

Administrator's Guide

HP Vertica Analytics Platform

Software Version: 7.0.x



Document Release Date:
12/18/2013

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

© Copyright 2006 - 2013 Hewlett-Packard Development Company, L.P.

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Contents

| | |
|---|----|
| Contents | 3 |
| Administration Overview | 51 |
| Managing Licenses | 52 |
| Copying Enterprise, Evaluation, and Flex Zone License Files | 52 |
| Obtaining a License Key File | 52 |
| Understanding HP Vertica Licenses | 52 |
| License Types | 53 |
| Installing or Upgrading a License Key | 54 |
| New HP Vertica License Installations | 54 |
| HP Vertica License Renewals or Upgrades | 54 |
| Uploading or Upgrading a License Key Using Administration Tools | 55 |
| Uploading or Upgrading a License Key Using Management Console | 55 |
| Flex Zone License Installations | 55 |
| Viewing Your License Status | 56 |
| Examining Your License Key | 56 |
| Viewing Your License Status | 57 |
| Viewing Your License Status Through MC | 57 |
| Calculating the Database Size | 57 |
| How HP Vertica Estimates Raw Data Size | 57 |
| Excluding Data From Raw Data Size Estimate | 58 |
| Evaluating Data Type Footprint Size | 58 |
| Using AUDIT to Estimate Database Size | 58 |
| Monitoring Database Size for License Compliance | 59 |
| Viewing the Current License State | 59 |
| Manually Running an Audit of the Entire Database | 59 |
| Targeted Auditing | 60 |
| Using Management Console to Monitor License Compliance | 60 |
| Managing License Warnings and Limits | 61 |
| Term License Warnings and Expiration | 61 |

| | |
|---|-----------|
| Data Size License Warnings and Remedies | 61 |
| If Your HP VerticaEnterprise Edition Database Size Exceeds Your Licensed Limits ... | 62 |
| If Your HP VerticaCommunity Edition Database Size Exceeds Your Licensed Limits .. | 62 |
| Configuring the Database | 63 |
| Configuration Procedure | 64 |
| IMPORTANT NOTES | 64 |
| Prepare Disk Storage Locations | 65 |
| Specifying Disk Storage Location During Installation | 65 |
| To Specify the Disk Storage Location When You install: | 66 |
| Notes | 66 |
| Specifying Disk Storage Location During Database Creation | 66 |
| Notes | 67 |
| Specifying Disk Storage Location on MC | 67 |
| Configuring Disk Usage to Optimize Performance | 67 |
| Using Shared Storage With HP Vertica | 68 |
| Viewing Database Storage Information | 68 |
| Disk Space Requirements for HP Vertica | 68 |
| Disk Space Requirements for Management Console | 68 |
| Prepare the Logical Schema Script | 68 |
| Prepare Data Files | 69 |
| How to Name Data Files | 69 |
| Prepare Load Scripts | 70 |
| Create an Optional Sample Query Script | 70 |
| Create an Empty Database | 72 |
| Creating a Database Name and Password | 72 |
| Database Passwords | 72 |
| Create an Empty Database Using MC | 74 |
| How to Create an Empty Database on an MC-managed Cluster | 74 |
| Notes | 75 |
| Create a Database Using Administration Tools | 76 |
| Create the Logical Schema | 77 |

| | |
|---|-----|
| Perform a Partial Data Load | 78 |
| Test the Database | 78 |
| Optimize Query Performance | 78 |
| Complete the Data Load | 79 |
| Test the Optimized Database | 79 |
| Set Up Incremental (Trickle) Loads | 80 |
| Implement Locales for International Data Sets | 81 |
| ICU Locale Support | 81 |
| Changing DB Locale for a Session | 81 |
| Specify the Default Locale for the Database | 82 |
| Override the Default Locale for a Session | 83 |
| Best Practices for Working with Locales | 83 |
| Server Locale | 84 |
| vsqI Client | 84 |
| ODBC Clients | 84 |
| JDBC and ADO.NET Clients | 85 |
| Notes and Restrictions | 85 |
| Change Transaction Isolation Levels | 87 |
| Notes | 88 |
| Configuration Parameters | 89 |
| Configuring HP Vertica Settings Using MC | 89 |
| Configuring HP Vertica At the Command Line | 91 |
| General Parameters | 91 |
| Tuple Mover Parameters | 94 |
| Epoch Management Parameters | 95 |
| Monitoring Parameters | 97 |
| Profiling Parameters | 99 |
| Security Parameters | 100 |
| Database Designer Parameters | 100 |
| Internationalization Parameters | 100 |
| Data Collector Parameters | 101 |

| | |
|---|-----|
| Kerberos Authentication Parameters | 102 |
| Designing a Logical Schema | 104 |
| Using Multiple Schemas | 105 |
| Multiple Schema Examples | 105 |
| Using Multiple Private Schemas | 105 |
| Using Combinations of Private and Shared Schemas | 107 |
| Creating Schemas | 107 |
| Specifying Objects in Multiple Schemas | 108 |
| Setting Search Paths | 108 |
| Creating Objects That Span Multiple Schemas | 110 |
| Tables in Schemas | 111 |
| About Base Tables | 111 |
| Automatic Projection Creation | 111 |
| About Temporary Tables | 112 |
| Local Temporary Tables | 113 |
| Automatic Projection Creation and Characteristics | 113 |
| Implementing Views | 115 |
| Creating Views | 115 |
| Using Views | 115 |
| Notes | 117 |
| Creating a Database Design | 118 |
| What Is a Design? | 118 |
| How Database Designer Creates a Design | 119 |
| Who Can Run Database Designer | 120 |
| Granting and Enabling the DBDUSER Role | 120 |
| Allowing the DBDUSER to Run Database Designer Using Management Console | 121 |
| Allowing the DBDUSER to Run Database Designer Programmatically | 122 |
| DBDUSER Capabilities and Limitations | 123 |
| Workflow for Running Database Designer | 124 |
| Specifying Parameters for Database Designer | 126 |
| Design Name | 126 |

| | |
|--|-----|
| Design Types | 126 |
| Comprehensive Design | 126 |
| Incremental Design | 127 |
| Optimization Objectives | 127 |
| Design Tables with Sample Data | 127 |
| Design Queries | 128 |
| Query Repository | 128 |
| K-Safety for Design | 128 |
| Replicated and Unsegmented Projections | 129 |
| Replicated Projections | 129 |
| Unsegmented Projections | 129 |
| Statistics Analysis | 130 |
| Building a Design | 130 |
| Resetting a Design | 131 |
| Deploying a Design | 132 |
| Deploying Designs Using Database Designer | 132 |
| Deploying Designs Manually | 133 |
| How to Create a Design | 133 |
| Using Management Console to Create a Design | 134 |
| Using the Wizard to Create a Design | 135 |
| Creating a Design Manually | 137 |
| Using Administration Tools to Create a Design | 140 |
| Creating Custom Designs | 142 |
| The Design Process | 142 |
| Planning Your Design | 143 |
| Design Requirements | 143 |
| Determining the Number of Projections to Use | 143 |
| Designing for K-Safety | 144 |
| Requirements for a K-Safe Physical Schema Design | 144 |
| Requirements for a Physical Schema Design with No K-Safety | 145 |
| Designing for Segmentation | 145 |

| | |
|---|-----|
| Design Fundamentals | 146 |
| Writing and Deploying Custom Projections | 146 |
| Anatomy of a Projection | 146 |
| Column List and Encoding | 147 |
| Base Query | 147 |
| Sort Order | 147 |
| Segmentation | 148 |
| Designing Superprojections | 148 |
| Minimizing Storage Requirements | 148 |
| Maximizing Query Performance | 149 |
| Designing Replicated Projections for K-Safety | 149 |
| Designing Segmented Projections for K-Safety | 150 |
| Segmenting Projections | 151 |
| Creating Buddy Projections | 151 |
| Projection Design for Merge Operations | 151 |
| Maximizing Projection Performance | 153 |
| Choosing Sort Order: Best Practices | 153 |
| Combine RLE and Sort Order | 153 |
| Maximize the Advantages of RLE | 154 |
| Put Lower Cardinality Column First for Functional Dependencies | 154 |
| Sort for Merge Joins | 155 |
| Sort on Columns in Important Queries | 156 |
| Sort Columns of Equal Cardinality By Size | 156 |
| Sort Foreign Key Columns First, From Low to High Distinct Cardinality | 156 |
| Prioritizing Column Access Speed | 156 |
| Projection Examples | 158 |
| New K-Safe=2 Database | 158 |
| Creating Segmented Projections Example | 158 |
| Creating Unsegmented Projections Example | 160 |
| Adding Node to a Database | 160 |
| Creating Segmented Projections Example | 161 |

| | |
|--|------------|
| Creating Unsegmented Projections Example | 162 |
| Implementing Security | 164 |
| Client Authentication | 164 |
| Connection Encryption | 164 |
| Client Authorization | 165 |
| Implementing Client Authentication | 166 |
| Supported Client Authentication Types | 166 |
| If You Want Communication Layer Authentication | 167 |
| Password Authentication | 168 |
| About Password Creation and Modification | 168 |
| Default Password Authentication | 168 |
| Profiles | 168 |
| How You Create and Modify Profiles | 169 |
| Password Expiration | 170 |
| Account Locking | 170 |
| How to Unlock a Locked Account | 170 |
| Password Guidelines | 171 |
| What to Use | 171 |
| What to Avoid | 172 |
| About External Authentication | 173 |
| Setting up Your Environment to Create Authentication Records | 173 |
| About Local Password Authentication | 174 |
| How to Create Authentication Records | 174 |
| How to Create Authentication Records | 174 |
| If You Do Not Specify a Client Authentication Method | 175 |
| Authentication Record Format and Rules | 175 |
| Formatting Rules | 178 |
| Configuring LDAP Authentication | 179 |
| LDAP Bind | 179 |
| LDAP Bind and Search | 180 |
| Using LDAP Over SSL and TLS | 181 |

| | |
|--|-----|
| LDAP Anonymous Binding | 182 |
| Configuring Multiple LDAP Servers | 182 |
| Configuring Ident Authentication | 182 |
| ClientAuthentication Records for Ident Authentication | 183 |
| Installing and Configuring an Ident Server | 184 |
| Example Authentication Records | 185 |
| Using an IP Range and Trust Authentication Method | 185 |
| Using Multiple Authentication Records | 185 |
| Record Order | 186 |
| How to Modify Authentication Records | 186 |
| Using the Administration Tools | 186 |
| Using the ClientAuthentication Configuration Parameter | 186 |
| Examples | 187 |
| Implementing Kerberos Authentication | 187 |
| Kerberos Prerequisites | 187 |
| Configure HP Vertica for Kerberos Authentication | 188 |
| Point machines at the KDC and configure realms | 192 |
| Configure Clients for Kerberos Authentication | 194 |
| Configure ODBC and vsql Clients on Linux, HP-UX, AIX, MAC OSX, and Solaris | 195 |
| Configure ADO.NET, ODBC, and vsql Clients on Windows | 197 |
| Windows KDC on Active Directory with Windows built-in Kerberos client and HP Vertica | 197 |
| Linux KDC with Windows-built-in Kerberos client and HP Vertica | 198 |
| Configuring Windows clients for Kerberos authentication | 198 |
| Authenticate and connect clients | 198 |
| Configure JDBC Clients on All Platforms | 199 |
| Determining the Client Authentication Method | 201 |
| Troubleshooting Kerberos Authentication | 202 |
| Server's principal name doesn't match the host name | 202 |
| JDBC client authentication | 204 |
| Working Domain Name Service (DNS) | 204 |

| | |
|--|-----|
| Clock synchronization | 204 |
| Encryption algorithm choices | 205 |
| Kerberos passwords | 205 |
| Using the ODBC Data Source Configuration utility | 205 |
| Implementing SSL | 206 |
| Certificate Authority | 206 |
| Public/private Keys | 206 |
| SSL Prerequisites | 207 |
| Prerequisites for SSL Server Authentication and SSL Encryption | 207 |
| Optional Prerequisites for SSL Server and Client Mutual Authentication | 208 |
| Generating Certifications and Keys | 208 |
| JDBC Certificates | 211 |
| Generating Certifications and Keys for MC | 211 |
| Signed Certificates | 211 |
| Self-Signed Certificates | 212 |
| Importing a New Certificate to MC | 213 |
| To Import a New Certificate | 213 |
| Distributing Certifications and Keys | 213 |
| Configuring SSL | 214 |
| To Enable SSL: | 214 |
| Configuring SSL for ODBC Clients | 214 |
| SSLMode Parameter | 215 |
| SSLKeyFile Parameter | 215 |
| SSLCertFile Parameter | 215 |
| Configuring SSL for JDBC Clients | 215 |
| To Configure JDBC: | 215 |
| To Enable the Driver for SSL | 216 |
| To Configure Troubleshooting | 216 |
| Requiring SSL for Client Connections | 217 |
| Managing Users and Privileges | 218 |
| About Database Users | 219 |

| | |
|--|-----|
| Types of Database Users | 220 |
| DBADMIN User | 220 |
| Object Owner | 220 |
| PUBLIC User | 221 |
| Creating a Database User | 221 |
| Notes | 221 |
| Example | 221 |
| Locking/unlocking a user's Database Access | 222 |
| Changing a user's Password | 223 |
| Changing a user's MC Password | 223 |
| About MC Users | 224 |
| Permission Group Types | 224 |
| MC User Types | 224 |
| Creating Users and Choosing an Authentication Method | 225 |
| Default MC Users | 225 |
| Creating an MC User | 225 |
| Prerequisites | 226 |
| Create a New MC-authenticated User | 226 |
| Create a New LDAP-authenticated User | 227 |
| How MC Validates New Users | 228 |
| Managing MC Users | 228 |
| Who Manages Users | 228 |
| What Kind of User Information You Can Manage | 229 |
| About User Names | 229 |
| About Database Privileges | 230 |
| Default Privileges for All Users | 230 |
| Default Privileges for MC Users | 231 |
| Privileges Required for Common Database Operations | 231 |
| Schemas | 231 |
| Tables | 231 |
| Views | 233 |

| | |
|---|-----|
| Projections | 234 |
| External Procedures | 234 |
| Libraries | 234 |
| User-Defined Functions | 235 |
| Sequences | 235 |
| Resource Pools | 236 |
| Users/Profiles/Roles | 237 |
| Object Visibility | 237 |
| I/O Operations | 238 |
| Comments | 240 |
| Transactions | 240 |
| Sessions | 241 |
| Tuning Operations | 241 |
| Privileges That Can Be Granted on Objects | 242 |
| Database Privileges | 242 |
| Schema Privileges | 243 |
| Schema Privileges and the Search Path | 243 |
| Table Privileges | 244 |
| Projection Privileges | 245 |
| Explicit Projection Creation and Privileges | 245 |
| Implicit Projection Creation and Privileges | 246 |
| Selecting From Projections | 246 |
| Dropping Projections | 246 |
| View Privileges | 246 |
| Sequence Privileges | 247 |
| External Procedure Privileges | 248 |
| User-Defined Function Privileges | 248 |
| Library Privileges | 249 |
| Resource Pool Privileges | 249 |
| Storage Location Privileges | 249 |
| Role, profile, and User Privileges | 250 |

| | |
|---|-----|
| Metadata Privileges | 251 |
| I/O Privileges | 252 |
| Comment Privileges | 253 |
| Transaction Privileges | 253 |
| Session Privileges | 254 |
| Tuning Privileges | 254 |
| Granting and Revoking Privileges | 254 |
| About Superuser Privileges | 254 |
| About Schema Owner Privileges | 255 |
| About Object Owner Privileges | 255 |
| How to Grant Privileges | 256 |
| How to Revoke Privileges | 256 |
| Privilege Ownership Chains | 258 |
| Modifying Privileges | 260 |
| Changing a Table Owner | 260 |
| Notes | 260 |
| Example | 260 |
| Table Reassignment with Sequences | 262 |
| Changing a Sequence Owner | 263 |
| Example | 263 |
| Viewing Privileges Granted on Objects | 264 |
| About Database Roles | 267 |
| Role Hierarchies | 267 |
| Creating and Using a Role | 267 |
| Roles on Management Console | 267 |
| Types of Database Roles | 268 |
| DBADMIN Role | 268 |
| View a List of Database Superusers | 269 |
| DBDUSER Role | 269 |
| PSEUDOSUPERUSER Role | 270 |
| PUBLIC Role | 270 |

| | |
|--|-----|
| Example | 271 |
| Default Roles for Database Users | 271 |
| Notes | 272 |
| Using Database Roles | 272 |
| Role Hierarchy | 273 |
| Example | 273 |
| Creating Database Roles | 275 |
| Deleting Database Roles | 275 |
| Granting Privileges to Roles | 275 |
| Example | 275 |
| Revoking Privileges From Roles | 276 |
| Granting Access to Database Roles | 276 |
| Example | 277 |
| Revoking Access From Database Roles | 278 |
| Example | 278 |
| Granting Administrative Access to a Role | 279 |
| Example | 279 |
| Revoking Administrative Access From a Role | 280 |
| Example | 280 |
| Enabling Roles | 280 |
| Disabling Roles | 281 |
| Viewing Enabled and Available Roles | 281 |
| Viewing Named Roles | 282 |
| Viewing a user's Role | 282 |
| How to View a user's Role | 282 |
| Users | 283 |
| Roles | 284 |
| Grants | 284 |
| Viewing User Roles on Management Console | 284 |
| About MC Privileges and Roles | 285 |
| MC Permission Groups | 285 |

| | |
|--|-----|
| MC's Configuration Privileges and Database Access | 285 |
| MC Configuration Privileges | 286 |
| MC Configuration Privileges By User Role | 286 |
| SUPER Role (mc) | 288 |
| ADMIN Role (mc) | 289 |
| About the MC Database Administrator Role | 290 |
| IT Role (mc) | 290 |
| About the MC IT (database) Role | 291 |
| NONE Role (mc) | 291 |
| MC Database Privileges | 292 |
| MC Database Privileges By Role | 293 |
| ADMIN Role (db) | 294 |
| About the ADMIN (MC configuration) Role | 295 |
| IT Role (db) | 295 |
| About the IT (MC configuration) Role | 295 |
| USER Role (db) | 296 |
| Granting Database Access to MC Users | 296 |
| Prerequisites | 296 |
| Grant a Database-Level Role to an MC user: | 297 |
| How MC Validates New Users | 297 |
| Mapping an MC User to a Database user's Privileges | 298 |
| How to Map an MC User to a Database User | 298 |
| What If You Map the Wrong Permissions | 301 |
| Adding Multiple MC Users to a Database | 301 |
| How to Find Out an MC user's Database Role | 302 |
| Adding Multiple Users to MC-managed Databases | 303 |
| Before You Start | 303 |
| How to Add Multiple Users to a Database | 304 |
| MC Mapping Matrix | 304 |
| Using the Administration Tools | 307 |
| Running the Administration Tools | 307 |

| | |
|--|-----|
| First Time Only | 308 |
| Between Dialogs | 308 |
| Using the Administration Tools Interface | 309 |
| Enter [Return] | 309 |
| OK - Cancel - Help | 309 |
| Menu Dialogs | 310 |
| List Dialogs | 310 |
| Form Dialogs | 310 |
| Help Buttons | 311 |
| K-Safety Support in Administration Tools | 311 |
| Notes for Remote Terminal Users | 312 |
| Using the Administration Tools Help | 312 |
| In a Menu Dialog | 313 |
| In a Dialog Box | 314 |
| Scrolling | 314 |
| Password Authentication | 314 |
| Distributing Changes Made to the Administration Tools Metadata | 315 |
| Administration Tools and Management Console | 315 |
| Administration Tools Reference | 318 |
| Viewing Database Cluster State | 318 |
| Connecting to the Database | 319 |
| Starting the Database | 320 |
| Starting the Database Using MC | 320 |
| Starting the Database Using the Administration Tools | 320 |
| Starting the Database At the Command Line | 321 |
| Stopping a Database | 321 |
| Error | 321 |
| Description | 322 |
| Resolution | 322 |
| Controlling Sessions | 324 |
| Notes | 325 |

| | |
|--|------------|
| Restarting HP Vertica on Host | 326 |
| Configuration Menu Item | 327 |
| Creating a Database | 327 |
| Dropping a Database | 329 |
| Notes | 329 |
| Viewing a Database | 330 |
| Setting the Restart Policy | 330 |
| Best Practice for Restoring Failed Hardware | 331 |
| Installing External Procedure Executable Files | 332 |
| Advanced Menu Options | 333 |
| Rolling Back Database to the Last Good Epoch | 333 |
| Important note: | 333 |
| Stopping HP Vertica on Host | 334 |
| Killing the HP Vertica Process on Host | 335 |
| Upgrading an Enterprise or Evaluation License Key | 336 |
| Managing Clusters | 337 |
| Using Cluster Management | 337 |
| Using the Administration Tools | 337 |
| Administration Tools Metadata | 337 |
| Writing Administration Tools Scripts | 338 |
| Syntax | 338 |
| Parameters | 338 |
| Tools | 339 |
| Using Management Console | 349 |
| Connecting to MC | 349 |
| Managing Client Connections on MC | 350 |
| Managing Database Clusters on MC | 351 |
| Create an Empty Database Using MC | 352 |
| How to Create an Empty Database on an MC-managed Cluster | 352 |
| Notes | 353 |
| Import an Existing Database Into MC | 354 |

| | |
|---|------------|
| How to Import an Existing Database on the Cluster | 354 |
| Using MC on an AWS Cluster | 355 |
| Managing MC Settings | 355 |
| Modifying Database-Specific Settings | 355 |
| Changing MC or Agent Ports | 356 |
| If You Need to Change the MC Default Ports | 356 |
| How to Change the Agent Port | 356 |
| Change the Agent Port in config.py | 356 |
| Change the Agent Port on MC | 357 |
| How to Change the MC Port | 357 |
| Backing Up MC | 357 |
| Troubleshooting Management Console | 359 |
| What You Can diagnose: | 359 |
| Viewing the MC Log | 359 |
| Exporting the User Audit Log | 360 |
| To Manually Export MC User Activity | 360 |
| Restarting MC | 361 |
| How to Restart MC Through the MC Interface (using Your browser) | 361 |
| How to Restart MC At the Command Line | 361 |
| Starting over | 362 |
| Resetting MC to Pre-Configured State | 362 |
| Avoiding MC Self-Signed Certificate Expiration | 362 |
| Operating the Database | 363 |
| Starting and Stopping the Database | 363 |
| Starting the Database | 363 |
| Starting the Database Using MC | 363 |
| Starting the Database Using the Administration Tools | 363 |
| Starting the Database At the Command Line | 364 |
| Stopping the Database | 364 |
| Stopping a Running Database Using MC | 365 |
| Stopping a Running Database Using the Administration Tools | 365 |

| | |
|---|------------|
| Stopping a Running Database At the Command Line | 365 |
| Working with the HP Vertica Index Tool | 367 |
| Syntax | 367 |
| Parameters | 368 |
| Permissions | 368 |
| Controlling Expression Analysis | 368 |
| Performance and CRC | 368 |
| Running the Reindex Option | 369 |
| Running the CheckCRC Option | 370 |
| Handling CheckCRC Errors | 371 |
| Running the Checksort Option | 371 |
| Viewing Details of Index Tool Results | 372 |
| Working with Tables | 375 |
| Creating Base Tables | 375 |
| Creating Tables Using the /*+direct*/ Clause | 375 |
| Automatic Projection Creation | 376 |
| Characteristics of Default Automatic Projections | 377 |
| Creating a Table Like Another | 378 |
| Epochs and Node Recovery | 378 |
| Storage Location and Policies for New Tables | 379 |
| Simple Example | 379 |
| Using CREATE TABLE LIKE | 379 |
| Creating Temporary Tables | 380 |
| Global Temporary Tables | 381 |
| Local Temporary Tables | 381 |
| Creating a Temp Table Using the /*+direct*/ Clause | 381 |
| Characteristics of Default Automatic Projections | 382 |
| Preserving GLOBAL Temporary Table Data for a Transaction or Session | 383 |
| Specifying Column Encoding | 384 |
| Creating External Tables | 384 |
| Required Permissions for External Tables | 385 |

| | |
|--|-----|
| COPY Statement Definition | 385 |
| Developing User-Defined Load (UDL) Functions for External Tables | 385 |
| Examples | 385 |
| Validating External Tables | 386 |
| Limiting the Maximum Number of Exceptions | 386 |
| Working with External Tables | 386 |
| Managing Resources for External Tables | 387 |
| Backing Up and Restoring External Tables | 387 |
| Using Sequences and Identity Columns in External Tables | 387 |
| Viewing External Table Definitions | 387 |
| External Table DML Support | 387 |
| Using External Table Values | 388 |
| Using External Tables | 389 |
| Using CREATE EXTERNAL TABLE AS COPY Statement | 390 |
| Storing HP Vertica Data in External Tables | 390 |
| Using External Tables with User-Defined Load (UDL) Functions | 390 |
| Organizing External Table Data | 390 |
| Altering Table Definitions | 391 |
| External Table Restrictions | 391 |
| Exclusive ALTER TABLE Clauses | 391 |
| Using Consecutive ALTER TABLE Commands | 392 |
| Adding Table Columns | 392 |
| Updating Associated Table Views | 393 |
| Specifying Default Expressions | 393 |
| About Using Volatile Functions | 393 |
| Updating Associated Table Views | 393 |
| Altering Table Columns | 394 |
| Adding Columns with a Default Derived Expression | 394 |
| Add a Default Column Value Derived From Another Column | 394 |
| Add a Default Column Value Derived From a UDSF | 396 |
| Changing a column's Data Type | 397 |

| | |
|--|-----|
| Examples | 397 |
| How to Perform an Illegitimate Column Conversion | 398 |
| Adding Constraints on Columns | 400 |
| Adding and Removing NOT NULL Constraints | 400 |
| Examples | 401 |
| Dropping a Table Column | 401 |
| Restrictions | 401 |
| Using CASCADE to Force a Drop | 401 |
| Examples | 402 |
| Moving a Table to Another Schema | 403 |
| Changing a Table Owner | 403 |
| Notes | 404 |
| Example | 404 |
| Table Reassignment with Sequences | 406 |
| Changing a Sequence Owner | 407 |
| Example | 407 |
| Renaming Tables | 407 |
| Using Rename to Swap Tables Within a Schema | 408 |
| Using Named Sequences | 409 |
| Types of Incrementing Value Objects | 409 |
| Using a Sequence with an Auto_Increment or Identity Column | 410 |
| Named Sequence Functions | 410 |
| Using DDL Commands and Functions With Named Sequences | 411 |
| Creating Sequences | 411 |
| Altering Sequences | 413 |
| Examples | 413 |
| Distributed Sequences | 414 |
| Loading Sequences | 424 |
| Creating and Instantiating a Sequence | 424 |
| Using a Sequence in an INSERT Command | 424 |
| Dropping Sequences | 425 |

| | |
|--|------------|
| Example | 425 |
| Synchronizing Table Data with MERGE | 425 |
| Optimized Versus Non-Optimized MERGE | 426 |
| Troubleshooting the MERGE Statement | 428 |
| Dropping and Truncating Tables | 429 |
| Dropping Tables | 429 |
| Truncating Tables | 429 |
| About Constraints | 431 |
| Adding Constraints | 432 |
| Adding Column Constraints with CREATE TABLE | 432 |
| Adding Two Constraints on a Column | 433 |
| Adding a Foreign Key Constraint on a Column | 433 |
| Adding Multicolumn Constraints | 434 |
| Adding Constraints on Tables with Existing Data | 435 |
| Adding and Changing Constraints on Columns Using ALTER TABLE | 435 |
| Adding and Dropping NOT NULL Column Constraints | 436 |
| Enforcing Constraints | 436 |
| Primary Key Constraints | 437 |
| Foreign Key Constraints | 437 |
| Examples | 438 |
| Unique Constraints | 439 |
| Not NULL Constraints | 440 |
| Dropping Constraints | 441 |
| Notes | 441 |
| Enforcing Primary Key and Foreign Key Constraints | 443 |
| Enforcing Primary Key Constraints | 443 |
| Enforcing Foreign Key Constraints | 443 |
| Detecting Constraint Violations Before You Commit Data | 443 |
| Detecting Constraint Violations | 444 |
| Fixing Constraint Violations | 449 |
| Reenabling Error Reporting | 452 |

| | |
|---|-----|
| Working with Table Partitions | 453 |
| Differences Between Partitioning and Segmentation | 453 |
| Partition Operations | 453 |
| Defining Partitions | 454 |
| Table 3: Partitioning Expression and Results | 455 |
| Partitioning By Year and Month | 455 |
| Restrictions on Partitioning Expressions | 455 |
| Best Practices for Partitioning | 456 |
| Dropping Partitions | 456 |
| Examples | 457 |
| Partitioning and Segmenting Data | 458 |
| Partitioning and Data Storage | 460 |
| Partitions and ROS Containers | 460 |
| Partition Pruning | 460 |
| Managing Partitions | 460 |
| Notes | 462 |
| Partitioning, repartitioning, and Reorganizing Tables | 462 |
| Reorganizing Data After Partitioning | 463 |
| Monitoring Reorganization | 463 |
| Auto Partitioning | 464 |
| Examples | 464 |
| Eliminating Partitions | 466 |
| Making Past Partitions Eligible for Elimination | 467 |
| Verifying the ROS Merge | 468 |
| Examples | 468 |
| Moving Partitions | 469 |
| Archiving Steps | 470 |
| Preparing and Moving Partitions | 470 |
| Creating a Snapshot of the Intermediate Table | 470 |
| Copying the Config File to the Storage Location | 471 |
| Drop the Intermediate Table | 471 |

| | |
|---|------------|
| Restoring Archived Partitions | 471 |
| Bulk Loading Data | 473 |
| Checking Data Format Before or After Loading | 474 |
| Converting Files Before Loading Data | 475 |
| Checking UTF-8 Compliance After Loading Data | 475 |
| Performing the Initial Database Load | 475 |
| Extracting Data From an Existing Database | 476 |
| Checking for Delimiter Characters in Load Data | 476 |
| Moving Data From an Existing Database to HP Vertica Nodes | 476 |
| Loading From a Local Hard Disk | 477 |
| Loading Over the Network | 477 |
| Loading From Windows | 477 |
| Using Load Scripts | 477 |
| Using Absolute Paths in a Load Script | 478 |
| Running a Load Script | 478 |
| Using COPY and COPY LOCAL | 479 |
| Copying Data From an HP Vertica Client | 479 |
| Transforming Data During Loads | 480 |
| Understanding Transformation Requirements | 480 |
| Loading FLOAT Values | 480 |
| Using Expressions in COPY Statements | 480 |
| Handling Expression Errors | 481 |
| Transformation Example | 481 |
| Deriving Table Columns From Data File Columns | 482 |
| Specifying COPY FROM Options | 483 |
| Loading From STDIN | 483 |
| Loading From a Specific Path | 483 |
| Loading with Wildcards (glob) ON ANY NODE | 484 |
| Loading From a Local Client | 484 |
| Choosing a Load Method | 484 |
| Loading Directly into WOS (AUTO) | 485 |

| | |
|---|-----|
| Loading Directly to ROS (DIRECT) | 485 |
| Loading Data Incrementally (TRICKLE) | 485 |
| Loading Data Without Committing Results (NO COMMIT) | 485 |
| Using NO COMMIT to Detect Constraint Violations | 486 |
| Using COPY Interactively | 486 |
| Canceling a COPY Statement | 486 |
| Specifying a COPY Parser | 487 |
| Specifying Load Metadata | 487 |
| Interpreting Last Column End of Row Values | 489 |
| Using a Single End of Row Definition | 489 |
| Using a Delimiter and Record Terminator End of Row Definition | 490 |
| Loading UTF-8 Format Data | 491 |
| Loading Special Characters As Literals | 491 |
| Using a Custom Column Separator (DELIMITER) | 491 |
| Using a Custom Column Option DELIMITER | 492 |
| Defining a Null Value (NULL) | 492 |
| Loading NULL Values | 493 |
| Filling Columns with Trailing Nulls (TRAILING NULLCOLS) | 493 |
| Attempting to Fill a NOT NULL Column with TRAILING NULLCOLS | 494 |
| Changing the Default Escape Character (ESCAPE AS) | 495 |
| Eliminating Escape Character Handling | 495 |
| Delimiting Characters (ENCLOSED BY) | 496 |
| Using ENCLOSED BY for a Single Column | 497 |
| Specifying a Custom End of Record String (RECORD TERMINATOR) | 497 |
| Examples | 498 |
| Loading Native Varchar Data | 498 |
| Loading Binary (Native) Data | 499 |
| Loading Hexadecimal, Octal, and Bitstring Data | 499 |
| Hexadecimal Data | 501 |
| Octal Data | 501 |
| BitString Data | 501 |

| | |
|---|-----|
| Examples | 501 |
| Loading Fixed-Width Format Data | 502 |
| Supported Options for Fixed-Width Data Loads | 502 |
| Using Nulls in Fixed-Width Data | 503 |
| Defining a Null Character (Statement Level) | 503 |
| Defining a Custom Record Terminator | 504 |
| Copying Fixed-Width Data | 504 |
| Skipping Content in Fixed-Width Data | 505 |
| Trimming Characters in Fixed-Width Data Loads | 506 |
| Using Padding in Fixed-Width Data Loads | 506 |
| Ignoring Columns and Fields in the Load File | 507 |
| Using the FILLER Parameter | 507 |
| FILLER Parameter Examples | 508 |
| Loading Data into Pre-Join Projections | 508 |
| Foreign and Primary Key Constraints | 509 |
| Concurrent Loads into Pre-Join Projections | 510 |
| Using Parallel Load Streams | 511 |
| Monitoring COPY Loads and Metrics | 512 |
| Using HP Vertica Functions | 512 |
| Using the LOAD_STREAMS System Table | 512 |
| Using the STREAM NAME Parameter | 513 |
| Other LOAD_STREAMS Columns for COPY Metrics | 513 |
| Capturing Load Exceptions and Rejections | 514 |
| COPY Parameters for Handling Rejections and Exceptions | 515 |
| Enforcing Truncating or Rejecting Rows (ENFORCELENGTH) | 516 |
| Specifying Maximum Rejections Before a Load Fails (REJECTMAX) | 516 |
| Aborting Data Loads for Any Error (ABORT ON ERROR) | 516 |
| Understanding Row Rejections and Rollback Errors | 517 |
| Saving Load Exceptions (EXCEPTIONS) | 518 |
| Saving Load Rejections (REJECTED DATA) | 519 |
| Saving Rejected Data to a Table | 520 |

| | |
|--|------------|
| Rejection Records for Table Files | 521 |
| Querying a Rejection Records Table | 521 |
| Exporting the Rejected Records Table | 523 |
| COPY Rejected Data and Exception Files | 524 |
| Specifying Rejected Data and Exceptions Files | 525 |
| Saving Rejected Data and Exceptions Files to a Single Server | 525 |
| Using VSQL Variables for Rejected Data and Exceptions Files | 526 |
| COPY LOCAL Rejection and Exception Files | 527 |
| Specifying Rejected Data and Exceptions Files | 527 |
| Referential Integrity Load Violation | 528 |
| Trickle Loading Data | 529 |
| Using INSERT, UPDATE, and DELETE | 529 |
| WOS Overflow | 529 |
| Copying and Exporting Data | 531 |
| Moving Data Directly Between Databases | 531 |
| Creating SQL Scripts to Export Data | 531 |
| Exporting Data | 532 |
| Exporting Identity Columns | 532 |
| Examples of Exporting Data | 533 |
| Copying Data | 534 |
| Importing Identity Columns | 534 |
| Examples | 535 |
| Using Public and Private IP Networks | 536 |
| Identify the Public Network to HP Vertica | 536 |
| Identify the Database or Nodes Used for Import/Export | 537 |
| Using EXPORT Functions | 538 |
| Saving Scripts for Export Functions | 538 |
| Exporting the Catalog | 539 |
| Function Summary | 539 |
| Exporting All Catalog Objects | 539 |
| Projection Considerations | 540 |

| | |
|--|------------|
| Exporting Database Designer Schema and Designs | 540 |
| Exporting Table Objects | 540 |
| Exporting Tables | 541 |
| Function Syntax | 542 |
| Exporting All Tables and Related Objects | 542 |
| Exporting a List Tables | 542 |
| Exporting a Single Table or Object | 543 |
| Exporting Objects | 543 |
| Function Syntax | 544 |
| Exporting All Objects | 544 |
| Exporting a List of Objects | 545 |
| Exporting a Single Object | 546 |
| Bulk Deleting and Purging Data | 547 |
| Choosing the Right Technique for Deleting Data | 548 |
| Best Practices for DELETE and UPDATE | 549 |
| Performance Considerations for DELETE and UPDATE Queries | 549 |
| Optimizing DELETES and UPDATES for Performance | 550 |
| Projection Column Requirements for Optimized Deletes | 550 |
| Optimized Deletes in Subqueries | 550 |
| Projection Sort Order for Optimizing Deletes | 551 |
| Purging Deleted Data | 553 |
| Setting a Purge Policy | 553 |
| Specifying the Time for Which Delete Data Is Saved | 554 |
| Specifying the Number of Epochs That Are Saved | 554 |
| Disabling Purge | 555 |
| Manually Purging Data | 555 |
| Managing the Database | 557 |
| Connection Load Balancing | 557 |
| Native Connection Load Balancing Overview | 557 |
| IPVS Overview | 558 |
| Choosing Whether to Use Native Connection Load Balancing or IPVS | 558 |

| | |
|---|-----|
| About Native Connection Load Balancing | 559 |
| Load Balancing Schemes | 560 |
| Enabling and Disabling Native Connection Load Balancing | 560 |
| Resetting the Load Balancing State | 561 |
| Monitoring Native Connection Load Balancing | 561 |
| Determining to which Node a Client Has Connected | 562 |
| Connection Load Balancing Using IPVS | 563 |
| Configuring HP Vertica Nodes | 565 |
| Notes | 565 |
| Set Up the Loopback Interface | 566 |
| Disable Address Resolution Protocol (ARP) | 567 |
| Configuring the Directors | 569 |
| Install the HP Vertica IPVS Load Balancer Package | 569 |
| Before You Begin | 569 |
| If You Are Using Red Hat Enterprise Linux 5.x: | 569 |
| If You Are Using Red Hat Enterprise Linux 6.x: | 570 |
| Configure the HP Vertica IPVS Load Balancer | 570 |
| Public and Private IPs | 571 |
| Set up the HP Vertica IPVS Load Balancer Configuration File | 572 |
| Connecting to the Virtual IP (VIP) | 573 |
| Monitoring Shared Node Connections | 574 |
| Determining Where Connections Are Going | 575 |
| Virtual IP Connection Problems | 577 |
| Issue | 577 |
| Resolution | 577 |
| Issue | 578 |
| Resolution | 578 |
| Expected E-mail Messages From the Keepalived Daemon | 578 |
| Troubleshooting Keepalived Issues | 579 |
| Managing Nodes | 581 |
| Fault Groups | 581 |

| | |
|--|-----|
| About the Fault Group Script | 581 |
| Creating Fault Groups | 583 |
| Modifying Fault Groups | 585 |
| How to modify a fault group | 585 |
| Dropping Fault Groups | 587 |
| How to drop a fault group | 587 |
| How to remove all fault groups | 587 |
| To add nodes back to a fault group | 587 |
| Monitoring Fault Groups | 588 |
| Monitoring fault groups through system tables | 588 |
| Monitoring fault groups through Management Console | 588 |
| Large Cluster | 589 |
| Control nodes on large clusters | 589 |
| Control nodes on small clusters | 590 |
| Planning a Large Cluster Arrangement | 590 |
| Installing a Large Cluster | 591 |
| If you want to install a new large cluster | 591 |
| Sample rack-based cluster hosts topology | 592 |
| If you want to expand an existing cluster | 593 |
| Defining and Realigning Control Nodes on an Existing Cluster | 593 |
| Rebalancing Large Clusters | 595 |
| How to rebalance the cluster | 595 |
| How long will rebalance take? | 596 |
| Expanding the Database for Large Cluster | 596 |
| Monitoring Large Clusters | 596 |
| Large Cluster Best Practices | 597 |
| Planning the number of control nodes | 597 |
| Allocate standby nodes | 598 |
| Plan for cluster growth | 598 |
| Write custom fault groups | 599 |
| Use segmented projections | 599 |

| | |
|--|-----|
| Use the Database Designer | 599 |
| Elastic Cluster | 599 |
| The Elastic Cluster Scaling Factor | 600 |
| Enabling and Disabling Elastic Cluster | 600 |
| Scaling Factor Defaults | 600 |
| Viewing Scaling Factor Settings | 601 |
| Setting the Scaling Factor | 601 |
| Local Data Segmentation | 602 |
| Enabling and Disabling Local Segmentation | 602 |
| Elastic Cluster Best Practices | 603 |
| When to Enable Local Data Segmentation | 603 |
| Upgraded Database Consideration | 603 |
| Monitoring Elastic Cluster Rebalancing | 604 |
| Historical Rebalance Information | 604 |
| Adding Nodes | 605 |
| Adding Hosts to a Cluster | 606 |
| Prerequisites and Restrictions | 606 |
| Procedure to Add Hosts | 606 |
| Examples: | 607 |
| Adding Nodes to a Database | 608 |
| To Add Nodes to a Database Using MC | 608 |
| To Add Nodes to a Database Using the Administration Tools: | 608 |
| Removing Nodes | 610 |
| Lowering the K-Safety Level to Allow for Node Removal | 610 |
| Removing Nodes From a Database | 610 |
| Prerequisites | 611 |
| Remove Unused Hosts From the Database Using MC | 611 |
| Remove Unused Hosts From the Database Using the Administration Tools ... | 611 |
| Removing Hosts From a Cluster | 612 |
| Prerequisites | 612 |
| Procedure to Remove Hosts | 612 |

| | |
|--|-----|
| Replacing Nodes | 614 |
| Prerequisites | 614 |
| Best Practice for Restoring Failed Hardware | 614 |
| Replacing a Node Using the Same Name and IP Address | 615 |
| Replacing a Failed Node Using a node with Different IP Address | 616 |
| Replacing a Functioning Node Using a Different Name and IP Address | 617 |
| Using the Administration Tools to Replace Nodes | 617 |
| Replace the Original Host with a New Host Using the Administration Tools | 617 |
| Using the Management Console to Replace Nodes | 618 |
| Rebalancing Data Across Nodes | 620 |
| K-safety and Rebalancing | 620 |
| Rebalancing Failure and Projections | 620 |
| Permissions | 621 |
| Rebalancing Data Using the Administration Tools UI | 621 |
| Rebalancing Data Using Management Console | 622 |
| Rebalancing Data Using SQL Functions | 622 |
| Redistributing Configuration Files to Nodes | 622 |
| Changing the IP Addresses of an HP Vertica Cluster | 623 |
| Stopping and Starting Nodes on MC | 625 |
| Managing Disk Space | 626 |
| Monitoring Disk Space Usage | 626 |
| Adding Disk Space to a Node | 626 |
| Replacing Failed Disks | 628 |
| Catalog and Data Files | 628 |
| Understanding the Catalog Directory | 629 |
| Reclaiming Disk Space From Deleted Records | 631 |
| Rebuilding a Table | 631 |
| Notes | 631 |
| Managing Tuple Mover Operations | 633 |
| Understanding the Tuple Mover | 634 |
| Moveout | 634 |

| | |
|--|-----|
| ROS Containers | 635 |
| Mergeout | 635 |
| Mergeout of Deletion Markers | 636 |
| Tuning the Tuple Mover | 636 |
| Tuple Mover Configuration Parameters | 637 |
| Resource Pool Settings | 638 |
| Loading Data | 639 |
| Using More Threads | 639 |
| Active Data Partitions | 639 |
| Managing Workloads | 641 |
| Statements | 641 |
| System Tables | 642 |
| The Resource Manager | 642 |
| Resource Manager Impact on Query Execution | 643 |
| Resource Pool Architecture | 644 |
| Modifying and Creating Resource Pools | 644 |
| Monitoring Resource Pools and Resource Usage By Queries | 644 |
| Examples | 644 |
| User Profiles | 648 |
| Example | 648 |
| Target Memory Determination for Queries in Concurrent Environments | 650 |
| Managing Resources At Query Run Time | 650 |
| Setting Run-Time Priority for the Resource Pool | 651 |
| Prioritizing Queries Within a Resource Pool | 651 |
| How to Set Run-Time Priority and Run-Time Priority Threshold | 651 |
| Changing Run-Time Priority of a Running Query | 652 |
| How To Change the Run-Time Priority of a Running Query | 652 |
| Using CHANGE_RUNTIME_PRIORITY | 652 |
| Restoring Resource Manager Defaults | 653 |
| Best Practices for Managing Workload Resources | 654 |
| Basic Principles for Scalability and Concurrency Tuning | 654 |

| | |
|--|-----|
| Guidelines for Setting Pool Parameters | 654 |
| Setting a Run-Time Limit for Queries | 659 |
| Example: | 660 |
| Using User-Defined Pools and User-Profiles for Workload Management | 661 |
| Scenario: Periodic Batch Loads | 661 |
| Scenario: The CEO Query | 662 |
| Scenario: Preventing Run-Away Queries | 663 |
| Scenario: Restricting Resource Usage of Ad Hoc Query Application | 664 |
| Scenario: Setting a Hard Limit on Concurrency For An Application | 665 |
| Scenario: Handling Mixed Workloads (Batch vs. Interactive) | 666 |
| Scenario: Setting Priorities on Queries Issued By Different Users | 667 |
| Scenario: Continuous Load and Query | 668 |
| Scenario: Prioritizing Short Queries At Run Time | 669 |
| Scenario: Dropping the Runtime Priority of Long Queries | 669 |
| Tuning the Built-In Pools | 671 |
| Scenario: Restricting HP Vertica to Take Only 60% of Memory | 671 |
| Scenario: Tuning for Recovery | 671 |
| Scenario: Tuning for Refresh | 671 |
| Scenario: Tuning Tuple Mover Pool Settings | 672 |
| Reducing Query Run-Time | 672 |
| Real-Time Profiling | 673 |
| Managing System Resource Usage | 674 |
| Managing Sessions | 674 |
| Viewing Sessions | 675 |
| Interrupting and Closing Sessions | 675 |
| Controlling Sessions | 676 |
| Managing Load Streams | 677 |
| Working With Storage Locations | 679 |
| How HP Vertica Uses Storage Locations | 679 |
| Viewing Storage Locations and Policies | 680 |
| Viewing Disk Storage Information | 680 |

| | |
|--|-----|
| Viewing Location Labels | 680 |
| Viewing Storage Tiers | 681 |
| Viewing Storage Policies | 681 |
| Adding Storage Locations | 682 |
| Planning Storage Locations | 682 |
| Adding the Location | 683 |
| Storage Location Subdirectories | 684 |
| Adding Labeled Storage Locations | 684 |
| Adding a Storage Location for USER Access | 685 |
| Altering Storage Location Use | 686 |
| USER Storage Location Restrictions | 686 |
| Effects of Altering Storage Location Use | 686 |
| Altering Location Labels | 687 |
| Adding a Location Label | 687 |
| Removing a Location Label | 687 |
| Effects of Altering a Location Label | 688 |
| Creating Storage Policies | 688 |
| Creating Policies Based on Storage Performance | 689 |
| Storage Levels and Priorities | 689 |
| Using the SET_OBJECT_STORAGE_POLICY Function | 690 |
| Effects of Creating Storage Policies | 690 |
| Moving Data Storage Locations | 691 |
| Moving Data Storage While Setting a Storage Policy | 692 |
| Effects of Moving a Storage Location | 692 |
| Clearing Storage Policies | 693 |
| Effects on Same-Name Storage Policies | 693 |
| Measuring Storage Performance | 694 |
| Measuring Performance on a Running HP Vertica Database | 695 |
| Measuring Performance Before a Cluster Is Set Up | 696 |
| Setting Storage Performance | 696 |
| How HP Vertica Uses Location Performance Settings | 696 |

| | |
|--|------------|
| Using Location Performance Settings With Storage Policies | 697 |
| Dropping Storage Locations | 697 |
| Altering Storage Locations Before Dropping Them | 697 |
| Dropping USER Storage Locations | 698 |
| Retiring Storage Locations | 698 |
| Restoring Retired Storage Locations | 698 |
| Backing Up and Restoring the Database | 699 |
| Compatibility Requirements for Using vbr.py | 699 |
| Automating Regular Backups | 699 |
| Types of Backups | 699 |
| Full Backups | 700 |
| Object-Level Backups | 700 |
| Hard Link Local Backups | 701 |
| When to Back up the Database | 701 |
| Configuring Backup Hosts | 701 |
| Configuring Single-Node Database Hosts for Backup | 702 |
| Creating Configuration Files for Backup Hosts | 702 |
| Estimating Backup Host Disk Requirements | 702 |
| Estimating Log File Disk Requirements | 703 |
| Making Backup Hosts Accessible | 703 |
| Setting Up Passwordless SSH Access | 704 |
| Testing SSH Access | 704 |
| Changing the Default SSH Port on Backup Hosts | 705 |
| Increasing the SSH Maximum Connection Settings for a Backup Host | 705 |
| Copying Rsync and Python to the Backup Hosts | 706 |
| Configuring Hard Link Local Backup Hosts | 706 |
| Listing Host Names | 706 |
| Creating vbr.py Configuration Files | 708 |
| Specifying a Backup Name | 708 |
| Backing Up the Vertica Configuration File | 709 |
| Saving Multiple Restore Points | 709 |

| | |
|---|-----|
| Specifying Full or Object-Level Backups | 709 |
| Entering the User Name | 710 |
| Saving the Account Password | 710 |
| Specifying the Backup Host and Directory | 710 |
| Saving the Configuration File | 711 |
| Continuing to Advanced Settings | 711 |
| Sample Configuration File | 711 |
| Changing the Overwrite Parameter Value | 712 |
| Configuring Required VBR Parameters | 712 |
| Sample Session Configuring Required Parameters | 713 |
| Configuring Advanced VBR Parameters | 713 |
| Example of Configuring Advanced Parameters | 714 |
| Configuring the Hard Link Local Parameter | 714 |
| Restrictions for Backup Encryption Option | 715 |
| Example Backup Configuration File | 715 |
| Using Hard File Link Local Backups | 717 |
| Planning Hard Link Local Backups | 717 |
| Specifying Backup Directory Locations | 717 |
| Understanding Hard Link Local Backups and Disaster Recovery | 718 |
| Creating Full and Incremental Backups | 718 |
| Running Vbr Without Optional Commands | 719 |
| Best Practices for Creating Backups | 719 |
| Object-Level Backups | 720 |
| Backup Locations and Storage | 720 |
| Saving Incremental Backups | 720 |
| When vbr.py Deletes Older Backups | 721 |
| Backup Directory Structure and Contents | 721 |
| Directory Tree | 722 |
| Multiple Restore Points | 722 |
| Creating Object-Level Backups | 724 |
| Invoking vbr.py Backup | 724 |

| | |
|---|-----|
| Backup Locations and Naming | 724 |
| Best Practices for Object-Level Backups | 725 |
| Naming Conventions | 725 |
| Creating Backups Concurrently | 726 |
| Determining Backup Frequency | 726 |
| Understanding Object-Level Backup Contents | 726 |
| Making Changes After an Object-Level Backup | 727 |
| Understanding the Overwrite Parameter | 727 |
| Changing Principal and Dependent Objects | 728 |
| Considering Constraint References | 728 |
| Configuration Files for Object-Level Backups | 728 |
| Backup Epochs | 729 |
| Maximum Number of Backups | 729 |
| Creating Hard Link Local Backups | 729 |
| Specifying the Hard Link Local Backup Location | 730 |
| Creating Hard Link Local Backups for Tape Storage | 730 |
| Interrupting the Backup Utility | 731 |
| Viewing Backups | 731 |
| List Backups With vbr.py | 731 |
| Monitor database_snapshots | 732 |
| Query database_backups | 732 |
| Restoring Full Database Backups | 733 |
| Restoring the Most Recent Backup | 733 |
| Restoring an Archive | 734 |
| Attempting to Restore a Node That Is UP | 734 |
| Attempting to Restore to an Alternate Cluster | 735 |
| Restoring Object-Level Backups | 735 |
| Backup Locations | 735 |
| Cluster Requirements for Object-Level Restore | 736 |
| Restoring Objects to a Changed Cluster Topology | 736 |
| Projection Epoch After Restore | 736 |

| | |
|--|-----|
| Catalog Locks During Backup Restore | 736 |
| Catalog Restore Events | 737 |
| Restoring Hard Link Local Backups | 737 |
| Restoring Full- and Object-Level Hard Link Local Backups | 737 |
| Avoiding OID and Epoch Conflicts | 737 |
| Transferring Backups to and From Remote Storage | 739 |
| Restoring to the Same Cluster | 740 |
| Removing Backups | 740 |
| Deleting Backup Directories | 740 |
| Copying the Database to Another Cluster | 741 |
| Identifying Node Names for Target Cluster | 742 |
| Configuring the Target Cluster | 743 |
| Creating a Configuration File for CopyCluster | 743 |
| Copying the Database | 744 |
| Backup and Restore Utility Reference | 745 |
| VBR Utility Reference | 745 |
| Syntax | 745 |
| Parameters | 746 |
| VBR Configuration File Reference | 746 |
| [Misc] Miscellaneous Settings | 746 |
| [Database] Database Access Settings | 748 |
| [Transmission] Data Transmission During Backup Process | 749 |
| [Mapping] | 750 |
| Recovering the Database | 753 |
| Failure Recovery | 753 |
| Recovery Scenarios | 754 |
| Notes | 755 |
| Restarting HP Vertica on a Host | 755 |
| Restarting HP Vertica on a Host Using the Administration Tools | 756 |
| Restarting HP Vertica on a Host Using the Management Console | 756 |
| Restarting the Database | 756 |

| | |
|--|------------|
| Recovering the Cluster From a Backup | 759 |
| Monitoring Recovery | 759 |
| Viewing Log Files on Each Node | 759 |
| Viewing the Cluster State and Recover Status | 759 |
| Using System Tables to Monitor Recovery | 760 |
| Monitoring Cluster Status After Recovery | 760 |
| Exporting a Catalog | 761 |
| Best Practices for Disaster Recovery | 761 |
| Monitoring HP Vertica | 763 |
| Monitoring Log Files | 763 |
| When a Database Is Running | 763 |
| When the Database / Node Is Starting up | 763 |
| Rotating Log Files | 764 |
| Using Administration Tools Logrotate Utility | 764 |
| Manually Rotating Logs | 764 |
| Manually Creating Logrotate Scripts | 765 |
| Monitoring Process Status (ps) | 767 |
| Monitoring Linux Resource Usage | 768 |
| Monitoring Disk Space Usage | 769 |
| Monitoring Database Size for License Compliance | 769 |
| Viewing the Current License State | 769 |
| Manually Running an Audit of the Entire Database | 770 |
| Targeted Auditing | 770 |
| Using Management Console to Monitor License Compliance | 771 |
| Monitoring Shared Node Connections | 771 |
| Monitoring Elastic Cluster Rebalancing | 773 |
| Historical Rebalance Information | 773 |
| Monitoring Parameters | 773 |
| Monitoring Events | 776 |
| Event Logging Mechanisms | 776 |
| Event Severity Types | 776 |

| | |
|--|-----|
| Event Data | 780 |
| Configuring Event Reporting | 783 |
| Configuring Reporting for Syslog | 783 |
| Enabling HP Vertica to Trap Events for Syslog | 783 |
| Defining Events to Trap for Syslog | 784 |
| Defining the SyslogFacility to Use for Reporting | 785 |
| Configuring Reporting for SNMP | 786 |
| Configuring Event Trapping for SNMP | 787 |
| To Configure HP Vertica to Trap Events for SNMP | 787 |
| To Enable Event Trapping for SNMP | 788 |
| To Define Where HP Vertica Send Traps | 788 |
| To Define Which Events HP Vertica Traps | 788 |
| Verifying SNMP Configuration | 789 |
| Event Reporting Examples | 790 |
| Vertica.log | 790 |
| SNMP | 790 |
| Syslog | 790 |
| Using System Tables | 792 |
| Where System Tables Reside | 792 |
| How System Tables Are Organized | 792 |
| Querying Case-Sensitive Data in System Tables | 793 |
| Examples | 794 |
| Retaining Monitoring Information | 796 |
| Data Collector | 796 |
| Where Is DC Information retained? | 796 |
| DC Tables | 797 |
| Enabling and Disabling Data Collector | 797 |
| Viewing Current Data Retention Policy | 797 |
| Configuring Data Retention Policies | 798 |
| Working with Data Collection Logs | 799 |
| Clearing the Data Collector | 800 |

| | |
|---|-----|
| Flushing Data Collector Logs | 801 |
| Monitoring Data Collection Components | 801 |
| Related Topics | 802 |
| Querying Data Collector Tables | 802 |
| Clearing PROJECTION_REFRESHES History | 803 |
| Monitoring Query Plan Profiles | 804 |
| Monitoring Partition Reorganization | 804 |
| Monitoring Resource Pools and Resource Usage By Queries | 805 |
| Examples | 805 |
| Monitoring Recovery | 808 |
| Viewing Log Files on Each Node | 808 |
| Viewing the Cluster State and Recover Status | 809 |
| Using System Tables to Monitor Recovery | 809 |
| Monitoring Cluster Status After Recovery | 810 |
| Monitoring HP Vertica Using MC | 811 |
| About Chart Updates | 811 |
| Viewing MC Home Page | 812 |
| Tasks | 812 |
| Recent Databases | 813 |
| Monitoring Same-Name Databases on MC | 813 |
| Monitoring Cluster Resources | 814 |
| Database | 814 |
| Messages | 814 |
| Performance | 815 |
| CPU/Memory Usage | 815 |
| User Query Type Distribution | 815 |
| Monitoring Cluster Nodes | 816 |
| Filtering What You See | 816 |
| If You don't See What You Expect | 816 |
| Monitoring Cluster CPU/Memory | 817 |
| Investigating Areas of Concern | 817 |

| | |
|---|-----|
| Monitoring Cluster Performance | 817 |
| How to Get Metrics on Your Cluster | 817 |
| Node Colors and What They Mean | 818 |
| Filtering Nodes From the View | 818 |
| Monitoring System Resources | 819 |
| How up to Date Is the information? | 819 |
| Monitoring Query Activity | 819 |
| Monitoring Key Events | 820 |
| Filtering Chart Results | 821 |
| Viewing More Detail | 821 |
| Monitoring Internal Sessions | 822 |
| Filtering Chart Results | 822 |
| Monitoring User Sessions | 822 |
| What Chart Colors Mean | 822 |
| Chart Results | 823 |
| Monitoring System Memory Usage | 823 |
| Types of System Memory | 823 |
| Monitoring System Bottlenecks | 824 |
| How MC Gathers System Bottleneck Data | 824 |
| The Components MC Reports on | 824 |
| How MC Handles Conflicts in Resources | 824 |
| Example | 825 |
| Monitoring Node Activity | 825 |
| Monitoring MC-managed Database Messages | 828 |
| Message Severity | 829 |
| Viewing Message Details | 829 |
| Search and Export Messages | 829 |
| Searching MC-managed Database Messages | 829 |
| Changing Message Search Criteria | 830 |
| Specifying Date Range Searches | 830 |
| Filtering Messages Client Side | 831 |

| | |
|---|------------|
| Exporting MC-managed Database Messages and Logs | 831 |
| Monitoring MC User Activity | 834 |
| Background Cleanup of Audit Records | 835 |
| Filter and Export Results | 836 |
| If You Perform a Factory Reset | 836 |
| Analyzing Workloads | 837 |
| About the Workload Analyzer | 837 |
| Getting Tuning Recommendations Through an API | 837 |
| What and When | 837 |
| Record the Events | 838 |
| Observation Count and Time | 839 |
| Knowing What to Tune | 839 |
| The Tuning Description (recommended action) and Command | 839 |
| What a Tuning Operation Costs | 839 |
| Examples | 839 |
| Getting Recommendations From System Tables | 841 |
| Understanding WLA's Triggering Events | 841 |
| Getting Tuning Recommendations Through MC | 841 |
| Understanding WLA Triggering Conditions | 842 |
| Collecting Database Statistics | 850 |
| Statistics Used By the Query Optimizer | 851 |
| How Statistics Are Collected | 851 |
| Using the ANALYZE ROW COUNT Operation | 851 |
| Using ANALYZE_STATISTICS | 852 |
| Using ANALYZE_HISTOGRAM | 852 |
| Examples | 853 |
| How Statistics Are Computed | 854 |
| How Statistics Are Reported | 854 |
| Determining When Statistics Were Last Updated | 855 |
| Reacting to Stale Statistics | 859 |
| Example | 860 |

| | |
|---|------------|
| Canceling Statistics Collection | 861 |
| Best Practices for Statistics Collection | 861 |
| When to Gather Full Statistics | 862 |
| Save Statistics | 863 |
| Using Diagnostic Tools | 864 |
| Determining Your Version of HP Vertica | 864 |
| Collecting Diagnostics (scrutinize Command) | 864 |
| Privileges | 868 |
| How to Run Scrutinize | 868 |
| How Scrutinize Collects and Packages Diagnostics | 868 |
| While Scrutinize Runs | 869 |
| After Scrutinize Finishes Running | 869 |
| Example | 870 |
| How to Upload Scrutinize Results to Support | 870 |
| How to Refer to Your Database Cluster | 871 |
| Example upload script | 871 |
| Examples for the Scrutinize Command | 872 |
| How to Include Gzipped Log Files | 872 |
| How to Include a Message in the Diagnostics Package | 872 |
| How to Send Results to Support | 873 |
| Collecting Diagnostics (diagnostics Command) | 873 |
| Syntax | 873 |
| Arguments | 873 |
| Using the Diagnostics Utility | 874 |
| Examples | 874 |
| Exporting a Catalog | 875 |
| Exporting Profiling Data | 875 |
| Syntax | 875 |
| Parameters | 876 |
| Example | 876 |
| Understanding Query Plans | 877 |

| | |
|---|-----|
| How to Get Query Plan Information | 878 |
| How to save query plan information | 879 |
| Viewing EXPLAIN output in Management Console | 880 |
| About the EXPLAIN Plan in Management Console | 881 |
| Expanding and collapsing query paths in EXPLAIN output | 882 |
| Clearing query data | 882 |
| Viewing projection and column metadata in EXPLAIN output | 882 |
| Viewing EXPLAIN Output in vsql | 883 |
| About EXPLAIN Output | 884 |
| Textual output of query plans | 884 |
| Viewing statistics query plan output | 885 |
| Viewing cost and rows path | 887 |
| Viewing projection path | 888 |
| Viewing join path | 889 |
| Outer versus inner join | 890 |
| Hash and merge joins | 891 |
| Inequality joins | 892 |
| Event series joins | 893 |
| Viewing path ID | 893 |
| Viewing filter path | 894 |
| Viewing the GROUP BY paths | 895 |
| GROUPBY HASH EXPLAIN plan example | 895 |
| GROUPBY PIPELINED EXPLAIN plan example | 896 |
| Partially sorted GROUPBY EXPLAIN plan example | 897 |
| Viewing sort path | 898 |
| Viewing limit path | 899 |
| Viewing data redistribution path | 899 |
| Viewing analytic function path | 901 |
| Viewing node down information | 901 |
| Viewing MERGE Path | 903 |
| Linking EXPLAIN Plans to Error Messages and Profiling Information | 904 |

| | |
|--|------------|
| Using the QUERY_PLAN_PROFILES table | 906 |
| Profiling Database Performance | 907 |
| How to Determine If Profiling Is Enabled | 908 |
| How to Enable Profiling | 908 |
| How to Disable Profiling | 909 |
| About Real-Time Profiling | 910 |
| About profiling counters | 910 |
| About query plan profiles | 910 |
| System tables with profile data | 911 |
| What to look for in query profiles | 912 |
| Viewing Profile Data in Management Console | 912 |
| Expanding and collapsing query path profile data | 913 |
| About Profile Data in Management Console | 914 |
| Projection metadata | 914 |
| Query phase duration | 915 |
| Profile metrics | 916 |
| Execution events | 916 |
| Optimizer events | 918 |
| Clearing query data | 919 |
| Viewing Profile Data in vsql | 919 |
| How to profile a single statement | 919 |
| Real-Time Profiling Example | 920 |
| How to Use the Linux watch Command | 920 |
| How to Find Out Which Counters are Available | 920 |
| Sample views for counter information | 921 |
| Running scripts to create the sample views | 921 |
| Viewing counter values using the sample views | 921 |
| Combining sample views | 922 |
| Viewing real-time profile data | 922 |
| How to label queries for profiling | 923 |
| Label syntax | 923 |

| | |
|---|------------|
| Profiling query plans | 925 |
| What you need for query plan profiling | 925 |
| How to get query plan status for small queries | 926 |
| How to get query plan status for large queries | 927 |
| Improving the readability of QUERY_PLAN_PROFILES output | 929 |
| Managing query profile data | 930 |
| Configuring data retention policies | 930 |
| Reacting to suboptimal query plans | 930 |
| About Locales | 933 |
| Unicode Character Encoding: UTF-8 (8-bit UCS/Unicode Transformation Format) | 933 |
| Locales | 933 |
| Notes | 933 |
| Locale Aware String Functions | 934 |
| UTF-8 String Functions | 935 |
| Locale Specification | 936 |
| Long Form | 936 |
| Syntax | 936 |
| Parameters | 936 |
| Collation Keyword Parameters | 940 |
| Notes | 942 |
| Examples | 942 |
| Short Form | 943 |
| Determining the Short Form of a Locale | 943 |
| Specifying a Short Form Locale | 943 |
| Supported Locales | 944 |
| Locale Restrictions and Workarounds | 955 |
| Appendix: Binary File Formats | 959 |
| Creating Native Binary Format Files | 959 |
| File Signature | 959 |
| Column Definitions | 959 |
| Row Data | 962 |

| | |
|------------------------------------|-----|
| Example | 963 |
| We appreciate your feedback! | 969 |

Administration Overview

This document describes the functions performed by an HP Vertica database administrator (DBA). Perform these tasks using only the dedicated database administrator account that was created when you installed HP Vertica. The examples in this documentation set assume that the administrative account name is dbadmin.

- To perform certain cluster configuration and administration tasks, the DBA (users of the administrative account) must be able to supply the root password for those hosts. If this requirement conflicts with your organization's security policies, these functions must be performed by your IT staff.
- If you perform administrative functions using a different account from the account provided during installation, HP Vertica encounters file ownership problems.
- If you share the administrative account password, make sure that only one user runs the **Administration Tools** at any time. Otherwise, automatic configuration propagation does not work correctly.
- The Administration Tools require that the calling user's shell be `/bin/bash`. Other shells give unexpected results and are not supported.

Managing Licenses

You must license HP Vertica in order to use it. Hewlett-Packard supplies your license information to you in the form of a license file named `vlicense.dat`, into which the terms of your license are encoded.

To prevent inadvertently introducing special characters (such as line endings or file terminators) into the license key file, do not open the file in an editor or e-mail client. Though such characters are not always visible in an editor, their presence invalidates the license.

Copying Enterprise, Evaluation, and Flex Zone License Files

For ease of HP Vertica Enterprise Edition installation, HP recommends that you copy the license file to `/tmp/vlicense.dat` on the **Administration host**.

If you have a license for Flex Zone, HP recommends that you copy the license file to `/opt/vertica/config/share/license.com.vertica.flextables.key`.

Be careful not to alter the license key file in any way when copying the file between Windows and Linux, or to any other location. To help prevent applications from trying to alter the file, enclose the license file in an archive file (such as a `.zip` or `.tar` file).

After copying the license file from one location to another, check that the copied file size is identical to that of the one you received from HP Vertica.

Obtaining a License Key File

To obtain an Enterprise Edition, Evaluation, or Flex Zone license key, contact HP Vertica at: <http://www.vertica.com/about/contact-us/>

Your HP Vertica Community Edition download package includes the Community Edition license, which allows three nodes and 1TB of data. The HP Vertica Community Edition license does not expire.

Understanding HP Vertica Licenses

HP Vertica has flexible licensing terms. It can be licensed on the following bases:

- Term-based (valid until a specific date)
- Raw data size based (valid to store up to some amount of raw data)
- Both term-based and data-size-based
- Unlimited duration and data storage
- Raw data size based and a limit of 3 nodes (HP Vertica Community Edition)

Your license key has your licensing bases encoded into it. If you are unsure of your current license, you can [view your license information from within HP Vertica](#).

License Types

HP Vertica Community Edition. You can download and start using Community Edition for free. The Community Edition license allows customers the following:

- 3 node limit
- 1 terabyte columnar table data limit
- 1 terabyte Flex table data limit

HP Vertica Enterprise Edition. You can purchase the Enterprise Edition license. The Enterprise Edition license entitles customers to:

- No node limit
- Columnar data, amount specified by the license
- 1 terabyte Flex table data

Flex Zone. Flex Zone is a new offering that is based on the flex tables technology available in version 7.0. Customers can separately purchase and apply a Flex Zone license to their installation. The Flex Zone license entitles customers to the licensed amount of Flex table data and removes the 3 node restriction imposed by the Community Edition.

Note that Enterprise Edition customers who purchase Flex Zone must apply two licenses: their Enterprise Edition license and their Flex Zone license.

Customers whose prime goal is to work with Flex tables can purchase a Flex Zone license. When they purchase Flex Zone, customers receive a complimentary Enterprise License, which entitles them to one terabyte of columnar data and imposes no node limit.

Not that customers who purchase a Flex Zone license must apply two licenses: the complimentary Enterprise Edition license and their Flex Zone license.

| Allowances | Community Edition | Enterprise Edition | Enterprise Edition + Flex Zone | Flex Zone |
|---------------|-------------------|--------------------|--------------------------------|-------------|
| Node Limit | 3 nodes | Unlimited | Unlimited | Unlimited |
| Columnar Data | 1 terabyte | Per license | Per license | 1 terabyte |
| Flex Data | 1 terabyte | 1 terabyte | Per license | Per license |

Installing or Upgrading a License Key

The steps you follow to apply your HP Vertica license key vary, depending on the type of license you are applying and whether you are upgrading your license. This section describes the following:

- [New HP Vertica License Installations](#)
- [HP Vertica License Renewals or Upgrades](#)

New HP Vertica License Installations

1. Copy the license key file to your **Administration Host**.
2. Ensure the license key's file permissions are set to at least 666 (read and write permissions for all users).
3. Install HP Vertica as described in the Installation Guide if you have not already done so. The interface prompts you for the license key file.
4. To install Community Edition, leave the default path blank and press **OK**. To apply your evaluation or Enterprise Edition license, enter the absolute path of the license key file you downloaded to your Administration Host and press **OK**. The first time you log in as the **Database Administrator** and run the **Administration Tools**, the interface prompts you to accept the End-User License Agreement (EULA).

Note: If you installed **Management Console**, the MC administrator can point to the location of the license key during Management Console configuration.

5. Choose **View EULA** to review the EULA.
6. Exit the EULA and choose **Accept EULA** to officially accept the EULA and continue installing the license, or choose **Reject EULA** to reject the EULA and return to the Advanced Menu.

HP Vertica License Renewals or Upgrades

If your license is expiring or you want your database to grow beyond your licensed data size, you must renew or upgrade your license. Once you have obtained your renewal or upgraded license key file, you can install it using Administration Tools or Management Console.

Uploading or Upgrading a License Key Using Administration Tools

1. Copy the license key file to your **Administration Host**.
2. Ensure the license key's file permissions are set to at least 666 (read and write permissions for all users).
3. Start your database, if it is not already running.
4. In the Administration Tools, select Advanced > Upgrade License Key and click **OK**.
5. Enter the path to your new license key file and click **OK**. The interface prompts you to accept the End-User License Agreement (EULA).
6. Choose **View EULA** to review the EULA.
7. Exit the EULA and choose **Accept EULA** to officially accept the EULA and continue installing the license, or choose **Reject EULA** to reject the EULA and return to the Advanced Tools menu.

Uploading or Upgrading a License Key Using Management Console

You can upload a new license from the Settings page for each database.

Browse to the location of the license key from your local computer (where the web browser is installed), upload the file, and save.

Note: As soon as you renew or upgrade your license key from either your **Administration Host** or Management Console, HP Vertica applies the license update. No further warnings appear.

Flex Zone License Installations

1. Install HP Vertica as described in the Installation Guide if you have not already done so.
2. Copy the Flex Zone license key file to your **Administration Host**. HP recommends that you copy the license file to
`/opt/vertica/config/share/license.com.vertica.flextables.key`.
3. Start your database, if it is not already running.
4. In the Administration Tools, connect to your database.
5. At the vsql prompt, select `INSTALL_LICENSE` as described in the SQL Reference Manual.

```
=> SELECT INSTALL_LICENSE('/opt/vertica/config/share/license.com.vertica.flextables.
key');
```

Viewing Your License Status

HP Vertica has several functions to show you your license terms and current status.

Examining Your License Key

Use the [DISPLAY_LICENSE](#) SQL function described in the SQL Reference Manual to display the license information. This function displays the dates for which your license is valid (or "Perpetual" if your license does not expire) and any raw data allowance. For example:

```
=> SELECT DISPLAY_LICENSE();
          DISPLAY_LICENSE
-----
HP Vertica Systems, Inc.
1/1/2011
12/31/2011
30
50TB
(1 row)
```

Or, use the [LICENSES](#) table described in the SQL Reference Manual to view information on all your installed licenses. This table displays your license types, the dates for which your licenses are valid, and the size and node limits your licenses impose. In the example below, the licenses table displays the Community Edition license and the default license that controls HP Vertica's flex data capacity.

```
=> SELECT * FROM licenses; \x
-[ RECORD 1 ]-----+-----
license_id      | 45035996273704986
name            | vertica
licensee       | Vertica Community Edition
start_date     | 2011-11-22
end_date       | Perpetual
size           | 1TB
is_community_edition | t
node_restriction | 3
-[ RECORD 2 ]-----+-----
license_id      | 45035996274085644
name            | com.vertica.flextable
licensee       | Vertica Community Edition, Flextable
start_date     | 2013-10-29
end_date       | Perpetual
size           | 1TB
is_community_edition | t
node_restriction |
```


Viewing Your License Status

If your license includes a raw data size allowance, HP Vertica periodically audits your database's size to ensure it remains compliant with the license agreement. If your license has an end date, HP Vertica also periodically checks to see if the license has expired. You can see the result of the latest audits using the [GET_COMPLIANCE_STATUS](#) function.

```
GET_COMPLIANCE_STATUS
-----
Raw Data Size: 2.00GB +/- 0.003GB
License Size : 4.000GB
Utilization  : 50%
Audit Time   : 2011-03-09 09:54:09.538704+00
Compliance Status : The database is in compliance with respect to raw data size.
License End Date: 04/06/2011
Days Remaining: 28.59
(1 row)
```

Viewing Your License Status Through MC

Information about license usage is on the Settings page. See [Monitoring Database Size for License Compliance](#).

Calculating the Database Size

You can use your HP Vertica software until your data reaches the maximum raw data size that the license agreement provides. This section describes when data is monitored, what data is included in the estimate, and the general methodology used to produce an estimate. For more information about monitoring for data size, see [Monitoring Database Size for License Compliance](#).

How HP Vertica Estimates Raw Data Size

HP Vertica uses statistical sampling to calculate an accurate estimate of the raw data size of the database. In this context, *raw data* means the uncompressed, unfederated data stored in a single HP Vertica database. For the purpose of license size audit and enforcement, HP Vertica evaluates the raw data size as if the data had been exported from the database in text format, rather than as compressed data.

HP Vertica conducts your database size audit using statistical sampling. This method allows HP Vertica to estimate the size of the database without significantly impacting database performance. The trade-off between accuracy and impact on performance is a small margin of error, inherent in statistical sampling. Reports on your database size include the margin of error, so you can assess the accuracy of the estimate. To learn more about simple random sampling, see the Wikipedia entry for [Simple Random Sample](#).

Excluding Data From Raw Data Size Estimate

Not all data in the HP Vertica database is evaluated as part of the raw data size. Specifically, HP Vertica excludes the following data:

- Multiple projections (underlying physical copies) of data from a logical database entity (table). Data appearing in multiple projections of the same table is counted only once.
- Data stored in temporary tables.
- Data accessible through external table definitions.
- Data that has been deleted, but which remains in the database. To understand more about deleting and purging data, see [Purging Deleted Data](#).
- Data stored in the WOS.
- Data stored in system and work tables such as monitoring tables, **Data Collector** tables, and Database Designer tables.

Evaluating Data Type Footprint Size

The data sampled for the estimate is treated as if it had been exported from the database in text format (such as printed from **vsq!l**). This means that HP Vertica evaluates the data type footprint sizes as follows:

- Strings and binary types (CHAR, VARCHAR, BINARY, VARBINARY) are counted as their actual size in bytes using UTF-8 encoding. NULL values are counted as 1-byte values (zero bytes for the NULL, and 1-byte for the delimiter).
- Numeric data types are counted as if they had been printed. Each digit counts as a byte, as does any decimal point, sign, or scientific notation. For example, -123.456 counts as eight bytes (six digits plus the decimal point and minus sign).
- Date/time data types are counted as if they had been converted to text, including any hyphens or other separators. For example, a timestamp column containing the value for noon on July 4th, 2011 would be 19 bytes. As text, vsq!l would print the value as 2011-07-04 12:00:00, which is 19 characters, including the space between the date and the time.

Note: Each column has an additional byte for the column delimiter.

Using AUDIT to Estimate Database Size

To supply a more accurate database size estimate than statistical sampling can provide, use the **AUDIT** function to perform a full audit. This function has parameters to set both the `error_tolerance` and `confidence_level`. Using one or both of these parameters increases or decreases the function's performance impact.

For instance, lowering the `error_tolerance` to zero (0) and raising the `confidence_level` to 100, provides the most accurate size estimate, and increases the performance impact of calling the `AUDIT` function. During a detailed, low error-tolerant audit, all of the data in the database is dumped to a raw format to calculate its size. Since performing a stringent audit can significantly impact database performance, never perform a full audit of a production database. See [AUDIT](#) for details.

Note: Unlike estimating raw data size using statistical sampling, a full audit performs SQL queries on the full database contents, *including* the contents of the WOS.

Monitoring Database Size for License Compliance

If your HP Vertica license includes a raw data storage allowance, you should regularly monitor the size of your database. This monitoring allows you to plan to either schedule deleting old data to keep your database in compliance with your license agreement, or budget for a license upgrade to allow for the continued growth of your database.

Viewing the Current License State

HP Vertica periodically runs an audit of the database size to verify that your database remains compliant with your license. You can view the results of the most recent audit by calling the [GET_COMPLIANCE_STATUS](#) function.

```
GET_COMPLIANCE_STATUS
-----
Raw Data Size: 2.00GB +/- 0.003GB
License Size : 4.000GB
Utilization  : 50%
Audit Time   : 2011-03-09 09:54:09.538704+00
Compliance Status : The database is in compliance with respect to raw data size.
License End Date: 04/06/2011
Days Remaining: 28.59
(1 row)
```

Periodically running `GET_COMPLIANCE_STATUS` to monitor your database's license status is usually enough to ensure that your database remains compliant with your license. If your database begins to near its data allowance, you may want to use the other auditing functions described below to determine where your database is growing and how recent deletes have affected the size of your database.

Manually Running an Audit of the Entire Database

You can trigger HP Vertica's automatic audit of your database at any time using the [AUDIT_LICENSE_SIZE](#) SQL function. This function triggers the same audit that HP Vertica performs periodically. The audit runs in the background, so you need to wait for the audit to complete. You can then view the audit results using [GET_COMPLIANCE_STATUS](#).

An alternative to `AUDIT_LICENSE_SIZE` is to use the [AUDIT](#) SQL function to audit the size of your entire database by passing it an empty string. Unlike `AUDIT_LICENSE_SIZE`, this function operates synchronously, returning when it has estimated the size of the database.

```
=> SELECT AUDIT('');  
  AUDIT  
-----  
 76376696  
(1 row)
```

The size of the database is reported in bytes. The `AUDIT` function also allows you to control the accuracy of the estimated database size using additional parameters. See the entry for the [AUDIT](#) function in the SQL Reference Manual for full details

Note: HP Vertica does not count the results of the `AUDIT` function as an official audit. It takes no license compliance actions based on the results.

Targeted Auditing

If your audits find your database to be unexpectedly large, you may want to find which schemas, tables, or partitions are using the most storage. You can use the `AUDIT` function to perform targeted audits of schemas, tables, or partitions by supplying the name of the entity whose size you want to find. For example, to find the size of the `online_sales` schema in the [VMart](#) example database, run the following command:

```
VMart=> SELECT AUDIT('online_sales');  
  AUDIT  
-----  
 35716504  
(1 row)
```

You can also change the granularity of an audit to report the size of each entity in a larger entity (for example, each table in a schema) by using the `granularity` argument of the `AUDIT` function. See the [AUDIT](#) function's entry in the SQL Reference Manual.

Using Management Console to Monitor License Compliance

You can also get information about raw data storage through the Management Console. This information is available in the database **Overview** page, which displays a grid view of the database's overall health.

- The needle in the license meter adjusts to reflect the amount used in megabytes.
- The grace period represents the term portion of the license.

- The Audit button returns the same information as the AUDIT() function in a graphical representation.
- The Details link within the License grid (next to the Audit button) provides historical information about license usage. This page also shows a progress meter of percent used toward your license limit.

Managing License Warnings and Limits

Term License Warnings and Expiration

The term portion of an HP Vertica license is easy to manage—you are licensed to use HP Vertica until a specific date. If the term of your license expires, HP Vertica alerts you with messages appearing in the **Administration Tools** and **vsq1**. For example:

```
=> CREATE TABLE T (A INT);NOTICE: Vertica license is in its grace period  
HINT: Renew at http://www.vertica.com/  
CREATE TABLE
```

Contact HP Vertica at <http://www.vertica.com/about/contact-us/> as soon as possible to renew your license, and then [install the new license](#). After the grace period expires, HP Vertica stops processing queries.

Data Size License Warnings and Remedies

If your HP Vertica license includes a raw data size allowance, HP Vertica periodically audits the size of your database to ensure it remains compliant with the license agreement. For details of this audit, see [Calculating the Database Size](#). You should also monitor your database size to know when it will approach licensed usage. Monitoring the database size helps you plan to either upgrade your license to allow for continued database growth or delete data from the database so you remain compliant with your license. See [Monitoring Database Size for License Compliance](#) for details.

If your database's size approaches your licensed usage allowance, you will see warnings in the **Administration Tools** and **vsq1**. You have two options to eliminate these warnings:

- Upgrade your license to a larger data size allowance.
- Delete data from your database to remain under your licensed raw data size allowance. The warnings disappear after HP Vertica's next audit of the database size shows that it is no longer close to or over the licensed amount. You can also manually run a database audit (see [Monitoring Database Size for License Compliance](#) for details).

If your database continues to grow after you receive warnings that its size is approaching your licensed size allowance, HP Vertica displays additional warnings in more parts of the system after a grace period passes.

If Your HP VerticaEnterprise Edition Database Size Exceeds Your Licensed Limits

If your Enterprise Edition database size exceeds your licensed data allowance, all successful queries from ODBC and JDBC clients return with a status of SUCCESS_WITH_INFO instead of the usual SUCCESS. The message sent with the results contains a warning about the database size. Your ODBC and JDBC clients should be prepared to handle these messages instead of assuming that successful requests always return SUCCESS.

If Your HP VerticaCommunity Edition Database Size Exceeds Your Licensed Limits

If your Community Edition database size exceeds your licensed data allowance, you will no longer be able to load or modify data in your database. In addition, you will not be able to delete data from your database.

To bring your database under compliance, you can choose to:

- Drop database tables
- Upgrade to HP Vertica Enterprise Edition (or an evaluation license)

Configuring the Database

This section provides information about:

- The [Configuration Procedure](#)
- [Configuration Parameters](#)
- Designing a [logical schema](#)
- Creating the [physical schema](#)

You'll also want to set up a security scheme. See [Implementing Security](#).

See also implementing [locales](#) for international data sets.

Note: Before you begin this section, HP strongly recommends that you follow the [Tutorial](#) in the Getting Started Guide to quickly familiarize yourself with creating and configuring a fully-functioning example database.

Configuration Procedure

This section describes the tasks required to set up an HP Vertica database. It assumes that you have obtained a valid license key file, installed the HP Vertica rpm package, and run the installation script as described in the Installation Guide.

You'll complete the configuration procedure using the:

- **Administration Tools**

If you are unfamiliar with **Dialog**-based user interfaces, read [Using the Administration Tools Interface](#) before you begin. See also the [Administration Tools Reference](#) for details.

- **vsqI** interactive interface
- The Database Designer, described fully in [Creating a Database Design](#)

Note: Users can also perform certain tasks using the [Management Console](#). Those tasks will point to the appropriate topic.

IMPORTANT NOTES

- Follow the configuration procedure in the order presented in this book.
- HP strongly recommends that you first use the [Tutorial](#) in the Getting Started Guide to experiment with creating and configuring a database.
- Although you may create more than one database (for example, one for production and one for testing), you may create only one active database for each installation of Vertica Analytics Platform
- The generic configuration procedure described here can be used several times during the development process and modified each time to fit changing goals. You can omit steps such as preparing actual data files and sample queries, and run the Database Designer without optimizing for queries. For example, you can create, load, and query a database several times for development and testing purposes, then one final time to create and load the production database.

Prepare Disk Storage Locations

You must create and specify directories in which to store your catalog and data files (**physical schema**). You can specify these locations when you install or configure the database, or later during database operations.

The directory you specify for your catalog files (the catalog path) is used across all nodes. That is, if you specify `/home/catalog` for your catalog files, HP Vertica will use `/home/catalog` as the catalog path on all nodes. The catalog directory should always be separate from any data files.

Note: : Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

The same is true for your data path. If you specify that your data should be stored in `/home/data`, HP Vertica ensures this is the data path used on all database nodes.

Do not use a single directory to contain both catalog and data files. You can store the catalog and data directories on different drives, which can be either on drives local to the host (recommended for the catalog directory) or on a shared storage location, such as an external disk enclosure or a **SAN**.

Both the catalog and data directories must be owned by the **database administrator**.

Before you specify a catalog or data path, be sure the parent directory exists on all nodes of your database. The database creation process in `admintools` creates the actual directories, but the parent directory must exist on all nodes.

You do not need to specify a disk storage location during installation. However, you can by using the `--data-dir` parameter to the `install_vertica` script. See [Specifying Disk Storage Location During Installation](#)

See Also

- [Specifying Disk Storage Location on MC](#)
- [Specifying Disk Storage Location During Database Creation](#)
- [Configuring Disk Usage to Optimize Performance](#)
- [Using Shared Storage With HP Vertica](#)

Specifying Disk Storage Location During Installation

There are three ways to specify the disk storage location. You can specify the location when you:

- Install HP Vertica
- Create a database using the Administration Tools
- Install and configure Management Console

To Specify the Disk Storage Location When You install:

When you install HP Vertica, the `--data_dir` parameter in the `install_vertica` script (see [Installing with the Script](#)) lets you specify a directory to contain database data and catalog files. The script defaults to the Database Administrator's default home directory: `/home/dbadmin`.

You should *replace this default* with a directory that has adequate space to hold your data and catalog files.

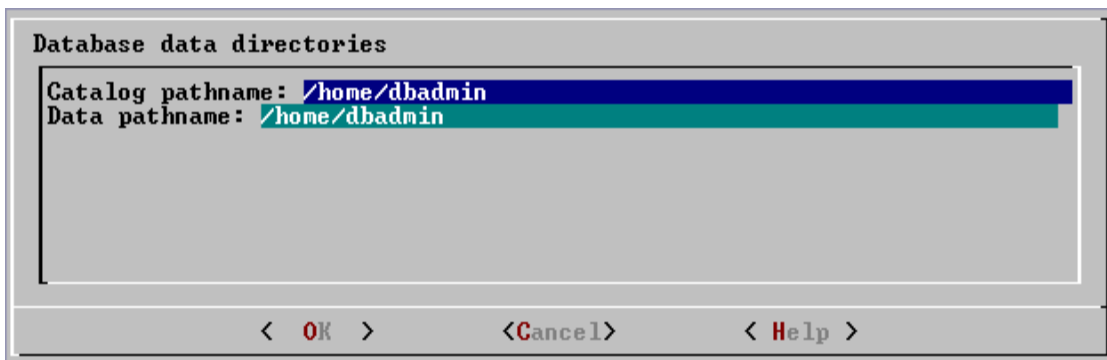
Before you create a database, verify that the data and catalog directory exists on each node in the cluster. Also verify that the directory on each node is owned by the database administrator.

Notes

- Catalog and data path names must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions will result in database creation failure.
- HP Vertica refuses to overwrite a directory if it appears to be in use by another database. Therefore, if you created a database for evaluation purposes, dropped the database, and want to reuse the database name, make sure that the disk storage location previously used has been completely cleaned up. See [Working With Storage Locations](#) for details.

Specifying Disk Storage Location During Database Creation

When you invoke the [Create Database](#) command in the **Administration Tools**, a dialog box allows you to specify the catalog and data locations. These locations must exist on each host in the cluster and must be owned by the database administrator.



When you click **OK**, HP Vertica automatically creates the following subdirectories:

```
catalog-pathname/database-name/node-name_catalog/data-pathname/database-name/node-name_data/
```

For example, if you use the default value (the database administrator's home directory) of `/home/dbadmin` for the Stock Exchange example database, the catalog and data directories are created on each node in the cluster as follows:

```
/home/dbadmin/Stock_Schema/stock_schema_node1_host01_catalog/home/dbadmin/Stock_Schema/stock_schema_node1_host01_data
```

Notes

- Catalog and data path names must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions will result in database creation failure.
- HP Vertica refuses to overwrite a directory if it appears to be in use by another database. Therefore, if you created a database for evaluation purposes, dropped the database, and want to reuse the database name, make sure that the disk storage location previously used has been completely cleaned up. See [Working With Storage Locations](#) for details.

Specifying Disk Storage Location on MC

You can use the MC interface to specify where you want to store database metadata on the cluster in the following ways:

- When you configure MC the first time
- When you create new databases using on MC

See [Configuring Management Console](#).

Configuring Disk Usage to Optimize Performance

Once you have created your initial storage location, you can add additional storage locations to the database later. Not only does this provide additional space, it lets you control disk usage and increase I/O performance by isolating files that have different I/O or access patterns. For example, consider:

- Isolating execution engine temporary files from data files by creating a separate storage location for **temp space**.
- Creating labeled storage locations and storage policies, in which selected database objects are stored on different storage locations based on measured performance statistics or predicted access patterns.

See [Working With Storage Locations](#) for details.

Using Shared Storage With HP Vertica

If using shared **SAN** storage, ensure there is no contention among the nodes for disk space or bandwidth.

- Each host must have its own catalog and data locations. Hosts cannot share catalog or data locations.
- Configure the storage so that there is enough I/O bandwidth for each node to access the storage independently.

Viewing Database Storage Information

You can view node-specific information on your HP Vertica cluster through the **Management Console**. See [Monitoring HP Vertica Using MC](#) for details.

Disk Space Requirements for HP Vertica

In addition to actual data stored in the database, HP Vertica requires disk space for several data reorganization operations, such as **mergeout** and [managing nodes](#) in the cluster. For best results, HP recommends that disk utilization per node be no more than sixty percent (60%) for a **K-Safe=1** database to allow such operations to proceed.

In addition, disk space is temporarily required by certain query execution operators, such as hash joins and sorts, in the case when they cannot be completed in memory (RAM). Such operators might be encountered during queries, recovery, refreshing projections, and so on. The amount of disk space needed (known as **temp space**) depends on the nature of the queries, amount of data on the node and number of concurrent users on the system. By default, any unused disk space on the data disk can be used as temp space. However, HP recommends provisioning temp space separate from data disk space. See [Configuring Disk Usage to Optimize Performance](#).

Disk Space Requirements for Management Console

You can install MC on any node in the cluster, so there are no special disk requirements for MC—other than disk space you would normally allocate for your database cluster. See [Disk Space Requirements for HP Vertica](#).

Prepare the Logical Schema Script

Designing a logical schema for an HP Vertica database is no different from designing one for any other SQL database. Details are described more fully in [Designing a Logical Schema](#).

To create your logical schema, prepare a SQL script (plain text file, typically with an extension of .sql) that:

1. Creates additional schemas (as necessary). See [Using Multiple Schemas](#).
2. Creates the tables and column **constraints** in your database using the [CREATE TABLE](#) command.
3. Defines the necessary table constraints using the [ALTER TABLE](#) command.
4. Defines any views on the table using the [CREATE VIEW](#) command.

You can generate a script file using:

- A schema designer application.
- A schema extracted from an existing database.
- A text editor.
- One of the example database `example-name_define_schema.sql` scripts as a template. (See the example database directories in `/opt/vertica/examples`.)

In your script file, make sure that:

- Each statement ends with a semicolon.
- You use [data types](#) supported by HP Vertica, as described in the SQL Reference Manual.

Once you have created a database, you can test your schema script by executing it as described in [Create the Logical Schema](#). If you encounter errors, drop all tables, correct the errors, and run the script again.

Prepare Data Files

Prepare two sets of data files:

- Test data files. Use test files to test the database after the partial data load. If possible, use part of the actual data files to prepare the test data files.
- Actual data files. Once the database has been tested and optimized, use your data files for your initial [Bulk Loading Data](#).

How to Name Data Files

Name each data file to match the corresponding table in the logical schema. Case does not matter.

Use the extension `.tbl` or whatever you prefer. For example, if a table is named `Stock_Dimension`, name the corresponding data file `stock_dimension.tbl`. When using multiple data files, append `_nnn` (where `nnn` is a positive integer in the range 001 to 999) to the file name. For example, `stock_dimension.tbl_001`, `stock_dimension.tbl_002`, and so on.

Prepare Load Scripts

Note: You can postpone this step if your goal is to test a logical schema design for validity.

Prepare SQL scripts to load data directly into physical storage using the [COPY...DIRECT](#) statement using **vsql**, or through **ODBC** as described in the Programmer's Guide.

You need scripts that load the:

- Large tables
- Small tables

HP recommends that you load large tables using multiple files. To test the load process, use files of 10GB to 50GB in size. This size provides several advantages:

- You can use one of the data files as a sample data file for the **Database Designer**.
- You can load just enough data to [Perform a Partial Data Load](#) before you load the remainder.
- If a single load fails and rolls back, you do not lose an excessive amount of time.
- Once the load process is tested, for multi-terabyte tables, break up the full load in file sizes of 250–500GB.

See the [Bulk Loading Data](#) and the following additional topics for details:

- [Bulk Loading Data](#)
- [Using Load Scripts](#)
- [Using Parallel Load Streams](#)
- [Loading Data into Pre-Join Projections](#)
- [Enforcing Constraints](#)
- [About Load Errors](#)

Tip: You can use the load scripts included in the example databases in the Getting Started Guide as templates.

Create an Optional Sample Query Script

The purpose of a sample query script is to test your schema and load scripts for errors.

Include a sample of queries your users are likely to run against the database. If you don't have any real queries, just write simple SQL that collects counts on each of your tables. Alternatively, you can skip this step.

Create an Empty Database

Two options are available for creating an empty database:

- Using the **Management Console**
- Using **Administration Tools**

Creating a Database Name and Password

Database name must conform to the following rules:

- Be between 1-30 characters
- Begin with a letter
- Follow with any combination of letters (upper and lowercase), numbers, and/or underscores.

Database names are case sensitive; however, HP strongly recommends that you do not create databases with the same name that uses different case; for example, do not create a database called `mydatabase` and another database called `MyDataBase`.

Database Passwords

Database passwords may contain letters, digits, and certain special characters; however, no non-ASCII Unicode characters may be used. The following table lists special (ASCII) characters that HP Vertica permits in database passwords. Special characters can appear anywhere within a password string; for example, `mypas$word` or `$mypassword` or `mypassword$` are all permitted.

Caution: Using special characters in database passwords that are not listed in the following table could cause database instability.

| Character | Description |
|-----------|-------------------|
| # | pound sign |
| ! | exclamation point |
| + | plus sign |
| * | asterisk |
| ? | question mark |
| , | comma |
| . | period |

| | |
|----|-------------------|
| / | forward slash |
| = | equals sign |
| ~ | tilde |
| - | minus sign |
| \$ | dollar sign |
| _ | underscore |
| : | colon |
| | space |
| " | double quote |
| ' | single quote |
| % | percent sign |
| & | ampersand |
| (| parenthesis |
|) | parenthesis |
| ; | semicolon |
| < | less than sign |
| > | greater than sign |
| @ | at sign |
| ` | back quote |
| [| square bracket |
|] | square bracket |
| \ | backslash |
| ^ | caret |
| | vertical bar |
| { | curly bracket |
| } | curly bracket |

See Also

- [Password Guidelines](#)

Create an Empty Database Using MC

You can create a new database on an existing HP Vertica cluster through the **Management Console** interface.

Database creation can be a long-running process, lasting from minutes to hours, depending on the size of the target database. You can close the web browser during the process and sign back in to MC later; the creation process continues unless an unexpected error occurs. See the **Notes** section below the procedure on this page.

You currently need to use command line scripts to define the database schema and load data. Refer to the topics in [Configuration Procedure](#). You should also run the **Database Designer**, which you access through the **Administration Tools**, to create either a comprehensive or incremental design. Consider using the [Tutorial](#) in the Getting Started Guide to create a sample database you can start monitoring immediately.

How to Create an Empty Database on an MC-managed Cluster

1. If you are already on the **Databases and Clusters** page, skip to the next step; otherwise:
 - a. [Connect](#) to MC and sign in as an MC administrator.
 - b. On the [Home page](#), click the **Databases and Clusters** task.
2. If no databases exist on the cluster, continue to the next step; otherwise:
 - a. If a database is running on the cluster on which you want to add a new database, select the database and click **Stop**.
 - b. Wait for the running database to have a status of *Stopped*.
3. Click the cluster on which you want to create the new database and click **Create Database**.
4. The Create Database wizard opens. Provide the following information:
 - Database name and password. See [Creating a Database Name and Password](#) for rules.
 - Optionally click **Advanced** to open the advanced settings and change the port and catalog, data, and temporary data paths. By default the MC application/web server port is 5450 and paths are `/home/dbadmin`, or whatever you defined for the paths when you ran the Cluster Creation Wizard or the `install_vertica` script. Do not use the default agent port 5444 as a new setting for the MC port. See **MC Settings > Configuration** for port values.
5. Click **Continue**.

6. Select nodes to include in the database.

The Database Configuration window opens with the options you provided and a graphical representation of the nodes appears on the page. By default, all nodes are selected to be part of this database (denoted by a green check mark). You can optionally click each node and clear **Include host in new database** to exclude that node from the database. Excluded nodes are gray. If you change your mind, click the node and select the **Include** check box.

7. Click **Create** in the **Database Configuration** window to create the database on the nodes.

The creation process takes a few moments, after which the database starts and a **Success** message appears on the interface.

8. Click **OK** to close the success message.

MC's Manage page opens and displays the database nodes. Nodes not included in the database are colored gray, which means they are standby nodes you can include later. To add nodes to or remove nodes from your HP Vertica cluster, which are not shown in standby mode, you must run the `install_vertica` script.

Notes

- If warnings occur during database creation, nodes will be marked on the UI with an Alert icon and a message.
 - Warnings do not prevent the database from being created, but you should address warnings after the database creation process completes by viewing the database **Message Center** from the MC Home page.
 - Failure messages display on the database **Manage** page with a link to more detailed information and a hint with an actionable task that you must complete before you can continue. Problem nodes are colored red for quick identification.
 - To view more detailed information about a node in the cluster, double-click the node from the Manage page, which opens the **Node Details** page.
- To create MC users and grant them access to an MC-managed database, see [About MC Users](#) and [Creating an MC User](#).

See Also

- [Creating a Cluster Using MC](#)
- [Troubleshooting Management Console](#)
- [Restarting MC](#)

Create a Database Using Administration Tools

1. Run the **Administration Tools** from your **Administration Host** as follows:

```
$ /opt/vertica/bin/admintools
```

If you are using a remote terminal application, such as PuTTY or a Cygwin bash shell, see [Notes for Remote Terminal Users](#).

2. Accept the license agreement and specify the location of your license file. See [Managing Licenses](#) for more information.

This step is necessary only if it is the first time you have run the Administration Tools

3. On the Main Menu, click **Configuration Menu**, and click **OK**.
4. On the Configuration Menu, click **Create Database**, and click **OK**.
5. Enter the name of the database and an optional comment, and click **OK**.
6. Establish the superuser password for your database.
 - To provide a password enter the password and click **OK**. Confirm the password by entering it again, and then click **OK**.
 - If you don't want to provide the password, leave it blank and click **OK**. If you don't set a password, HP Vertica prompts you to verify that you truly do not want to establish a superuser password for this database. Click **Yes** to create the database without a password or **No** to establish the password.

Caution: If you do not enter a password at this point, the superuser password is set to empty. Unless the database is for evaluation or academic purposes, HP strongly recommends that you enter a superuser password. See [Creating a Database Name and Password](#) for guidelines.

7. Select the hosts to include in the database from the list of hosts specified when HP Vertica was installed (`install_vertica -s`), and click **OK**.
8. Specify the directories in which to store the data and **catalog** files, and click **OK**.

Note: Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

9. Catalog and data pathnames must contain only alphanumeric characters and cannot have

leading spaces. Failure to comply with these restrictions results in database creation failure.

For example:

Catalog pathname: /home/dbadmin

Data Pathname: /home/dbadmin

10. Review the **Current Database Definition** screen to verify that it represents the database you want to create, and then click **Yes** to proceed or **No** to modify the database definition.
11. If you click **Yes**, HP Vertica creates the database you defined and then displays a message to indicate that the database was successfully created.

Note: : For databases created with 3 or more nodes, HP Vertica automatically sets **K-safety** to 1 to ensure that the database is fault tolerant in case a node fails. For more information, see the [Failure Recovery](#) in the Administrator's Guide and [MARK_DESIGN_KSAFE](#) in the SQL Reference Manual.

12. Click **OK** to acknowledge the message.

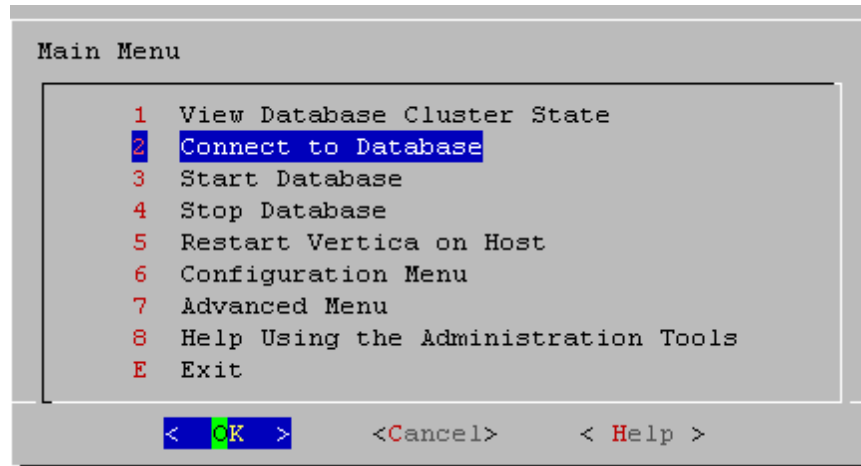
If you receive an error message, see Startup Problems.

Create the Logical Schema

1. **Connect to the database.**

In the Administration Tools Main Menu, click **Connect to Database** and click **OK**.

See [Connecting to the Database](#) for details.



The vsql welcome script appears:

```
Welcome to vsql, the Vertica Analytic Database interactive terminal.  
Type: \h or \? for help with vsql commands  
      \g or terminate with semicolon to execute query  
      \q to quit  
  
=>
```

2. Run the logical schema script

Using the [\i meta-command](#) in vsql to run the SQL [logical schema script](#) that you prepared earlier.

3. Disconnect from the database

Use the [\q meta-command](#) in vsql to return to the Administration Tools.

Perform a Partial Data Load

HP recommends that for large tables, you perform a partial data load and then test your database before completing a full data load. This load should load a representative amount of data.

1. Load the small tables.

Load the small table data files using the SQL [load scripts](#) and [data files](#) you prepared earlier.

2. Partially load the large tables.

Load 10GB to 50GB of table data for each table using the SQL [load scripts](#) and [data files](#) that you prepared earlier.

For more information about projections, see [Physical Schema](#) in the Concepts Guide.

Test the Database

Test the database to verify that it is running as expected.

Check queries for syntax errors and execution times.

1. Use the vsql [\timing meta-command](#) to enable the display of query execution time in milliseconds.
2. Execute the SQL sample query script that you prepared earlier.
3. Execute several ad hoc queries.

Optimize Query Performance

Optimizing the database consists of optimizing for compression and tuning for queries. (See [Creating a Database Design](#).)

To optimize the database, use the Database Designer to create and deploy a design for optimizing the database. See the [Tutorial](#) in the Getting Started Guide for an example of using the Database Designer to create a **Comprehensive Design**.

After you have run the Database Designer, use the techniques described in [Optimizing Query Performance](#) in the Programmer's Guide to improve the performance of certain types of queries.

Note: The database response time depends on factors such as type and size of the application query, database design, data size and data types stored, available computational power, and network bandwidth. Adding nodes to a database cluster does not necessarily improve the system response time for every query, especially if the response time is already short, e.g., less than 10 seconds, or the response time is not hardware bound.

Complete the Data Load

To complete the load:

1. Monitor system resource usage

Continue to run the `top`, `free`, and `df` utilities and watch them while your load scripts are running (as described in [Monitoring Linux Resource Usage](#)). You can do this on any or all nodes in the cluster. Make sure that the system is not swapping excessively (watch `kswapd` in `top`) or running out of swap space (watch for a large amount of used swap space in `free`).

Note: HP Vertica requires a dedicated server. If your loader or other processes take up significant amounts of RAM, it can result in swapping.

2. Complete the large table loads

Run the remainder of the large table load scripts.

Test the Optimized Database

Check query execution times to test your optimized design:

1. Use the `vsq \timing` meta-command to enable the display of query execution time in milliseconds.

Execute a SQL sample query script to test your schema and load scripts for errors.

Note: Include a sample of queries your users are likely to run against the database. If you don't have any real queries, just write simple SQL that collects counts on each of your tables. Alternatively, you can skip this step.

2. Execute several ad hoc queries

- a. Run **Administration Tools** and select [Connect to Database](#).
- b. Use the `\i` [meta-command](#) to execute the query script; for example:

```
vmartdb=> \i vmart_query_03.sql customer_name | annual_income
-----+-----
James M. McNulty |          999979
Emily G. Vogel   |          999998
(2 rows)
Time: First fetch (2 rows): 58.411 ms. All rows formatted: 58.448 ms
vmartdb=> \i vmart_query_06.sql
store_key | order_number | date_ordered
-----+-----+-----
      45 |      202416 | 2004-01-04
     113 |       66017 | 2004-01-04
     121 |     251417 | 2004-01-04
      24 |     250295 | 2004-01-04
       9 |     188567 | 2004-01-04
     166 |      36008 | 2004-01-04
      27 |     150241 | 2004-01-04
     148 |     182207 | 2004-01-04
     198 |      75716 | 2004-01-04
(9 rows)
Time: First fetch (9 rows): 25.342 ms. All rows formatted: 25.383 ms
```

Once the database is optimized, it should run queries efficiently. If you discover queries that you want to optimize, you can modify and update the design. See [Creating an Incremental Design Using Database Designer](#) in the Administrator's Guide.

Set Up Incremental (Trickle) Loads

Once you have a working database, you can use trickle loading to load new data while concurrent queries are running.

Trickle load is accomplished by using the COPY command (without the DIRECT keyword) to load 10,000 to 100,000 rows per transaction into the **WOS**. This allows HP Vertica to batch multiple loads when it writes data to disk. While the COPY command defaults to loading into the WOS, it will write ROS if the WOS is full.

See [Trickle Loading Data](#) for details.

See Also

- [COPY](#)
- [Loading Data Through ODBC](#)

Implement Locales for International Data Sets

The locale is a parameter that defines the user's language, country, and any special variant preferences, such as collation. HP Vertica uses the locale to determine the behavior of various string functions as well for collation for various SQL commands that require ordering and comparison; for example, GROUP BY, ORDER BY, joins, the analytic ORDER BY clause, and so forth.

By default, the locale for the database is `en_US@collation=binary` (English US). You can establish a new default locale that is used for all sessions on the database, as well as override individual sessions with different locales. Additionally the locale can be set through [ODBC](#), [JDBC](#), and [ADO.net](#).

ICU Locale Support

HP Vertica uses the **ICU** library for locale support; thus, you must specify locale using the ICU Locale syntax. While the locale used by the database session is not derived from the operating system (through the `LANG` variable), HP Vertica does recommend that you set the `LANG` appropriately for each node running `vsq`, as described in the next section.

While ICU library services can specify collation, currency, and calendar preferences, HP Vertica supports only the collation component. Any keywords not relating to collation are rejected. Projections are always collated using the `en_US@collation=binary` collation regardless of the session collation. Any locale-specific collation is applied at query time.

The `SET DATESTYLE TO ...` command provides some aspects of the calendar, but HP Vertica supports only dollars as currency.

Changing DB Locale for a Session

This examples sets the session locale to Thai.

1. At the OS level for each node running `vsq`, set the `LANG` variable to the locale language as follows:

```
export LANG=th_TH.UTF-8
```

Note: If setting the `LANG=` as noted does not work, OS support for locales may not be installed.

2. For each HP Vertica session (from ODBC/JDBC or `vsq`) set the language locale.

From `vsq`:

```
\locale th_TH
```

3. From ODBC/JDBC:

```
"SET LOCALE TO th_TH;"
```

4. In PUTTY (or ssh terminal), change the settings as follows:

```
settings > window > translation > UTF-8
```

5. Click Apply, and Save.

All data being loaded must be in UTF-8 format, not an ISO format, as described in [Loading UTF-8 Format Data](#). Character sets like ISO 8859-1 (Latin1), which are incompatible with UTF-8 are not supported, so functions like substring do not work correctly for multi-byte characters. Thus, ISO settings for locale should NOT work correctly. If the translation setting ISO-8859-11:2001 (Latin/Thai) works, the data is loaded incorrectly. To convert data correctly, use a utility program such as Linux `iconv` (see the man page).

Note: The maximum length parameter for VARCHAR and CHAR data type refers to the number of **octets** (bytes) that can be stored in that field and not number of characters. When using multi-byte UTF-8 characters, size fields to accommodate from 1 to 4 bytes per character, depending on the data

See Also

- [Supported Locales](#)
- [About Locales](#)
- [SET LOCALE](#)
- [ICU User Guide](#)

Specify the Default Locale for the Database

The default locale configuration parameter sets the initial locale for every database session once the database has been restarted. Sessions may override this value.

To set the local for the database, use the configuration parameter as follows:

```
SELECT SET_CONFIG_PARAMETER('DefaultSessionLocale' ,  
'<ICU-locale-identifier>');
```

For example:

```
mydb=> SELECT SET_CONFIG_PARAMETER('DefaultSessionLocale','en_GB');
```

```
SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

Override the Default Locale for a Session

To override the default locale for a specific session, use one of the following commands:

- The vsql command `\locale <ICU-locale-identifier>`.

For example:

```
\locale en_GB
INFO: Locale: 'en_GB'
INFO: English (United Kingdom)
INFO: Short form: 'LEN'
```

- The statement `SET LOCALE TO <ICU-locale-identifier>`.

```
SET LOCALE TO en_GB;
INFO: Locale: 'en_GB'
INFO: English (United Kingdom)
INFO: Short form: 'LEN'
```

You can also use the [Short Form](#) of a locale in either of these commands:

```
SET LOCALE TO LEN;
INFO: Locale: 'en'
INFO: English
INFO: Short form: 'LEN'
\locale LEN
INFO: Locale: 'en'
INFO: English
INFO: Short form: 'LEN'
```

You can use these commands to override the locale as many times as needed within a session. The session locale setting applies to any subsequent commands issued in the session.

See Also

- [SET LOCALE](#)

Best Practices for Working with Locales

It is important to understand the distinction between the locale settings on the database server and locale settings at the client application level. The server locale settings impact only the collation behavior for server-side query processing. The client application is responsible for ensuring that the

correct locale is set in order to display the characters correctly. Below are the best practices recommended by HP to ensure predictable results:

Server Locale

Server session locale should be set using the set as described in [Specify the Default Locale for the Database](#). If using different locales in different session, set the server locale at the start of each session from your client.

vsq Client

- If there is no default session locale at database level, the server locale for the session should be set to the desired locale, as described in [Override the Default Locale for a Session](#).
- The locale setting in the terminal emulator where vsq client is run should be set to be equivalent to session locale setting on server side (**ICU** locale) so data is collated correctly on the server and displayed correctly on the client.
- All input data for vsq should be in UTF-8 and all output data is encoded in UTF-8
- Non UTF-8 encodings and associated locale values should not be used because they are not supported.
- Refer to the documentation of your terminal emulator for instructions on setting locale and encoding.

ODBC Clients

- ODBC applications can be either in ANSI or Unicode mode. If Unicode, the encoding used by ODBC is UCS-2. If the user application is ANSI, the data must be in single-byte ASCII, which is compatible with UTF-8 used on the database server. The ODBC driver converts UCS-2 to UTF-8 when passing to the HP Vertica server and converts data sent by the HP Vertica server from UTF-8 to UCS-2.
- If the user application is not already in UCS-2, the application is responsible for converting the input data to UCS-2, or unexpected results could occur. For example:
 - On non-UCS-2 data passed to ODBC APIs, when it is interpreted as UCS-2, it could result in an invalid UCS-2 symbol being passed to the APIs, resulting in errors.
 - The symbol provided in the alternate encoding could be a valid UCS-2 symbol; in this case, incorrect data is inserted into the database.
- If there is no default session locale at database level, ODBC applications should set the desired server session locale using `SQLSetConnectAttr` (if different from database wide setting) in order to get expected collation and string functions behavior on the server.

JDBC and ADO.NET Clients

- JDBC and ADO.NET applications use a UTF-16 character set encoding and are responsible for converting any non-UTF-16 encoded data to UTF-16. The same cautions apply as for ODBC if this encoding is violated.
- The JDBC and ADO.NET drivers convert UTF-16 data to UTF-8 when passing to the HP Vertica server and convert data sent by HP Vertica server from UTF-8 to UTF-16.
- If there is no default session locale at the database level, JDBC and ADO.NET applications should set the correct server session locale by executing the [SET LOCALE TO](#) command in order to get expected collation and string functions behavior on the server. See the [SET LOCALE](#) command in the SQL Reference Manual.

Notes and Restrictions

Session related:

- The locale setting is session scoped and applies to queries only (no DML/DDDL) run in that session. You cannot specify a locale for an individual query.
- The default locale for new sessions can be set using a configuration parameter

Query related:

The following restrictions apply when queries are run with locale other than the default `en_US@collation=binary`:

- Multicolumn NOT IN subqueries are not supported when one or more of the left-side NOT IN columns is of CHAR or VARCHAR data type. For example:

```
=> CREATE TABLE test (x VARCHAR(10), y INT);
=> SELECT ... FROM test WHERE (x,y) NOT IN (SELECT ...);
      ERROR: Multi-expression NOT IN subquery is not supported because a left hand expression could be NULL
```

Note: An error is reported even if columns `test.x` and `test.y` have a "NOT NULL" constraint.

- Correlated HAVING clause subqueries are not supported if the outer query contains a GROUP BY on a CHAR or a VARCHAR column. In the following example, the GROUP BY `x` in the outer query causes the error:

```
=> DROP TABLE test CASCADE;
```

```
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT COUNT(*) FROM test t GROUP BY x HAVING x
      IN (SELECT x FROM test WHERE t.x||'a' = test.x||'a' );
ERROR: subquery uses ungrouped column "t.x" from outer query
```

- Subqueries that use analytic functions in the HAVING clause are not supported. For example:

```
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT MAX(x)OVER(PARTITION BY 1 ORDER BY 1)
      FROM test GROUP BY x HAVING x IN (SELECT MAX(x) FROM test);
ERROR: Analytics query with having clause expression that involves aggregates and subq
very is not supported
```

DML/DDDL related:

- SQL identifiers (such as table names, column names, and so on) can use UTF-8 Unicode characters. For example, the following CREATE TABLE statement uses the ß (German eszett) in the table name:

```
=> CREATE TABLE straÙe(x int, y int);
CREATE TABLE
```

- Projection sort orders are made according to the default en_US@collation=binary collation. Thus, regardless of the session setting, issuing the following command creates a projection sorted by col1 according to the binary collation:

```
=> CREATE PROJECTION p1 AS SELECT * FROM table1 ORDER BY col1;
```

Note that in such cases, `straÙe` and `strasse` would not be near each other on disk.

Sorting by binary collation also means that [sort optimizations](#) do not work in locales other than binary. HP Vertica returns the following warning if you create tables or projections in a non-binary locale:

```
WARNING: Projections are always created and persisted in the default HP Vertica local
e. The current locale is de_DE
```

- When creating pre-join projections, the projection definition query does not respect the locale or collation setting. This means that when you insert data into the fact table of a pre-join projection, referential integrity checks are not locale or collation aware.

For example:

```
\locale LDE_S1 -- German
=> CREATE TABLE dim (col1 varchar(20) primary key);
=> CREATE TABLE fact (col1 varchar(20) references dim(col1));
=> CREATE PROJECTION pj AS SELECT * FROM fact JOIN dim
    ON fact.col1 = dim.col1 UNSEGMENTED ALL NODES;
=> INSERT INTO dim VALUES('ß');
=> COMMIT;
```

The following INSERT statement fails with a "nonexistent FK" error even though 'ß' is in the dim table, and in the German locale 'SS' and 'ß' refer to the same character.

```
=> INSERT INTO fact VALUES('SS');
    ERROR: Nonexistent foreign key value detected in FK-PK join (fact x dim)
    using subquery and dim_node0001; value SS
=> => ROLLBACK;
=> DROP TABLE dim, fact CASCADE;
```

- When the locale is non-binary, the collation function is used to transform the input to a binary string which sorts in the proper order.

This transformation increases the number of bytes required for the input according to this formula:

```
result_column_width = input_octet_width * CollationExpansion + 4
```

CollationExpansion defaults to 5.

- CHAR fields are displayed as fixed length, including any trailing spaces. When CHAR fields are processed internally, they are first stripped of trailing spaces. For VARCHAR fields, trailing spaces are usually treated as significant characters; however, trailing spaces are ignored when sorting or comparing either type of character string field using a non-BINARY locale.

Change Transaction Isolation Levels

By default, HP Vertica uses the `READ COMMITTED` isolation level for every session. If you prefer, you can change the default isolation level for the database or for a specific session.

To change the isolation level for a specific session, use the [SET SESSION CHARACTERISTICS](#) command.

To change the isolation level for the database, use the `TransactionIsolationLevel` configuration parameter. Once modified, HP Vertica uses the new transaction level for every new session.

The following examples set the default isolation for the database to `SERIALIZABLE` and then back to `READ COMMITTED`:

```
=> SELECT SET_CONFIG_PARAMETER('TransactionIsolationLevel','SERIALIZABLE');
=> SELECT SET_CONFIG_PARAMETER('TransactionIsolationLevel','READ COMMITTED');
```

Notes

- A change to isolation level only applies to future sessions. Existing sessions and their transactions continue to use the original isolation level.

A transaction retains its isolation level until it completes, even if the session's transaction isolation level changes mid-transaction. HP Vertica internal processes (such as the **Tuple Mover** and **refresh** operations) and DDL operations are always run at `SERIALIZABLE` isolation level to ensure consistency.

See Also

- [Transactions](#)
- [Configuration Parameters](#)

Configuration Parameters

You can modify certain parameters to configure your HP Vertica database using one of the following options:

- Dynamically through the Management Console [browser-based interface](#)
- At the [command line](#) directly
- From **vsqI**

Important: Before you modify a database parameter, review all documentation about the parameter to determine the context under which you can change it. Parameter changes take effect after you restart the database.

Configuring HP Vertica Settings Using MC

To change database settings for any MC-managed database, click the **Settings** tab at the bottom of the Overview, Activity, or Manage pages. The database must be running.

The Settings page defaults to parameters in the General category. To change other parameters, click an option from the tab panel on the left.

The screenshot shows the HP Vertica Management Console interface. At the top, there's a navigation bar with the HP logo, 'VERTICA', and user information 'uidbadmin Log out'. Below that, a breadcrumb trail shows 'Databases and Clusters > smartdb settings: General'. On the right, there are buttons for 'Reset Defaults', 'Apply', and 'Done'. A left-hand navigation menu lists categories: General (selected), Tuple mover, Epoch mgmt, Monitoring, Password, Profiling, Snapshot, I18N, and License. The main content area displays a table of settings for the 'General' category.

| | | Current value | Default value |
|------------|-----------------------------|----------------|----------------|
| General | Analyze row count interval | 60 seconds | 60 |
| Epoch mgmt | Compress network data | false | false |
| Monitoring | Maximum client sessions | 50 | 50 |
| Password | Transaction isolation level | READ COMMITTED | READ COMMITTED |
| Profiling | Transaction mode | READ WRITE | READ WRITE |
| Snapshot | | | |
| I18N | | | |
| License | | | |

At the bottom of the page, a horizontal navigation bar contains tabs: Overview, Activity, Manage, License, Settings (selected), Design, and Explain.

Some settings require that you restart the database, and MC will prompt you to do so. You can ignore the prompt, but those changes will not take effect until after you restart the database.

If you want to change settings that are specific to Management Console, such as change MC or agent port assignments, see [Managing MC Settings](#) for more information.

See Also

- [Configuration Parameters](#)

Configuring HP Vertica At the Command Line

The tables in this section list parameters for configuring HP Vertica at the command line.

General Parameters

The following table describes the general parameters for configuring HP Vertica.

| Parameters | Default | Description |
|-------------------------------|------------|---|
| AnalyzeRowCountInterval | 60 seconds | Automatically runs every 60 seconds to collect the number of rows in the projection and aggregates row counts calculated during loads. See Collecting Statistics . |
| CompressCatalogOnDisk | 0 | When enabled (set value to 1 or 2) compresses the size of the catalog on disk. Values can be: <ul style="list-style-type: none"> • 1—Compress checkpoints, but not logs • 2—Compress checkpoints and logs Consider enabling this parameter if the catalog disk partition is small (<50GB) and the metadata is large (hundreds of tables, partitions, or nodes). |
| CompressNetworkData | 0 | When enabled (set to value 1), HP Vertica will compress all of the data it sends over the network. This speeds up network traffic at the expense of added CPU load. You can enable this if you find that the network is throttling your database performance. |
| EnableCooperativeParse | 1 | Enabled by default. Implements multi-threaded cooperative parsing capabilities for delimited and fixed-width loads. |
| EnableResourcePoolCPUAffinity | 1 | Enabled by default. When disabled (set value to 0), queries run on any CPU, regardless of the CPU_AFFINITY_SET of the resource pool. |

| Parameters | Default | Description |
|----------------------------------|---------|--|
| ExternalTablesExceptionsLimit | 100 | Determines the maximum number of COPY exceptions and rejections that can occur when a SELECT statement references an external table. Setting this parameter to -1 removes any exceptions limit. For more information, see Validating External Tables . |
| FencedUDxMemoryLimitMB | -1 | Sets the maximum amount of memory (in MB) that a fenced-mode UDF can use. Any UDF that attempts to allocate more memory than this limit triggers an exception. When set to -1, there is no limit on the amount of memory a UDF can allocate. For more information, see Fenced Mode in the Programmer's Guide. |
| FlexTableDataTypeGuessMultiplier | 2.0 | Specifies the multiplier to use for a key value when creating a view for a flex keys table. See Specifying Unstructured Parameters . |
| FlexTableRowSize | 130000 | The default value (in bytes) of the <code>__raw__</code> column size of flex table. The maximum value of is 32000000. You can change the default value as with other configuration parameters, or update the <code>__raw__</code> column size on a per-table basis using ALTER TABLE once an unstructured table exists. See Specifying Unstructured Parameters . |
| MaxAutoSegColumns | 32 | Specifies the number of columns (0 - 1024) to segment automatically when creating auto-projections from COPY and INSERT INTO statements. Setting this parameter to zero (0) indicates to use all columns in the hash segmentation expression. |

| Parameters | Default | Description |
|----------------------------|----------------|--|
| MaxClientSessions | 50 | <p>Determines the maximum number of client sessions that can run on a single node of the database. The default value allows for 5 additional administrative logins, which prevents DBAs from being locked out of the system if the limit is reached by non-dbadmin users.</p> <p>Tip: Setting this parameter to 0 is useful for preventing new client sessions from being opened while you are shutting down the database. Be sure to restore the parameter to its original setting once you've restarted the database. See the section "Interrupting and Closing Sessions" in Managing Sessions.</p> |
| PatternMatchAllocator | 0 | <p>If set to 1, overrides the heap memory allocator for the Perl Compatible Regular Expressions (PCRE) pattern-match library, which evaluates regular expressions. You must restart the database for this parameter to take effect. For more information, see Regular Expression Functions.</p> |
| PatternMatchStackAllocator | 1 | <p>If set to 1, overrides the stack memory allocator for the Perl Compatible Regular Expressions (PCRE) pattern-match library, which evaluates regular expressions. You must restart the database for this parameter to take effect. For more information, see Regular Expression Functions.</p> |
| SegmentAutoProjection | 1 | <p>Determines whether auto-projections are segmented by default. Setting this parameter to zero (0) disables the feature.</p> |
| TransactionIsolationLevel | READ COMMITTED | <p>Changes the isolation level for the database. Once modified, HP Vertica uses the new transaction level for every new session. Existing sessions and their transactions continue to use the original isolation level. See Change Transaction Isolation Levels.</p> |

| Parameters | Default | Description |
|-----------------|---------------|--|
| TransactionMode | READ WRITE | Controls whether transactions are read/write or read-only. Read/write is the default. Existing sessions and their transactions continue to use the original isolation level. |

Setting Configuration Parameters

You can set a new value for a configuration parameter with a select statement as follows. These examples illustrate changing the parameters listed in the table:

```
SELECT SET_CONFIG_PARAMETER ('AnalyzeRowCountInterval',3600);
SELECT SET_CONFIG_PARAMETER ('CompressNetworkData',1);
SELECT SET_CONFIG_PARAMETER ('ExternalTablesExceptionsLimit',-1);
SELECT SET_CONFIG_PARAMETER ('MaxClientSessions', 100);
SELECT SET_CONFIG_PARAMETER ('PatternMatchAllocator',1);
SELECT SET_CONFIG_PARAMETER ('PatternMatchStackAllocator',0);
SELECT SET_CONFIG_PARAMETER ('TransactionIsolationLevel','SERIALIZABLE');
SELECT SET_CONFIG_PARAMETER ('TransactionMode','READ ONLY');
SELECT SET_CONFIG_PARAMETER ('FlexTableRowSize',16000000);
SELECT SET_CONFIG_PARAMETER ('CompressCatalogOnDisk',2);
```

Tuple Mover Parameters

These parameters control how the **Tuple Mover** operates.

| Parameters | Description |
|----------------------|---|
| ActivePartitionCount | <p>Sets the number of partitions, called <i>active partitions</i>, that are currently being loaded. For information about how the Tuple Mover treats active (and inactive) partitions during a mergeout operation, see Understanding the Tuple Mover.</p> <p>Default Value: 1</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('ActivePartitionCount', 2);</pre> |

| Parameters | Description |
|-------------------|---|
| MergeOutInterval | <p>The number of seconds the Tuple Mover waits between checks for new ROS files to merge out. If ROS containers are added frequently, you may need to decrease this value.</p> <p>Default Value: 600</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('MergeOutInterval', 1200);</pre> |
| MoveOutInterval | <p>The number of seconds the Tuple mover waits between checks for new data in the WOS to move to ROS.</p> <p>Default Value: 300</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('MoveOutInterval', 600);</pre> |
| MoveOutMaxAgeTime | <p>The specified interval (in seconds) after which the tuple mover is forced to write the WOS to disk. The default interval is 30 minutes.</p> <p>Tip: If you had been running the <code>force_moveout.sh</code> script in previous releases, you no longer need to run it.</p> <p>Default Value: 1800</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('MoveOutMaxAgeTime', 1200);</pre> |
| MoveOutSizePct | <p>The percentage of the WOS that can be filled with data before the Tuple Mover performs a moveout operation.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('MoveOutSizePct', 50);</pre> |

Epoch Management Parameters

The following table describes the epoch management parameters for configuring HP Vertica.

| Parameters | Description |
|--------------------|--|
| AdvanceAHMInterval | <p>Determines how frequently (in seconds) HP Vertica checks the history retention status. By default the AHM interval is set to 180 seconds (3 minutes).</p> <p>Note: AdvanceAHMInterval cannot be set to a value less than the EpochMapInterval.</p> <p>Default Value: 180</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('AdvanceAHMInterval', '3600');</pre> |
| EpochMapInterval | <p>Determines the granularity of mapping between epochs and time available to historical queries. When a historical queries AT TIME T is issued, HP Vertica maps it to an epoch within a granularity of EpochMapInterval seconds. It similarly affects the time reported for Last Good Epoch during Failure Recovery. Note that it does not affect internal precision of epochs themselves.</p> <p>By default, EpochMapInterval is set to 180 seconds (3 minutes).</p> <p>Tip: Decreasing this interval increases the number of epochs saved on disk. Therefore, you might want to reduce the HistoryRetentionTime parameter to limit the number of history epochs that HP Vertica retains.</p> <p>Default Value: 180</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('EpochMapInterval', '300');</pre> |

| Parameters | Description |
|------------------------|---|
| HistoryRetentionTime | <p>Determines how long deleted data is saved (in seconds) as a historical reference. The default is 0, which means that HP Vertica saves historical data only when nodes are down. Once the specified time has passed since the delete, the data is eligible to be purged. Use the -1 setting if you prefer to use <code>HistoryRetentionEpochs</code> for determining which deleted data can be purged.</p> <p>Note: The default setting of 0 effectively prevents the use of the Administration Tools 'Roll Back Database to Last Good Epoch' option because the AHM remains close to the current epoch and a rollback is not permitted to an epoch prior to the AHM.</p> <p>Tip: If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window for removing loaded data; for example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('HistoryRetentionTime', '86400');</pre> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('HistoryRetentionTime', '240');</pre> |
| HistoryRetentionEpochs | <p>Specifies the number of historical epochs to save, and therefore, the amount of deleted data.</p> <p>Unless you have a reason to limit the number of epochs, HP recommends that you specify the time over which delete data is saved. The -1 setting disables this configuration parameter.</p> <p>If both <code>History</code> parameters are specified, <code>HistoryRetentionTime</code> takes precedence, and if both parameters are set to -1, all historical data is preserved.</p> <p>See Setting a Purge Policy.</p> <p>Default Value: -1</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('HistoryRetentionEpochs', '40');</pre> |

Monitoring Parameters

The following table describes the monitoring parameters for configuring HP Vertica.

| Parameters | Description |
|---------------------------|--|
| SnmptTrapDestinationsList | <p>Defines where HP Vertica send traps for SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: none</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SnmptTrapDestinationsList', 'localhost 162 public');</pre> |
| SnmptTrapsEnabled | <p>Enables event trapping for SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SnmptTrapsEnabled', 1);</pre> |
| SnmptTrapEvents | <p>Define which events HP Vertica traps through SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value:Low Disk Space, Read Only File System, Loss of K Safety, Current Fault Tolerance at Critical Level, Too Many ROS Containers, WOS Over Flow, Node State Change, Recovery Failure, and Stale Checkpoint</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SnmptTrapEvents', 'Low Disk Space, Recovery Failure');</pre> |
| SyslogEnabled | <p>Enables event trapping for syslog. See Configuring Reporting for Syslog.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SyslogEnabled', 1);</pre> |

| Parameters | Description |
|----------------|---|
| SyslogEvents | <p>Defines events that generate a syslog entry. See Configuring Reporting for Syslog.</p> <p>Default Value: none</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SyslogEvents', 'Low Disk Space, Recovery Failure');</pre> |
| SyslogFacility | <p>Defines which SyslogFacility HP Vertica uses. See Configuring Reporting for Syslog.</p> <p>Default Value: user</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SyslogFacility' , 'ftp');</pre> |

Profiling Parameters

The following table describes the profiling parameters for configuring HP Vertica. See [Profiling Database Performance](#) for more information on profiling queries.

| Parameters | Description |
|------------------------|---|
| GlobalEETProfiling | <p>Enables profiling for query execution runs in all sessions, on all nodes.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('GlobalEETProfiling',1);</pre> |
| GlobalQueryProfiling | <p>Enables query profiling for all sessions on all nodes.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('GlobalQueryProfiling',1);</pre> |
| GlobalSessionProfiling | <p>Enables session profiling for all sessions on all nodes.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('GlobalSessionProfiling',1);</pre> |

Security Parameters

The following table describes the parameters for configuring the client authentication method and enabling SSL for HP Vertica.

| Parameters | Description |
|----------------------|---|
| ClientAuthentication | <p>Configures client authentication. By default, HP Vertica uses user name and password (if supplied) to grant access to the database.</p> <p>The preferred method for establishing client authentication is to use the Administration Tools. See Implementing Client Authentication and How to Create Authentication Records.</p> <p>Default Value: local all trust</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('ClientAuthentication', 'hostnossl dbadmin 0.0.0.0/0 trust');</pre> |
| EnableSSL | <p>Configures SSL for the server. See Implementing SSL.</p> <p>Default Value:0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER('EnableSSL', '1');</pre> |

See Also

[Kerberos Authentication Parameters](#)

Database Designer Parameters

The following table describes the parameters for configuring the HP Vertica Database Designer.

| Parameter | Description |
|----------------------------|--|
| DBDCollationSampleRowCount | <p>Minimum number of table rows at which Database Designer discovers and records correlated columns.</p> <p>Default value: 4000</p> <p>Example:</p> <pre>SELECT SET_CONFIGURATION_PARAMETER ('DBDCorrelationSampleRowCou nt', 3000);</pre> |

Internationalization Parameters

The following table describes the internationalization parameters for configuring HP Vertica.

| Parameters | Description |
|---------------------------|---|
| DefaultIntervalStyle | <p>Sets the default interval style to use. If set to 0 (default), the interval is in PLAIN style (the SQL standard), no interval units on output. If set to 1, the interval is in UNITS on output. This parameter does not take effect until the database is restarted.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('DefaultIntervalStyle', 1);</pre> |
| DefaultSessionLocale | <p>Sets the default session startup locale for the database. This parameter does not take effect until the database is restarted.</p> <p>Default Value: en_US@collation=binary</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('DefaultSessionLocale', 'en_GB');</pre> |
| EscapeStringWarning | <p>Issues a warning when back slashes are used in a string literal. This is provided to help locate back slashes that are being treated as escape characters so they can be fixed to follow the Standard conforming string syntax instead.</p> <p>Default Value: 1</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('EscapeStringWarning', '1');</pre> |
| StandardConformingStrings | <p>In HP Vertica 4.0, determines whether ordinary string literals ('...') treat backslashes (\) as string literals or escape characters. When set to '1', backslashes are treated as string literals, when set to '0', back slashes are treated as escape characters.</p> <p>Tip: To treat backslashes as escape characters, use the Extended string syntax:</p> <pre>(E'...');</pre> <p>See String Literals (Character) in the SQL Reference Manual.</p> <p>Default Value: 1</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER('StandardConformingStrings' , '0');</pre> |

Data Collector Parameters

The following table lists the **Data Collector** parameter for configuring HP Vertica.

| Parameter | Description |
|---------------------|--|
| EnableDataCollector | <p>Enables and disables the Data Collector (the Workload Analyzer's internal diagnostics utility) for all sessions on all nodes. Default is 1, enabled. Use 0 to turn off data collection.</p> <p>Default value: 1</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('EnableDataCollector', 0);</pre> |

For more information, see the following topics in the SQL Reference Manual:

- [Data Collector Functions](#)
- [ANALYZE_WORKLOAD](#)
- [V_MONITOR.DATA_COLLECTOR](#)
- [V_MONITOR.TUNING_RECOMMENDATIONS](#)

See also the following topics in the Administrator's Guide

- [Retaining Monitoring Information](#)
- [Analyzing Workloads](#) and [Tuning Recommendations](#)
- [Analyzing Workloads Through Management Console](#) and [Through an API](#)

Kerberos Authentication Parameters

The following parameters let you configure the HP Vertica principal for Kerberos authentication and specify the location of the Kerberos keytab file.

| Parameter | Description |
|---------------------|---|
| KerberosServiceName | <p>Provides the service name portion of the HP Vertica Kerberos principal. By default, this parameter is 'vertica'. For example:</p> <p>vertica/host@EXAMPLE.COM.</p> |

| Parameter | Description |
|--------------------|--|
| KerberosHostname | <p>[Optional] Provides the instance or host name portion of the HP Vertica Kerberos principal. For example: <code>vertica/host@EXAMPLE.COM</code></p> <p>Notes:</p> <ul style="list-style-type: none"> • If you omit the optional KerberosHostname parameter, HP Vertica uses the return value from the <code>gethostname()</code> function. Assuming each cluster node has a different host name, those nodes will each have a different principal, which you must manage in that node's keytab file. • Consider specifying the KerberosHostname parameter to get a single, cluster-wide principal that is easier to manage than multiple principals. |
| KerberosRealm | <p>Provides the realm portion of the HP Vertica Kerberos principal. A realm is the authentication administrative domain and is usually formed in uppercase letters; for example: <code>vertica/host@EXAMPLE.COM</code>.</p> |
| KerberosKeytabFile | <p>Provides the location of the keytab file that contains credentials for the HP Vertica Kerberos principal. By default, this file is located in <code>/etc</code>. For example: <code>KerberosKeytabFile=/etc/krb5.keytab</code>.</p> <p>Notes:</p> <ul style="list-style-type: none"> • The principal must take the form <code>KerberosServiceName/KerberosHostName@KerberosRealm</code> • The keytab file must be readable by the file owner who is running the process (typically the Linux <code>dbadmin</code> user assigned file permissions <code>0600</code>). |

Designing a Logical Schema

Designing a logical schema for an HP Vertica database is no different than designing for any other SQL database. A logical schema consists of objects such as **Schemas**, **Tables**, Views and **Referential Integrity constraints** that are visible to SQL users. HP Vertica supports any relational schema design of your choice.

Using Multiple Schemas

Using a single schema is effective if there is only one database user or if a few users cooperate in sharing the database. In many cases, however, it makes sense to use additional schemas to allow users and their applications to create and access tables in separate namespaces. For example, using additional schemas allows:

- Many users to access the database without interfering with one another.

Individual schemas can be configured to grant specific users access to the schema and its tables while restricting others.

- Third-party applications to create tables that have the same name in different schemas, preventing table collisions.

Unlike other RDBMS, a schema in an HP Vertica database is not a collection of objects bound to one user.

Multiple Schema Examples

This section provides examples of when and how you might want to use multiple schemas to separate database users. These examples fall into two categories: using multiple private schemas and using a combination of private schemas (i.e. schemas limited to a single user) and shared schemas (i.e. schemas shared across multiple users).

Using Multiple Private Schemas

Using multiple private schemas is an effective way of separating database users from one another when sensitive information is involved. Typically a user is granted access to only one schema and its contents, thus providing database security at the schema level. Database users can be running different applications, multiple copies of the same application, or even multiple instances of the same application. This enables you to consolidate applications on one database to reduce management overhead and use resources more effectively. The following examples highlight using multiple private schemas.

- **Using Multiple Schemas to Separate Users and Their Unique Applications**

In this example, both database users work for the same company. One user (HRUser) uses a Human Resource (HR) application with access to sensitive personal data, such as salaries, while another user (MedUser) accesses information regarding company healthcare costs through a healthcare management application. HRUser should not be able to access company healthcare cost information and MedUser should not be able to view personal employee data.

To grant these users access to data they need while restricting them from data they should not see, two schemas are created with appropriate user access, as follows:

- HRSchema—A schema owned by HRUser that is accessed by the HR application.
- HealthSchema—A schema owned by MedUser that is accessed by the healthcare management application.

- **Using Multiple Schemas to Support Multitenancy**

This example is similar to the last example in that access to sensitive data is limited by separating users into different schemas. In this case, however, each user is using a virtual instance of the same application.

An example of this is a retail marketing analytics company that provides data and software as a service (SaaS) to large retailers to help them determine which promotional methods they use are most effective at driving customer sales.

In this example, each database user equates to a retailer, and each user only has access to its own schema. The retail marketing analytics company provides a virtual instance of the same application to each retail customer, and each instance points to the user's specific schema in which to create and update tables. The tables in these schemas use the same names because they are created by instances of the same application, but they do not conflict because they are in separate schemas.

Example of schemas in this database could be:

- MartSchema—A schema owned by MartUser, a large department store chain.
- PharmSchema—A schema owned by PharmUser, a large drug store chain.

- **Using Multiple Schemas to Migrate to a Newer Version of an Application**

Using multiple schemas is an effective way of migrating to a new version of a software application. In this case, a new schema is created to support the new version of the software, and the old schema is kept as long as necessary to support the original version of the software. This is called a "rolling application upgrade."

For example, a company might use a HR application to store employee data. The following schemas could be used for the original and updated versions of the software:

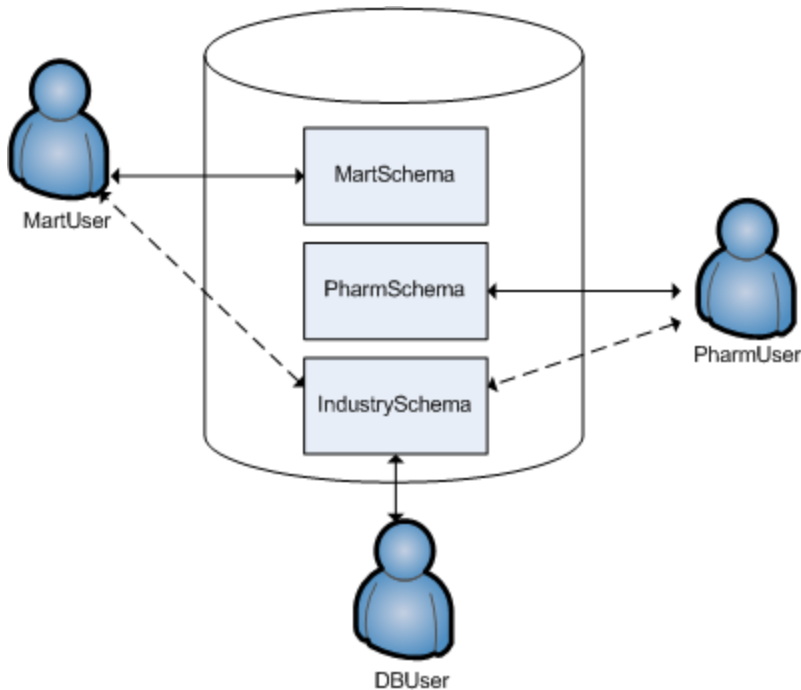
- HRSchema—A schema owned by HRUser, the schema user for the original HR application.
- V2HRSchema—A schema owned by V2HRUser, the schema user for the new version of the HR application.

Using Combinations of Private and Shared Schemas

The previous examples illustrate cases in which all schemas in the database are private and no information is shared between users. However, users might want to share common data. In the retail case, for example, MartUser and PharmUser might want to compare their per store sales of a particular product against the industry per store sales average. Since this information is an industry average and is not specific to any retail chain, it can be placed in a schema on which both users are granted USAGE privileges. (For more information about schema privileges, see [Schema Privileges](#).)

Example of schemas in this database could be:

- MartSchema—A schema owned by MartUser, a large department store chain.
- PharmSchema—A schema owned by PharmUser, a large drug store chain.
- IndustrySchema—A schema owned by DBUser (from the retail marketing analytics company) on which both MartUser and PharmUser have USAGE privileges. It is unlikely that retailers would be given any privileges beyond USAGE on the schema and SELECT on one or more of its tables.



Creating Schemas

You can create as many schemas as necessary for your database. For example, you could create a schema for each database user. However, schemas and users are not synonymous as they are in Oracle.

By default, only a superuser can create a schema or give a user the right to create a schema. (See [GRANT \(Database\)](#) in the SQL Reference Manual.)

To create a schema use the [CREATE SCHEMA](#) statement, as described in the SQL Reference Manual.

Specifying Objects in Multiple Schemas

Once you create two or more schemas, each SQL statement or function must identify the schema associated with the object you are referencing. You can specify an object within multiple schemas by:

- Qualifying the object name by using the schema name and object name separated by a dot. For example, to specify `MyTable`, located in `Schema1`, qualify the name as `Schema1.MyTable`.
- Using a search path that includes the desired schemas when a referenced object is unqualified. By [Setting Search Paths](#), HP Vertica will automatically search the specified schemas to find the object.

Setting Search Paths

The search path is a list of schemas where HP Vertica looks for tables and User Defined Functions (UDFs) that are referenced without a schema name. For example, if a statement references a table named `Customers` without naming the schema that contains the table, and the search path is `public`, `Schema1`, and `Schema2`, HP Vertica first searches the `public` schema for a table named `Customers`. If it does not find a table named `Customers` in `public`, it searches `Schema1` and then `Schema2`.

HP Vertica uses the first table or UDF it finds that matches the unqualified reference. If the table or UDF is not found in any schema in the search path, HP Vertica reports an error.

Note: HP Vertica only searches for tables and UDFs in schemas to which the user has access privileges. If the user does not have access to a schema in the search path, HP Vertica silently skips the schema. It does not report an error or warning if the user's search path contains one or more schemas to which the user does not have access privileges. Any schemas in the search path that do not exist (for example, schemas that have been deleted since being added to the search path) are also silently ignored.

The first schema in the search path to which the user has access is called the *current schema*. This is the schema where HP Vertica creates tables if a `CREATE TABLE` statement does not specify a schema name.

The default schema search path is `"$user", public, v_catalog, v_monitor, v_internal`.

```
=> SHOW SEARCH_PATH;
  name | setting
-----+-----
 search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

The `$user` entry in the search path is a placeholder that resolves to the current user name, and `public` references the public schema. The `v_catalog` and `v_monitor` schemas contain HP Vertica system tables, and the `v_internal` schema is for HP Vertica's internal use.

Note: HP Vertica always ensures that the `v_catalog`, `v_monitor`, and `v_internal` schemas are part of the schema search path.

The default search path has HP Vertica search for unqualified tables first in the user's schema, assuming that a separate schema exists for each user and that the schema uses their user name. If such a user schema does not exist, or if HP Vertica cannot find the table there, HP Vertica next search the public schema, and then the `v_catalog` and `v_monitor` built-in schemas.

A database administrator can set a user's default search schema when creating the user by using the `SEARCH_PATH` parameter of the [CREATE USER](#) statement. An administrator or the user can change the user's default search path using the [ALTER USER](#) statement's `SEARCH_PATH` parameter. Changes made to the default search path using `ALTER USER` affect new user sessions—they do not affect any current sessions.

A user can use the [SET SEARCH_PATH](#) statement to override the schema search path for the current session.

Tip: The `SET SEARCH_PATH` statement is equivalent in function to the `CURRENT_SCHEMA` statement found in some other databases.

To see the current search path, use the [SHOW SEARCH_PATH](#) statement. To view the current schema, use [SELECT CURRENT_SCHEMA\(\)](#). The function `SELECT CURRENT_SCHEMA()` also shows the resolved name of `$user`.

The following example demonstrates displaying and altering the schema search path for the current user session:

```
=> SHOW SEARCH_PATH;
  name | setting
-----+-----
 search_path | "$user", PUBLIC, v_catalog, v_monitor, v_internal
(1 row)

=> SET SEARCH_PATH TO SchemaA, "$user", public;
SET

=> SHOW SEARCH_PATH;
  name | setting
-----+-----
 search_path | SchemaA, "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

You can use the `DEFAULT` keyword to reset the schema search path to the default.

```
=> SET SEARCH_PATH TO DEFAULT;SET
=> SHOW SEARCH_PATH;
  name | setting
```

```
-----+-----
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

To view the default schema search path for a user, query the `search_path` column of the [V_CATALOG.USERS](#) system table:

```
=> SELECT search_path from USERS WHERE user_name = 'ExampleUser';
search_path
-----
"$user", public, v_catalog, v_monitor, v_internal
(1 row)

=> ALTER USER ExampleUser SEARCH_PATH SchemaA,"$user",public;
ALTER USER

=> SELECT search_path from USERS WHERE user_name = 'ExampleUser';
search_path
-----
SchemaA, "$user", public, v_catalog, v_monitor, v_internal
(1 row)

=> SHOW SEARCH_PATH;
name | setting
-----+-----
search_path | "$user", public, v_catalog, v_monitor, v_internal
(1 row)
```

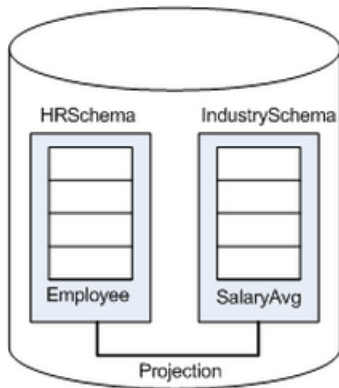
Note that changing the default search path has no effect on the user's current session. Even using the `SET SEARCH_PATH DEFAULT` statement does not set the search path to the newly-defined default value. It only has an effect in new sessions.

See Also

- [System Tables](#)

Creating Objects That Span Multiple Schemas

HP Vertica supports **views** or **pre-join projections** that reference tables across multiple schemas. For example, a user might need to compare employee salaries to industry averages. In this case, the application would query a shared schema (IndustrySchema) for salary averages in addition to its own private schema (HRSchema) for company-specific salary information.



Best Practice: When creating objects that span schemas, use qualified table names. This naming convention avoids confusion if the query path or table structure within the schemas changes at a later date.

Tables in Schemas

In HP Vertica you can create both base tables and temporary tables, depending on what you are trying to accomplish. For example, base tables are created in the HP Vertica **logical schema** while temporary tables are useful for dividing complex query processing into multiple steps.

For more information, see [Creating Tables](#) and [Creating Temporary Tables](#).

About Base Tables

The [CREATE TABLE](#) statement creates a table in the HP Vertica **logical schema**. The example databases described in the [Getting Started Guide](#) include sample SQL scripts that demonstrate this procedure. For example:

```
CREATE TABLE vendor_dimension (
  vendor_key      INTEGER      NOT NULL PRIMARY KEY,
  vendor_name     VARCHAR(64),
  vendor_address  VARCHAR(64),
  vendor_city     VARCHAR(64),
  vendor_state    CHAR(2),
  vendor_region   VARCHAR(32),
  deal_size       INTEGER,
  last_deal_update DATE
);
```

Automatic Projection Creation

To get your database up and running quickly, HP Vertica automatically creates a default **projection** for each table created through the [CREATE TABLE](#) and [CREATE TEMPORARY TABLE](#) statements. Each projection created automatically (or manually) includes a base projection name

prefix. You must use the projection prefix when altering or dropping a projection ([ALTER PROJECTION RENAME](#), [DROP PROJECTION](#)).

How you use the CREATE TABLE statement determines when the projection is created:

- If you create a table without providing the projection-related clauses, HP Vertica automatically creates a **superprojection** for the table when you use an INSERT INTO or COPY statement to load data into the table for the first time. The projection is created in the same schema as the table. Once HP Vertica has created the projection, it loads the data.
- If you use CREATE TABLE AS SELECT to create a table from the results of a query, the table is created first and a projection is created immediately after, using some of the properties of the underlying SELECT query.
- (Advanced users only) If you use any of the following parameters, the default projection is created immediately upon table creation using the specified properties:
 - [column-definition](#) (ENCODING encoding-type and ACCESSRANK integer)
 - ORDER BY table-column
 - [hash-segmentation-clause](#)
 - UNSEGMENTED { NODE *node* | ALL NODES }
 - KSAFE

Note: Before you define a superprojection in the above manner, read [Creating Custom Designs](#) in the Administrator's Guide.

See Also

- [Creating Base Tables](#)
- [Projection Concepts](#)
- [CREATE TABLE](#)

About Temporary Tables

A common use case for a temporary table is to divide complex query processing into multiple steps. Typically, a reporting tool holds intermediate results while reports are generated (for example, first get a result set, then query the result set, and so on). You can also write [subqueries](#).

Note: The default retention when creating temporary tables is ON COMMIT DELETE ROWS, which discards data at transaction completion. The non-default value is ON COMMIT PRESERVE ROWS, which discards data when the current session ends.

You create temporary tables Using the CREATE TEMPORARY TABLE statement.

Global Temporary Tables

HP Vertica creates global temporary tables in the public schema, with the data contents private to the transaction or session through which data is inserted.

Global temporary table definitions are accessible to all users and sessions, so that two (or more) users can access the same global table concurrently. However, whenever a user commits or rolls back a transaction, or ends the session, HP Vertica removes the global temporary table data automatically, so users see only data specific to their own transactions or session.

Global temporary table definitions persist in the database catalogs until they are removed explicitly through a [DROP TABLE](#) statement.

Local Temporary Tables

Local temporary tables are created in the `V_TEMP_SCHEMA` namespace and inserted into the user's search path transparently. Each local temporary table is visible only to the user who creates it, and only for the duration of the session in which the table is created.

When the session ends, HP Vertica automatically drops the table definition from the database catalogs. You cannot preserve non-empty, session-scoped temporary tables using the `ON COMMIT PRESERVE ROWS` statement.

Creating local temporary tables is significantly faster than creating regular tables, so you should make use of them whenever possible.

Automatic Projection Creation and Characteristics

Once local or global table exists, HP Vertica creates auto-projections for temporary tables whenever you load or insert data.

The default auto-projection for a temporary table has the following characteristics:

- It is a **superprojection**.
- It uses the default encoding-type `AUTO`.
- It is automatically unsegmented on the initiator node, if you do not specify a [hash-segmentation-clause](#).
- The projection is not **pinned**.
- Temp tables are not recoverable, so the superprojection is not K-Safe (`K-SAFE=0`), and you cannot make it so.

Auto-projections are defined by the table properties and creation methods, as follows:

| If table... | Sort order is... | Segmentation is... |
|--|--|---|
| Is created from input stream (COPY or INSERT INTO) | Same as input stream, if sorted. | On PK column (if any), on all FK columns (if any), on the first 31 configurable columns of the table |
| Is created from CREATE TABLE AS SELECT query | Same as input stream, if sorted. If not sorted, sorted using following rules. | Same segmentation columns f query output is segmented The same as the load, if output of query is unsegmented or unknown |
| Has FK and PK constraints | FK first, then PK columns | PK columns |
| Has FK constraints only (no PK) | FK first, then remaining columns | Small data type (< 8 byte) columns first, then large data type columns |
| Has PK constraints only (no FK) | PK columns | PK columns |
| Has no FK or PK constraints | On all columns | Small data type (< 8 byte) columns first, then large data type columns |

Advanced users can modify the default projection created through the CREATE TEMPORARY TABLE statement by defining one or more of the following parameters:

- [column-definition \(temp table\)](#) (ENCODING encoding-type and ACCESSRANK integer)
- ORDER BY table-column
- [hash-segmentation-clause](#)
- UNSEGMENTED { NODE node | ALL NODES }
- NO PROJECTION

Note: Before you define the superprojection in this manner, read [Creating Custom Designs](#) in the Administrator's Guide.

See Also

- [Creating Temporary Tables](#)
- [Projection Concepts](#)
- [CREATE TEMPORARY TABLE](#)

Implementing Views

A view is a stored query that dynamically accesses and computes data from the database at execution time. It differs from a projection in that it is not **materialized**: it does not store data on disk. This means that it doesn't need to be refreshed whenever the data in the underlying tables change, but it does require additional time to access and compute data.

Views are read-only and they support references to tables, temp tables, and other views. They do not support inserts, deletes, or updates. You can use a view as an abstraction mechanism to:

- Hide the complexity of `SELECT` statements from users for support or security purposes. For example, you could create a view that selects specific columns from specific tables to ensure that users have easy access to the information they need while restricting them from confidential information.
- Encapsulate the details of the structure of your tables, which could change as your application evolves, behind a consistent user interface.

Creating Views

A view contains one or more `SELECT` statements that reference any combination of one or more tables, temp tables, or views. Additionally, views can specify the column names used to display results.

The user who creates the view must be a **superuser** or have the following privileges:

- `CREATE` on the schema in which the view is created.
- `SELECT` on all the tables and views referenced within the view's defining query.
- `USAGE` on all the schemas that contain the tables and views referenced within the view's defining query.

To create a view:

1. Use the `CREATE VIEW` statement to create the view.
2. Use the `GRANT (View)` statement to grant users the privilege to use the view.

Note: Once created, a view cannot be actively altered. It can only be deleted and recreated.

Using Views

Views can be used in the `FROM` clause of any SQL query or subquery. At execution, HP Vertica internally substitutes the name of the view used in the query with the actual contents of the view. The following example defines a view (`ship`) and illustrates how a query that refers to the view is transformed internally at execution time.

- **New view**

```
=> CREATE VIEW ship AS SELECT * FROM public.shipping_dimension;
```

- **Original query**

```
=> SELECT * FROM ship;
```

- **Transformed query**

```
=> SELECT * FROM (SELECT * FROM public.shipping_dimension) AS ship;
```

Tip: : To use a view, a user must be granted SELECT permissions on the view. See [GRANT \(View\)](#).

The following example creates a view named `myview` that sums all individual incomes of customers listed in the `store.store_sales_fact` table by state. The results are grouped in ascending order by state.

```
=> CREATE VIEW myview AS
  SELECT SUM(annual_income), customer_state
  FROM public.customer_dimension
  WHERE customer_key IN
    (SELECT customer_key
     FROM store.store_sales_fact)
  GROUP BY customer_state
  ORDER BY customer_state ASC;
```

The following example uses the `myview` view with a `WHERE` clause that limits the results to combined salaries of greater than 2,000,000,000.

```
=> SELECT * FROM myview where sum > 2000000000;
```

| SUM | customer_state |
|-------------|----------------|
| 2723441590 | AZ |
| 29253817091 | CA |
| 4907216137 | CO |
| 3769455689 | CT |
| 3330524215 | FL |
| 4581840709 | IL |
| 3310667307 | IN |
| 2793284639 | MA |
| 5225333668 | MI |
| 2128169759 | NV |
| 2806150503 | PA |
| 2832710696 | TN |

```
14215397659 | TX
2642551509 | UT
(14 rows)
```

Notes

If HP Vertica does not have to evaluate an expression that would generate a run-time error in order to answer a query, the run-time error might not occur. See the following sequence of commands for an example of this scenario.

If you run a query like the following, HP Vertica returns an error:

```
=> SELECT TO_DATE('F','dd mm yyyy') FROM customer_dimension;
ERROR: Invalid input for DD: "F"
```

Now create a view using the same query. Note that the view gets created when you would expect it to return the same error:

```
=> CREATE VIEW temp AS SELECT TO_DATE('F','dd mm yyyy')
FROM customer_dimension;
CREATE VIEW
```

The view, however, cannot be used in all queries without generating the same error message. For example, the following query returns the same error, which is what you would expect:

```
=> SELECT * FROM temp;
ERROR: Invalid input for DD: "F"
```

When you then issue a COUNT command, the returned rowcount is correct:

```
=> SELECT COUNT(*) FROM temp;
COUNT
-----
100
(1 row)
```

This behavior works as intended. You might want to create views that contain subqueries, where not every row is intended to pass the predicate.

Creating a Database Design

Data in HP Vertica is physically stored in projections. When you initially load data into a table using `INSERT`, `COPY` (or `COPY LOCAL`), HP Vertica creates a default **superprojection** for the table. This superprojection ensures that all of the data is available for queries. However, these superprojections might not optimize database performance, resulting in slow query performance and low data compression.

To improve performance, create a physical design for your database that optimizes both query performance and data compression. You can [use the Database Designer](#) or [create this design by hand](#).

Database Designer is a tool that recommends the design of design (**projections**) that provide the best query performance. Using Database Designer minimizes the time you spend on manual database tuning and provides the ability to re-design the database incrementally to optimize for changing workloads over time.

Database Designer runs as a background process. If non-superusers are running Database Designer on, or deploying for the same tables at the same time, Database Designer may not be able to complete.

Tip: HP recommends that you first globally optimize your database using the Comprehensive setting in Database Designer. If the performance of the comprehensive design is not adequate, you can design custom projections using an incremental design and manually, as described in [Creating Custom Designs](#).

What Is a Design?

A *design* is a physical storage plan that optimizes query performance. Database Designer uses sophisticated strategies to create a design that provides excellent performance for ad-hoc queries and specific queries while using disk space efficiently. Database Designer bases the design on the following information that you provide:

- Design type (comprehensive or incremental)
- Optimization objective (query, load, or balanced)
- K-safety
- Design queries: Typical queries that you run during normal database operations. Each query can be assigned a weight that indicates its relative importance so that Database Designer can prioritize it when creating the design. Database Designer groups queries that affect the design that Database Designer creates in the same way and considers one weighted query when creating a design.
- Design tables that contain sample data.

- Setting that specifies that Database Designer be guided to create only unsegmented projections.
- Setting that specifies that Database Designer analyze statistics before creating the design.

The result of a Database Designer run is:

- A design script that creates the projections for the design in a way that meets the optimization objectives and distributes data uniformly across the cluster.
- A deployment script that creates and refreshes the projections for your design. For comprehensive designs, the deployment script contains commands that remove non-optimized projections. The deployment script includes the full design script.
- A backup script that contains SQL statements to deploy the design that existed on the system before deployment. This file is useful in case you need to revert to the pre-deployment design.

While running Database Designer, you can choose to deploy your design automatically after the deployment script is created, or to deploy it manually, after you have reviewed and tested the design. HP Vertica recommends that you test the design on a non-production server before deploying the design to your production server.

How Database Designer Creates a Design

During the design process, Database Designer analyzes the logical schema definition, sample data, and sample queries, and creates a physical schema (**projections**) in the form of a SQL script that you deploy automatically or manually. The script creates a minimal set of superprojections to ensure K-safety.

In most cases, the projections that Database Designer creates provide excellent query performance within physical constraints while using disk space efficiently. Database Designer:

- Recommends **buddy projections** with the same sort order, which can significantly improve load, recovery, and site node performance. All buddy projections have the same base name so that they can be identified as a group.

Note: If you manually create projections, Database Designer recommends a buddy with the same sort order, if one does not already exist. By default, Database Designer recommends both super and non-super segmented projections with a buddy of the same sort order and segmentation.

- Automatically rebalances data after you add or remove nodes.
- Accepts queries as design input.
- Runs the design and deployment processes in the background.

This is useful if you have a large design that you want to run overnight. An active SSH session is not required, so design/deploy operations continue to run uninterrupted, even if the session is terminated.

- Accepts a file of sample queries for Database Designer to consider when creating a design. Providing this file is optional for comprehensive designs. If you do not provide this file, Database Designer recommends a generic design that does not consider specific queries. For incremental designs, you *must* provide sample queries; the query file can contain up to 100 queries.
- Accepts unlimited queries for a comprehensive design.
- Allows you to analyze column correlations, which the Database Designer and query optimizer exploit to improve data compression and query performance. Correlation analysis typically only needs to be performed once, and only if the table has more than `DBDCorrelationSampleRowCount` (default: 4000) rows.

By default, Database Designer does not analyze column correlations. To set the correlation analysis mode, use [DESIGNER_SET_ANALYZE_CORRELATIONS_MODE](#)

- Identifies similar design queries and assigns them a signature. Of queries with the same signature, Database Designer weights the queries depending on how many have that signature and considers the weighted query when creating a design.
- Creates projections in a way that minimizes data skew by distributing data uniformly across the cluster.
- Produces higher quality designs by considering UPDATE and DELETE statements, as well as SELECT statements.
- Does not sort, segment, or partition projections on LONG VARBINARY and LONG VARCHAR columns.

Who Can Run Database Designer

To use Administration Tools to run Database Designer and create an optimal database design, you must be a DBADMIN user.

To run Database Designer programmatically or using Management Console, you must be one of two types of users:

- DBADMIN user
- Have been assigned the DBDUSER role and be the owner of database tables for which you are creating a design

Granting and Enabling the DBDUSER Role

For a non-DBADMIN user to be able to run Database Designer using Management Console, follow the steps described in [Allowing the DBDUSER to Run Database Designer Using Management](#)

Console.

For a non-DBADMIN user to be able to run Database Designer programmatically, following the steps described in [Allowing the DBDUSER to Run Database Designer Programmatically](#).

Important: When you grant the DBDUSER role, make sure to associate a resource pool with that user to manage resources during Database Designer runs. (For instructions about how to associate a resource pool with a user, see [User Profiles](#).)

Multiple users can run Database Designer concurrently without interfering with each other or using up all the cluster resources. When a user runs Database Designer, either using the Management Console or programmatically, its execution is mostly contained by the user's resource pool, but may spill over into system resource pools for less-intensive tasks.

Allowing the DBDUSER to Run Database Designer Using Management Console

To allow a user with the DBDUSER role to run Database Designer using Management Console, you first need to create the user on the HP Vertica server.

As DBADMIN, take these steps on the server:

1. Add a /tmp folder to all cluster nodes.

```
=> SELECT ADD_LOCATION('/tmp');
```

2. Create the user who needs access to Database Designer.

```
=> CREATE USER new_user;
```

3. Grant the user the privilege to create schemas on the database for which they want to create a design.

```
=> GRANT CREATE on new_database TO new_user;
```

4. Grant the DBDUSER role to the new user.

```
=> GRANT DBDUSER TO new_user;
```

5. On all nodes in the cluster, grant the user access to the /tmp folder.

```
=> GRANT ALL ON LOCATION '/tmp' TO new_user;
```

6. Grant the new user access to the database schema and its tables.

```
=> GRANT ALL ON SCHEMA user_schema TO new_user;
=> GRANT ALL ON ALL TABLES IN SCHEMA user_schema TO new_user;
```

After you have completed this task, you need to do the following to map the MC user to the new_user you created in the previous steps:

1. Log in to Management Console as an MC Super user.
2. Click **MC Settings**.
3. Click **User Management**.
4. To create a new MC user, click **Add**. To use an existing MC user, select the user and click **Edit**.
5. Next to the **DB access level** window, click **Add**.
6. In the **Add Permissions** window, do the following:
 - a. From the **Choose a database** drop-down list, select the database for which you want the user to be able to create a design.
 - b. In the **Database username** field, enter the user name you created on the HP Vertica server, new_user in this example.
 - c. In the Database password field, enter the password for the database you selected in step a.
 - d. In the **Restrict access** drop-down list, select the level of MC user you want for this user.
7. Click **OK** to save your changes.
8. Log out of the MC Super user account.

The MC user is now mapped to the user that you created on the HP Vertica server. Log in as the MC user and use Database Designer to create an optimized design for your database.

For more information about mapping MC users, see [Mapping an MC User to a Database user's Privileges](#).

Allowing the DBDUSER to Run Database Designer Programmatically

To allow a user with the DBDUSER role to run Database Designer programmatically, take these steps:

1. The DBADMIN user must grant the DBDUSER role:

```
=> GRANT DBDUSER TO <username>;
```

This role persists until the DBADMIN user revokes it.

2. For a non-DBADMIN user to run the Database Designer programmatically or using Management Console, one of the following two steps must happen first:

- If the user's default role is already DBDUSER, skip this step. Otherwise, The user must enable the DBDUSER role:

```
=> SET ROLE DBDUSER;
```

- The DBADMIN must add DBDUSER as the default role for that user:

```
=> ALTER USER <username> DEFAULT ROLE DBDUSER;
```

DBDUSER Capabilities and Limitations

The DBDUSER role has the following capabilities and limitations:

- A DBDUSER cannot create a design with a K-safety less than the system K-safety. If the designs violate the current K-safet by not having enough buddy projections for the tables, the design does not complete.
- A DBDUSER cannot explicitly change the ancient history mark (AHM), even during deployment of their design.

When you create a design, you automatically have privileges to manipulate the design. Other tasks may require that the DBDUSER have additional privileges:

| To... | DBDUSER must have... |
|------------------------------|---|
| Add design tables | <ul style="list-style-type: none"> • USAGE privilege on the design table schema • OWNER privilege on the design table |
| Add a single design query | <ul style="list-style-type: none"> • Privilege to execute the design query |
| Add a file of design queries | <ul style="list-style-type: none"> • Read privilege on the storage location that contains the query file • Privilege to execute all the queries in the file |

| To... | DBDUSER must have... |
|--|--|
| Add design queries from the result of a user query | <ul style="list-style-type: none"> • Privilege to execute the user query • Privilege to execute each design query retrieved from the results of the user query |
| Create the design and deployment scripts | <ul style="list-style-type: none"> • WRITE privilege on the storage location of the design script • WRITE privilege on the storage location of the deployment script |

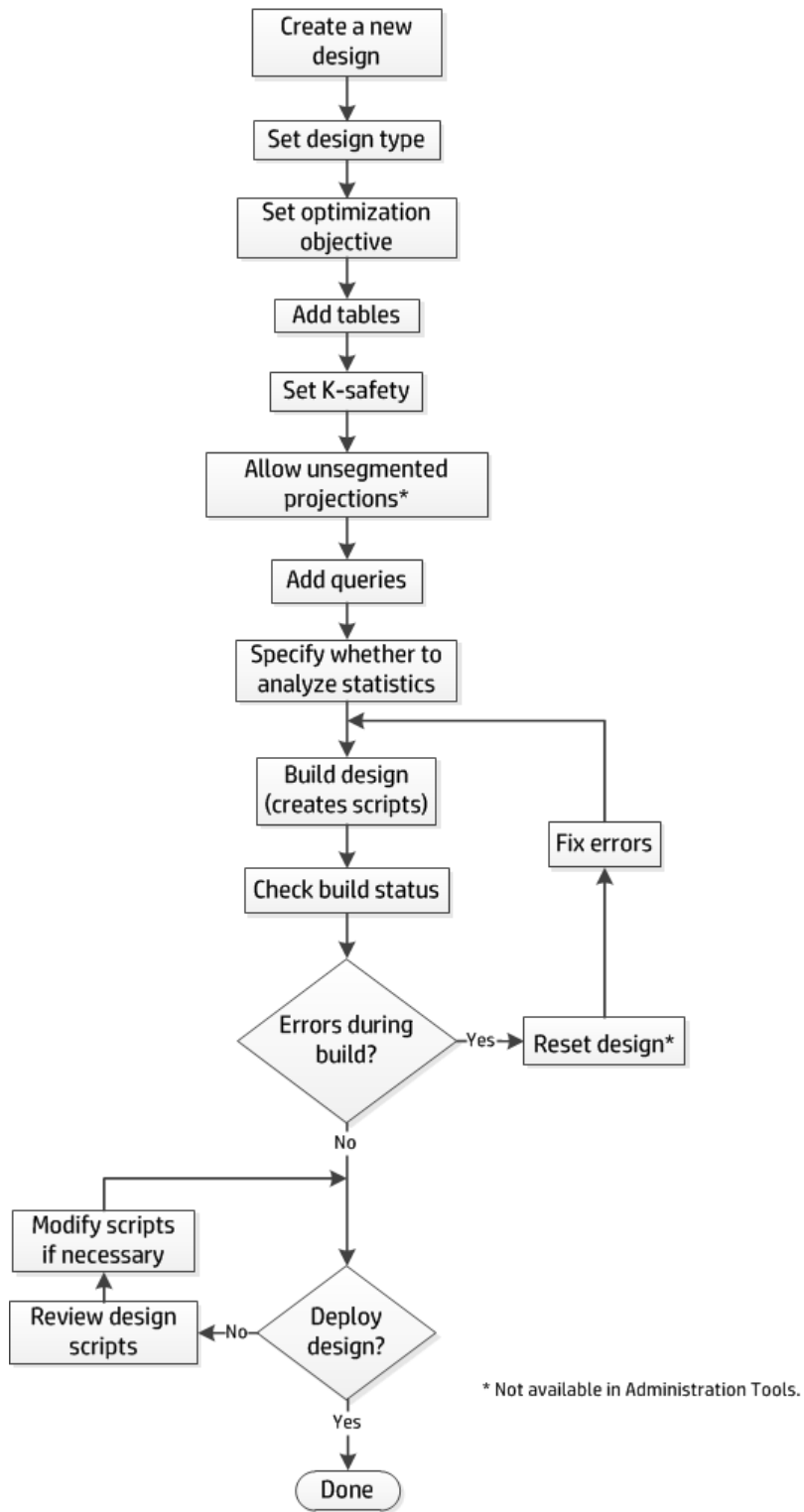
Workflow for Running Database Designer

HP Vertica provides three ways to run Database Designer:

- [Using Management Console to Create a Design](#)
- [Using Administration Tools to Create a Design](#)
- [About Running Database Designer Programmatically](#)

The following workflow is common to all these ways to run Database Designer:

Typical Workflow for Creating a Design with Database Designer



Specifying Parameters for Database Designer

Before you run Database Designer to create a design, provide information that allows Database Designer to create the optimal physical schema:

- [Design Name](#)
- [Design Types](#)
- [Optimization Objectives](#)
- [Design Tables with Sample Data](#)
- [Design Queries](#)
- [K-Safety for Design](#)
- [Replicated and Unsegmented Projections](#)
- [Statistics Analysis](#)

Design Name

All designs that Database Designer creates must have a name that you specify. The design name must be alphanumeric or underscore (_) characters, and can be no more than 32 characters long. (Administrative Tools and Management Console limit the design name to 16 characters.)

The design name becomes part of the files that Database Designer generates, including the deployment script, allowing the files to be easily associated with a particular Database Designer run.

Design Types

The Database Designer can create two distinct design types. The design you choose depends on what you are trying to accomplish:

- [Comprehensive Design](#)
- [Incremental Design](#)

Comprehensive Design

A comprehensive design creates an initial or replacement design for all the tables in the specified schemas. Create a comprehensive design when you are creating a new database.

To help Database Designer create an efficient design, load representative data into the tables before you begin the design process. When you load data into a table, HP Vertica creates an unoptimized **superprojection** so that Database Designer has projections to optimize. If a table has no data, Database Designer cannot optimize it.

Optionally, supply Database Designer with representative queries that you plan to use so Database Designer can optimize the design for them. If you do not supply any queries, Database Designer creates a generic optimization of the superprojections that minimizes storage, with no query-specific projections.

During a comprehensive design, Database Designer creates deployment scripts that:

- Create new projections to optimize query performance, only when they do not already exist.
- Create replacement buddy projections when Database Designer changes the encoding of pre-existing projections that it has decided to keep.

Incremental Design

An incremental design creates an enhanced design with additional projections, if required, that are optimized specifically for the queries that you provide. Create an incremental design when you have one or more queries that you want to optimize.

Optimization Objectives

When creating a design, Database Designer can optimize the design for one of three objectives:

- **Load** Database Designer creates a design that is optimized for loads, minimizing database size, potentially at the expense of query performance.
- **Performance** Database Designer creates a design that is optimized for fast query performance. Because it recommends a design for optimized query performance, this design might recommend more than the Load or Balanced objectives, potentially resulting in a larger database storage size.
- **Balanced** Database Designer creates a design whose objectives are balanced between database size and query performance.

Design Tables with Sample Data

You *must* specify one or more design tables for Database Designer to deploy a design. If your schema is empty, it does not appear as a design table option.

To create the most efficient projections for your database, load a moderate amount of representative data into tables before running Database Designer. Database Designer considers the data in this table when creating the design.

If your design tables have a large amount of data, the Database Designer run takes a long time; if your tables have too little data, the design is not optimized. HP Vertica recommends that 10 GB of sample data is sufficient for creating an optimal design.

If you submit a design table with no data, Database Designer ignores it.

Design Queries

If you supply representative queries that you run on your database to Database Designer, it optimizes the performance of those queries.

If you are creating an incremental design, you *must* supply design queries; if you are creating a comprehensive design, HP Vertica recommends you supply design queries to create an optimal design.

Database Designer checks the validity of all queries when you add them to your design and again when it builds the design. If a query is invalid, Database Designer ignores it.

Query Repository

Using Management Console, you can submit design queries from the [QUERY_REQUESTS](#) system table. This is called *the query repository*.

The QUERY_REQUESTS table contains queries that users have run recently. For a comprehensive design, you can submit up to 200 queries from the QUERY_REQUESTS table to Database Designer to be considered when creating the design. For an incremental design, you can submit up to 100 queries from the QUERY_REQUESTS table.

K-Safety for Design

When you create a comprehensive design, you can set a K-safety value for your design. Valid values are 0, 1, and 2, and the value you specify must be less than or equal to the K-safety value of your cluster.

The default K-safety is as follows:

- If your cluster has one or two nodes, the default K-safety is 0.
- If your cluster has three or more nodes, the default K-safety is 1.

For a comprehensive design, you can make the following changes to the design K-safety before deploying the design:

- If your cluster has one or two nodes, you cannot change the K-safety.
- If your cluster has three or four nodes, you change the K-safety to 1 or 0.
- If your cluster has five or more nodes, you can change the K-safety to 2, 1, or 0.

You cannot change the K-safety value of an incremental design. Incremental designs assume the K-safety value of your cluster.

For more information about K-safety, see [K-Safety](#) in the Concepts Guide.

Replicated and Unsegmented Projections

When creating a comprehensive design, Database Designer creates projections based on data statistics and queries. It also reviews the submitted design tables to decide whether projections should be segmented (distributed across the cluster nodes) or replicated (duplicated on all cluster nodes).

For detailed information, see the following sections:

- [Replicated Projections](#)
- [Unsegmented Projections](#)

Replicated Projections

Replication occurs when HP Vertica stores identical copies of data across all nodes in a cluster.

If you are running on a single-node database, *all* projections are replicated because segmentation is not possible in a single-node database.

Assuming that *largest-row-count* equals the number of rows in the design table with the largest number of rows, Database Designer recommends that a projection be replicated if any one of the following is true:

- Condition 1: *largest-row-count* < 1,000,000 and number of rows in the table <= 10% of *largest-row-count*.
- Condition 2: *largest-row-count* >= 10,000,000 and number of rows in the table <= 1% of *largest-row-count*.
- Condition 3: The number of rows in the table <= 100,000.

For more information about replication, see [High Availability Through Projections](#) in the Concepts Guide.

Unsegmented Projections

Segmentation occurs when HP Vertica distributes data evenly across multiple database nodes so that all nodes participate in query execution. Projection segmentation provides high availability and recovery, and optimizes query execution.

When running Database Designer programmatically or using Management Console, you can specify to allow Database Designer to recommend unsegmented projections in the design. If you do not specify this, Database Designer recommends only segmented projections.

Database Designer recommends segmented superprojections for large tables when deploying to multiple node clusters, and recommends replicated superprojections for smaller tables.

Database Designer does not segment projections on:

- Single-node clusters
- LONG VARCHAR and LONG VARBINARY columns

For more information about segmentation, see [High Availability Through Projections](#) in the Concepts Guide.

Statistics Analysis

By default, Database Designer analyzes statistics for the design tables when adding them to the design. This option is optional, but HP HP Vertica recommends that you analyze statistics because accurate statistics help Database Designer optimize compression and query performance.

Analyzing statistics takes time and resources. If the current statistics for the design tables are up to date, do not bother analyzing the statistics. When in doubt, analyze the statistics to make sure they are current.

For more information, see [Collecting Statistics](#).

Building a Design

After you have created design tables and loaded data into them, and then specified the parameters you want Database Designer to use when creating the physical schema, direct Database Designer to create the scripts necessary to build the design.

When you build a database design, HP Vertica generates two scripts:

- **Deployment script**—<design_name>_deploy.sql—Contains the SQL statements that create projections for the design you are deploying, deploy the design, and drop unused projections. When the deployment script runs, it creates the optimized design. For details about how to run this script and deploy the design, see [Deploying a Design](#).
- **Design script**—<design_name>_design.sql—Contains the CREATE PROJECTION statements that Database Designer uses to create the design. Review this script to make sure you are happy with the design.

The design script is a subset of the deployment script. It serves as a backup of the DDL for the projections that the deployment script creates.

If you run Database Designer from Administrative Tools, HP HP Vertica also creates a backup script named <design_name>_projection_backup_<unique id #>.sql. This script contains SQL statements to deploy the design that existed on the system before deployment. This file is useful in case you need to revert to the pre-deployment design.

When you create a design using Management Console:

- If you submit a large number of queries to your design and build it right immediately, a timing issue could cause the queries not to load before deployment starts. If this occurs, you may see

one of the following errors:

- No queries to optimize for
- No tables to design projections for

To accommodate this timing issue, you may need to reset the design, check the **Queries** tab to make sure the queries have been loaded, and then rebuild the design. Detailed instructions are in:

- [Using the Wizard to Create a Design](#)
 - [Creating a Design Manually](#)
- The scripts are deleted when deployment completes. To save a copy of the deployment script after the design is built but before the deployment completes, go to the **Output** window and copy and paste the SQL statements to a file.

Resetting a Design

You must reset a design when:

- You build a design and the output scripts described in [Building a Design](#) are not created.
- You build a design but Database Designer cannot complete the design because the queries it expects are not loaded.

Resetting a design discards all the run-specific information of the previous Database Designer build, but retains its configuration (design type, optimization objectives, K-safety, etc.) and tables and queries.

After you reset a design, review the design to see what changes you need to make. For example, you can fix errors, change parameters, or check for and add additional tables or queries. Then you can rebuild the design.

You can only reset a design in Management Console or by using the [DESIGNER_RESET_DESIGN](#) function.

Deploying a Design

After running Database Designer to generate a deployment script, HP Vertica recommends that you test your design on a non-production server before you deploy it to your production server.

Both the design and deployment processes run in the background. This is useful if you have a large design that you want to run overnight. Because an active SSH session is not required, the design/deploy operations continue to run uninterrupted, even if the session is terminated.

Database Designer runs as a background process. Multiple users can run Database Designer concurrently without interfering with each other or using up all the cluster resources. However, if multiple users are deploying a design on the same tables at the same time, Database Designer may not be able to complete the deployment. To avoid problems, consider the following:

- Schedule potentially conflicting Database Designer processes to run sequentially overnight so that there are no concurrency problems.
- Avoid scheduling Database Designer runs on the same set of tables at the same time.

There are two ways to deploy your design:

- [Deploying Designs Using Database Designer](#)
- [Deploying Designs Manually](#)

Deploying Designs Using Database Designer

HP recommends that you run Database Designer and deploy optimized projections right after loading your tables with sample data because Database Designer provides projections optimized for the current state of your database.

If you choose to allow Database Designer to automatically deploy your script during a comprehensive design and are running Administrative Tools, Database Designer creates a backup script of your database's current design. This script helps you re-create the design of projections that may have been dropped by the new design. The backup script is located in the output directory you specified during the design process.

If you choose not to have Database Designer automatically run the deployment script (for example, if you want to maintain projections from a pre-existing deployment), you can manually run the deployment script later. See [Deploying Designs Manually](#).

To deploy a design while running Database Designer, do one of the following:

- In Management Console, select the design and click **Deploy Design**.
- In the Administration Tools, select **Deploy design** in the **Design Options** window.

If you are running Database Designer programmatically, use `DESIGNER_RUN_POPULATE_DESIGN_AND_DEPLOY` and set the `deploy` parameter to 'true'.

Once you have deployed your design, query the `DEPLOY_STATUS` system table to see the steps that the deployment took:

```
vmartdb=> SELECT * FROM V_MONITOR.DEPLOY_STATUS;
```

Deploying Designs Manually

If you chose *not* to have Database Designer deploy your design at design time, you can deploy the design later using the deployment script:

1. Make sure that you have a database that contains the same tables and projections as the database on which you ran Database Designer. The database should also contain sample data.
2. To deploy the projections to a test or production environment, use the following `vsql` command to execute the deployment script, where `<design_name>` is the name of the database design:

```
=> \i <design_name>_deploy.sql
```

How to Create a Design

There are three ways to create a design using Database Designer:

- From Management Console, open a database and select the **Design** page at the bottom of the window.

For details about using Management Console to create a design, see [Using Management Console to Create a Design](#)

- Programmatically, using the techniques described in [About Running Database Designer Programmatically](#) in the Programmer's Guide. To run Database Designer programmatically, you must be a `DBADMIN` or have been granted the `DBDUSER` role and enabled that role.
- From the Administration Tools menu, by selecting **Configuration Menu > Run Database Designer**. You must be a `DBADMIN` user to run Database Designer from the Administration Tools.

For details about using Administration Tools to create a design, see [Using Administration Tools to Create a Design](#).

The following table shows what Database Designer capabilities are available in each tool:

| Database Designer Capability | Management Console | Running Database Designer Programmatically | Administrative Tools |
|------------------------------|--------------------|--|----------------------|
| Create design | Yes | Yes | Yes |

| Database Designer Capability | Management Console | Running Database Designer Programmatically | Administrative Tools |
|---|--------------------|--|----------------------|
| Design name length (# of characters) | 16 | 32 | 16 |
| Build design (create design and deployment scripts) | Yes | Yes | Yes |
| Create backup script | | | Yes |
| Set design type (comprehensive or incremental) | Yes | Yes | Yes |
| Set optimization objective | Yes | Yes | Yes |
| Add design tables | Yes | Yes | Yes |
| Add design queries file | Yes | Yes | Yes |
| Add single design query | | Yes | |
| Use query repository | Yes | Yes | |
| Set K-safety | Yes | Yes | Yes |
| Analyze statistics | Yes | Yes | Yes |
| Require all unsegmented projections | Yes | Yes | |
| View event history | Yes | Yes | |
| Set correlation analysis mode (Default = 0) | | Yes | |

Using Management Console to Create a Design

To use Management Console to create an optimized design for your database, you must be a DBADMIN user or have been assigned the DBDUSER role.

Management Console provides two ways to create a design

- **Wizard**—This option walks you through the process of configuring a new design. Click **Back** and **Next** to navigate through the Wizard steps, or **Cancel** to cancel creating a new design.

To learn how to use the Wizard to create a design, see [Using the Wizard to Create a Design](#).

- **Manual**—This option creates and saves a design with the default parameters.

To learn how to create a design manually, see [Creating a Design Manually](#)

Tip: If you have many design tables that you want Database Designer to consider, it might be easier to use the Wizard to create your design. In the Wizard, you can submit all the tables in a schema at once; creating a design manually requires that you submit the design tables one at a time.

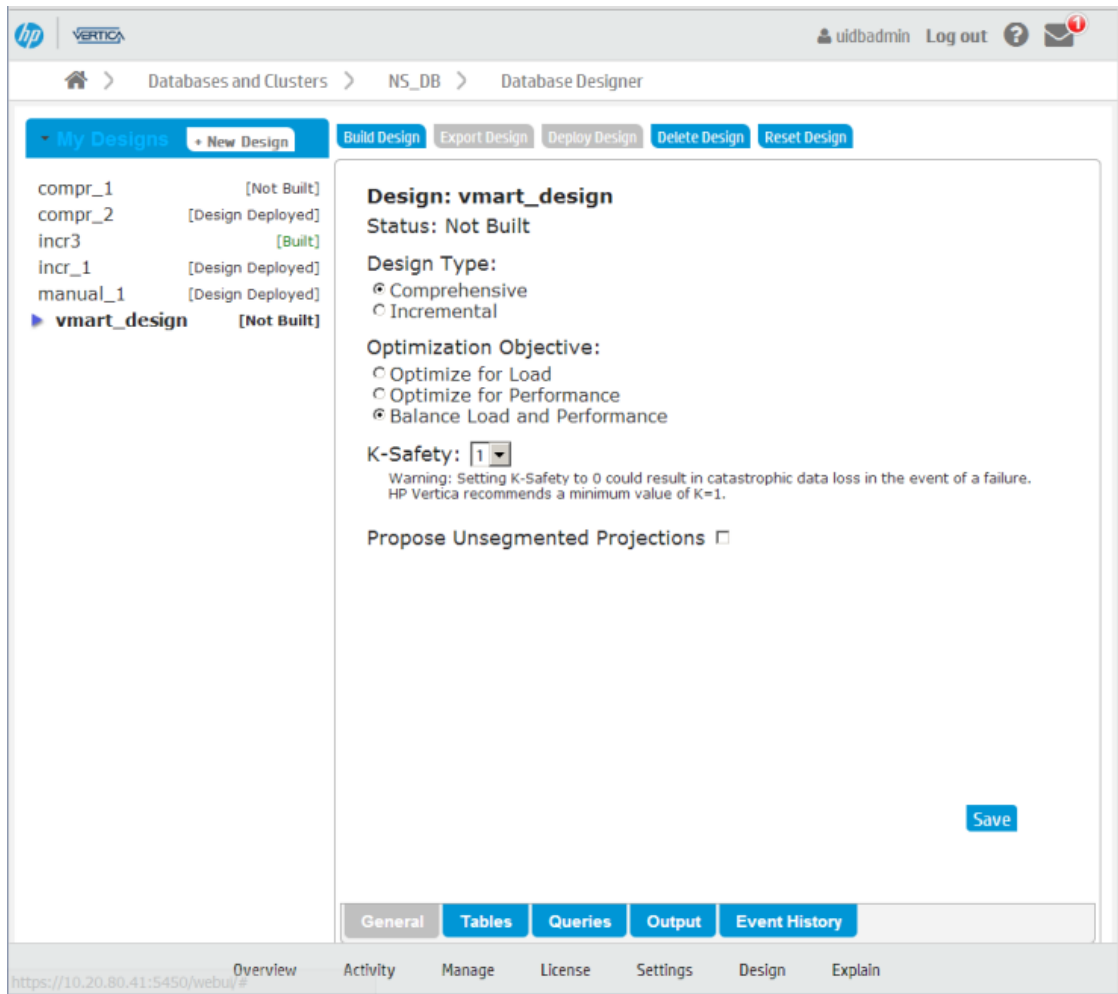
Using the Wizard to Create a Design

Take these steps to create a design using the Management Console's Wizard:

1. Log in to Management Console, select and start your database, and click **Design** at the bottom of the window. The Database Designer window appears. If there are no existing designs, the **New Design** window appears.

The left-hand side of the Database Designer window lists the database designs for which you are the owner, with the most recent design you worked on selected. That pane also lists the current status of the design.

The main pane contains details about the selected design.



2. To create a new design, click **New Design**.

3. Enter a name for your design, and click **Wizard**.

For more information, see [Design Name](#).

4. Navigate through the Wizard using the **Back** and **Next** buttons.

5. To build the design immediately after exiting the Wizard, on the **Execution Options** window, select **Auto-build**.

Important: Hewlett-Packard does not recommend that you auto-deploy the design from the Wizard. There may be a delay in adding the queries to the design, so if the design is deployed but the queries have not yet loaded, deployment may fail. If this happens, reset the design, check the **Queries** tab to make sure the queries have been loaded, and deploy the design.

- When you have entered all the information, the Wizard displays a summary of your choices. Click **Submit Design** to build your design.

Summary

Review these design choices. Click **Submit Design** when you finish configuring your design.

Design Name: VMart_design
Design Type: comprehensive
Optimization Objective: balanced
Schemas: public, store, online_sales
K-Safety: 1
Propose Unsegmented Projections: true
Query File to Upload:
User Query Repository: true
Execution Options - Analyze Statistics: true
Execution Options - Auto-build: true
Execution Options - Auto-deploy: false

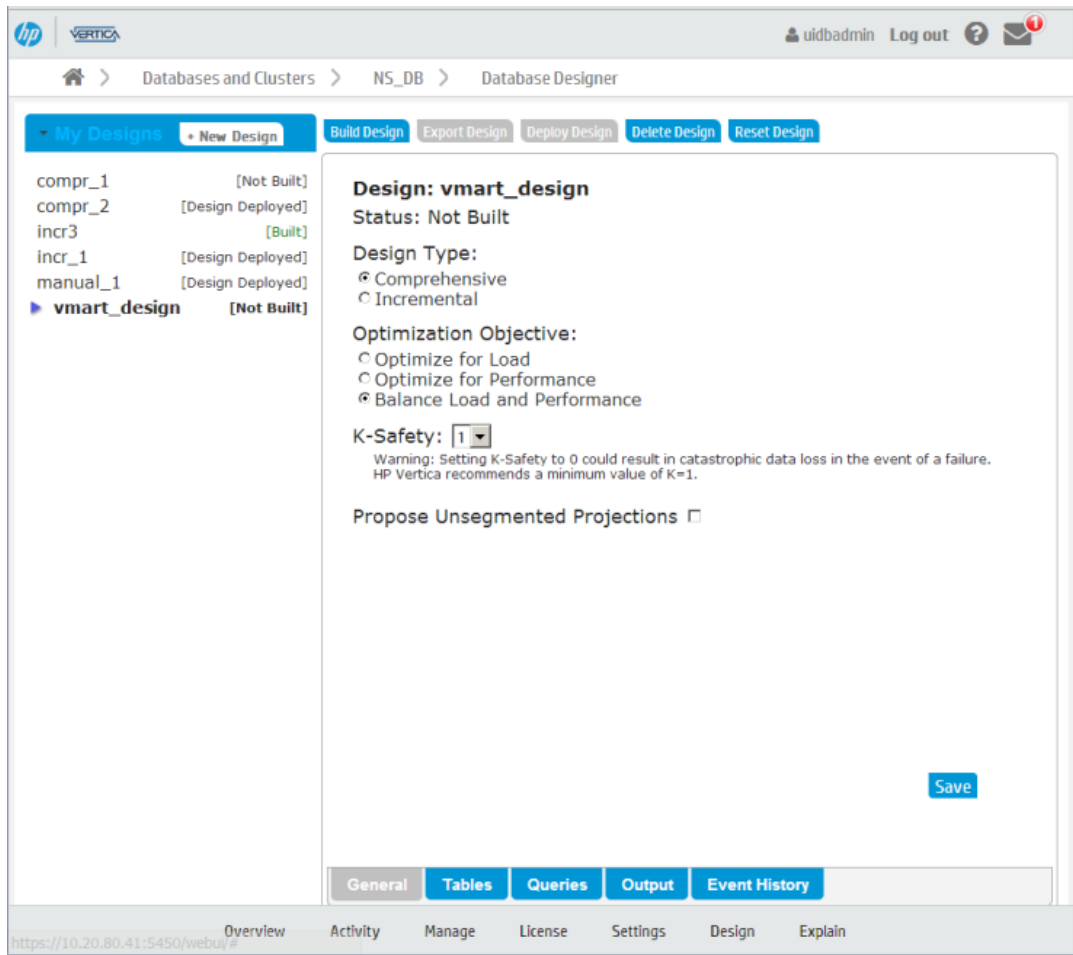
Creating a Design Manually

To create a design using the Management Console's Wizard, take these steps.

- Log in to Management Console, select and start your database, and click **Design** at the bottom of the window. The Database Designer window appears.

The left-hand side of the Database Designer window lists the database designs for which you are the owner, with the most recent design you worked on highlighted. That pane also lists the current status of the design. Details about the most recent design appears in the main pane.

The main pane contains details about the selected design.



2. To create a new design, click **New Design**.

3. Enter a name for your design and select **Manual**.

After a few seconds, the main **Database Design** window opens, displaying the default design parameters. HP Vertica has created and saved a design with the name you specified, and assigned it the default parameters.

For more information, see [Design Name](#).

4. On the **General** window, modify the design type, optimization objectives, K-safety, and the setting that allows Database Designer to create unsegmented projections.

If you choose **Incremental**, the design automatically optimizes for load and the K-safety defaults to the value of the cluster K-safety; you cannot change these values for an incremental design.

5. Click the **Tables** tab. You must submit tables to your design.

6. To add tables of sample data to your design, click **Add Tables**. A list of available tables appears; select the tables you want and click **Save**. If you want to remove tables from your design, click the tables you want to remove, and click **Remove Selected**.
7. Click the **Queries** tab. To add queries to your design, do one of the following:
 - To add queries from the [QUERY_REQUESTS](#) system table, click **Query Repository**, select the desired queries and click **Save**. All valid queries that you selected appear in the **Queries** window.
 - To add queries from a file, select **Choose File**. All valid queries in the file that you select are added to the design and appear in the **Queries** window.

Database Designer checks the validity of the queries when you add the queries to the design and again when you build the design. If it finds invalid queries, it ignores them.

If you have a large number of queries, it may take time to load them. Make sure that all the queries you want Database Designer to consider when creating the design are listed in the **Queries** window.

8. Once you have specified all the parameters for your design, you should build the design. To do this, select your design and click **Build Design**.
9. Select **Analyze Statistics** if you want Database Designer to analyze the statistics before building the design.

For more information see [Statistics Analysis](#).

10. If you do not need to review the design before deploying it, select **Deploy Immediately**. Otherwise, leave that option unselected.
11. Click **Start**. On the left-hand pane, the status of your design displays as **Building** until it is complete.
12. To follow the progress of a build, click **Event History**. Status messages appear in this window and you can see the current phase of the build operation. The information in the Event History tab contains data from the [OUTPUT_EVENT_HISTORY](#) system table.
13. When the build completes, the left-hand pane displays **Built**. To view the deployment script, select your design and click **Output**.
14. After you deploy the design using Management Console, the deployment script is deleted. To keep a permanent copy of the deployment script, copy and paste the SQL commands from the **Output** window to a file.
15. Once you have reviewed your design and are ready to deploy it, select the design and click **Deploy Design**.
16. To follow the progress of the deployment, click **Event History**. Status messages appear in this

window and you can see the current phase of the deployment operation.

17. When the deployment completes, the left-hand pane displays **Deployment Completed**. To view the deployment script, select your design and click **Output**.

Your database is now optimized according to the parameters you set.

Using Administration Tools to Create a Design

To use the Administration Tools interface to create an optimized design for your database, you must be a DBADMIN user. Follow these steps:

1. Log in as the dbadmin user and start Administration Tools.
2. From the main menu, start the database for which you want to create a design. The database must be running before you can create a design for it.
3. On the main menu, select **Configuration Menu** and click **OK**.
4. On the Configuration Menu, select **Run Database Designer** and click **OK**.
5. On the **Select a database to design** window, enter the name of the database for which you are creating a design and click **OK**.
6. On the **Enter the directory for Database Designer output** window, enter the full path to the directory to contain the design script, deployment script, backup script, and log files, and click **OK**.

For information about the scripts, see [Building a Design](#).

7. On the **Database Designer** window, enter a name for the design and click **OK**.

For more information about design names, see [Design Name](#).

8. On the **Design Type** window, choose which type of design to create and click **OK**.

For a description of the design types, see [Design Types](#)

9. The **Select schema(s) to add to query search path** window lists all the schemas in the database that you selected. Select the schemas that contain representative data that you want Database Designer to consider when creating the design and click **OK**.

For more information about choosing schema and tables to submit to Database Designer, see [Design Tables with Sample Data](#).

10. On the **Optimization Objectives** window, select the objective you want for the database optimization:

- **Optimize with Queries**

For more information, see [Design Queries](#).

- **Update statistics**

For more information see [Statistics Analysis](#).

- **Deploy design**

For more information, see [Deploying a Design](#).

For details about these objectives, see [Optimization Objectives](#).

11. The final window summarizes the choices you have made and offers you two choices:
 - **Proceed** with building the design, and deploying it if you specified to deploy it immediately. If you did not specify to deploy, you can review the design and deployment scripts and deploy them manually, as described in [Deploying Designs Manually](#).
 - **Cancel** the design and go back to change some of the parameters as needed.
12. Creating a design can take a long time. To cancel a running design from the Administration Tools window, enter **Ctrl+C**.

To create a design for the VMart example database, see [Using Database Designer to Create a Comprehensive Design](#) in the Getting Started Guide.

Creating Custom Designs

HP strongly recommends that you use the physical schema design produced by **Database Designer**, which provides **K-safety**, excellent query performance, and efficient use of storage space. If you find that any of your queries are not running as efficiently as you would like, you can use the Database Designer incremental design process to optimize the database design for the query.

If the projections created by Database Designer still do not meet your needs, you can write custom projections, from scratch or based on projection designs created by Database Designer.

If you are unfamiliar with writing custom projections, start by modifying an existing design generated by Database Designer.

The Design Process

To customize an existing design or create a new one, take these steps:

1. Plan the design or design modification.

As with most successful projects, a good design requires some up-front planning. See [Planning Your Design](#).

2. Create or modify projections.

For an overview of the CREATE PROJECTION statement and guidelines for creating common projections, see [Design Fundamentals](#). The [CREATE PROJECTION](#) section in the SQL Reference Manual also provides more detail.

3. Deploy the projections to a test environment. See [Writing and Deploying Custom Projections](#).
4. Test the projections.
5. Modify the projections as necessary.
6. Once you have finalized the design, deploy the projections to the production environment.

Planning Your Design

The syntax for creating a design is easy for anyone who is familiar with SQL. As with any successful project, however, a successful design requires some initial planning. Before you create your first design:

- Become familiar with standard design requirements and plan your design to include them. See [Design Requirements](#).
- Determine how many projections you need to include in the design. See [Determining the Number of Projections to Use](#).
- Determine the type of compression and encoding to use for columns. See [Data Encoding and Compression](#).
- Determine whether or not you want the database to be K-safe. HP Vertica recommends that all production databases have a minimum K-safety of one (K=1). Valid K-safety values are 0, 1, and 2. See [Designing for K-Safety](#).

Design Requirements

A physical schema design is a script that contains CREATE PROJECTION statements. These statements determine which columns are included in projections and how they are optimized.

If you use Database Designer as a starting point, it automatically creates designs that meet all fundamental design requirements. If you intend to create or modify designs manually, be aware that all designs must meet the following requirements:

- Every design must create at least one superprojection for every table in the database that is used by the client application. These projections provide complete coverage that enables users to perform ad-hoc queries as needed. They can contain joins and they are usually configured to maximize performance through sort order, compression, and encoding.
- Query-specific projections are optional. If you are satisfied with the performance provided through superprojections, you do not need to create additional projections. However, you can maximize performance by tuning for specific query work loads.
- HP recommends that all production databases have a minimum K-safety of one (K=1) to support high availability and recovery. (K-safety can be set to 0, 1, or 2.) See [High Availability Through Projections](#) in the Concepts Guide and [Designing for K-Safety](#).

Determining the Number of Projections to Use

In many cases, a design that consists of a set of superprojections (and their buddies) provides satisfactory performance through compression and encoding. This is especially true if the sort orders for the projections have been used to maximize performance for one or more query predicates (WHERE clauses).

However, you might want to add additional query-specific projections to increase the performance of queries that run slowly, are used frequently, or are run as part of business-critical reporting. The number of additional projections (and their buddies) that you create should be determined by:

- Your organization's needs
- The amount of disk space you have available on each node in the cluster
- The amount of time available for loading data into the database

As the number of projections that are tuned for specific queries increases, the performance of these queries improves. However, the amount of disk space used and the amount of time required to load data increases as well. Therefore, you should create and test designs to determine the optimum number of projections for your database configuration. On average, organizations that choose to implement query-specific projections achieve optimal performance through the addition of a few query-specific projections.

Designing for K-Safety

Before creating custom physical schema designs, determine whether you want the database to be **K-safe** and adhere to the appropriate design requirements for K-safe databases or databases with no K-safety. HP requires that all production databases have a minimum K-safety of one (K=1). Valid K-safety values for production databases are 1 and 2. Non-production databases do not have to be K-safe and can be set to 0. You can start by creating a physical schema design with no K-safety, and then modify it to be K-safe at a later point in time. See [High Availability and Recovery](#) and [High Availability Through Projections](#) in the Concepts Guide for an explanation of how HP Vertica implements high availability and recovery through replication and segmentation.

Requirements for a K-Safe Physical Schema Design

Database Designer automatically generates designs with a K-safety of 1 for clusters that contain at least three nodes. (If your cluster has one or two nodes, it generates designs with a K-safety of 0. You can modify a design created for a three-node (or greater) cluster, and the K-safe requirements are already set.

If you create custom projections, your physical schema design must meet the following requirements to be able to successfully recover the database in the event of a failure:

- Segmented projections must be **segmented** across all nodes. Refer to [Designing for Segmentation](#) and [Designing Segmented Projections for K-Safety](#).
- Replicated projections must be **replicated** on all nodes. See [Designing Replicated Projections for K-Safety](#).
- Segmented projections must have **K buddy** projections (projections that have identical columns and segmentation criteria, except that corresponding segments are placed on different nodes).

You can use the [MARK_DESIGN_KSAFE](#) function to find out whether your schema design meets requirements for K-safety.

Requirements for a Physical Schema Design with No K-Safety

If you use Database Designer to generate a comprehensive design that you can modify and you do not want the design to be K-safe, set K-safety level to 0 (zero).

If you want to start from scratch, do the following to establish minimal projection requirements for a functioning database with no K-safety (K=0):

1. Define at least one **superprojection** for each table in the **logical schema**.
2. Replicate (define an exact copy of) each dimension table superprojection on each **node**.

Designing for Segmentation

You segment projections using hash segmentation. Hash segmentation allows you to segment a projection based on a built-in hash function that provides even distribution of data across multiple nodes, resulting in optimal query execution. In a projection, the data to be hashed consists of one or more column values, each having a large number of unique values and an acceptable amount of skew in the value distribution. Primary key columns that meet the criteria could be an excellent choice for hash segmentation.

Note: For detailed information about using hash segmentation in a projection, see [CREATE PROJECTION](#) in the SQL Reference Manual.

When segmenting projections, determine which columns to use to segment the projection. Choose one or more columns that have a large number of unique data values and acceptable skew in their data distribution. Primary key columns are an excellent choice for hash segmentation. The columns must be unique across all the tables being used in a query.

Design Fundamentals

Although you can write custom projections from scratch, HP Vertica recommends that you use Database Designer to create a design to use as a starting point. This ensures that you have projections that meet basic requirements.

Writing and Deploying Custom Projections

Before you write custom projections, be sure to review the topics in [Planning Your Design](#) carefully. Failure to follow these considerations can result in non-functional projections.

To manually modify or create a projection:

1. Write a script to create the projection, using the [CREATE PROJECTION](#) statement.
2. Use the [\i meta-command](#) in vsql to run the script.

Note: You must have a database loaded with a logical schema.

3. For a K-safe database, use the function `SELECT get_projections('table_name')` to verify that the projections were properly created. Good projections are noted as being "safe." This means that the projection has enough buddies to be **K-safe**.
4. If you added the new projection to a database that already has projections that contain data, you need to update the newly created projection to work with the existing projections. By default, the new projection is **out-of-date** (not available for query processing) until you refresh it.
5. Use the [MAKE_AHM_NOW](#) function to set the Ancient History Mark (AHM) to the greatest allowable epoch (now).
6. Use the [DROP_PROJECTION](#) function to drop any previous projections that are no longer needed.

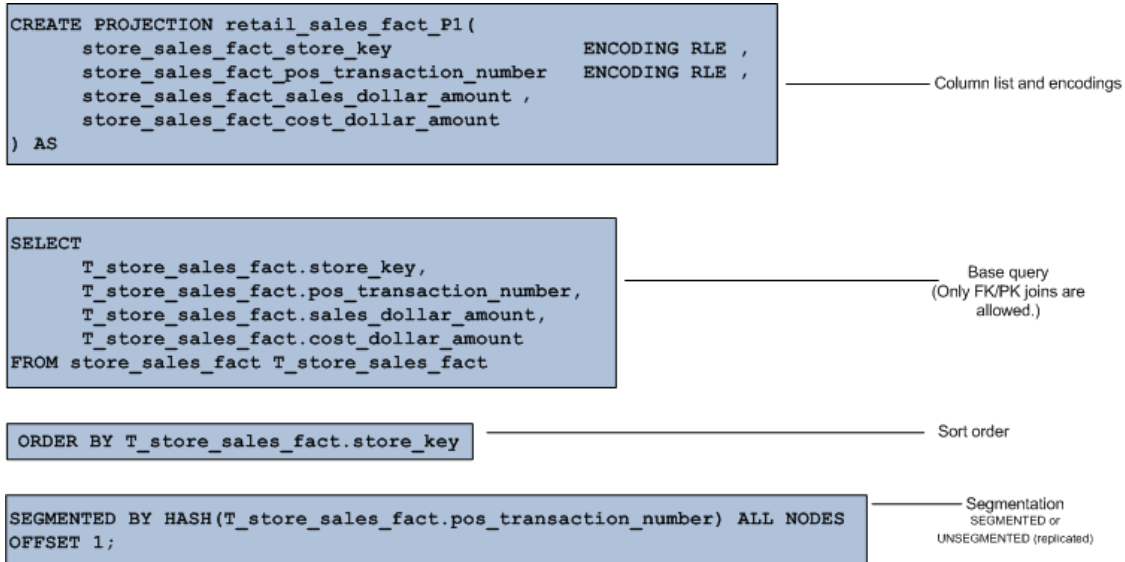
These projections can waste disk space and reduce load speed if they remain in the database.

7. Run the [ANALYZE_STATISTICS](#) function on all projections in the database. This function collects and aggregates data samples and storage information from all nodes on which a projection is stored, and then writes statistics into the catalog. For example:

```
=>SELECT ANALYZE_STATISTICS ('');
```

Anatomy of a Projection

The [CREATE PROJECTION](#) statement defines the individual elements of a projection, as the following graphic shows.



The previous example contains the following significant elements:

Column List and Encoding

Lists every column in the projection and defines the encoding for each column. Unlike traditional database architectures, HP Vertica operates on encoded data representations. Therefore, HP recommends that you use data encoding because it results in less disk I/O.

Base Query

Identifies all the columns to incorporate in the projection through column name and table name references. The base query for large table projections can contain PK/FK joins to smaller tables.

Sort Order

The sort order optimizes for a specific query or commonalities in a class of queries based on the query predicate. The best sort orders are determined by the WHERE clauses. For example, if a projection's sort order is (x, y), and the query's WHERE clause specifies (x=1 AND y=2), all of the needed data is found together in the sort order, so the query runs almost instantaneously.

You can also optimize a query by matching the projection's sort order to the query's GROUP BY clause. If you do not specify a sort order, HP Vertica uses the order in which columns are specified in the column definition as the projection's sort order.

The ORDER BY clause specifies a projection's sort order, which localizes logically grouped values so that a disk read can pick up many results at once. For maximum performance, do not sort projections on LONG VARBINARY and LONG VARCHAR columns.

Segmentation

The segmentation clause determines whether a projection is segmented across nodes within the database. Segmentation distributes contiguous pieces of projections, called *segments*, for large and medium tables across database nodes. Segmentation maximizes database performance by distributing the load. Use SEGMENTED BY HASH to segment large table projections.

For small tables, use the UNSEGMENTED keyword to direct HP Vertica to replicate these tables, rather than segment them. Replication creates and stores identical copies of projections for small tables across all nodes in the cluster. Replication ensures high availability and recovery.

For maximum performance, do not segment projections on LONG VARBINARY and LONG VARCHAR columns.

Designing Superprojections

Superprojections have the following requirements:

- They must contain every column within the table.
- For a K-safe design, superprojections must either be replicated on all nodes within the database cluster (for dimension tables) or paired with buddies and segmented across all nodes (for very large tables and medium large tables). See [Physical Schema](#) and [High Availability Through Projections](#) in the Concepts Guide for an overview of projections and how they are stored. See [Designing for K-Safety](#) for design specifics.

To provide maximum usability, superprojections need to minimize storage requirements while maximizing query performance. To achieve this, the sort order for columns in superprojections is based on storage requirements and commonly used queries.

Minimizing Storage Requirements

Minimizing storage not only saves on physical resources, it increases performance by requiring the database to perform less disk I/O. To minimize storage space for a projection:

- Analyze the type of data stored in each projection column and choose the most effective encoding method. See the [CREATE PROJECTION](#) statement and [encoding-type](#) in the SQL Reference Manual.

The HP Vertica optimizer gives Run-Length Encoding (RLE) preference, so be sure to use it whenever appropriate. Run Length Encoding (RLE) replaces sequences (runs) of identical values with a single pair that contains the value and number of occurrences. Therefore, use it only when the run length is large, such as when sorting low-cardinality columns.

- Prioritize low-cardinality columns in the column sort order. This minimizes the number of rows that HP Vertica stores and accesses to retrieve query results.

For more information about minimizing storage requirements, see [Choosing Sort Order: Best Practices](#).

Maximizing Query Performance

In addition to minimizing storage requirements, the column sort order facilitates the most commonly used queries for the table. This means that the column sort order prioritizes the lowest cardinality columns that are actually used in queries. For examples that take into account both storage and query requirements, see [Choosing Sort Order: Best Practices](#).

Note: For maximum performance, do not sort projections on LONG VARBINARY and LONG VARCHAR columns.

Projections within a buddy set can all have different sort orders. This enables you to maximize query performance for groups of queries with common WHERE clauses, but different sort orders. If, for example, you have a three-node cluster, your buddy set contains three interrelated projections, each having its own sort order.

In a database with a K-safety of 1 or 2, buddy projections are used for data recovery. If a node fails, it queries the other nodes to recover data through buddy projections. (See How Result Sets are Stored in the Concepts Guide.) If a projection's buddies use different sort orders, it takes longer to recover the projection because the data has to be resorted during recovery to match the sort order of the projection. Therefore, consider using identical sort orders for tables that are rarely queried or that are repeatedly accessed by the same query, and use multiple sort orders for tables that are accessed by queries with common WHERE clauses, but different sort orders.

If you have queries that access multiple tables or you want to maintain the same sort order for projections within buddy sets, create query-specific projections. Designs that contain projections for specific queries are called *optimized designs*.

Designing Replicated Projections for K-Safety

If you are creating or modifying a design for a K-safe database, make sure that projections for dimension tables are replicated on each node in the database.

You can accomplish this using a single [CREATE PROJECTION](#) command for each dimension table. The UNSEGMENTED ALL NODES syntax within the segmentation clause automatically creates an unsegmented projection on each node in the database.

When you run your design script, HP Vertica generates a list of nodes based on the number of nodes in the database and replicates the projection accordingly. Replicated projections have the name:

projection-name_node-name

If, for example, the nodes are named NODE01, NODE02, and NODE03, the projections are named ABC_NODE01, ABC_NODE02, and ABC_NODE03.

Note: This naming convention can affect functions that provide information about projections, for example, [GET_PROJECTIONS](#) or [GET_PROJECTION_STATUS](#), where you must provide the name ABC_NODE01 instead of just ABC. To view a list of the nodes in a database, use the [View Database](#) command in the **Administration Tools**.

The following script uses the UNSEGMENTED ALL NODES syntax to create one unsegmented superprojection for the store_dimension table on each node.

```
CREATE PROJECTION store_dimension( C0_store_dimension_floor_plan_type ENCODING RLE ,
  C1_store_dimension_photo_processing_type ENCODING RLE ,
  C2_store_dimension_store_key ,
  C3_store_dimension_store_name ,
  C4_store_dimension_store_number ,
  C5_store_dimension_store_street_address ,
  C6_store_dimension_store_city ,
  C7_store_dimension_store_state ,
  C8_store_dimension_store_region ,
  C9_store_dimension_financial_service_type ,
  C10_store_dimension_selling_square_footage ,
  C11_store_dimension_total_square_footage ,
  C12_store_dimension_first_open_date ,
  C13_store_dimension_last_remodel_date )
AS SELECT T_store_dimension.floor_plan_type,
  T_store_dimension.photo_processing_type,
  T_store_dimension.store_key,
  T_store_dimension.store_name,
  T_store_dimension.store_number,
  T_store_dimension.store_street_address,
  T_store_dimension.store_city,
  T_store_dimension.store_state,
  T_store_dimension.store_region,
  T_store_dimension.financial_service_type,
  T_store_dimension.selling_square_footage,
  T_store_dimension.total_square_footage,
  T_store_dimension.first_open_date,
  T_store_dimension.last_remodel_date
FROM store_dimension T_store_dimension
ORDER BY T_store_dimension.floor_plan_type, T_store_dimension.photo_processing_type
UNSEGMENTED ALL NODES;
```

Note: Large dimension tables can be segmented. A dimension table is considered to be large when it is approximately the same size as a fact table.

Designing Segmented Projections for K-Safety

If you are creating or modifying a design for a K-safe database, you need to create K-safe projections for fact tables and large dimension tables. (A dimension table is considered to be large if it is similar in size to a fact table.) To accomplish this, you must:

- Create a segmented projection for each fact and large dimension table.
- Create segmented buddy projections for each of these projections. The total number of projections in a buddy set must be two for a K=1 database or three for a K=2 database.

For an overview of segmented projections and their buddies, see [Projection Segmentation](#) in the Concepts Guide. For information about designing for K-safety, see [Designing for K-Safety](#) and [Designing for Segmentation](#).

Segmenting Projections

To segment a projection, use the segmentation clause to specify the:

- Segmentation method to use.
- Column to use to segment the projection.
- Nodes on which to segment the projection. You can segment projections across all the nodes, or just the number of nodes necessary to maintain K-safety, either three for a K=1 database or five for a K=2 database.

See the [CREATE PROJECTION](#) statement in the SQL Reference Manual.

The following segmentation clause uses hash segmentation to segment the projection across all nodes based on the `T_retail_sales_fact.pos_transaction_number` column:

```
CREATE PROJECTION retail_sales_fact_P1... SEGMENTED BY HASH(T_retail_sales_fact.pos_trans
action_number) ALL NODES;
```

Creating Buddy Projections

To create a buddy projection, copy the original projection and modify it as follows:

- Rename it to something similar to the name of the original projection. For example, a projection named `retail_sales_fact_P1` could have buddies named `retail_sales_fact_P1_B1` and `retail_sales_fact_P1_B2`.
- Modify the sort order as needed.
- Create an offset to store the segments for the buddy on different nodes. For example, the first buddy in a projection set would have an offset of one (`OFFSET1;`) the second buddy in a projection set would have an offset of two (`OFFSET2;`), and so on.

To create a buddy for the projection created in the previous example:

```
CREATE PROJECTION retail_sales_fact_P1_B1... SEGMENTED BY HASH(T_retail_sales_fact.pos_tr
ansaction_number) ALL NODES OFFSET 1;
```

Projection Design for Merge Operations

The HP Vertica query optimizer automatically picks the best projections to use for queries, but you can help improve the performance of `MERGE` operations by ensuring projections are designed for optimal use.

Good projection design lets HP Vertica choose the faster merge join between the **target** and **source** tables without having to perform additional sort and data transfer operations.

HP recommends that you first use **Database Designer** to generate a comprehensive design and then customize projections, as needed. Be sure to first review the topics in [Planning Your Design](#). Failure to follow those considerations could result in non-functioning projections.

In the following `MERGE` statement, HP Vertica inserts and/or updates records from the source table's column `b` into the target table's column `a`:

```
=> MERGE INTO target t USING source s ON t.a = s.b WHEN ....
```

HP Vertica can use a local merge join if tables `target` and `source` use one of the following projection designs, where their inputs are pre-sorted through the `CREATE PROJECTION ORDER BY` clause:

- **Replicated** projections that are sorted on:
 - Column `a` for target
 - Column `b` for source
- **Segmented** projections that are [identically segmented](#) on:
 - Column `a` for target
 - Column `b` for source
 - Corresponding segmented columns

Tip: For best merge performance, the source table should be smaller than the target table.

See Also

- [Optimized Versus Non-Optimized MERGE](#)
- [Best Practices for Optimizing MERGE Statements](#)

Maximizing Projection Performance

This section explains how to design your projections in order to optimize their performance.

Choosing Sort Order: Best Practices

When choosing sort orders for your projections, HP Vertica has several recommendations that can help you achieve maximum query performance, as illustrated in the following examples.

Combine RLE and Sort Order

When dealing with predicates on low-cardinality columns, use a combination of RLE and sorting to minimize storage requirements and maximize query performance.

Suppose you have a `students` table contain the following values and encoding types:

| Column | # of Distinct Values | Encoded With |
|------------------------|--|--------------|
| <code>gender</code> | 2 (M or F) | RLE |
| <code>pass_fail</code> | 2 (P or F) | RLE |
| <code>class</code> | 4 (freshman, sophomore, junior, or senior) | RLE |
| <code>name</code> | 10000 (too many to list) | Auto |

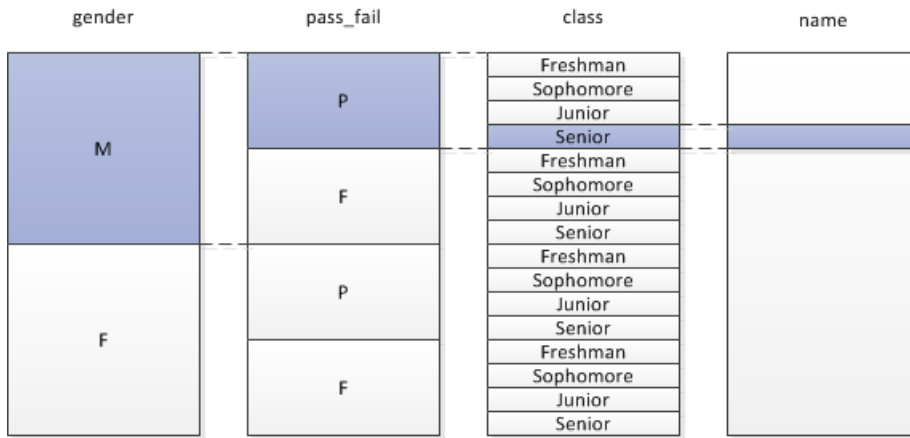
You might have queries similar to this one:

```
SELECT name FROM students WHERE gender = 'M' AND pass_fail = 'P' AND class = 'senior';
```

The fastest way to access the data is to work through the low-cardinality columns with the smallest number of distinct values before the high-cardinality columns. The following sort order minimizes storage and maximizes query performance for queries that have equality restrictions on `gender`, `class`, `pass_fail`, and `name`. Specify the `ORDER BY` clause of the projection as follows:

```
ORDER BY students.gender, students.pass_fail, students.class, students.name
```

In this example, the `gender` column is represented by two RLE entries, the `pass_fail` column is represented by four entries, and the `class` column is represented by 16 entries, regardless of the cardinality of the `students` table. HP Vertica efficiently finds the set of rows that satisfy all the predicates, resulting in a huge reduction of search effort for RLE encoded columns that occur early in the sort order. Consequently, if you use low-cardinality columns in local predicates, as in the previous example, put those columns early in the projection sort order, in increasing order of distinct cardinality (that is, in increasing order of the number of distinct values in each column).



If you sort this table with `student.class` first, you improve the performance of queries that restrict only on the `student.class` column, and you improve the compression of the `student.class` column (which contains the largest number of distinct values), but the other columns do not compress as well. Determining which projection is better depends on the specific queries in your workload, and their relative importance.

Storage savings with compression decrease as the cardinality of the column increases; however, storage savings with compression increase as the number of bytes required to store values in that column increases.

Maximize the Advantages of RLE

To maximize the advantages of RLE encoding, use it only when the average run length of a column is greater than 10 when sorted. For example, suppose you have a table with the following columns, sorted in order of cardinality from low to high:

```
address.country, address.region, address.state, address.city, address.zipcode
```

The `zipcode` column might not have 10 sorted entries in a row with the same zip code, so there is probably no advantage to run-length encoding that column, and it could make compression worse. But there are likely to be more than 10 countries in a sorted run length, so applying RLE to the `country` column can improve performance.

Put Lower Cardinality Column First for Functional Dependencies

In general, put columns that you use for local predicates (as in the previous example) earlier in the join order to make predicate evaluation more efficient. In addition, if a lower cardinality column is uniquely determined by a higher cardinality column (like `city_id` uniquely determining a `state_id`), it is always better to put the lower cardinality, functionally determined column earlier in the sort order than the higher cardinality column.

For example, in the following sort order, the `Area_Code` column is sorted before the `Number` column in the `customer_info` table:

```
ORDER BY = customer_info.Area_Code, customer_info.Number, customer_info.Address
```

In the query, put the Area_Code column first, so that only the values in the Number column that start with 978 are scanned.

```
SELECT Address FROM customer_info WHERE Area_Code='978' AND Number='9780123457';
```

| Area_code | Number | Address |
|-----------|--------------|-----------------|
| 508 | 508 012 3456 | 1 Elm Street |
| | 508 012 3457 | 1 School Street |
| | ... | ... |
| 617 | 617 012 3456 | 1 Maple Avenue |
| | 617 012 3457 | 1 Post Street |
| | ... | ... |
| 978 | 978 012 3456 | 1 Olive Road |
| | 978 012 3457 | 1 Main Street |
| | ... | ... |

Sort for Merge Joins

When processing a join, the HP Vertica optimizer chooses from two algorithms:

- **Merge join**—If both inputs are pre-sorted on the join column, the optimizer chooses a merge join, which is faster and uses less memory.
- **Hash join**—Using the hash join algorithm, HP Vertica uses the smaller (inner) joined table to build an in-memory hash table on the join column. A hash join has no sort requirement, but it consumes more memory because Vertica builds a hash table with the values in the inner table. The optimizer chooses a hash join when projections are not sorted on the join columns.

If both inputs are pre-sorted, merge joins do not have to do any pre-processing, making the join perform faster. HP Vertica uses the term sort-merge join to refer to the case when at least one of the inputs must be sorted prior to the merge join. HP Vertica sorts the inner input side but only if the outer input side is already sorted on the join columns.

To give the Vertica query optimizer the option to use an efficient merge join for a particular join, create projections on both sides of the join that put the join column first in their respective projections. This is primarily important to do if both tables are so large that neither table fits into memory. If all tables that a table will be joined to can be expected to fit into memory simultaneously, the benefits of merge join over hash join are sufficiently small that it probably isn't worth creating a projection for any one join column.

Sort on Columns in Important Queries

If you have an important query, one that you run on a regular basis, you can save time by putting the columns specified in the WHERE clause or the GROUP BY clause of that query early in the sort order.

If that query uses a high-cardinality column such as Social Security number, you may sacrifice storage by placing this column early in the sort order of a projection, but your most important query will be optimized.

Sort Columns of Equal Cardinality By Size

If you have two columns of equal cardinality, put the column that is larger first in the sort order. For example, a CHAR(20) column takes up 20 bytes, but an INTEGER column takes up 8 bytes. By putting the CHAR(20) column ahead of the INTEGER column, your projection compresses better.

Sort Foreign Key Columns First, From Low to High Distinct Cardinality

Suppose you have a fact table where the first four columns in the sort order make up a foreign key to another table. For best compression, choose a sort order for the fact table such that the foreign keys appear first, and in increasing order of distinct cardinality. Other factors also apply to the design of projections for fact tables, such as partitioning by a time dimension, if any.

In the following example, the table `inventory` stores inventory data, and `product_key` and `warehouse_key` are foreign keys to the `product_dimension` and `warehouse_dimension` tables:

```
CREATE TABLE inventory (
    date_key INTEGER NOT NULL,
    product_key INTEGER NOT NULL,
    warehouse_key INTEGER NOT NULL,
    ...
);
ALTER TABLE inventory
    ADD CONSTRAINT fk_inventory_warehouse FOREIGN KEY(warehouse_key)
    REFERENCES warehouse_dimension(warehouse_key);
ALTER TABLE inventory
    ADD CONSTRAINT fk_inventory_product FOREIGN KEY(product_key)
    REFERENCES product_dimension(product_key);
```

The `inventory` table should be sorted by `warehouse_key` and then `product`, since the cardinality of the `warehouse_key` column is probably lower than the cardinality of the `product_key`.

Prioritizing Column Access Speed

If you measure and set the performance of storage locations within your cluster, HP Vertica uses this information to determine where to store columns based on their rank. For more information, see [Setting Storage Performance](#).

How Columns are Ranked

HP Vertica stores columns included in the projection sort order on the fastest storage locations. Columns not included in the projection sort order are stored on slower disks. Columns for each projection are ranked as follows:

- Columns in the sort order are given the highest priority (numbers > 1000).
- The last column in the sort order is given the rank number 1001.
- The next-to-last column in the sort order is given the rank number 1002, and so on until the first column in the sort order is given 1000 + # of sort columns.
- The remaining columns are given numbers from 1000–1, starting with 1000 and decrementing by one per column.

HP Vertica then stores columns on disk from the highest ranking to the lowest ranking, with the highest ranking columns placed on the fastest disks, and the lowest ranking columns placed on the slowest disks.

Overriding Default Column Ranking

You can modify which columns are stored on fast disks by manually overriding the default ranks for these columns. To accomplish this, set the `ACCESSRANK` keyword in the column list. Make sure to use an integer that is not already being used for another column. For example, if you want to give a column the fastest access rank, use a number that is significantly higher than 1000 + the number of sort columns. This allows you to enter more columns over time without bumping into the access rank you set.

The following example sets the access rank for the `C1_retail_sales_fact_store_key` column to 1500.

```
CREATE PROJECTION retail_sales_fact_P1 ( C1_retail_sales_fact_store_key ENCODING RLE ACC
ESSRANK 1500,
C2_retail_sales_fact_pos_transaction_number ,
C3_retail_sales_fact_sales_dollar_amount ,
C4_retail_sales_fact_cost_dollar_amount )
```

Projection Examples

This section provides examples that show you how to create projections.

New K-Safe=2 Database

In this example, projections are created for a new five-node database with a K-safety of 2. To simplify the example, this database contains only two tables: `retail_sale_fact` and `store_dimension`. Creating projections for this database consists of creating the following segmented and unsegmented (replicated) superprojections:

- **Segmented projections**

To support K-safety=2, the database requires three segmented projections (one projection and two buddy projections) for each fact table. In this case, it requires three segmented projections for the `retail_sale_fact` table:

| Projection | Description |
|------------|---|
| P1 | The primary projection for the <code>retail_sale_fact</code> table. |
| P1_B1 | The first buddy projection for P1. This buddy is required to provide K-safety=1. |
| P1_B2 | The second buddy projection for P1. This buddy is required to provide K-safety=2. |

- **Unsegmented Projections**

To support the database, one unsegmented superprojection must be created for each dimension table on each node. In this case, one unsegmented superprojection must be created on each node for the `store_dimension` table:

| Node | Unsegmented Projection |
|--------|-------------------------------------|
| Node01 | <code>store_dimension_Node01</code> |
| Node02 | <code>store_dimension_Node02</code> |
| Node03 | <code>store_dimension_Node03</code> |
| Node04 | <code>store_dimension_Node04</code> |
| Node05 | <code>store_dimension_Node05</code> |

Creating Segmented Projections Example

The following SQL script creates the P1 projection and its buddies, P1_B1 and P1_B2, for the `retail_sales_fact` table. The following syntax is significant:

- **CREATE PROJECTION** creates the named projection (retail_sales_fact_P1, retail_sales_fact_P1_B1, or retail_sales_fact_P1_B2).
- **ALL NODES** automatically segments the projections across all five nodes in the cluster without specifically referring to each node.
- **HASH** evenly distributes the data across these nodes.
- **OFFSET** ensures that the same data is not stored on the same nodes for each of the buddies. The first buddy uses **OFFSET 1** to shift the storage locations by 1 and the second buddy uses **OFFSET 2** to shift the storage locations by 1. This is critical to ensure K-safety.

```

CREATE PROJECTION retail_sales_fact_P1 (
  C1_retail_sales_fact_store_key ENCODING RLE ,
  C2_retail_sales_fact_pos_transaction_number ,
  C3_retail_sales_fact_sales_dollar_amount ,
  C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
  T_retail_sales_fact.pos_transaction_number,
  T_retail_sales_fact.sales_dollar_amount,
  T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES;
-----
-- Projection #           : 6
-- Projection storage (KBytes) : 4.8e+06
-- Note: This is a super projection for table: retail_sales_fact
CREATE PROJECTION retail_sales_fact_P1_B1 (
  C1_retail_sales_fact_store_key ENCODING RLE ,
  C2_retail_sales_fact_pos_transaction_number ,
  C3_retail_sales_fact_sales_dollar_amount ,
  C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
  T_retail_sales_fact.pos_transaction_number,
  T_retail_sales_fact.sales_dollar_amount,
  T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODESOFFSET 1;
-----
-- Projection #           : 6
-- Projection storage (KBytes) : 4.8e+06
-- Note: This is a super projection for table: retail_sales_fact
CREATE PROJECTION retail_sales_fact_P1_B2 (
  C1_retail_sales_fact_store_key ENCODING RLE ,
  C2_retail_sales_fact_pos_transaction_number ,
  C3_retail_sales_fact_sales_dollar_amount ,
  C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
  T_retail_sales_fact.pos_transaction_number,
  T_retail_sales_fact.sales_dollar_amount,
  T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODESOFFSET 2;

```

Creating Unsegmented Projections Example

The following script uses the UNSEGMENTED ALL NODES syntax to create one unsegmented superprojection for the store_dimension table on each node.

```
CREATE PROJECTION store_dimension ( C0_store_dimension_floor_plan_type ENCODING RLE ,
  C1_store_dimension_photo_processing_type ENCODING RLE ,
  C2_store_dimension_store_key ,
  C3_store_dimension_store_name ,
  C4_store_dimension_store_number ,
  C5_store_dimension_store_street_address ,
  C6_store_dimension_store_city ,
  C7_store_dimension_store_state ,
  C8_store_dimension_store_region ,
  C9_store_dimension_financial_service_type ,
  C10_store_dimension_selling_square_footage ,
  C11_store_dimension_total_square_footage ,
  C12_store_dimension_first_open_date ,
  C13_store_dimension_last_remodel_date )
AS SELECT T_store_dimension.floor_plan_type,
  T_store_dimension.photo_processing_type,
  T_store_dimension.store_key,
  T_store_dimension.store_name,
  T_store_dimension.store_number,
  T_store_dimension.store_street_address,
  T_store_dimension.store_city,
  T_store_dimension.store_state,
  T_store_dimension.store_region,
  T_store_dimension.financial_service_type,
  T_store_dimension.selling_square_footage,
  T_store_dimension.total_square_footage,
  T_store_dimension.first_open_date,
  T_store_dimension.last_remodel_date
FROM store_dimension T_store_dimension
ORDER BY T_store_dimension.floor_plan_type, T_store_dimension.photo_processing_type
UNSEGMENTED ALL NODES;
```

Adding Node to a Database

In this example, a fourth node (Node04) is being added to a three-node database cluster. The database contains two tables: retail_sale_fact and store_dimension. It also contains the following segmented and unsegmented (replicated) **superprojections**:

- **Segmented projections**

P1 and its buddy, B1, are projections for the retail_sale_fact table. They were created using the ALL NODES syntax, so HP Vertica automatically segments the projections across all three nodes.

- **Unsegmented Projections**

Currently three unsegmented superprojections exist for the `store_dimension` table, one for each node, as follows:

| Node | Unsegmented Projection |
|--------|------------------------|
| Node01 | store_dimension_Node01 |
| Node02 | store_dimension_Node02 |
| Node03 | store_dimension_Node03 |

To support an additional node, replacement projections need to be created for the segmented projections, P1 and B1. The new projections could be called P2 and B2, respectively. Additionally, an unsegmented superprojection (`store_dimension_Node04`) needs to be created for the dimension table on the new node (Node04).

Creating Segmented Projections Example

The following SQL script creates the original P1 projection and its buddy, B1, for the `retail_sales_fact` table. Since the script uses the ALL NODES syntax, creating a new projection that includes the fourth node is as easy as copying the script and changing the names of the projection and its buddy to unique names (for example, P2 for the projection and P2_B2 for its buddy). The names that need to be changed are highlighted within the example.

```
CREATE PROJECTION retail_sales_fact_P1 ( C1_retail_sales_fact_store_key ENCODING RLE ,
    C2_retail_sales_fact_pos_transaction_number ,
    C3_retail_sales_fact_sales_dollar_amount ,
    C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
    T_retail_sales_fact.pos_transaction_number,
    T_retail_sales_fact.sales_dollar_amount,
    T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES;
-----
-- Projection #                : 6
-- Projection storage (KBytes) : 4.8e+06
-- Note: This is a super projection for table: retail_sales_fact
CREATE PROJECTION retail_sales_fact_P1_B1 (
    C1_retail_sales_fact_store_key ENCODING RLE ,
    C2_retail_sales_fact_pos_transaction_number ,
    C3_retail_sales_fact_sales_dollar_amount ,
    C4_retail_sales_fact_cost_dollar_amount )
AS SELECT T_retail_sales_fact.store_key,
    T_retail_sales_fact.pos_transaction_number,
    T_retail_sales_fact.sales_dollar_amount,
    T_retail_sales_fact.cost_dollar_amount
FROM retail_sales_fact T_retail_sales_fact
ORDER BY T_retail_sales_fact.store_key
```

```
SEGMENTED BY HASH(T_retail_sales_fact.pos_transaction_number) ALL NODES
OFFSET 1;
-----
```

Creating Unsegmented Projections Example

The following script used the ALL NODES syntax to create the original three unsegmented superprojections for the `store_dimension` table, one per node.

The following syntax is significant:

- CREATE PROJECTION creates a superprojection called `store_dimension`.
- ALL NODES automatically places a complete copy of the superprojection on each of the three original nodes.

```
CREATE PROJECTION store_dimension (
  C0_store_dimension_floor_plan_type ENCODING RLE ,
  C1_store_dimension_photo_processing_type ENCODING RLE ,
  C2_store_dimension_store_key ,
  C3_store_dimension_store_name ,
  C4_store_dimension_store_number ,
  C5_store_dimension_store_street_address ,
  C6_store_dimension_store_city ,
  C7_store_dimension_store_state ,
  C8_store_dimension_store_region ,
  C9_store_dimension_financial_service_type ,
  C10_store_dimension_selling_square_footage ,
  C11_store_dimension_total_square_footage ,
  C12_store_dimension_first_open_date ,
  C13_store_dimension_last_remodel_date )
AS SELECT T_store_dimension.floor_plan_type,
  T_store_dimension.photo_processing_type,
  T_store_dimension.store_key,
  T_store_dimension.store_name,
  T_store_dimension.store_number,
  T_store_dimension.store_street_address,
  T_store_dimension.store_city,
  T_store_dimension.store_state,
  T_store_dimension.store_region,
  T_store_dimension.financial_service_type,
  T_store_dimension.selling_square_footage,
  T_store_dimension.total_square_footage,
  T_store_dimension.first_open_date,
  T_store_dimension.last_remodel_date
FROM store_dimension T_store_dimension
ORDER BY T_store_dimension.floor_plan_type, T_store_dimension.photo_processing_type
UNSEGMENTED ALL NODES;
```

To create another copy of the superprojection on the fourth node (Node04), the best approach is to create a copy of that projection on Node04 only. This means avoiding the ALL NODES syntax. The following script shows how to create the fourth superprojection.

The following syntax is significant:

- CREATE PROJECTION creates a superprojection called store_dimension_Node04.
- UNSEGMENTED SITE Node04 creates the projection on just Node04.

```

CREATE PROJECTION store_dimension_Node04 ( C0_store_dimension_floor_plan_type ENCODING R
LE ,
  C1_store_dimension_photo_processing_type ENCODING RLE ,
  C2_store_dimension_store_key ,
  C3_store_dimension_store_name ,
  C4_store_dimension_store_number ,
  C5_store_dimension_store_street_address ,
  C6_store_dimension_store_city ,
  C7_store_dimension_store_state ,
  C8_store_dimension_store_region ,
  C9_store_dimension_financial_service_type ,
  C10_store_dimension_selling_square_footage ,
  C11_store_dimension_total_square_footage ,
  C12_store_dimension_first_open_date ,
  C13_store_dimension_last_remodel_date )
AS SELECT T_store_dimension.floor_plan_type,
  T_store_dimension.photo_processing_type,
  T_store_dimension.store_key,
  T_store_dimension.store_name,
  T_store_dimension.store_number,
  T_store_dimension.store_street_address,
  T_store_dimension.store_city,
  T_store_dimension.store_state,
  T_store_dimension.store_region,
  T_store_dimension.financial_service_type,
  T_store_dimension.selling_square_footage,
  T_store_dimension.total_square_footage,
  T_store_dimension.first_open_date,
  T_store_dimension.last_remodel_date
FROM store_dimension T_store_dimension
ORDER BY T_store_dimension.floor_plan_type, T_store_dimension.photo_processing_type
UNSEGMENTED NODE Node04;

```

Implementing Security

In HP Vertica, there are three primary security concerns:

- Client authentication prevents unauthorized access to the database
- Connection encryption prevents the interception of data, as well as authenticating the identity of the server and the client
- Client authorization (managing users and privileges) controls what users can access and change in the database

Client Authentication

To gain access to HP Vertica, a user or client application must supply the name of a valid user account. You can configure HP Vertica to require just a user name, but a more common practice is to require an additional means of authentication, such as a password. There are several ways to implement this added authentication:

- [Password Authentication](#) using passwords stored in the database.
- [Authentication using outside means](#), such as LDAP or Kerberos.

You can use different authentication methods based on:

- Connection type
- Client IP address range
- User name for the client that is attempting to access the server

See [Implementing Client Authentication](#).

Connection Encryption

To secure the connection between the client and the server, you can configure HP Vertica and database clients to use Secure Socket Layer (SSL) to communicate. HP Vertica uses SSL to:

- Authenticate the server so the client can confirm the server's identity. HP Vertica supports mutual authentication in which the server can also confirm the identity of the client. This authentication helps prevent **"man-in-the-middle" attacks**.
- Encrypt data sent between the client and database server to significantly reduce the likelihood that the data can be read if the connection between the client and server is compromised.
- Verify that data sent between the client and server has not been altered during transmission.

See [Implementing SSL](#).

Client Authorization

Database users should have access to just the database resources they need to perform their tasks. For example, some users need to query only specific sets of data. To prevent unauthorized access to additional data, you can limit their access to just the data that they need to perform their queries. Other users should be able to read the data but not be able to modify or insert new data. Still other users might need more permissive access, such as the right to create and modify schemas, tables, and views or even grant other users access to database resources.

A collection of SQL statements control authorization for the resources users can access. See [Managing Users and Privileges](#), in particular [About Database Privileges](#). You can also use roles to grant users access to a set of privileges, rather than directly grant the privileges for each user. See [About Database Roles](#).

Use the [GRANT Statements](#) to assign privileges to users and the [REVOKE Statements](#) to repeal privileges. See the SQL Reference Manual for details.

Implementing Client Authentication

When a client (the user who runs a client application or the client application itself) connects to the HP Vertica database server, it supplies the HP Vertica database user name to gain access. HP Vertica restricts which database users can connect through **client authentication**, a process where the database server establishes the identity of the requesting client and determines whether that client is authorized to connect to the HP Vertica server using the supplied credentials.

HP Vertica offers several client authentication methods, which you set up using the **Administration Tools** (see [How to Create Authentication Records](#)). Although you can configure HP Vertica to require just a user name for connections, you likely require more secure means of authentication, such as a password at a minimum.

Supported Client Authentication Types

HP Vertica supports the following types of authentication to prove a client's identity. For information about syntax and formatting rules, see [Authentication Record Format and Rules](#).

- Trust authentication—Authorizes any user that connects to the server using a valid user name.
- Reject authentication—Blocks the connection and prevents additional records from being evaluated for the requesting client.
- Kerberos authentication—Uses a secure, single-sign-on, trusted third-party, mutual authentication service to connect to HP Vertica using one of the following methods:
 - `krb5` uses the MIT Kerberos APIs (deprecated in HP Vertica 7.0. Use the `gss` method).
 - `gss` authentication uses the GSSAPI standard and provides better compatibility with non-MIT Kerberos implementations, such as for Java and Windows clients.
- Password authentication—Uses either `md5` or `password` methods, which are similar except for the manner in which the password is sent across the network:
 - `md5` method sends encrypted MD5-hashed passwords over the network, and the server provides the client with **salt**.
 - `password` method sends passwords over the network in clear text.
- LDAP authentication—Works like password authentication except the `ldap` method authenticates the client against a Lightweight Directory Access Protocol server.
- Ident-based authentication—Authenticates the client against the username in an Ident server.

The method HP Vertica uses to authenticate a particular client connection can be automatically selected on the basis of the connection type, client IP address, and user name. See [About External Authentication](#) for more information.

If You Want Communication Layer Authentication

Topics in this section describe authentication methods supported at the database server layer. For communication layer authentication between server and client, see [Implementing SSL](#).

Password Authentication

The simplest method to authenticate a client connection is to assign the user account a password in HP Vertica. If a user account has a password set, then the user or client using the account to connect to the database must supply the correct password. If the user account does not have a password set and HP Vertica is not configured to use another form of client authentication, the user account is always allowed to log in.

Passwords are stored in the database in an encrypted format to prevent others from potentially stealing them. However, the transmission of the password to HP Vertica is in plain text. This means it is possible for a "man in the middle" attack to intercept the password. To secure the login, consider [implementing SSL security](#) or MD5 authentication.

About Password Creation and Modification

A **superuser** creates passwords for user accounts when he or she runs the [CREATE USER](#) statement. You can add a password afterward by using the [ALTER USER](#) statement. To change a password, use [ALTER USER](#) or the `vsq! \password` command. A superuser can set any user account's password. Users can change their own passwords.

To make password authentication more effective, enforce password policies that control how often users are forced to change passwords and the required content of a password. These policies are set using [Profiles](#).

Default Password Authentication

By default, the `vertica.conf` file does not contain any authentication records. When the file is empty, HP Vertica defaults to using password authentication for user accounts that have passwords.

If you add authentication methods to `vertica.conf`, even for remote hosts, password authentication is disabled. You must explicitly enable password authentication. To always enable local users to log in using password authentication, you would add the following to the `vertica.conf` file:

```
ClientAuthentication = local all password
```

The above entry allows users logged into a database host to connect to the database using HP Vertica-based passwords, rather than some other form of authentication.

Profiles

You set password policies using profiles. A profile is a group of parameters that sets requirements for user passwords. You assign users to a profile to set their password policy.

A profile controls:

- How often users must change their passwords.
- How many times users must change their passwords before they can reuse an old password.
- How many times users can fail to log in before their account is locked.
- The required length and content of the password (maximum and minimum amount of characters and the minimum number of letters, capital letters, lowercase letters, digits, and symbols that must be in a password).

You can create multiple profiles to enforce different password policies for different users. For example, you might decide to create one profile for interactive users that requires them to frequently change their passwords and another profile for user accounts that applications use to access the database that aren't required to change passwords.

How You Create and Modify Profiles

You create profiles using the [CREATE PROFILE](#) statement and change profiles using [ALTER PROFILE](#). You can assign a user to a profile when you create the user ([CREATE USER](#)), or afterwards using the [ALTER USER](#) statement. A user can be assigned to only one profile at a time.

All newly-created databases contain an initial profile named DEFAULT. All database users are assigned to the DEFAULT profile if:

- You do not explicitly assign users a profile when you create them
- You drop the profile to which a user is currently assigned

You can change the policy parameters in the DEFAULT profile, but you cannot delete it.

Note: When upgrading from versions of HP Vertica prior to version 5.0, a DEFAULT profile is added to each database, and all users are assigned to it.

The profiles you create can inherit some or all of their policy parameters from the DEFAULT profile. When creating a profile using the [CREATE PROFILE](#) statement, any parameter you set to the special value DEFAULT or any parameter to which you do not assign a value inherits its value from the DEFAULT profile. Changing a parameter in the the DEFAULT profile changes that parameter's value in every profile that inherited the parameter from DEFAULT.

When you assign users to a profile (or alter an existing profile that has users assigned to it), the profile's policies for password content (maximum and minimum length, number of specific types of characters) do not have an immediate effect on the users—HP Vertica does not test user's passwords to ensure they comply with the new password criteria. These settings only affect the users the next time they change their password. If you want to ensure users comply with the new password policy, use the [ALTER USER](#) statement to expire user passwords. Users with expired passwords are prompted to their change passwords when they next log in.

Note: Only the profile settings for how many failed login attempts trigger [Account Locking](#) and

how long accounts are locked have an effect on external password authentication methods such as LDAP or Kerberos. All password [complexity](#), reuse, and [lifetime settings](#) have an effect on passwords managed by HP Vertica only.

See Also

- [PROFILES](#)

Password Expiration

Use profiles to control how often users must change their passwords. Initially, the DEFAULT profile is set so that passwords never expire. You can change this default value, or you can create additional profiles that set time limits for passwords and assign users to them.

When a password expires, the user is required to change his or her password when next logging in, unless the profile to which the user is assigned has a PASSWORD_GRACE_TIME set. In that case, the user is allowed to log in after the expiration, but HP Vertica warns about the password expiration. Once the grace period elapses, the user is forced to change their password, unless they have manually changed the password during the grace period.

Password expiration has no effect on any of the user's current sessions.

Note: You can expire a user's password immediately using the [ALTER USER](#) statement's PASSWORD_EXPIRE argument. Expiring a password is useful to force users to comply with a change to their password policy, or when setting a new password for users who have forgotten their old one.

Account Locking

One password policy you can set in a profile is how many consecutive failed login attempts (giving the wrong password when trying to log in) a user account is allowed before the account is locked. You set this value using the FAILED_LOGIN_ATTEMPTS parameter in the [CREATE PROFILE](#) or [ALTER PROFILE](#) statement.

HP Vertica locks any user account that has more sequential failed login attempts than the value to which you set FAILED_LOGIN_ATTEMPTS. A locked account is not allowed to log in, even if the user supplies the correct password.

How to Unlock a Locked Account

There are two ways to unlock an account:

- A **superuser** can manually unlock the account using the [ALTER USER](#) command.
- HP Vertica automatically unlocks the account after the number of days set in the PASSWORD_LOCK_TIME parameter of the user's profile has passed. However, if this parameter is set to

UNLIMITED, the user's account is never automatically unlocked and a superuser must be manually unlock it.

This locking mechanism helps prevent dictionary-style brute-force attempts to crack users' passwords.

Note: A superuser account cannot be locked, since it is the only user that can unlock accounts. For this reason, you should ensure that you choose a very secure password for a superuser account. See [Password Guidelines](#) for suggestions on choosing a secure password.

The following examples demonstrates failing to log in to an account whose profile is set to lock accounts after three failed tries:

```
> vsql -U dbuserPassword:
vsq1: FATAL: Invalid username or password
> vsql -U dbuser
Password:
vsq1: FATAL: Invalid username or password
> vsql -U dbuser
Password:
vsq1: FATAL: The user account "dbuser" is locked due to too many invalid logins
HINT: Please contact the database administrator
> vsql -U dbuser
Password:
vsq1: FATAL: The user account "dbuser" is locked due to too many invalid logins
HINT: Please contact the database administrator
```

Password Guidelines

For passwords to be effective, they must be hard to guess. You need to protect passwords from:

- Dictionary-style brute-force attacks
- Users who have knowledge of the password holder (family names, dates of birth, etc.)

Use [Profiles](#) to enforce good password practices (password length and required content), and make sure database users know not to use personal information in their passwords.

What to Use

Consider the following password guidelines, published by the Internet Engineering Task Force (IETF), when you create passwords:

- Use mixed-case characters.
- Use non-alphabetic characters (for example, numeric digits and punctuation).

- Use a password that is easy to remember, so you don't need to write it down; for example, i3atSandw1ches! instead of !a#^*!\$&D)z.
- Use a password that you can type quickly without having to look at the keyboard.

What to Avoid

Avoid using the following practices to create a password:

- Do not use your login or user name in any form (as-is, reversed, capitalized, doubled, and so on).
- Do not use your first, middle, or last name in any form.
- Do not use your spouse's, partner's, child's, parent's, friend's, or pet's name in any form.
- Do not use other information easily obtained about you, including your date of birth, license plate number, telephone number, Social Security number, make of your automobile, house address, and so on.
- Do not use a password of all digits or all the same letter.
- Do not use a word contained in English or foreign language dictionaries, spelling lists, acronym or abbreviation lists, or other lists of words.
- Do not use a password that contains fewer than six characters.
- Do not give your password to another person for any reason.

See Also

- [Creating a Database Name and Password](#)

About External Authentication

To help you implement external authentication methods, HP Vertica provides an editing environment within the **Administration Tools** that lets you create, edit, and maintain authentication records. The Administration Tools also verifies that the authentication records are correctly formed, inserts the records into the `vertica.conf` configuration file, and implements the changes on all cluster nodes.

The `vertica.conf` file supports multiple records, one per line, to provide options for client sessions that might require a variety of authentication methods. Each record establishes the authentication method to use based on:

- Connection type
- Client IP address range
- User name for the client that is attempting to access the database

For example, you could use multiple records to have application logins authenticated using HP Vertica-based passwords, and interactive users authenticated using LDAP. See [Example Authentication Records](#).

HP Vertica uses the first record with a matching connection type, client address, and user name to authenticate that connection. If authentication fails, the client is denied access to HP Vertica. Access is also denied if no records match the client session. If, however, there are no records (the DBA did not configure `vertica.conf`), HP Vertica reverts to using the user name and password (if created) to control client access to the database.

Setting up Your Environment to Create Authentication Records

Editing of `vertica.conf` is performed by the text editor set in your Linux or UNIX account's `VISUAL` or `EDITOR` environment variable. If you have not specified a text editor, HP Vertica uses the `vim` (`vi`) editor.

To switch your editor from `vi` to GNU Emacs, run the following command before you run the Administration Tools:

```
$ export EDITOR=/usr/bin/emacs
```

You can also add the above line to the `.profile` file in your home directory to always use GNU Emacs to edit the authentication records.

Caution: Never edit `vertica.conf` directly, because Administration Tools performs error checking on your entries before adding them to the `vertica.conf`.

About Local Password Authentication

If you add authentication methods to `vertica.conf` but still want password authentication to work locally, you must explicitly add a password authentication entry. See [Password Authentication](#) for details.

How to Create Authentication Records

In this procedure, you'll use the Administration Tools to specify the authentication methods to use for various client sessions.

How to Create Authentication Records

1. On the Main Menu in the **Administration Tools**, select **View Database Cluster State**, verify that all cluster nodes are UP, and click **OK**.
2. Select **Configuration Menu**, and click **OK**.
3. On the **Configuration Menu**, select **Edit Authentication**, and click **OK**.
4. Select the database you want to create authentication records for and click **OK**.

Your system's default editor opens the `vertica.conf` file .

5. Enter one or more authentication records.

Tip: See [Authentication Record Format and Rules](#) and [Authentication Record Format and Rules](#) for information about the content and rules required to create a record.

6. When you have finished entering authentication records, exit the editor. For example, in `vi`, press the Esc key and type `:wq` to complete your editing session.

The Administration Tools verifies that the records are correctly formed and does one of the following, the first of which prompts you for action:

- If the records are properly formed, they are inserted into the `vertica.conf` file, and the file is automatically copied to other nodes in the database cluster. You are prompted to restart the database. Click **OK** and go to step 7.
- If the records are not properly formed, a message describes the problem and gives you the opportunity to: edit your errors (e), exit without saving your changes (a), or save and implement your changes anyway (s). Saving your changes is not recommended because it can cause client authentication to fail.

7. [Restart the database](#).

If You Do Not Specify a Client Authentication Method

If you do not insert records into the `vertica.conf` file, HP Vertica defaults to the username and password (if supplied) to grant access to the database. If you later add authentication methods, the username/password default is no longer enabled. To continue using password authentication, you must explicitly add it as described in [Password Authentication](#).

See Also

- [How to Modify Authentication Records](#)

Authentication Record Format and Rules

If the `ClientAuthentication` record introduced in [Security Parameters](#) does not exist, HP Vertica uses the `password` method to authenticate client connections. Otherwise, each authentication record takes the following format:

Syntax

```
ClientAuthentication = connection_type user_name address method
```

Arguments

| | |
|------------------------|--|
| <i>connection_type</i> | <p>The access method the client uses to connect to an instance. Valid values are:</p> <ul style="list-style-type: none">• <code>local</code> — Matches connection attempts made using local domain sockets. When using the local connection type, do not specify the <code><address></code> parameter.• <code>host</code> — Matches connection attempts made using TCP/IP. Connection attempts can be made using a plain (non-SSL) or SSL-wrapped TCP socket.• <code>hostssl</code> — Matches a SSL TCP connection only.• <code>hostnoss1</code> — Matches a plain TCP socket only. <p>Notes about client connections:</p> <ul style="list-style-type: none">• Avoid using <code>-h <hostname></code> from the client if a "local" connection type is specified and you want to match the client authentication entry.• Use <code>-h <hostname></code> from the client if you specify a Kerberos connection method (<code>gss</code> or <code>krb5</code>) connection method. <p>See the vsq1 command line option h Hostname -host Hostname.</p> |
| <i>user_name</i> | <p>Identifies the client's user name that match this record. Valid user names are:</p> <ul style="list-style-type: none">• <code>all</code> — Matches all users.• One or more specific user names. <p>The <i>user_name</i> argument accepts either a single value or concatenated values. To concatenate values, use a plus sign between the values, for example: <code>user1+user2</code>.</p> |

| | |
|----------------|--|
| <i>address</i> | <p>Identifies the client machine IP address range that match this record. Use a format of <code><IP_Address>/<netmask_value></code>. You must specify the IP address numerically, not as domain or host names. HP Vertica supports the following formats:</p> <ul style="list-style-type: none">• <code>w.x.y.z/<mask_format></code> (For example, 10.10.0.25/24.)• The mask length indicates the number of high-order bits of the client IP address that must match. Do not insert white space between the IP address, the slash (/), and the Classless Inter-Domain Routing (CIDR) mask length.• Separate dotted-IP address and mask values (For example, 10.10.0.25 255.255.255.0.)• To allow users to connect from any IP address, use the value 0.0.0.0/0. <p>Note: If you are working with a multi-node cluster, be aware that any IP/netmask settings in host-based <code>ClientAuthentication</code> parameters (<code>host</code>, <code>hostssl</code>, or <code>hostnossl</code>) must match all nodes in the cluster. This setup allows the database owner to authenticate with and administer every node in the cluster. For example, specifying 10.10.0.8/30 would allow a CIDR address range of 10.10.0.8–10.10.0.11.</p> |
|----------------|--|

| | |
|---------------|---|
| <i>method</i> | <p>Identifies the authentication method to use for clients that match this record. Use one of the following methods:</p> <ul style="list-style-type: none">• <code>trust</code> — Authenticates clients based on valid user names only. You might implement <code>trust</code> if a user connection has already been authenticated through some external means, such as SSL or a firewall.• <code>reject</code> — Rejects the connection and prevents additional records from being evaluated for the client. This method is useful for blocking clients by user name or IP address.• <code>gss</code> — Authenticates the client using the GSSAPI standard, allowing for better compatibility with non-MIT kerberos implementations, such as Java and Windows clients. (HP Vertica follows RFC 1964.) <p><code>krb5</code> — Authenticates the client using the MIT Kerberos APIs. This method is deprecated in HP Vertica 7.0. Use the <code>gss</code> method for all new records and modify existing <code>krb5</code> records to use <code>gss</code> as soon as possible.</p> <ul style="list-style-type: none">• <code>password</code> — Requires that the client supply an unencrypted password for authentication. Because the password is sent over the network in clear text, never use this method on untrusted networks.• <code>md5</code> — Requires that the client supply a Message-Digest Algorithm 5 (MD5) password across the network for authentication. By default, passwords are encrypted MD5-hashed passwords and the server provides the client with salt.• <code>ldap</code> — Authenticates the client against a Lightweight Directory Access Protocol (LDAP) server. This method is useful if your application uses LDAP to query directory services.• <code>ident</code> — Authenticates the client using an Ident server (HP Vertica follows RFC 1413). Use this method only when the Ident server is installed on the same system as the HP Vertica database server. |
|---------------|---|

Formatting Rules

When you create authentication records, be aware of the following formatting rules:

- Only one authentication record is allowed per line.
- Each authentication record must be on one line.
- Fields that make up the authentication record can be separated by white space or tabs.
- Other than IP addresses and mask columns, field values cannot contain white space.
- Place more specific rules (a specific user or IP address) before broader rules (all users or a range

of IP addresses).

Note: The order of rules is important. HP Vertica scans the list of rules from top to bottom and uses the first rule that matches the incoming connection.

See Also

- [Security Parameters](#)
- [How to Create Authentication Records](#)
- [Example Authentication Records](#)

Configuring LDAP Authentication

To use LDAP as the authentication method to validate user name/password pairs:

- You must be connected to one or more preconfigured LDAP servers.
- The LDAP directory must contain a record for each client you need to authenticate.

HP Vertica supports two types of LDAP client authentication:

- LDAP bind
- LDAP bind and search

LDAP Bind

If HP Vertica can create a distinguished name (DN) for a user, specify LDAP as the authentication method by creating an authentication record in the `vertica.conf` file similar to the following:

```
ClientAuthentication = host all 10.0.0.0/8 ldap "ldap://ldap.example.com/basedn;cn=;,dc=example,dc=com"
```

Where:

- You must include the URL for the LDAP server in the `ClientAuthentication` parameter. In this example, the URL for the LDAP server is `ldap://ldap.example.com`. For connections over SSL, use `S_HTTP`, as in the following example:

```
ClientAuthentication = local all 10.0.0.0/8 ldap "ldaps://ldap.example.com/basedn;cn=;,dc=qa_domain,dc=com"
```

- The HP Vertica server binds the distinguished name constructed as

prefix username suffix

- Typically, the *prefix* specifies the common name (cn), and the *suffix* specifies the remaining part of the DN. For example, the DN for user "jsmith" would be

```
cn=jsmith,dc=example,dc=com
```

- If the LDAP server does not find that DN, authentication fails.
- For ODBC, the `SQLConnect` function sends the user name and password to HP Vertica for authentication. If the client IP address and user name/password combination matches an LDAP `ClientAuthentication` record in `vertica.conf`, HP Vertica contacts the LDAP server.
- For JDBC, the `java.sql.DriverManager.getConnection()` function passes the user name and password to the database for authentication. If the client IP address and user name/password combination matches an LDAP `ClientAuthentication` record in `vertica.conf`, HP Vertica contacts the LDAP server.

If you have some of the information needed to create a DN in the authentication record, use the Linux tool `ldapsearch` to find the DN. `ldapsearch` opens a connection to an LDAP server, searches the LDAP directory using the specified parameters, and returns the DN if it has enough information and finds a match.

For example, the following `ldapsearch` command connects to an LDAP/Active Directory server and searches for the user. The following command searches the server for user `jsmith`:

```
$ ldapsearch -LLL -H -x ldap://ad.example.com -b 'dc=example,dc=com' -D 'DOMAIN\jsmith' -w 'password' "DC=ad,DC=example,DC=com" '(sAMAccountName=jsmith)' dn cn uid
```

The `ldapsearch` command returns:

```
dn: CN=jsmith,OU=Users,DC=ad,DC=example,DC=comcn: jsmith
```

Using this information, create the `ClientAuthentication` record for this LDAP/Active Directory record as:

```
ClientAuthentication = host all 10.0.0.0/8 ldap "ldap://ad.example.com/basedn;cn=;,OU=users,DC=ad,DC=example,DC=com"
```

LDAP Bind and Search

If HP Vertica does not have enough information to create the DN for a user attempting to authenticate, the authentication record must specify to use LDAP bind and search. For LDAP bind and search authentication, the authentication record must contain the word `search` in the URL of the LDAP server, for example:

```
ldap://ldap.example.com/search
```

When HP Vertica sees `search` in the authentication record, it directs the LDAP server to search for a DN with the information in the record.

The format of an authentication record for LDAP bind and search is:

```
ClientAuthentication = host all 10.0.0.0/8 ldap
                      "ldap://ldap.example.com/search;
                      basedn=<root DN>;
                      binddn=<bind DN>;
                      bindpasswd=<password>;
                      searchattribute=<attribute_name>"
```

In this authentication record:

- `basedn`: Root DN where the search should begin (required)
- `binddn`: DN of the user to search for (Default: blank)
- `bindpasswd`: Password of the `binddn` user (Default: blank)
- `searchattribute`: Attribute of the `binddn` user (Default: UID). Use this parameter to search for the user name within a specific attribute.
- Parameters can appear in any order after `search;`. They must be separated by semicolons.
- The `basedn` parameter is required. All other parameters are optional.
- If HP Vertica passes a user name to the LDAP server, that name cannot contain any of the following characters because they have special meaning on the LDAP server:
 - Asterisks (*)
 - Parentheses (or)
 - Forward slashes (/)
 - Backward slashes (\)

Using LDAP Over SSL and TLS

If the schema in the authentication record in `vertica.conf` is `ldaps`, the HP Vertica server uses SSL on the specified port, or on the LDAPS port (636). If the LDAP server does not support SSL on that port, authentication fails.

If the schema record in the authentication record is `ldap`, the HP Vertica server sends a StartTLS request to see if the LDAP server supports TLS on the specified port or on the default LDAP port

(389). If the LDAP server does not support TLS on that port, the HP Vertica server proceeds with the authentication without TLS.

To use LDAP over SSL and TLS, you must specify the location of your certificate file in your `ldap.conf` file on all nodes:

```
TLS_CACERT /full-path-to-ca-certificate-file.crt
```

LDAP Anonymous Binding

Unless you specifically configure the LDAP server to deny anonymous binds, the underlying LDAP protocol will not cause MC's [Configure Authentication](#) process to fail if you choose "Bind anonymously" for the MC administrator. Before you use anonymous bindings for LDAP authentication on MC, be sure that your LDAP server is configured to explicitly disable/enable this option. For more information, see the article on [Infusion Technology Solutions](#) and the [OpenLDAP documentation](#) on access control.

Configuring Multiple LDAP Servers

In the `vertica.conf` file, the `ClientAuthentication` record can contain multiple LDAP URLs, separated by single spaces. The following record instructs the LDAP server to search the entire directory (`basedn=dc=example,dc=com`) for a DN with an OU (office unit) attribute that matches `Sales`. If the search returns no results or otherwise fails, the LDAP server searches for a DN with the OU attribute that matches `Marketing`:

```
ClientAuthentication = host all 10.0.0.0/8 ldap  
  "ldap://ldap.example.com/search;  
  basedn=dc=example,dc=com; OU=Sales"  
  "ldap://ldap.example.com/search;  
  basedn=dc=example,dc=com;  
  OU=Marketing"
```

Configuring Ident Authentication

The Ident protocol, defined in [RFC 1413](#), identifies the system user of a particular connection. You configure HP Vertica client authentication to query an Ident server to see if that system user can log in as a certain database user without specifying a password. With this feature, system users can run automated scripts to execute tasks on the HP Vertica server.

Caution: Ident responses can be easily spoofed by untrusted servers. Ident authentication should take place only on local connections, where the Ident server is installed on the same computer as the HP Vertica database server.

ClientAuthentication Records for Ident Authentication

To configure Ident authentication, the `ClientAuthentication` record in the `vertica.conf` file must have one of the following formats:

```
ClientAuthentication = local <database_user> ident systemusers=<systemuser1:systemuser2:..  
..> [ continue ]ClientAuthentication = local <database_user> ident [ continue ]
```

Where:

- `local` indicates that the Ident server is installed on the same computer as the database, a requirement for Ident authentication on HP Vertica.
- `<database_user>`: The name of any valid user of the database. To allow the specified system users to log in as any database user, use the word `all` instead of a database user name.
- `<systemuser1:systemuser2:...>`: Colon-delimited list of system user names.
- `continue`: Allows system users not specified in the `systemusers` list to authenticate using methods specified in subsequent `ClientAuthentication` records. The `continue` keyword can be used with or without the `systemusers` list.

The following examples show how to configure Ident authentication in HP Vertica:

- Allow the system's root user to log in to the database as the `dbadmin` user:

```
ClientAuthentication = local dbadmin ident systemusers=root
```

- Allow system users `jsmith`, `tbrown`, and `root` to log in as database user `user1`:

```
ClientAuthentication = local user1 ident systemusers=jsmith:tbrown:root
```

- Allow system user `jsmith` to log in as any database user:

```
ClientAuthentication = local all ident systemusers=jsmith
```

- Allow any system user to log in as the database user of the same name:

```
ClientAuthentication = local all ident
```

- Allow any system user to log in as `user1`:

```
ClientAuthentication = local user1 ident systemusers=*
```

- Allow the system user backup to log in as dbadmin without a password and allow all other system users to log in a dbadmin with a password:

```
ClientAuthentication = local dbadmin ident systemusers=backup continue, local dbadmin password
```

- Allow all system users to log in as the database user with the same name without a password, and log in as other database users with a password:

```
ClientAuthentication = local all ident continue, local all password
```

Installing and Configuring an Ident Server

To use Ident authentication, you must install the oidentd server and enable it on your HP Vertica server. oidentd is an Ident daemon that is compatible with HP Vertica and compliant with RFC 1413.

To install and configure oidentd on Red Hat Linux for use with your HP Vertica database, take these steps:

1. To install oidentd on Red Hat Linux, run this command:

```
$ yum install oidentd
```

Note: The source code and installation instructions for oidentd are available at the [oidentd website](#).

2. For Ident authentication to work, the Ident server must accept IPv6 connections. To make sure this happens, you need to start oidentd with the argument `-a ::`. In the script `/etc/init.d/oidentd`, change the line

```
exec="/usr/sbin/oidentd"
```

to

```
exec="/usr/sbin/oidentd -a ::"
```


3. Restart the server with the following command:

```
/etc/init.d/oidentd restart
```

Example Authentication Records

The following examples show several different authentication records.

Using an IP Range and Trust Authentication Method

The following example allows the dbadmin account to connect from any IP address in the range of 10.0.0.0 to 10.255.255.255 without a password, as long as the connection is made without using SSL:

```
ClientAuthentication = hostnossl dbadmin 10.0.0.0/8 trust
```

Note: If this is the only authentication record in `vertica.conf` file, dbadmin will be the only user that is able to log in.

Using Multiple Authentication Records

When the `vertica.conf` file contains multiple authentication records, HP Vertica scans them from top to bottom and uses the first entry that matches the incoming connection to authenticate (or reject) the user. If the user fails to authenticate using the method specified in the record, HP Vertica denies access to that user. You can use this behavior to include records that enable or reject specific connections and end with one or more "catch-all" records. The following example demonstrates setting up some specific records, followed by some catch-all records:

```
ClientAuthentication = host alice 192.168.1.100/32 reject
ClientAuthentication = host alice 192.168.1.101/32 trust
ClientAuthentication = host all 0.0.0.0/0 password
ClientAuthentication = local all password
```

The first two records apply only to the user alice. If alice attempts to connect from 192.168.1.100, the first record is used to authenticate her, which rejects her connection attempt. If she attempts to connect from 192.168.1.101, she is allowed to connect automatically. If alice attempts to log in from any other remote system, the third record matches, and she must enter a password. Finally, if she attempts to connect locally from a node in the cluster, the fourth record applies, and she again has to enter a password to authenticate herself. For all other users, the third and fourth record are used to authenticate them using password authentication. The first two records are ignored, since their user name doesn't match the name in the record.

Record Order

The ordering of the records is important. If the order of the records were reversed, so that the wildcard rule was first, the rules that are specific to alice would never be used. The wildcard or local rule would always match, and HP Vertica would use the password authentication, no matter where alice connected from.

How to Modify Authentication Records

To modify an existing authentication record, use the **Administration Tools** or set the `ClientAuthentication` configuration parameter.

Using the Administration Tools

The advantages of using the Administration Tools are:

- You do not have to connect to the database
- The editor verifies that records are correctly formed
- The editor maintains records so they are available to you to edit later

Note: You must restart the database to implement your changes.

For information about using the Administration Tools to create and edit authentication records, see [How to Create Authentication Records](#).

Using the ClientAuthentication Configuration Parameter

The advantage of using the `ClientAuthentication` configuration parameter is that the changes are implemented immediately across all nodes within the database cluster. You do not need to restart the database.

However, all the database nodes must be up and you must [connect to the database](#) before you set this parameter. Most importantly, this method does not verify that records are correctly formed and it does not maintain the records so you can modify them later.

New authentication records are appended to the list of existing authentication records. Because HP Vertica scans the list of records from top to bottom and uses the first record that matches the incoming connection, you might find your newly-added record does not have an effect if HP Vertica used an earlier record instead.

To configure client authentication through a connection parameter, use the `SET_CONFIG_PARAMETER` function:

```
=> SELECT SET_CONFIG_PARAMETER('ClientAuthentication,'
```

```
'connection type user name address method');
```

When you specify authentication records, make sure to adhere to the following guidelines:

- Fields that make up the record can be separated by white space or tabs
- Other than IP addresses and mask columns, field values cannot contain white space

For more information, see [Authentication Record Format and Rules](#).

Examples

The following example creates an authentication record for the trust method:

```
=> SELECT SET_CONFIG_PARAMETER('ClientAuthentication',  
'hostnossl dbadmin 0.0.0.0/0 trust');
```

The following example creates an authentication record for the LDAP method:

```
=> SELECT SET_CONFIG_PARAMETER('ClientAuthentication', 'host all 10.0.0.0/8  
ldap "ldap://summit.vertica.com;cn=;,dc=vertica,dc=com"');
```

The following example specifies three authentication records. In a single command, separate each authentication record by a comma:

```
=> SELECT SET_CONFIG_PARAMETER('ClientAuthentication',  
'hostnossl dbadmin 0.0.0.0/0 trust, hostnossl all 0.0.0.0/0 md5,  
local all trust');
```

Implementing Kerberos Authentication

Kerberos authentication is different from user name/password authentication. Instead of authenticating each user to each network service, Kerberos uses symmetric encryption through a trusted third party, called the Key Distribution Center (KDC). In this environment, clients and servers validate their authenticity by obtaining a shared secret (ticket) from the KDC, after which clients and servers can talk to each other directly.

Note: Topics in this section describe how to configure the HP Vertica server and clients for Kerberos authentication. This section does not describe how to install, configure, or administer a Key Distribution Center. You can obtain the Kerberos 5 GSSAPI distribution for your operating system from the [MIT Kerberos Distribution Page](#).

Kerberos Prerequisites

At a minimum you must meet the following requirements to use Kerberos authentication with the HP Vertica server and client drivers.

Kerberos server

Your network administrator should have already installed and configured one or more Kerberos Key Distribution Centers (KDC), and the KDC must be accessible from every node in your Vertica Analytics Platform cluster.

The KDC must support Kerberos 5 via GSSAPI. For details, see the [MIT Kerberos Distribution Page](#).

Client package

The Kerberos 5 client package contains software that communicates with the KDC server. This package is not included as part of the HP Vertica Analytics Platform installation. If the Kerberos 5 client package is not present on your system, you must download and install it on all clients and servers involved in Kerberos authentication (for example, each HP Vertica and each HP Vertica client), with the exception of the KDC itself.

Kerberos software is built into Microsoft Windows. If you are using another operating system, you must obtain and install the client package.

Refer to the Kerberos documentation for installation instructions, such as on the [MIT website](#), including the [MIT Kerberos Distribution page](#).

Client/server identity

Each client (users or applications that will connect to HP Vertica) and the HP Vertica server must be configured as Kerberos principals. These principals authenticate using the KDC.

Each client platform has a different security framework, so the steps required to configure and authenticate against Kerberos differ among clients. See the following topics for more information:

- [Configure HP Vertica for Kerberos Authentication](#)
- [Configure Clients for Kerberos Authentication](#).

Configure HP Vertica for Kerberos Authentication

To set up HP Vertica for Kerberos authentication, perform a series of short procedures described in the following sections:

- [Install the Kerberos 5 client package](#)
- [Create the HP Vertica principal](#)
- [Create the keytab](#)
- [Specify the location of the keytab file](#)

- [Point machines at the KDC and configure realms](#)
- [Inform HP Vertica about the Kerberos principal](#)
- [Configure the authentication method for all clients](#)
- [Restart the database](#)
- [Get the ticket and authenticate HP Vertica with the KDC](#)

Install the Kerberos 5 client package

See [Kerberos Prerequisites](#).

Create the HP Vertica principal

You can create the Vertica Analytics Platform principal on any machine in the Kerberos realm, though generally, you'll perform this task on the KDC. This section describes how to create the Vertica Analytics Platform principal on Linux and Active Directory KDCs.

Creating the Vertica Analytics Platform principal on a Linux KDC

Start the Kerberos 5 database administration utility (`kadmin` or `kadmin.local`) to create an Vertica Analytics Platform principal on a Linux KDC.

- Use `kadmin` if you are accessing the KDC on a remote server. You can use `kadmin` on any machine that has the Kerberos 5 client package installed, as long as you have access to the Kerberos administrator's password. When you start `kadmin`, the utility will prompt you for the Kerberos administrator's password. You might need root privileges on the client system in order to run `kadmin`.
- Use `kadmin.local` if the KDC is on the machine you're logging in to and you have root privileges on that server. You might also need to modify your path to include the location of the `kadmin.local` command; for example, try setting the following path:
`/usr/kerberos/sbin/kadmin.local`.

The following example creates the principal `vertica` on the `EXAMPLE.COM` Kerberos realm:

```
$ sudo /usr/kerberos/sbin/kadmin.local  
kadmin.local add_principal vertica/vcluster.example.com
```

For more information about the `kadmin` command, refer to the `kadmin` documentation.

Creating the Vertica Analytics Platform principal on an Active Directory KDC

To configure Vertica Analytics Platform for Kerberos authentication on Active Directory, you will generally can most likely add the Vertica Analytics Platform server and clients to an existing Active Directory domain. You'll need to modify the Kerberos configuration file (`krb5.conf`) on the Vertica

Analytics Platform server to make sure all parties support encryption types used by the Active Directory KDC.

If you need to configure encryption on the KDC:

- Open the Local Group Policy Editor (gpedit.msc) and expand Computer Configuration > Windows Settings > Security Settings > Local Policies > Security Options, and double-click Network security: Configure encryption types allowed for Kerberos.
- Refresh the local and Active Directory-based Group Policy settings, including security settings, by running the command `gpupdate /force`
- Use the `ktpass` command to configure the server principal name for the host or service in Active Directory and generate a `.keytab` file; for example:

```
ktpass -out ./host.vcluster.example.com.keytab
-princ host/vcluster.example.com@EXAMPLE.COM
-mapuser vcluster
-mapop set -pass <password>
-crypto <encryption_type> -ptype <principal_type>

ktpass -out ./vertica.vcluster.example.com.keytab
-princ vertica/vcluster.example.com@EXAMPLE.COM
-mapuser vertica
-mapop set -pass <password>
-crypto <encryption_type> -ptype <principal_type>
```

For more information, see the [Technet.Microsoft.com Ktpass page](https://technet.microsoft.com/Ktpass). See also "[Create the keytab](#)" below.

You can view a list of the Service Principal Names that a computer has registered with Active Directory by running the `setspn -l hostname` command, where *hostname* is the host name of the computer object that you want to query; for example:

```
setspn -L vertica
Registered ServicePrincipalNamefor CN=vertica,CN=Users,
EXAMPLE=example,EXAMPLE=com
vertica/vcluster.example.com

setspn -L vcluster
Registered ServicePrincipalNamefor CN=vertica,CN=Users,
EXAMPLE=example,EXAMPLE=com
host/vcluster.example.com
```

Create the keytab

The keytab is an encrypted, local copy of the host's key that contains the credentials for the HP Vertica principal (its own principal and key), so the HP Vertica server can authenticate itself to the KDC. The keytab is required so that Vertica Analytics Platform doesn't have to prompt for a password.

Before you create the keytab file

You do not need to create a keytab file on the KDC; however, a keytab entry must reside on each node in the HP Vertica cluster. The absolute path to the keytab file must be the same on every cluster node.

You can generate a keytab on any machine in the Kerberos realm that already has the Kerberos 5 client package installed, as long as you have access to the Kerberos administrator's password. Then you can copy that file to each Vertica Analytics Platform node.

Generating a keytab file on Linux systems

On Linux, the default location for the keytab file is `/etc/krb5.keytab`. You might need root privileges on the client system to run the `kadmin` utility.

1. To generate a keytab or add a principal to an existing keytab entry, use the `ktadd` command from the `kadmin` utility, as in the following example:

```
$ sudo /usr/kerberos/sbin/kadmin -p vertica/vcluster.example.com -q  
"ktadd vertica/vcluster.example.com" -r EXAMPLE.COM  
Authenticating as principal vertica/vcluster.example.com  
with password.
```

2. Make the keytab file readable by the file owner who is running the process (typically the Linux `dbadmin` user); for example, you can change ownership of the files to `dbadmin` as follows:

```
$ sudo chown dbadmin *.keytab
```

Important: In a production environment, you must control who can access the keytab file to prevent unauthorized users from impersonating your server.

3. Copy the keytab file to the `/etc` folder on each cluster node.

After you create the keytab file, you can use the `klist` command to view keys stored in the file:

```
$ sudo /usr/kerberos/bin/klist -ke -t  
Keytab name: FILE:/etc/krb5.keytab  
KVNO Principal  
-----  
4 vertica/vcluster.example.com@EXAMPLE.COM  
4 vertica/vcluster.example.com@EXAMPLE.COM
```

Generating a keytab file for Active Directory

Use the `ktutil` command to read, write, or edit entries in a Kerberos 5 keytab file. The keytab entries were created in the previous example, and they created a principal name for the host and service in Active Directory.

1. On any Vertica Analytics Platform node use the `ktutil` command to read the Kerberos 5 keytab file `keytab` into the current keylist (from the keytab entries you created using `ktpass` -

out):

```
$ /usr/kerberos/sbin/ktutil
ktutil: rkt host.vcluster.example.com.keytab
ktutil: rkt vertica.vcluster.example.com.keytab

ktutil: list

KVNO Principal
-----
   3 host/vcluster.example.com@EXAMPLE.COM
  16 vertica/vcluster.example.com@EXAMPLE.COM

ktutil: wkt vcluster.example.com.keytab
ktutil: exit
```

2. Make the keytab file readable by the file owner who is running the process (the administrator) with no permissions for group or other:

```
$ chmod 600 vcluster.example.com.keytab
```

3. Copy the keytab file to the catalog directory.

Specify the location of the keytab file

Using the Administration Tools, log in to the database as the database administrator (usually dbadmin) and set the KerberosKeyTabFile configuration parameter to point to the location of the keytab file; for example:

```
> SELECT set_config_parameter('KerberosKeytabFile', '/etc/krb5.keytab');
```

See [Kerberos Authentication Parameters](#) for more information.

Point machines at the KDC and configure realms

Each client and Vertica Analytics Platform server in the Kerberos realm must have a valid, identically-configured Kerberos configuration (krb5.conf) file in order to know how to reach the KDC.

If you use Microsoft Active Directory, you won't need to perform this step. Refer to the Kerberos documentation for your platform for more information about the Kerberos configuration file on Active Directory.

At a minimum, you must configure the following sections in the krb5.conf file. See the Kerberos documentation for information about other sections in this configuration file.

- [libdefaults] Settings used by the Kerberos 5 library
- [realms] Realm-specific contact information and settings
- [domain_realm] Maps server hostnames to Kerberos realms

You need to update the `/etc/krb5.conf` file to reflect your site's Kerberos configuration. The easiest way to ensure consistency among all clients/servers in the Kerberos realm is to copy the `/etc/krb5.conf` file from the KDC to the `/etc` directory on each HP Vertica cluster node.

Inform HP Vertica about the Kerberos principal

Follow these steps to inform HP Vertica about the `KerberosServiceName/KerberosHostname@KerberosRealm` principal. This procedure provides an example principal called `vertica/vcluster@EXAMPLE.COM`.

About the host name parameter

If you omit the optional `KerberosHostname` parameter in Step 2 below, HP Vertica uses the return value from the `gethostname()` function. Assuming each cluster node has a different host name, those nodes will each have a different principal, which you must manage in that node's keytab file. HP recommends that you specify the `KerberosHostname` parameter to get a single, cluster-wide principal that is easier to manage than multiple principals.

Configure the Vertica Analytics Platform principal parameters

For information about the parameters that you'll set in this procedure, see [Kerberos Authentication Parameters](#).

1. Log in to the database as an administrator (typically `dbadmin`) and set the service name for the HP Vertica principal; for example, `vertica`.

```
> SELECT set_config_parameter('KerberosServiceName', 'vertica');
```

2. Optionally provide the instance or hostname portion of the principal; for example, `vcluster` (see "About the host name parameter" above this procedure):

```
> SELECT set_config_parameter('KerberosHostname', 'vcluster.example.com');
```

3. Provide the realm portion of the principal; for example, `EXAMPLE.COM`:

```
> SELECT set_config_parameter('KerberosRealm', 'EXAMPLE.COM');
```

Configure the authentication method for all clients

To make sure that all clients use the gss Kerberos authentication method, run the following command:

```
> SELECT set_config_parameter('ClientAuthentication', 'host all 0.0.0.0/0 gss');
```

For more information, see [Implementing Client Authentication](#).

Restart the database

For all settings to take effect, you must [restart the database](#).

Get the ticket and authenticate HP Vertica with the KDC

The example below shows how to get the ticket and authenticate Vertica Analytics Platform with the KDC using the `kinit` command. You'll commonly perform this final step from the `vsq` client.

```
/etc/krb5.conf
EXAMPLE.COM = {
  kdc = myserver.example.com:11
  admin_server = myadminserver.example.com:000
  kpasswd_protocol = SET_CHANGE
  default_domain = example /etc/krb5.conf
}
```

Calling the `kinit` utility requests a ticket from the KDC server.

```
$ kinit kuser@EXAMPLE.COM
Password for kuser@EXAMPLE.COM:
```

Configure Clients for Kerberos Authentication

Each supported platform has a different security framework, so the steps required to configure and authenticate against Kerberos differ among clients.

On the server side, you construct the HP Vertica Kerberos service name principal using this format:

```
KerberosServiceName/KerberosHostname@KerberosRealm
```

On the client side, the GSS libraries require the following format for the HP Vertica service principal:

```
KerberosServiceName@KerberosHostName
```

The realm portion of the principal can be omitted because GSS libraries use the realm name of the configured default (`KerberosRealm`) realm.

For information about client connection strings, see the following topics in the Programmer's Guide:

- [ODBC DSN Parameters](#)
- [JDBC Connection Properties](#)
- [ADO.NET Connection Properties](#)
- (vsq) [Command Line Options](#)

Note: A few scenarios exist in which HP Vertica server's principal name might not match the host name in the connection string. See [Troubleshooting Kerberos Authentication](#) for more information.

In This Section

- [Configure ODBC and vsq Clients on Linux, HP-UX, AIX, MAC OSX, and Solaris](#)
- [Configure ODBC and vsq Clients on Windows and ADO.NET](#)
- [Configure JDBC Clients on all Platforms](#)

Configure ODBC and vsq Clients on Linux, HP-UX, AIX, MAC OSX, and Solaris

This topic describes the requirements for configuring an ODBC or vsq client on Linux, HP-UX, AIX, MAC OSX, or Solaris.

Install the Kerberos 5 client package

See [Kerberos Prerequisites](#).

Provide clients with a valid Kerberos configuration file

The Kerberos configuration (`krb5.conf`) file contains Kerberos-specific information, such as how to reach the KDC, default realm name, domain, the path to log files, DNS lookup, encryption types to use, ticket lifetime, and other settings. The default location for the Kerberos configuration file is `/etc/krb5.conf`.

Each client participating in Kerberos authentication must have a valid, identically-configured `krb5.conf` file in order to communicate with the KDC. When configured properly, the client can authenticate with Kerberos and retrieve a ticket through the `kinit` utility. Likewise, the server can then use `ktutil` to store its credentials in a keytab file

Tip: The easiest way to ensure consistency among clients, Vertica Analytics Platform, and the KDC is to copy the `/etc/krb5.conf` file from the KDC to the client's `/etc` directory.

Authenticate and connect clients

ODBC and vsql use the client's ticket established by `kinit` to perform Kerberos authentication. These clients rely on the security library's default mechanisms to find the ticket file and the Kerberos configuration file.

To authenticate against Kerberos, call the `kinit` utility to obtain a ticket from the Kerberos KDC server. The following two examples show how to send the ticket request using ODBC and vsql clients.

ODBC authentication request/connection

1. On an ODBC client, call the `kinit` utility to acquire a ticket for the `kuser` user:

```
$ kinit kuser@EXAMPLE.COM
Password for kuser@EXAMPLE.COM:
```

2. Connect to HP Vertica and provide the principals in the connection string:

```
char outStr[100];
SQLLEN len;
SQLDriverConnect(handle, NULL, "Database=VMart;User=kuser;
Server=myserver.example.com;Port=5433;KerberosHostname=vcluster.example.com",
SQL_NTS, outStr, &len);
```

vsql authentication request/connection

If the vsql client is on the same machine you're connecting to, vsql connects through a UNIX domain socket and bypasses Kerberos authentication. When you are authenticating with Kerberos, especially if the [ClientAuthentication](#) record `connection_type` is 'local', you must include the `-h` hostname option, described in [Command Line Options](#) in the Programmer's Guide.

1. On the vsql client, call the `kinit` utility:

```
$ kinit kuser@EXAMPLE.COM
Password for kuser@EXAMPLE.COM:
```

2. Connect to HP Vertica and provide the host and user principals in the connection string:

```
$ ./vsql -K vcluster.example.com -h myserver.example.com -U kuser

Welcome to vsql, the Vertica Analytic Database
interactive terminal.

Type: \h or \? for help with vsql commands
\g or terminate with semicolon to execute query
\q to quit
```

In the future, when you log in to vsql as kuser, vsql uses your cached ticket without prompting you for a password.

You can verify the authentication method by querying the SESSIONS system table:

```
kuser=> SELECT authentication_method FROM sessions;
authentication_method
-----
GSS-Kerberos
(1 row)
```

See Also

- [Kerberos Client/Server Requirements](#)
- [ODBC DSN Parameters](#) in the Programmer's Guide
- (vsq) [Command Line Options](#) in the Programmer's Guide

Configure ADO.NET, ODBC, and vsql Clients on Windows

The HP Vertica client drivers support the Windows SSPI library for Kerberos authentication. Windows Kerberos configuration is stored in the registry.

You can choose between two different setup scenarios for Kerberos authentication on ODBC and vsql clients on Windows and ADO.NET:

- [Windows KDC on Active Directory with Windows built-in Kerberos client and HP Vertica](#)
- [Linux KDC with Windows-built-in Kerberos client and HP Vertica](#)

Windows KDC on Active Directory with Windows built-in Kerberos client and HP Vertica

Kerberos authentication on Windows is commonly used with Active Directory, Microsoft's enterprise directory service/Kerberos implementation, and is most likely set up by your organization's network or IT administrator.

Windows clients have Kerberos authentication built into the authentication process. This means when you log in to Windows from a client machine, and your Windows instance has been configured to use Kerberos through Active Directory, your login credentials authenticate you to the Kerberos server (KDC). There is no additional software to set up. To use Kerberos authentication on Windows clients, log in as REALM\user.

ADO.NET IntegratedSecurity

When you use the ADO.NET driver to connect to HP Vertica, you can optionally specify `IntegratedSecurity=true` in the connection string. This Boolean setting informs the driver to authenticate the calling user against his or her Windows credentials. As a result, you do not need to

include a user name or password in the connection string. If you add a `user=<username>` entry to the connection string, the ADO.NET driver ignores it.

Linux KDC with Windows-built-in Kerberos client and HP Vertica

A simpler, but less common scenario is to configure Windows to authenticate against a non-Windows KDC. In this implementation, you use the `ksetup` utility to point the Windows operating system's native Kerberos capabilities at a non-Active Directory KDC. The act of logging in to Windows obtains a ticket granting ticket, similar to the Active Directory implementation, except in this case, Windows is internally communicating with a Linux KDC. See the Microsoft Windows Server [Ksetup page](#) for more information.

Configuring Windows clients for Kerberos authentication

Depending on which implementation you want to configure, refer to one of the following pages on the Microsoft Server website:

- To set up Windows clients with Active Directory, refer to [Step-by-Step Guide to Kerberos 5 \(krb5 1.0\) Interoperability](#).
- To set up Windows clients with the `ksetup` utility, refer to the [Ksetup page](#).

Authenticate and connect clients

This section shows you how to authenticate an ADO.NET and vsql client to the KDC, respectively.

Note: Use the fully-qualified domain name as the server in your connection string; for example, use `host.example.com` instead of just `host`.

ADO.NET authentication request/connection

The following example uses the `IntegratedSecurity=true`, setting, which instructs the ADO.NET driver to authenticate the calling user's Windows credentials:

```
VerticaConnection conn = new  
VerticaConnection("Database=VMart;Server=host.example.com;  
Port=5433;IntegratedSecurity=true;  
KerberosServiceName=vertica;KerberosHostname=vcluster.example.com");  
conn.open();
```

vsql authentication request/connection

1. Log in to your Windows client, for example as `EXAMPLE\kuser`.
2. Run the vsql client and supply the connection string to HP Vertica:

```
C:\Users\kuser\Desktop>vsql.exe -h host.example.com -K vcluster -U kuser
```

```
Welcome to vsql, the Vertica Analytic Database interactive terminal.  
Type: \h or \? for help with vsql commands  
\g or terminate with semicolon to execute query  
\q to quit
```

See Also

- [Kerberos Client/Server Requirements](#)
- [vsql Command Line Options](#) in the Programmer's Guide
- [ADO.NET Connection Properties](#) in the Programmer's Guide

Configure JDBC Clients on All Platforms

Kerberos authentication on JDBC clients uses Java™ Authentication and Authorization Service (JAAS) to acquire the initial Kerberos credentials. JAAS is an API framework that hides platform-specific authentication details and provides a consistent interface for other applications.

The client login process is determined by the JAAS Login Configuration File, which contains options that specify the authentication method and other settings to use for Kerberos. Options allowed in the configuration file are defined by a class called the `LoginModule`.

The JDBC client principal is crafted as `jdbc-username@server-from-connection-string`.

About the LoginModule

Many vendors can provide a `LoginModule` implementation that you can use for Kerberos authentication, but HP recommends that you use the JAAS public class `com.sun.security.auth.module.Krb5LoginModule` provided in the Java Runtime Environment (JRE). The `Krb5LoginModule` authenticates users using Kerberos protocols and is implemented differently on non-Windows and Windows platforms:

- **On non-Windows platforms:** The `Krb5LoginModule` defers to a native Kerberos client implementation, which means you can use the same `/etc/krb5.conf` setup as you use to [configure ODBC and vsql clients](#) on Linux, HP-UX, AIX, MAC OSX, and Solaris platforms.
- **On Windows platforms:** The `Krb5LoginModule` uses a custom Kerberos client implementation bundled with the Java Runtime Environment (JRE). Windows settings are stored in a `%WINDIR%\krb5.ini` file, which has similar syntax and conventions to the non-Windows `krb5.conf` file. You can copy a `krb5.conf` from a non-Windows client to `%WINDIR%\krb5.ini`.

Documentation for the `LoginModules` is in the `com.sun.security.auth` package, and on the [Krb5LoginModule](#) web page.

Create the JAAS login configuration

The `JAASConfigName` connection property identifies a specific configuration within a JAAS configuration that contains the `Krb5LoginModule` and its settings. The `JAASConfigName` setting lets multiple JDBC applications with different Kerberos settings coexist on a single host. The default configuration name is `verticajdbc`.

Note: Carefully construct the JAAS login configuration file. If syntax is incorrect, authentication will fail.

You can configure JAAS-related settings in the `java.security` master security properties file, which is located in the `lib/security` directory of the JRE. For more information, see [Appendix A](#) in the Java™ Authentication and Authorization Service (JAAS) Reference Guide.

Create a JDBC login context

The following example creates a login context for Kerberos authentication on a JDBC client that uses the default `JAASConfigName` of `verticajdbc` and specifies that:

- The ticket granting ticket will be obtained from the ticket cache
- The user will not be prompted for a password if credentials can't be obtained from the cache, the keytab file, or through shared state

```
verticajdbc {  
    com.sun.security.auth.module.Krb5LoginModule  
    required  
    useTicketCache=true  
    doNotPrompt=true;  
};
```

JDBC authentication request/connection

You can configure the `Krb5LoginModule` to use a cached ticket, keytab, or the driver can acquire one automatically if the calling user provides a password.

In the previous example, the login process uses a cached ticket and won't ask for a password because both `useTicketCache` and `doNotPrompt` are set to `true`. If `doNotPrompt=false` and you provide a user name and password during the login process, the driver provides that information to the `LoginModule` and calls the `kinit` utility on your behalf.

1. On a JDBC client, call the `kinit` utility to acquire a ticket:

```
kinit kuser@EXAMPLE.COM
```

If you prefer to use a password instead of calling the `kinit` utility, see ["Have the driver call kinit for you"](#).

2. Connect to HP Vertica:

```
Properties props = new Properties();
props.setProperty("user", "kuser");
props.setProperty("KerberosServiceName", "vertica");
props.setProperty("KerberosHostName", "vcluster.example.com");
props.setProperty("JAASConfigName", "verticajdbc");
Connection conn = DriverManager.getConnection
    "jdbc:vertica://myserver.example.com:5433/VMart", props);
```

Have the driver call kinit for you

If you want to bypass calling the `kinit` utility yourself but still benefit from encrypted, mutual authentication, you can optionally pass the driver a clear text password to acquire the ticket from the KDC. The password is encrypted when sent across the network. For example, `useTicketCache` and `doNotPrompt` are now both `false` in the example below, which means that the calling user's credentials will not be obtained through the ticket cache or keytab.

```
verticajdbc {
    com.sun.security.auth.module.Krb5LoginModule
    required
    useTicketCache=false
    doNotPrompt=false;
};
```

In the above example, the driver no longer looks for a cached ticket and you don't have to call `kinit`. Instead, the driver takes the password and user name and calls `kinit` on your behalf.

Note: The above is an example to demonstrate the flexibility of JAAS.

See Also

- [Kerberos Client/Server Requirements](#)
- [JDBC Connection Properties](#) in the Programmer's Guide
- [Java™ Authentication and Authorization Service \(JAAS\) Reference Guide](#) (external website)

Determining the Client Authentication Method

To determine the details behind the type client authentication used for a particular user session, query the `V_MONITOR.SESIONS` system table.

The following example output indicates that the GSS Kerberos authentication method is set:

```
> SELECT authentication_method FROM sessions;
authentication_method
-----
GSS-Kerberos
(1 row)
```

For a list of possible values in the `authentication_method` column, see [Implementing Client Authentication](#).

Tip: You can also view details about the authentication method by querying the `V_MONITOR.USER_SESSIONS` system table.

Troubleshooting Kerberos Authentication

This topic provides tips to help you avoid and troubleshoot issues related to Kerberos authentication with Vertica Analytics Platform.

Server's principal name doesn't match the host name

In some cases during client connection, the HP Vertica server's principal name might not match the host name in the connection string. (See also [Using the ODBC Data Source Configuration utility](#) in this topic.)

On ODBC, JDBC, and ADO.NET clients, you set the host name portion of the server's principal using the `KerberosHostName` connection string. See the following topics in the Programmer's Guide:

- [ODBC DSN Parameters](#)
- [JDBC Connection Properties](#)
- [ADO.NET Connection Properties](#)

On vsql clients, you set the host name portion of the server's principal name using the `-K KRB_HOST` command line option, where the default value is specified by the `-h` switch (the host name of the machine on which the HP Vertica server is running). `-K` is equivalent to the driver's `KerberosHostName` connection string value. See [Command Line Options](#) in the Programmer's Guide.

Principal/host mismatch issues and resolutions

Here are some common scenarios where the server's principal name might not match the host name with a workaround, when available.

- The `KerberosHostName` configuration parameter has been overridden.

For example, consider the following connection string:

```
jdbc:vertica://node01.example.com/vmart?user=kuser
```

Because the above connection string includes no explicit `KerberosHostName` parameter, the driver defaults to the host in the URL (`node01.example.com`). If you overwrote the server-side `KerberosHostName` parameter as “abc”, the client would generate an incorrect principal. To resolve this issue, explicitly set the client’s `KerberosHostName` to the connection string; for example:

```
jdbc:vertica://node01.example.com/vmart?user=kuser&kerberoshostname=abc
```

- Connection load balancing is enabled, but the node against which the client authenticates might not be the node in the connection string.

In this situation, consider setting all nodes to use the same `KerberosHostName` setting. When you default to the host originally specified in the connection string, load balancing cannot interfere with Kerberos authentication.

- You have a DNS name that does not match the Kerberos hostname.

For example, imagine a cluster of six servers, where you want `hr-servers` and `finance-servers` to connect to different nodes on the Vertica Analytics Platform cluster. Kerberos authentication, however, occurs on a single (the same) KDC. In this example, the Kerberos service hostname of the servers is `server.example.com`, and here's the list of example servers:

```
server1.example.com 192.16.10.11  
server2.example.com 192.16.10.12  
server3.example.com 192.16.10.13  
server4.example.com 192.16.10.14  
server5.example.com 192.16.10.15  
server6.example.com 192.16.10.16
```

Now assume you have the following DNS entries:

```
finance-servers.example.com 192.168.10.11, 192.168.10.13, 192.168.10.13  
hr-servers.example.com 192.168.10.14, 192.168.10.15, 192.168.10.16
```

When you connect to `finance-servers.example.com`, specify the Kerberos `-h` hostname option as `server.example.com` and the `-K` host option for `hr-servers.example.com`; for example:

```
$ vsql -h fianance-servers.example.com -K server.example.com
```

- You do not have DNS set up on the client machine, so you have to connect by IP only.

To resolve this issue, specify the Kerberos `-h` hostname option for the IP address and the `-K` host option for `server.example.com`; for example:

```
$ vsql -h 192.168.1.12 -K server.example.com
```

- There is a load balancer involved (Virtual IP), but there is no DNS name for the VIP.

Specify the Kerberos `-h` hostname option for the Virtual IP address and the `-K` host option for `server.example.com`; for example:

```
$ vsql -h <virtual IP> -K server.example.com
```

- You connect to HP Vertica using an IP address, but there is no host name to construct the Kerberos principal name.
- You set the server-side `KerberosHostName` configuration parameter to a name other than the HP Vertica node's host name, but the client can't determine the host name based on the hostname in the connection string alone.

JDBC client authentication

If Kerberos authentication fails on a JDBC client, check the JAAS login configuration file for syntax issues. If syntax is incorrect, authentication will fail.

Working Domain Name Service (DNS)

Make sure that the DNS entries and hosts on the network are all properly configured. Refer to the Kerberos documentation for your platform for details.

Clock synchronization

Systems clocks in your network must remain in sync for Kerberos authentication to work properly, so you need to:

- Install NTP on the Kerberos server (KDC)
- Install NTP on each server in your network
- Synchronize system clocks on all machines that participate in the Kerberos realm within a few minutes of the KDC and each other

Clock skew can be problematic on Linux Virtual Machines that need to sync with Windows Time Service. You can try the following to keep time in sync:

1. Use any text editor to open `/etc/ntp.conf`.
2. Under the `Undisciplined Local Clock` section, add the IP address for the Vertica Analytics Platform server and remove existing server entries.
3. Log in to the server as root and set up a cron job to sync time with the added IP address every half hour, or as often as needed; for example:

```
# 0 */2 * * * /etc/init.d/ntpd restart
```

4. Alternatively, run the following command to force clock sync immediately:

```
$ sudo /etc/init.d/ntpd restart
```

For more information, see [Set Up Time Synchronization](#) in the Installation Guide and the [Network Time Protocol website](#).

Encryption algorithm choices

Kerberos is based on symmetric encryption, so be sure that all Kerberos parties involved in the Kerberos realm agree on the encryption algorithm to use. If they don't agree, authentication fails. You can review the exceptions in the `vertica.log`.

On a Windows client, be sure the encryption types match the types set on Active Directory (see [Configure HP Vertica for Kerberos Authentication](#)).

Be aware that Kerberos is used for securing the log-in process only. After the log-in process completes, information travels between client and server without encryption, by default. If you want to encrypt traffic, use SSL. For details, see [Implementing SSL](#).

Kerberos passwords

If you change your Kerberos password, you must recreate all of your keytab files.

Using the ODBC Data Source Configuration utility

On Windows vsql clients, if you use the ODBC Data Source Configuration utility and supply a client Data Source, be sure you enter a Kerberos host name in the Client Settings tab to avoid client connection failures with the Vertica Analytics Platform server.

Implementing SSL

To ensure privacy and verify data integrity, you can configure HP Vertica and database clients to use Secure Socket Layer (SSL) to communicate and secure the connection between the client and the server. The SSL protocol uses a trusted third-party called a Certificate Authority (CA), which means that both the owner of a certificate and the party that relies on the certificate trust the CA.

Certificate Authority

The CA issues electronic certificates to identify one or both ends of a transaction and to certify ownership of a public key by the name on the certificate.

Public/private Keys

A CA issues digital certificates that contain a public key and the identity of the owner.

The public key is available to all users through a publicly-accessible directory, while private keys are confidential to their respective owner. The private/public key pair ensures that the data can be encrypted by one key and decrypted by the other key pair only.

The public and private keys are similar and can be used alternatively; for example, what one key encrypts, the other key pair can decrypt.

- If encrypted with a public key, can be decrypted by its corresponding private key only
- If encrypted with a private key can be decrypted by its corresponding public key only

For example, if Alice wants to send confidential data to Bob and needs to ensure that only Bob can read it, she will encrypt the data with Bob's public key. Only Bob has access to his corresponding private key; therefore, he is the only person who can decrypt Alice's encrypted data back into its original form, even if someone else gains access to the encrypted data.

HP Vertica uses SSL to:

- Authenticate the server so the client can confirm the server's identity. HP Vertica also supports mutual authentication in which the server can confirm the identity of the client. This authentication helps prevent **man-in-the-middle attacks**.
- Encrypt data sent between the client and database server to significantly reduce the likelihood that the data can be read if the connection between the client and server is compromised.
- Verify that data sent between the client and server has not been altered during transmission.

HP Vertica supports the following authentication methods under SSL v3/Transport Layer Security (TLS) 1.0 protocol:

- **SSL server authentication** — Lets the client confirm the server's identity by verifying that the server's certificate and public key are valid and were issued by a certificate authority (CA) listed

in the client's list of trusted CAs. See "Required Prerequisites for SSL Server Authentication and SSL Encryption" in [SSL Prerequisites](#) and [Configuring SSL](#).

- **SSL client authentication** — (Optional) Lets the server confirm the client's identity by verifying that the client's certificate and public key are valid and were issued by a certificate authority (CA) listed in the server's list of trusted CAs. Client authentication is optional because HP Vertica can achieve authentication at the application protocol level through user name and password credentials. See "Additional Prerequisites for SSL Server and Client Mutual Authentication" in [SSL Prerequisites](#).
- **Encryption** — Encrypts data sent between the client and database server to significantly reduce the likelihood that the data can be read if the connection between the client and server is compromised. Encryption works both ways, regardless of whether SSL Client Authentication is enabled. See "Required Prerequisites for SSL Server Authentication and SSL encryption" in [SSL Prerequisites](#) and [Configuring SSL](#).
- **Data integrity** — Verifies that data sent between the client and server has not been altered during transmission.

Note: For server authentication, HP Vertica supports using RSA encryption with [ephemeral Diffie-Hellman](#) (DH). DH is the key agreement protocol.

SSL Prerequisites

Before you implement SSL security, obtain the appropriate certificate signed by a certificate authority (CA) and private key files and then copy the certificate to your system. (See the [OpenSSL](#) documentation.) These files must be in Privacy-Enhanced Mail (PEM) format.

Prerequisites for SSL Server Authentication and SSL Encryption

Follow these steps to set up SSL authentication of the server by the clients, which is also required in order to provide encrypted communication between server and client.

1. On each server host in the cluster, copy the server certificate file (`server.crt`) and private key (`server.key`) to the HP Vertica catalog directory. (See [Distributing Certifications and Keys](#).)

The public key contained within the certificate and the corresponding private key allow the SSL connection to encrypt the data and ensure its integrity.

Note: The `server.key` file must have read and write permissions for the `dbadmin` user only. Do not provide any additional permissions or extend them to any other users. Under Linux, for example, file permissions would be `0600`.

2. If you are using Mutual SSL Authentication, then copy the `root.crt` file to each client so that

the client's can verify the server's certificate. If you are using vsql, copy the file to:
/home/dbadmin/.vsq1/.

This ability is not available for ODBC clients at this time.

The `root.crt` file contains either the server's certificate or the CA that issued the server certificate.

Note: If you do not perform this step, the SSL connection is set up and ensures message integrity and confidentiality via encryption; however, the client cannot authenticate the server and is, therefore, susceptible to problems where a fake server with the valid certificate file masquerades as the real server. If the `root.crt` is present but does not match the CA used to sign the certificate, the database will not start.

Optional Prerequisites for SSL Server and Client Mutual Authentication

Follow these additional steps to optionally configure authentication of clients by the server.

Setting up client authentication by the server is optional because the server can use alternative techniques, like database-level password authentication, to verify the client's identity. Follow these steps only if you want to have both server and client mutually authenticate themselves with SSL keys.

1. On each server host in the cluster, copy the `root.crt` file to the HP Vertica catalog directory. (See [Distributing Certifications and Keys](#).)

The `root.crt` file has the same name on the client and server. However, these files do not need to be identical. They would be identical only if the client and server certificates were used by the same root certificate authority (CA).

2. On each client, copy the client certificate file (`client.crt`) and private key (`client.key`) to the client. If you are using vsql, copy the files to: `/home/dbadmin/.vsq1/`.

If you are using either ODBC or JDBC, you can place the files anywhere on your system and provide the location in the connection string (ODBC/JDBC) or ODBCINI (ODBC only). See [Configuring SSL for ODBC Clients](#) and [Configuring SSL for JDBC Clients](#).

Note: If you're using ODBC, the private key file (`client.key`) must have read and write permissions for the dbadmin user only. Do not provide any additional permissions or extend them to any other users. Under Linux, for example, file permissions would be 0600.

Generating Certifications and Keys

For testing purposes, you can create and use simple self-signed certificates. For production, you need to use certificates signed by a certificate authority (CA) so the client can verify the server's identity.

This section illustrates how to create certificate authority (CA) keys and self-signed certificates for testing purposes. It uses the CA private keys to sign "normal" certificates and to generate the server's and client's private key files. For detailed information about creating signed certificates, refer to the [OpenSSL](#) documentation.

The server and client keys can be rooted in different CAs.

1. **Create the CA private key:**

```
$>openssl genrsa -des3 -out rootkey.pem
```

The output file name can vary.

2. **Create the CA public certificate:**

```
$>openssl req -new -x509 -key rootkey.pem -out root.crt
```

The output file name can vary.

Important: The following is an example of the certificate's contents. When you create a certificate, there must be one unique name (a Distinguished Name (DN)), which is different for each certificate that you create. The examples in this procedure use the Organizational Unit Name for the DN.

```
Country Name (2 letter code) [GB]:USState or Province Name (full name) [Berkshire]:Massachusetts  
Locality Name (e.g., city) [Newbury]:Billerica  
Organization Name (e.g., company) [My Company Ltd]:HP Vertica  
Organizational Unit Name (e.g., section) []:Support_CA  
Common Name (e.g., your name or your server's hostname) []:myhost  
Email Address []:myhost@vertica.com
```

3. **Create the server's private key file:**

```
$>openssl genrsa -out server.key
```

Note that HP Vertica supports only unencrypted key files, so there is no `-des3` argument.

4. **Create the server certificate request:**

```
$>openssl req -new -out reqout.txt -key server.key
```

This step was not required for the CA because CA certificates are self-signed.

You are prompted to enter information that is incorporated into your certificate request. In this example, the `Organizational Unit Name` contains the unique DN (`Support_server`):

```
Country Name (2 letter code) [GB]:USState or Province Name (full name) [Berkshire]:Massachusetts
Locality Name (e.g., city) [Newbury]:Billerica
Organization Name (e.g., company) [My Company Ltd]:HP Vertica
Organizational Unit Name (e.g., section) []:Support_server
Common Name (e.g., your name or your server's hostname) []:myhost
Email Address []:myhost@vertica.com
```

5. Use the CA private key file to sign the server's certificate:

```
$>openssl x509 -req -in reqout.txt -days 3650 -sha1 -CAcreateserial -CA root.crt -CAkey rootkey.pem -out server.crt
```

6. Create the client's private key file:

```
$>openssl genrsa -out client.key
```

HP Vertica supports only unencrypted key files, so there is no `-des3` argument.

7. Create the client certificate request:

```
$>openssl req -new -out reqout.txt -key client.key
```

This step was not required for the CA because CA certificates are self-signed.

You are prompted to enter information that is incorporated into your certificate request. In this example, the `Organizational Unit Name` contains the unique DN (`Support_client`):

```
Country Name (2 letter code) [GB]:USState or Province Name (full name) [Berkshire]:Massachusetts
Locality Name (e.g., city) [Newbury]:Billerica
Organization Name (e.g., company) [My Company Ltd]:HP
Organizational Unit Name (e.g., section) []:Support_client
Common Name (e.g., your name or your server's hostname) []:myhost
Email Address []:myhost@vertica.com
```

8. Use the CA private key file to sign the client's certificate:

```
$>openssl x509 -req -in reqout.txt -days 3650 -sha1 -CAcreateserial -CA root.crt -CAkey rootkey.pem -out client.crt
```

JDBC Certificates

If you are using JDBC, perform the following steps after you have generated the key and self-signed certificate:

1. Convert the HP Vertica server certificate to a form that JAVA understands:

```
openssl x509 -in server.crt -out server.crt.der -outform der
```

2. Create a new truststore and imported the certificate into it:

```
keytool -keystore verticastore -alias verticasql -import -file server.crt.der
```

Generating Certifications and Keys for MC

A certificate signing request (CSR) is a block of encrypted text that you generate on the server on which the certificate will be used. You send the CSR to a certificate authority (CA) in order to apply for a digital identity certificate. The certificate authority uses the CSR to create your SSL certificate from information in your certificate; for example, organization name, common (domain) name, city, country, and so on.

MC uses a combination of OAuth (Open Authorization), Secure Socket Layer (SSL), and locally-encrypted passwords to secure HTTPS requests between a user's browser and MC, as well as between MC and the **agents**. Authentication occurs through MC and between agents within the cluster. Agents also authenticate and authorize jobs.

The MC configuration process sets up SSL automatically, but you must have the openssl package installed on your Linux environment first.

When you [connect to MC](#) through a client browser, HP Vertica assigns each HTTPS request a self-signed certificate, which includes a timestamp. To increase security and protect against password replay attacks, the timestamp is valid for several seconds only, after which it expires.

To avoid being blocked out of MC, synchronize time on the hosts in your HP Vertica cluster, as well as on the MC host if it resides on a dedicated server. To recover from loss or lack of synchronization, resync system time and the Network Time Protocol. See [Set Up Time Synchronization](#) in the Installation Guide. If you want to generate your own certificates and keys for MC, see [Generating Certifications and Keys for MC](#).

Signed Certificates

For production, you need to use certificates that are signed by a certificate authority. You can create and submit one now and [import the certificate into MC](#) when the certificate returns from the CA.

To generate a new CSR, enter the following command in a terminal window, like vsq:

```
openssl req -new -key /opt/vertica/config/keystore.key -out server.csr
```

When you press enter, you are prompted to enter information that will be incorporated into your certificate request. Some fields contain a default value, which you should change, and some you can leave blank, like password and optional company name. Enter ' .' to leave the field blank.

Important: The keystore.key value for the -key option creates private key for the keystore. If you generate a new key and import it using the Management Console interface, the MC process will not restart properly. You will have to restore the original keystore.jks file and [restart Management Console](#).

Here's an example of the information contained in the CSR, showing both the default and replacement values:

```
Country Name (2 letter code) [GB]:USState or Province Name (full name) [Berkshire]:Massachusetts
Locality Name (eg, city) [Newbury]: Billerica
Organization Name (eg, company) [My Company Ltd]:HP
Organizational Unit Name (eg, section) []:Information Management
Common Name (eg, your name or your server's hostname) []:console.vertica.com
Email Address []:madmin@vertica.com
```

The Common Name field is the fully qualified domain name of your server. The entry must be an exact match for what you type in your web browser, or you will receive a name mismatch error.

Self-Signed Certificates

If you want to test your new SSL implementation, you can self-sign a CSR using either a temporary certificate or your own internal CA, if one is available.

Note: A self-signed certificate will generate a browser-based error notifying you that the signing certificate authority is unknown and not trusted. For testing purposes, accept the risks and continue.

The following command generate a temporary certificate, which is good for 365 days:

```
openssl x509 -req -days 365 -in server.csr -signkey /opt/vertica/config/keystore.key -out server.crt
```

Here's an example of the command's output to the terminal window:

```
Signature oksubject=/C=US/ST=Massachusetts/L=Billerica/O=HP/OU=IT/
CN=console.vertica.com/emailAddress=madmin@vertica.com
Getting Private key
```

You can now [import the self-signed key](#), server.crt, into Management Console.

For additional information about certificates and keys, refer to the following external web sites:

Note: At the time of publication, the above links were valid. HP does not control this content, which could change between HP Vertica documentation releases.

See Also

- [How to Configure SSL](#)
- [Key and Certificate Management Tool](#)

Importing a New Certificate to MC

To generate a new certificate for Management Console, you must use the `keystore.key` file, which is located in `/opt/vconsole/config` on the server on which you installed MC. Any other generated key/certificate pair will cause MC to restart incorrectly. You will then have to restore the original `keystore.jks` file and [restart Management Console](#). See [Generating Certifications and Keys for Management Console](#).

To Import a New Certificate

1. [Connect to Management Console](#) and log in as an administrator.
2. On the Home page, click MC **Settings**.
3. In the button panel at left, click **SSL certificates**.
4. To the right of "Upload a new SSL certificate" click **Browse** to import the new key.
5. Click **Apply**.
6. [Restart Management Console](#).

Distributing Certifications and Keys

Once you have created the prerequisite certifications and keys for one host, you can easily distribute them cluster-wide by using the Administration Tools. Client files cannot be distributed through Administration Tools.

To distribute certifications and keys to all hosts in a cluster:

1. Log on to a host that contains the certifications and keys you want to distribute and start the Administration Tools.

See [Using the Administration Tools](#) for information about accessing the Administration Tools.

2. On the **Main Menu** in the Administration Tools, select **Configuration Menu**, and click **OK**.
3. On the **Configuration Menu**, select **Distribute Config Files**, and click **OK**.

4. Select **SSL Keys** and click **OK**.
5. Select the database where you want to distribute the files and click **OK**.
6. Fill in the fields with the directory `/home/dbadmin/.vsq1/` using the `root.crt`, `server.crt` and `server.key` files to distribute the files.
7. [Configure SSL](#).

Configuring SSL

Configure SSL for each server in the cluster.

To Enable SSL:

1. Ensure that you have performed the steps listed in [SSL Prerequisites](#) minimally for server authentication and encryption, and optionally for mutual authentication.
2. Set the `EnableSSL` parameter to `1`. By default, `EnableSSL` is set to `0` (disabled).

```
=> SELECT SET_CONFIG_PARAMETER('EnableSSL', '1');
```

Note: HP Vertica fails to start if SSL has been enabled and the server certificate files (`server.crt`, `server.key`) are not in the expected location.

3. [Restart the database](#).
4. If you are using either ODBC or JDBC, configure SSL for the appropriate client:
 - [Configuring SSL for ODBC Clients](#)
 - [Configuring SSL for JDBC Clients](#)

vsq1 automatically attempts to make connections using SSL. If a connection fails, **vsq1** attempts to make a second connection over clear text.

See Also

- [Configuration Parameters](#)

Configuring SSL for ODBC Clients

Configuring SSL for ODBC clients requires that you set the `SSLMode` parameter. If you want to configure optional SSL client authentication, you also need to configure the `SSLKeyFile` and `SSLCertFile` parameters.

The method you use to configure the DSN depends on the type of client operating system you are using:

- Linux and UNIX — Enter the parameters in the `odbc.ini` file. See [Creating an ODBC DSN for Linux, Solaris, AIX, and HP-UX Clients](#).
- Microsoft Windows — Enter the parameters in the Windows Registry. See [Creating an ODBC DSN for Windows Clients](#).

SSLMode Parameter

Set the `SSLMode` parameter to one of the following for the DSN:

- `require` — Requires the server to use SSL. If the server cannot provide an encrypted channel, the connection fails.
- `prefer` (the default) — Prefers the server to use SSL. If the server does not offer an encrypted channel, the client requests one. The first connection to the database tries to use SSL. If that fails, a second connection is attempted over a clear channel.
- `allow` — The first connection to the database tries to use a clear channel. If that fails, a second connection is attempted over SSL.
- `disable` — Never connects to the server using SSL. This setting is typically used for troubleshooting.

SSLKeyFile Parameter

To configure optional SSL client authentication, set the `SSLKeyFile` parameter to the file path and name of the client's private key. This key can reside anywhere on the client.

SSLCertFile Parameter

To configure optional SSL client authentication, set the `SSLCertFile` parameter to the file path and name of the client's public certificate. This file can reside anywhere on the client.

Configuring SSL for JDBC Clients

To Configure JDBC:

1. Enable the driver for SSL.
2. Configure troubleshooting if desired.

To Enable the Driver for SSL

For JDBC, the driver must be enabled for SSL. Use a connection parameter when connecting to the database to force a connection using SSL. You can specify a connection parameter within a connection URL or by using an additional properties object parameter to `DriverManager.getConnection`.

- Using a Connection URL

The following example forces a connection using SSL by setting the `ssl` connection parameter to `true`:

```
String url = "jdbc:vertica://VerticaHost://DatabaseName?user=username" +  
&password=password&ssl=true";  
Connection conn = DriverManager.getConnection (url);
```

Note: If the server is not SSL enabled, the connection fails. This differs from `vsq`, which can try an unencrypted connection.

- Using an Additional Properties Object Parameter

The following code fragment forces a connection using SSL by establishing an `ssl` connection property:

```
String url = "jdbc:vertica://VerticaHost/DatabaseName"; Properties props = new Propert  
ies();  
props.setProperty("user", "username"); props.setProperty("password", "password");  
props.setProperty("ssl", "true");  
Connection conn = new Connection(url, props);
```

Note: For compatibility with future versions, specify a value, even though the `ssl` property does not require that a value be associated with it. Specifying a `ssl` property, even without a value of `"true,"` automatically forces a connection using SSL.

To Configure Troubleshooting

To enable troubleshooting, configure the keystore file that contains trusted certificate authority (CA) certificates:

```
-Djavax.net.debug=ssl-Djavax.net.ssl.trustStore=<keystore file>
```

In the above command:

- Configuring `-Djavax.net.debug=ssl` is optional.
- The keystore file is the same keystore that was updated as part of [Generating Certifications and Keys](#) (JDBC Certificates). Normally, the keystore file is `$HOME/.keystore`. The `keytool` utility takes `server.crt.der` and places it in the keystore.

For details, see "Customizing the Default Key and Trust Stores, Store Types, and Store Passwords" on the java.sun.com web site.

Requiring SSL for Client Connections

You can require clients to use SSL when connecting to HP Vertica by creating a client authentication record for them that has a `connection_type` of `hostssl`. You can choose to limit specific users to only connecting using SSL (useful for specific clients that you know are connecting through an insecure network connection) or require all clients to use SSL.

See [Implementing Client Authentication](#) for more information about creating client authentication records.

Managing Users and Privileges

Database users should have access to only the database resources they need to perform their tasks. For example, most users should be able to read data but not modify or insert new data, while other users might need more permissive access, such as the right to create and modify schemas, tables, and views, as well as rebalance nodes on a cluster and start or stop a database. It is also possible to allow certain users to grant other users access to the appropriate database resources.

Client authentication controls what database objects users can access and change in the database. To prevent unauthorized access, a **superuser** limits access to what is needed, granting privileges directly to users or to roles through a series of GRANT statements. Roles can then be granted to users, as well as to other roles.

A Management Console administrator can also grant MC users access to one or more HP Vertica databases through the MC interface. See [About MC Users](#) and [About MC Privileges and Roles](#) for details.

This section introduces the [privilege role model](#) in HP Vertica and describes how to create and manage users.

See Also

- [About Database Privileges](#)
- [About Database Roles](#)
- [GRANT Statements](#)
- [REVOKE Statements](#)

About Database Users

Every HP Vertica database has one or more users. When users connect to a database, they must log on with valid credentials (username and password) that a **superuser** defined in the database.

Database users own the objects they create in a database, such as tables, procedures, and storage locations.

Note: By default, users have the right to [create temporary tables](#) in a database.

See Also

- [Creating a Database User](#)
- [CREATE USER](#)
- [About MC Users](#)

Types of Database Users

In an HP Vertica database, there are three types of users:

- Database administrator (DBADMIN)
- Object owner
- Everyone else (PUBLIC)

Note: External to an HP Vertica database, an MC administrator can create users through the Management Console and grant them database access. See [About MC Users](#) for details.

DBADMIN User

When you create a new database, a single database administrator account, DBADMIN, is automatically created along with the PUBLIC role. This database **superuser** bypasses all permission checks and has the authority to perform all database operations, such as bypassing all [GRANT/REVOKE](#) authorizations and any user granted the [PSEUDOSUPERUSER](#) role.

Note: Although the dbadmin user has the same name as the Linux database administrator account, do not confuse the concept of a database superuser with Linux superuser (root) privilege; they are not the same. A database superuser cannot have Linux superuser privileges.

The DBADMIN user can start and stop a database without a database password. To connect to the database, a password is required.

See Also

- [DBADMIN Role](#)
- [PSEUDOSUPERUSER Role](#)
- [PUBLIC Role](#)

Object Owner

An object owner is the user who creates a particular database object and can perform any operation on that object. By default, only an owner (or a **superuser**) can act on a database object. In order to allow other users to use an object, the owner or superuser must grant privileges to those users using one of the [GRANT statements](#).

Note: Object owners are [PUBLIC users](#) for objects that other users own.

See [About Database Privileges](#) for more information.

***PUBLIC* User**

All non-DBA (superuser) or object owners are PUBLIC users.

Note: Object owners are PUBLIC users for objects that other users own.

Newly-created users do not have access to schema PUBLIC by default. Make sure to GRANT USAGE ON SCHEMA PUBLIC to all users you create.

See Also

- [PUBLIC Role](#)

Creating a Database User

This procedure describes how to create a new user on the database.

1. From **vsqI**, connect to the database as a superuser.
2. Issue the [CREATE USER](#) statement with optional parameters.
3. Run a series of [GRANT statements](#) to grant the new user privileges.

Notes

- Newly-created users do not have access to schema PUBLIC by default. Make sure to GRANT USAGE ON SCHEMA PUBLIC to all users you create
- By default, database users have the right to create temporary tables in the database.
- If you plan to [create users on Management Console](#), the database user account needs to exist before you can associate an MC user with the database.
- You can change information about a user, such as his or her password, by using the [ALTER USER](#) statement. If you want to configure a user to not have any password authentication, you can set the empty password " " in CREATE or ALTER USER statements, or omit the IDENTIFIED BY parameter in CREATE USER.

Example

The following series of commands add user Fred to a database with password 'password'. The second command grants USAGE privileges to Fred on the public schema:

```
=> CREATE USER Fred IDENTIFIED BY 'password';=> GRANT USAGE ON SCHEMA PUBLIC to Fred;
```

User names created with double-quotes are case sensitive. For example:

```
=> CREATE USER "FrEd1";
```

In the above example, the logon name must be an exact match. If the user name was created without double-quotes (for example, FRED1), then the user can log on as FRED1, FrEd1, fred1, and so on.

ALTER USER and [DROP USER](#) syntax is not case sensitive.

See Also

- [Granting and Revoking Privileges](#)
- [Granting Access to Database Roles](#)
- [Creating an MC User](#)

Locking/unlocking a user's Database Access

A **superuser** can manually lock an existing database user's account with the [ALTER USER](#) statement. For example, the following command prevents user Fred from logging in to the database:

```
=> ALTER USER Fred ACCOUNT LOCK;
=> \c - Fred
FATAL 4974: The user account "Fred" is locked
HINT: Please contact the database administrator
```

To grant Fred database access, use UNLOCK syntax with the ALTER USER command:

```
=> ALTER USER Fred ACCOUNT UNLOCK;
=> \c - Fred
You are now connected as user "Fred".
```

Using CREATE USER to lock an account

Although not as common, you can create a new user with a locked account; for example, you might want to set up an account for a user who doesn't need immediate database access, as in the case of an employee who will join the company at a future date.

```
=> CREATE USER Bob ACCOUNT UNLOCK;
CREATE USER
```

CREATE USER also supports UNLOCK syntax; however, UNLOCK is the default, so you don't need to specify the keyword when you create a new user to whom you want to grant immediate database access.

Locking an account automatically

Instead of manually locking an account, a superuser can automate account locking by setting a maximum number of failed login attempts through the CREATE PROFILE statement. See Profiles.

Changing a user's Password

A **superuser** can change another user's database account, including reset a password, with the the [ALTER USER](#) statement.

Making changes to a database user account with does not affect current sessions.

```
=> ALTER USER Fred IDENTIFIED BY 'newpassword';
```

In the above command, Fred's password is now *newpassword*.

Note: Non-DBA users can change their own passwords using the IDENTIFIED BY 'new-password' option along with the REPLACE 'old-password' clause. See [ALTER USER](#) for details.

Changing a user's MC Password

On MC, users with ADMIN or IT privileges can reset a user's non-LDAP password from the MC interface.

Non-LDAP passwords on MC are for MC access only and are not related to a user's logon credentials on the HP Vertica database.

1. Sign in to Management Console and navigate to **MC Settings > User management**.
2. Click to select the user to modify and click **Edit**.
3. Click **Edit password** and enter the new password twice.
4. Click **OK** and then click **Save**.

About MC Users

Unlike database users, which you create on the HP Vertica database and then grant privileges and roles through SQL statements, you create MC users on the Management Console interface. MC users are external to the database; their information is stored on an internal database on the MC application/web server, and their access to both MC and to **MC-managed databases** is controlled by groups of privileges (also referred to as access levels). MC users are not system (Linux) users; they are entries in the MC internal database.

Permission Group Types

There are two types of permission groups on MC, those that apply to MC configuration and those that apply to database access:

- [MC configuration](#) privileges are made up of roles that control what users can configure on the MC, such as modify MC settings, create/import HP Vertica databases, restart MC, create an HP Vertica cluster through the MC interface, and create and manage MC users.
- [MC database](#) privileges are made up of roles that control what users can see or do on an MC-managed HP Vertica database, such as view the database cluster state, query and session activity, monitor database messages and read log files, replace cluster nodes, and stop databases.

If you are using MC, you might want to allow one or more users in your organization to configure and manage MC, and you might want other users to have database access only. You can meet these requirements by creating MC users and granting them a role from each privileges group. See [Creating an MC User](#) for details.

MC User Types

There are four types of role-based users on MC:

- The default superuser administrator (Linux account) who gets created when you install and configure MC and oversees the entire MC. See [SUPER Role \(mc\)](#).
- Users who can configure all aspects of MC and control all MC-managed databases. See [ADMIN Role \(mc\)](#).
- Users who can configure some aspects of MC and monitor all MC-managed databases. See [IT Role \(mc\)](#).
- Users who cannot configure MC and have access to one or more MC-managed databases only. See [NONE Role \(mc\)](#).

You create users and grant them privileges (through roles) on the **MC Settings** page by selecting **User management**, to add users who will be authenticated against the MC or **Authentication**, to authenticate MC users through your organization's LDAP repository.

Creating Users and Choosing an Authentication Method

You create users and grant them privileges (through roles) on the **MC Settings** page, where you can also choose how to authenticate their access to MC; for example:

- To add users who will be authenticated against the MC, click **User Management**
- To add users who will be authenticated through your organization's LDAP repository, click **Authentication**

MC supports only one method for authentication, so if you choose MC, all MC users will be authenticated using their MC login credentials.

Default MC Users

The **MC super** account is the only default user. The super or another MC administrator must create all other MC users.

See Also

- [Management Console](#)
- [About MC Privileges and Roles](#)
- [Granting Database Access to MC Users](#)
- [Mapping an MC User to a Database user's Privileges](#)

Creating an MC User

MC provides two authentication schemes for MC users: LDAP or MC (internal). Which method you choose will be the method MC uses to authenticate *all* MC users. It is not possible to authenticate some MC users against LDAP and other MC users against credentials in the database through MC.

- **MC (internal) authentication.** Internal user authorization is specific to the MC itself, where you create a user with a username and password combination. This method stores MC user information in an internal database on the MC application/web server, and encrypts passwords. Note that these MC users are not system (Linux) users; they are entries in the MC's internal database.
- **LDAP authentication.** All MC users—except for the **MC super** administrator, which is a Linux account—will be authenticated based on search criteria against your organization's LDAP repository. MC uses information from LDAP for authentication purposes only and does not modify LDAP information. Also, MC does not store LDAP passwords but passes them to the LDAP server for authentication.

Instructions for creating new MC users are in this topic.

- If you chose MC authentication, follow the instructions under **Create a new MC-authenticated user**.
- If you chose LDAP authentication, follow the instructions under **Create a new user from LDAP**.

See [About MC Users](#) and [Configuring LDAP Authentication](#) for more information.

Prerequisites

Before you create an MC user, you already:

- Created a database directly on the server or through the MC interface, or you imported an existing database cluster into the MC interface. See [Managing Database Clusters on MC](#).
- Created a database user account (source user) on the server, which has the privileges and/or roles you want to map to the new (target) MC user. See [Creating a Database User](#).
- Know what MC privileges you want to grant the new MC user. See [About MC Privileges and Roles](#).
- Are familiar with the concept of [mapping MC users to database users](#).

If you have not yet met the first two above prerequisites, you can still create new MC users; you just won't be able to map them to a database until after the database and target database user exist. To grant MC users database access later, see [Granting Database Access to MC Users](#).

Create a New MC-authenticated User

1. Sign in to Management Console as an administrator and navigate to **MC Settings > User management**.
2. Click **Add**.
3. Enter the MC username.

Note: It is not necessary to give the MC user the exact same name as the database user account you'll map the MC user to in Step 7. What matters is that the source database user has privileges and/or roles similar to the database role you want to grant the MC user. The most likely scenario is that you will map multiple MC users to a single database user account. See [MC Database Privileges](#) and [Mapping an MC User to a Database user's Privileges](#) for more information.

4. Let MC generate a password or create one by clicking **Edit password**. If LDAP has been configured, the MC password field will not appear.
5. Optionally enter the user's e-mail address.

6. Select an **MC configuration permissions** level. See [MC Configuration Privileges](#).
7. Next to the **DB access levels section**, click **Add** to grant this user database permissions. If you want to grant access later, proceed to Step 8. If you want to grant database access now, provide the following information:
 - i. **Choose a database.** Select a database from the list of MC-discovered (databases that were created on or imported into the MC interface).
 - ii. **Database username.** Enter an existing database user name or, if the database is running, click the ellipses [...] to browse for a list of database users, and select a name from the list.
 - iii. **Database password.** Enter the password to the database user account (not this username's password).
 - iv. **Restricted access.** Chose a database level ([ADMIN](#), [IT](#), or [USER](#)) for this user.
 - v. Click **OK** to close the **Add permissions** dialog box.

See [Mapping an MC User to a Database user's Privileges](#) for additional information about associating the two user accounts.

1. Leave the user's **Status** as enabled (the default). If you need to prevent this user from accessing MC, select disabled.
2. Click **Add User** to finish.

Create a New LDAP-authenticated User

When you add a user from LDAP on the MC interface, options on the **Add a new user** dialog box are slightly different from when you create users without LDAP authentication. Because passwords are store externally (LDAP server) the password field does not appear. An MC administrator can override the default LDAP search string if the user is found in another branch of the tree. The **Add user** field is pre-populated with the default search path entered when LDAP was configured.

1. Sign in to Management Console and navigate to **MC Settings > User management**.
2. Click **Add** and provide the following information:
 - a. LDAP user name.
 - b. LDAP search string.
 - c. User attribute, and click **Verify user**.
 - d. User's email address.
 - e. MC configuration role. NONE is the default. See [MC Configuration Privileges](#) for details.

- f. Database access level. See [MC Database Privileges](#) for details.
 - g. Accept or change the default user's **Status** (enabled).
3. Click **Add user**.

If you encounter issues when creating new users from LDAP, you'll need to contact your organization's IT department.

How MC Validates New Users

After you click OK to close the Add permissions dialog box, MC tries to validate the database username and password entered against the selected MC-managed database or against your organization's LDAP directory. If the credentials are found to be invalid, you are asked to re-enter them.

If the database is not available at the time you create the new user, MC saves the username/password and prompts for validation when the user accesses the Database and Clusters page later.

See Also

- [Configuring MC](#)
- [About MC Users](#)
- [About MC Privileges and Roles](#)
- [Granting Database Access to MC Users](#)
- [Creating a Database User](#)
- [Mapping an MC User to a Database user's Privileges](#)
- [Adding Multiple Users to MC-managed Databases](#)

Managing MC Users

You manage MC users through the following pages on the Management Console interface:

- **MC Settings > User management**
- **MC Settings > Resource access**

Who Manages Users

The MC superuser administrator ([SUPER Role \(mc\)](#)) and users granted [ADMIN Role \(mc\)](#) manage all aspects of users, including their access to MC and to MC-managed databases.

Users granted [IT Role \(mc\)](#) can enable and disable user accounts.

See [About MC Users](#) and [About MC Privileges and Roles](#) for more information.

Editing an MC user's information follows the same steps as [creating a new user](#), except the user's information will be pre-populated, which you then edit and save.

The only user account you cannot alter or remove from the MC interface is the MC super account.

What Kind of User Information You Can Manage

You can change the following user properties:

- MC password
- Email address. This field is optional; if the user is authenticated against LDAP, the email field is pre-populated with that user's email address if one exists.
- [MC Configuration Privileges](#) role
- [MC Database Privileges](#) role

You can also change a user's status (enable/disable access to MC) and delete users.

About User Names

After you create and save a user, you cannot change that user's MC user name, but you can delete the user account and create a new user account under a new name. The only thing you lose by deleting a user account is its audit activity, but MC immediately resumes logging activity under the user's new account.

See Also

- [About MC Users](#)
- [About MC Privileges and Roles](#)

About Database Privileges

When a database object is created, such as a schema, table, or view, that object is assigned an owner—the person who executed the CREATE statement. By default, database administrators (**superusers**) or object owners are the only users who can do anything with the object.

In order to allow other users to use an object, or remove a user's right to use an object, the authorized user must grant another user privileges on the object.

Privileges are granted (or revoked) through a collection of GRANT/REVOKE statements that assign the privilege—a type of permission that lets users perform an action on a database object, such as:

- Create a schema
- Create a table (in a schema)
- Create a view
- View (select) data
- Insert, update, or delete table data
- Drop tables or schemas
- Run procedures

Before HP Vertica executes a statement, it determines if the requesting user has the necessary privileges to perform the operation.

For more information about the privileges associated with these resources, see [Privileges That Can Be Granted on Objects](#).

Note: HP Vertica logs information about each grant (grantor, grantee, privilege, and so on) in the `V_CATALOG.GRANTS` system table.

See Also

- [GRANT Statements](#)
- [REVOKE Statements](#)

Default Privileges for All Users

To set the minimum level of privilege for all users, HP Vertica has the special [PUBLIC Role](#), which it grants to each user automatically. This role is automatically enabled, but the database administrator or a **superuser** can also grant higher privileges to users separately using GRANT statements.

The following topics discuss those higher privileges.

Default Privileges for MC Users

Privileges on Management Console (MC) are managed through roles, which determine a user's access to MC and to MC-managed HP Vertica databases through the MC interface. MC privileges do not alter or override HP Vertica privileges or roles. See [About MC Privileges and Roles](#) for details.

Privileges Required for Common Database Operations

This topic lists the required privileges for database objects in HP Vertica.

Unless otherwise noted, **superusers** can perform all of the operations shown in the following tables without any additional privilege requirements. Object owners have the necessary rights to perform operations on their own objects, by default.

Schemas

The PUBLIC schema is present in any newly-created HP Vertica database, and newly-created users have only USAGE privilege on PUBLIC. A database superuser must explicitly grant new users CREATE privileges, as well as grant them individual object privileges so the new users can create or look up objects in the PUBLIC schema.

| Operation | Required Privileges |
|-------------------------------------|------------------------------|
| CREATE SCHEMA | CREATE privilege on database |
| DROP SCHEMA | Schema owner |
| ALTER SCHEMA RENAME | CREATE privilege on database |

Tables

| Operation | Required Privileges |
|------------------------------|--|
| CREATE TABLE | CREATE privilege on schema Note: Referencing sequences in the CREATE TABLE statement requires the following privileges: <ul style="list-style-type: none">• SELECT privilege on sequence object• USAGE privilege on sequence schema |
| DROP TABLE | USAGE privilege on the schema that contains the table or schema owner |

| Operation | Required Privileges |
|--|---|
| <code>TRUNCATE TABLE</code> | USAGE privilege on the schema that contains the table or schema owner |
| <code>ALTER TABLE ADD/DROP/RENAME/ALTER-TYPE COLUMN</code> | USAGE privilege on the schema that contains the table |
| <code>ALTER TABLE ADD/DROP CONSTRAINT</code> | USAGE privilege on the schema that contains the table |
| <code>ALTER TABLE PARTITION (REORGANIZE)</code> | USAGE privilege on the schema that contains the table |
| <code>ALTER TABLE RENAME</code> | USAGE and CREATE privilege on the schema that contains the table |
| <code>ALTER TABLE SET SCHEMA</code> | <ul style="list-style-type: none"> • CREATE privilege on new schema • USAGE privilege on the old schema |
| <code>SELECT</code> | <ul style="list-style-type: none"> • SELECT privilege on table • USAGE privilege on schema that contains the table |
| <code>INSERT</code> | <ul style="list-style-type: none"> • INSERT privilege on table • USAGE privilege on schema that contains the table |
| <code>DELETE</code> | <ul style="list-style-type: none"> • DELETE privilege on table • USAGE privilege on schema that contains the table • SELECT privilege on the referenced table when executing a DELETE statement that references table column values in a WHERE or SET clause |
| <code>UPDATE</code> | <ul style="list-style-type: none"> • UPDATE privilege on table • USAGE privilege on schema that contains the table • SELECT privilege on the table when executing an UPDATE statement that references table column values in a WHERE or SET clause |
| REFERENCES | <ul style="list-style-type: none"> • REFERENCES privilege on table to create foreign key constraints that reference this table • USAGE privileges on schema that contains the constrained table and the source of the foreign k |
| <code>ANALYZE_STATISTICS()</code> | <ul style="list-style-type: none"> • INSERT/UPDATE/DELETE privilege on table • USAGE privilege on schema that contains the table |

| Operation | Required Privileges |
|----------------------------------|--|
| <code>ANALYZE_HISTOGRAM()</code> | <ul style="list-style-type: none"> • INSERT/UPDATE/DELETE privilege on table • USAGE privilege on schema that contains the table |
| <code>DROP_STATISTICS()</code> | <ul style="list-style-type: none"> • INSERT/UPDATE/DELETE privilege on table • USAGE privilege on schema that contains the table |
| <code>DROP_PARTITION()</code> | USAGE privilege on schema that contains the table |
| <code>MERGE_PARTITIONS()</code> | USAGE privilege on schema that contains the table |

Views

| Operation | Required Privileges |
|-----------------------------------|--|
| <code>CREATE VIEW</code> | <ul style="list-style-type: none"> • CREATE privilege on the schema to contain a view • SELECT privileges on base objects (tables/views) • USAGE privileges on schema that contains the base objects |
| <code>DROP VIEW</code> | USAGE privilege on schema that contains the view or schema owner |
| <code>SELECT ... FROM VIEW</code> | <ul style="list-style-type: none"> • SELECT privilege on view • USAGE privilege on the schema that contains the view <p>Note: Privileges required on base objects for view owner must be directly granted, not through roles:</p> <ul style="list-style-type: none"> • View owner must have <code>SELECT ... WITH GRANT OPTION</code> privileges on the view's base tables or views if non-owner runs a <code>SELECT</code> query on the view. This privilege must be directly granted to the owner, not through a role. • View owner must have <code>SELECT</code> privilege directly granted (not through a role) on a view's base objects (table or view) if owner runs a <code>SELECT</code> query on the view. |

Projections

| Operation | Required Privileges |
|-------------------------|--|
| CREATE PROJECTION | <ul style="list-style-type: none"> • SELECT privilege on base tables • USAGE privilege on schema that contains base tables or schema owner • CREATE privilege on schema to contain the projection <p>Note: If a projection is implicitly created with the table, no additional privilege is needed other than privileges for table creation.</p> |
| AUTO/DELAYED PROJECTION | On projections created during INSERT..SELECT or COPY operations: <ul style="list-style-type: none"> • SELECT privilege on base tables • USAGE privilege on schema that contains base tables |
| ALTER PROJECTION RENAME | USAGE and CREATE privilege on schema that contains the projection |
| DROP PROJECTION | USAGE privilege on schema that contains the projection or schema owner |

External Procedures

| Operation | Required Privileges |
|------------------|---|
| CREATE PROCEDURE | Superuser |
| DROP PROCEDURE | Superuser |
| EXECUTE | <ul style="list-style-type: none"> • EXECUTE privilege on procedure • USAGE privilege on schema that contains the procedure |

Libraries

| Operation | Required Privileges |
|----------------|---------------------|
| CREATE LIBRARY | Superuser |
| DROP LIBRARY | Superuser |

User-Defined Functions

The following abbreviations are used in the UDF table:

- UDF = Scalar
- UDT = Transform
- UDAnF= Analytic
- UDAF = Aggregate

| Operation | Required Privileges |
|--|--|
| CREATE FUNCTION (SQL) CREATE FUNCTION (UDF) CREATE TRANSFORM FUNCTION (UDF) CREATE ANALYTIC FUNCTION (UDAnF) CREATE AGGREGATE FUNCTION (UDAF) | <ul style="list-style-type: none"> • CREATE privilege on schema to contain the function • USAGE privilege on base library (if applicable) |
| DROP FUNCTION DROP TRANSFORM FUNCTION DROP ANALYTIC FUNCTION DROP AGGREGATE FUNCTION | <ul style="list-style-type: none"> • Superuser or function owner • USAGE privilege on schema that contains the function |
| ALTER FUNCTION RENAME TO | USAGE and CREATE privilege on schema that contains the function |
| ALTER FUNCTION SET SCHEMA | <ul style="list-style-type: none"> • USAGE privilege on schema that currently contains the function (old schema) • CREATE privilege on the schema to which the function will be moved (new schema) |
| EXECUTE (SQL/UDF/UDT/ UDAF/UDAnF) function | <ul style="list-style-type: none"> • EXECUTE privilege on function • USAGE privilege on schema that contains the function |

Sequences

| Operation | Required Privileges |
|---------------------------------|--|
| CREATE SEQUENCE | CREATE privilege on schema to contain the sequence Note: Referencing sequence in the CREATE TABLE statement requires SELECT privilege on sequence object and USAGE privilege on sequence schema. |

| Operation | Required Privileges |
|--|---|
| <code>CREATE TABLE</code> with <code>SEQUENCE</code> | <ul style="list-style-type: none"> • <code>SELECT</code> privilege on sequence • <code>USAGE</code> privilege on sequence schema |
| <code>DROP SEQUENCE</code> | <code>USAGE</code> privilege on schema containing the sequence or schema owner |
| <code>ALTER SEQUENCE RENAME TO</code> | <code>USAGE</code> and <code>CREATE</code> privileges on schema |
| <code>ALTER SEQUENCE SET SCHEMA</code> | <ul style="list-style-type: none"> • <code>USAGE</code> privilege on the schema that currently contains the sequence (old schema) • <code>CREATE</code> privilege on new schema to contain the sequence |
| <code>CURRVAL()</code> / <code>NEXTVAL()</code> | <ul style="list-style-type: none"> • <code>SELECT</code> privilege on sequence • <code>USAGE</code> privilege on sequence schema |

Resource Pools

| Operation | Required Privileges |
|--|---|
| <code>CREATE RESOURCE POOL</code> | Superuser |
| <code>ALTER RESOURCE POOL</code> | <p>Superuser on the resource pool to alter:</p> <ul style="list-style-type: none"> • <code>MAXMEMORYSIZE</code> • <code>PRIORITY</code> • <code>QUEUETIMEOUT</code> <p><code>UPDATE</code> privilege on the resource pool to alter:</p> <ul style="list-style-type: none"> • <code>PLANNEDCONCURRENCY</code> • <code>SINGLEINITIATOR</code> • <code>MAXCONCURRENCY</code> |
| <code>SET SESSION RESOURCE_POOL</code> | <ul style="list-style-type: none"> • <code>USAGE</code> privilege on the resource pool • Users can only change their own resource pool setting using <code>ALTER USER</code> syntax |
| <code>DROP RESOURCE POOL</code> | Superuser |

Users/Profiles/Roles

| Operation | Required Privileges |
|--|---------------------|
| CREATE USER CREATE PROFILE CREATE ROLE | Superuser |
| ALTER USER ALTER PROFILE ALTER ROLE RENAME | Superuser |
| DROP USER DROP PROFILE DROP ROLE | Superuser |

Object Visibility

You can use one or a combination of vsq! [\d \[pattern\]](#) meta commands and [SQL system tables](#) to view objects on which you have privileges to view.

- Use [\dn \[pattern\]](#) to view schema names and owners
- Use [\dt \[pattern\]](#) to view all tables in the database, as well as the system table [V_CATALOG.TABLES](#)
- Use [\dj \[pattern\]](#) to view projections showing the schema, projection name, owner, and node, as well as the system table [V_CATALOG.PROJECTIONS](#)

| Operation | Required Privileges |
|----------------|---|
| Look up schema | At least one privilege on schema that contains the object |

| Operation | Required Privileges |
|--|--|
| Look up Object in Schema or in System Tables | USAGE privilege on schema At least one privilege on any of the following objects: TABLE VIEW FUNCTION PROCEDURE SEQUENCE |
| Look up Projection | At least one privilege on all base tables USAGE privilege on schema of all base table |
| Look up resource pool | SELECT privilege on the resource pool |
| Existence of object | USAGE privilege on the schema that contains the object |

I/O Operations

| Operation | Required Privileges |
|-------------------|---------------------|
| CONNECTDISCONNECT | None |

| Operation | Required Privileges |
|-----------------------------------|--|
| <code>EXPORT TO HP Vertica</code> | <ul style="list-style-type: none"> • SELECT privileges on the source table • USAGE privilege on source table schema • INSERT privileges for the destination table in target database • USAGE privilege on destination table schema |
| <code>COPY FROM HP Vertica</code> | <ul style="list-style-type: none"> • SELECT privileges on the source table • USAGE privilege on source table schema • INSERT privileges for the destination table in target database • USAGE privilege on destination table schema |
| <code>COPY FROM file</code> | Superuser |
| <code>COPY FROM STDIN</code> | <ul style="list-style-type: none"> • INSERT privilege on table • USAGE privilege on schema |
| <code>COPY LOCAL</code> | <ul style="list-style-type: none"> • INSERT privilege on table • USAGE privilege on schema |

Comments

| Operation | Required Privileges |
|--|---------------------------|
| COMMENT ON { is one of }: <ul style="list-style-type: none"> • AGGREGATE FUNCTION • ANALYTIC FUNCTION • COLUMN • CONSTRAINT • FUNCTION • LIBRARY • NODE • PROJECTION • SCHEMA • SEQUENCE • TABLE • TRANSFORM FUNCTION • VIEW | Object owner or superuser |

Transactions

| Operation | Required Privileges |
|-------------------|---------------------|
| COMMIT | None |
| ROLLBACK | None |
| RELEASE SAVEPOINT | None |
| SAVEPOINT | None |

Sessions

| Operation | Required Privileges |
|---|---------------------|
| SET { is one of }: <ul style="list-style-type: none"> • DATESTYLE • ESCAPE_STRING_WARNING • INTERVALSTYLE • LOCALE • ROLE • SEARCH_PATH • SESSION AUTOCOMMIT • SESSION CHARACTERISTICS • SESSION MEMORYCAP • SESSION RESOURCE POOL • SESSION RUNTIMECAP • SESSION TEMPSPACE • STANDARD_CONFORMING_STRINGS • TIMEZONE | None |
| SHOW { name ALL } | None |

Tuning Operations

| Operation | Required Privileges |
|-----------|---|
| PROFILE | Same privileges required to run the query being profiled |
| EXPLAIN | Same privileges required to run the query for which you use the EXPLAIN keyword |

Privileges That Can Be Granted on Objects

The following table provides an overview of privileges that can be granted on (or revoked from) database objects in HP Vertica:

| | CREATE | USAGE | EXE- CUTE | SELECT | INSERT | DELETE | UPDATE | REFER- ENCES | READ | WRITE |
|------------------|--------|-------|--------------|--------|--------|--------|--------|-----------------|------|-------|
| Database | Y | | | | | | | | | |
| Schema | Y | Y | | | | | | | | |
| Table | | | | Y | Y | Y | Y | Y | | |
| View | | | | Y | | | | | | |
| Sequence | | Y | | Y | | | Y | | | |
| Procedure | | | Y | | | | | | | |
| UDx | | | Y | | | | | | | |
| Library | | Y | | | | | | | | |
| Resource Pool | | Y | | Y | | | Y | | | |
| Storage Location | | | | | | | | | Y | Y |

See Also

- [GRANT Statements](#)
- [REVOKE Statements](#)

Database Privileges

Only a database **superuser** can create a database. In a new database, the **PUBLIC Role** is granted USAGE on the automatically-created PUBLIC schema. It is up to the superuser to grant further privileges to users and roles.

The only privilege a superuser can grant on the database itself is CREATE, which allows the user to create a new schema in the database. For details on granting and revoking privileges on a database, see the [GRANT \(Database\)](#) and [REVOKE \(Database\)](#) topics in the SQL Reference Manual.

| Privilege | Grantor | Description |
|-----------|-----------|-----------------------------------|
| CREATE | Superuser | Allows a user to create a schema. |

Schema Privileges

By default, only a **superuser** and the schema owner have privileges to create objects within a **schema**. Additionally, only the schema owner or a superuser can drop or alter a schema. See [DROP SCHEMA](#) and [ALTER SCHEMA](#).

All new users have only USAGE privilege on the PUBLIC schema, which is present in any newly-created HP Vertica database. A superuser must then explicitly grant these new users CREATE privileges, as well as grant them individual object privileges, so the new users can create or look up objects in the PUBLIC schema. Without USAGE privilege, objects in the schema cannot be used or altered, even by the object owner.

CREATE gives the schema owner or user WITH GRANT OPTION permission to create new objects in the schema, including renaming an object in the schema or moving an object into this schema.

Note: The schema owner is typically the user who creates the schema. However, a superuser can create a schema and assign ownership of the schema to a different user at creation.

All other access to the schema and its objects must be explicitly granted to users or roles by the superuser or schema owner. This prevents unauthorized users from accessing the schema and its objects. A user can be granted one of the following privileges through the GRANT statement:

| Privilege | Description |
|-----------|--|
| CREATE | Allows the user to create new objects within the schema. This includes the ability to create a new object, rename existing objects, and move objects into the schema from other schemas. |
| USAGE | Permission to select, access, alter, and drop objects in the schema. The user must also be granted access to the individual objects in order to alter them. For example, a user would need to be granted USAGE on the schema and SELECT on a table to be able to select data from a table. You receive an error message if you attempt to query a table that you have SELECT privileges on, but do not have USAGE privileges for the schema that contains the table. |

Schema Privileges and the Search Path

The search path determines to which schema unqualified objects in SQL statements belong.

When a user specifies an object name in a statement without supplying the schema in which the object exists (called an unqualified object name) HP Vertica has two different behaviors, depending on whether the object is being accessed or created.

| Creating an object | Accessing/altering an object |
|---|---|
| <p>When a user creates an object—such as table, view, sequence, procedure, function—with an unqualified name, HP Vertica tries to create the object in the current schema (the first schema in the schema search path), returning an error if the schema does not exist or if the user does not have CREATE privileges in that schema.</p> <p>Use the SHOW search_path command to view the current search path.</p> <pre>=> SHOW search_path; name setting -----+----- search_path "\$user", public, v_catalog, v_monitor, v_internal (1 row)</pre> <p>Note: The first schema in the search path is the current schema, and the \$user setting is a placeholder that resolves to the current user's name.</p> | <p>When a user accesses or alters an object with an unqualified name, those statements search through all schemas for a matching object, starting with the current schema, where:</p> <ul style="list-style-type: none"> • The object name in the schema matches the object name in the statement. • The user has USAGE privileges on the schema in order to access object in it. • The user has at least one privilege on the object. |

See Also

- [Setting Search Paths](#)
- [GRANT \(Schema\)](#)
- [REVOKE \(Schema\)](#)

Table Privileges

By default, only a **superuser** and the table owner (typically the person who creates a table) have access to a table. The ability to drop or alter a table is also reserved for a superuser or table owner. This privilege cannot be granted to other users.

All other users or roles (including the user who owns the schema, if he or she does not also own the table) must be explicitly granted using WITH GRANT OPTION syntax to access the table.

These are the table privileges a superuser or table owner can grant:

| Privilege | Description |
|-----------|--|
| SELECT | Permission to run SELECT queries on the table. |

| Privilege | Description |
|------------|--|
| INSERT | Permission to INSERT data into the table. |
| DELETE | Permission to DELETE data from the table, as well as SELECT privilege on the table when executing a DELETE statement that references table column values in a WHERE or SET clause. |
| UPDATE | Permission to UPDATE and change data in the table, as well as SELECT privilege on the table when executing an UPDATE statement that references table column values in a WHERE or SET clause. |
| REFERENCES | Permission to CREATE foreign key constraints that reference this table. |

To use any of the above privileges, the user must also have USAGE privileges on the schema that contains the table. See [Schema Privileges](#) for details.

Referencing sequence in the [CREATE TABLE](#) statement requires the following privileges:

- SELECT privilege on sequence object
- USAGE privilege on sequence schema

For details on granting and revoking table privileges, see [GRANT \(Table\)](#) and [REVOKE \(Table\)](#) in the SQL Reference Manual.

Projection Privileges

Because [projections](#) are the underlying storage construct for tables, they are atypical in that they do not have an owner or privileges associated with them directly. Instead, the privileges to create, access, or alter a projection are based on the anchor and base tables that the projection references, as well as the schemas that contain them.

To be able run a query involving a projection, a user must have SELECT privileges on the table or tables that the projection references, and USAGE privileges on all the schemas that contain those tables.

There are two ways to create projection: explicitly and implicitly.

Explicit Projection Creation and Privileges

To explicitly create a projection using the [CREATE PROJECTION](#) statement, a user must be a **superuser** or owner of the anchor table or have the following privileges:

- CREATE privilege on the schema in which the projection is created
- SELECT on all the base tables referenced by the projection
- USAGE on all the schemas that contain the base tables referenced by the projection

Explicitly-created projections can only be dropped by the table owner on which the projection is based for a single-table projection, or the owner of the anchor table for **pre-join projections**.

Implicit Projection Creation and Privileges

Projections get implicitly created when you insert data into a table, an operation that automatically creates a **superprojection** for the table.

Implicitly-created projections do not require any additional privileges to create or drop, other than privileges for table creation. Users who can create a table or drop a table can also create and drop the associated superprojection.

Selecting From Projections

To select from projections requires the following privileges:

- SELECT privilege on each of the base tables
- USAGE privilege on the corresponding containing schemas

HP Vertica does not associate privileges directly with projections since they are the underlying storage construct. Privileges may only be granted on the logical storage containers: the tables and views.

Dropping Projections

Dropping projections are handled much the same way HP Vertica creates them:

- Explicitly with **DROP PROJECTION** statement
- Implicitly when you drop the table

View Privileges

By default, only a **superuser** and the view owner (typically the person who creates the view) have access to the base object for a **view**. All other users and roles must be directly granted access to the view. For example:

- If a non-owner runs a SELECT query on the view, the view owner must also have SELECT ... WITH GRANT OPTION privileges on the view's base tables or views. This privilege must be directly granted to the owner, rather than through a role.
- If a view owner runs a SELECT query on the view, the owner must also have SELECT privilege directly granted (not through a role) on a view's base objects (table or view).

The only privilege that can be granted to a user or role is SELECT, which allows the user to execute SELECT queries on the view. The user or role also needs to have USAGE privilege on the schema containing the view to be able to run queries on the view.

| Privilege | Description |
|-----------|---|
| SELECT | Permission to run SELECT queries on the view. |
| USAGE | Permission on the schema that contains the view |

For details on granting and revoking view privileges, see [GRANT \(View\)](#) and [REVOKE \(View\)](#) in the SQL Reference Manual.

Sequence Privileges

To create a sequence, a user must have CREATE privileges on schema that contains the sequence. Only the owner and **superusers** can initially access the sequence. All other users must be granted access to the sequence by a superuser or the owner.

Only the sequence owner (typically the person who creates the sequence) or can drop or rename a sequence, or change the schema in which the sequence resides:

- [DROP SEQUENCE](#): Only a sequence owner or schema owner can drop a sequence.
- [ALTER SEQUENCE RENAME TO](#): A sequence owner must have USAGE and CREATE privileges on the schema that contains the sequence to be renamed.
- [ALTER SEQUENCE SET SCHEMA](#): A sequence owner must have USAGE privilege on the schema that currently contains the sequence (old schema), as well as CREATE privilege on the schema where the sequence will be moved (new schema).

The following table lists the privileges that can be granted to users or roles on sequences.

The only privilege that can be granted to a user or role is SELECT, which allows the user to use [CURRVAL\(\)](#) and [NEXTVAL\(\)](#) on sequence and reference in table. The user or role also needs to have USAGE privilege on the schema containing the sequence.

| Privilege | Description |
|-----------|---|
| SELECT | Permission to use CURRVAL() and NEXTVAL() on sequence and reference in table. |
| USAGE | Permissions on the schema that contains the sequence. |

Note: Referencing sequence in the [CREATE TABLE](#) statement requires SELECT privilege on sequence object and USAGE privilege on sequence schema.

For details on granting and revoking sequence privileges, see [GRANT \(Sequence\)](#) and [REVOKE \(Sequence\)](#) in the SQL Reference Manual.

See Also

- [Using Named Sequences](#)

External Procedure Privileges

Only a **superuser** is allowed to create or drop an **external procedure**.

By default, users cannot execute external procedures. A superuser must grant users and roles this right, using the GRANT (Procedure) EXECUTE statement. Additionally, users must have USAGE privileges on the schema that contains the procedure in order to call it.

| Privilege | Description |
|-----------|---|
| EXECUTE | Permission to run an external procedure. |
| USAGE | Permission on the schema that contains the procedure. |

For details on granting and revoking external table privileges, see [GRANT \(Procedure\)](#) and [REVOKE \(Procedure\)](#) in the SQL Reference Manual.

User-Defined Function Privileges

User-defined functions (described in [CREATE FUNCTION Statements](#)) can be created by **superusers** or users with CREATE privileges on the schema that will contain the function, as well as USAGE privileges on the base library (if applicable).

Users or roles other than the function owner can use a function only if they have been granted EXECUTE privileges on it. They must also have USAGE privileges on the schema that contains the function to be able to call it.

| Privilege | Description |
|-----------|--|
| EXECUTE | Permission to call a user-defined function. |
| USAGE | Permission on the schema that contains the function. |

- [DROP FUNCTION](#): Only a superuser or function owner can drop the function.
- [ALTER FUNCTION RENAME TO](#): A superuser or function owner must have USAGE and CREATE privileges on the schema that contains the function to be renamed.
- [ALTER FUNCTION SET SCHEMA](#): A superuser or function owner must have USAGE privilege on the schema that currently contains the function (old schema), as well as CREATE privilege on the schema where the function will be moved (new schema).

For details on granting and revoking user-defined function privileges, see the following topics in the SQL Reference Manual:

- [GRANT \(User Defined Extension\)](#)
- [REVOKE \(User Defined Extension\)](#)

Library Privileges

Only a **superuser** can load an external library using the [CREATE LIBRARY](#) statement. By default, only a superuser can create user-defined functions (**UDFs**) based on a loaded library. A superuser can use the GRANT USAGE ON LIBRARY statement to allow users to create UDFs based on classes in the library. The user must also have CREATE privileges on the schema that will contain the UDF.

| Privilege | Description |
|-----------|---|
| USAGE | Permission to create UDFs based on classes in the library |

Once created, only a superuser or the user who created a UDF can use it by default. Either of them can grant other users or roles the ability to call the function using the GRANT EXECUTE ON FUNCTION statement. See the GRANT (Function) and REVOKE (Function) topics in the SQL Reference Manual for more information on granting and revoking privileges on functions.

In addition to EXECUTE privilege, users/roles also require USAGE privilege on the schema in which the function resides in order to execute the function.

For more information about libraries and UDFs, see [Developing and Using User Defined Functions](#) in the Programmer's Guide.

Resource Pool Privileges

Only a **superuser** can create, alter, or drop a **resource pool**.

By default, users are granted USAGE rights to the GENERAL pool, from which their queries and other statements allocate memory and get their priorities. A superuser must grant users USAGE rights to any additional resource pools by using the GRANT USAGE ON RESOURCE POOL statement. Once granted access to the resource pool, users can use the [SET SESSION RESOURCE POOL](#) statement and the RESOURCE POOL clause of the [ALTER USER](#) statement to have their queries draw their resources from the new pool.

| Privilege | Description |
|-----------|--|
| USAGE | Permission to use a resource pool. |
| SELECT | Permission to look up resource pool information/status in system tables. |
| UPDATE | Permission to adjust the tuning parameters of the pool. |

For details on granting and revoking resource pool privileges, see [GRANT \(Resource Pool\)](#) and [REVOKE \(Resource Pool\)](#) in the SQL Reference Manual.

Storage Location Privileges

Users and roles without **superuser** privileges can copy data to and from storage locations as long as the following conditions are met, where a **superuser**:

1. Creates a special class of storage location ([ADD_LOCATION](#)) specifying the 'USER' argument, which indicates the specified area is accessible to non-dbadmin users.
2. Grants users or roles READ and/or WRITE access to the specified location using the [GRANT \(Storage Location\)](#) statement.

Note: GRANT/REVOKE (Storage Location) statements are applicable only to 'USER' storage locations.

Once such storage locations exist and the appropriate privileges are granted, users and roles granted READ privileges can copy data from files in the storage location into a table. Those granted WRITE privileges can export data from a table to the storage location on which they have been granted access. WRITE privileges also let users save COPY statement exceptions and rejected data files from HP Vertica to the specified storage location.

Only a superuser can add, alter, retire, drop, and restore a location, as well as set and measure location performance. All non-dbadmin users or roles require READ and/or WRITE permissions on the location.

| Privilege | Description |
|-----------|---|
| READ | Allows the user to copy data from files in the storage location into a table. |
| WRITE | Allows the user to copy data to the specific storage location. Users with WRITE privileges can also save COPY statement exceptions and rejected data files to the specified storage location. |

See Also

- [GRANT \(Storage Location\)](#)
- [Storage Management Functions](#)
- [ADD_LOCATION](#)

Role, profile, and User Privileges

Only a **superuser** can create, alter or drop a:

- role
- profile
- user

By default, only the superuser can grant or revoke a role to another user or role. A user or role can be given the privilege to grant and revoke a role by using the WITH ADMIN OPTION clause of the GRANT statement.

For details on granting and revoking role privileges, see [GRANT \(Role\)](#) and [REVOKE \(Role\)](#) in the SQL Reference Manual.

See Also

- [CREATE USER](#)
- [ALTER USER](#)
- [DROP USER](#)
- [CREATE PROFILE](#)
- [ALTER PROFILE](#)
- [DROP PROFILE](#)
- [CREATE ROLE](#)
- [ALTER ROLE RENAME](#)
- [DROP ROLE](#)

Metadata Privileges

A **superuser** has unrestricted access to all database metadata. Other users have significantly reduced access to metadata based on their privileges, as follows:

| Type of Metadata | User Access |
|---|--|
| Catalog objects: <ul style="list-style-type: none"> • Tables • Columns • Constraints • Sequences • External Procedures • Projections • ROS containers • WOS | <p>Users must possess USAGE privilege on the schema and any type of access (SELECT) or modify privilege on the object to see catalog metadata about the object. See also Schema Privileges.</p> <p>For internal objects like projections, WOS and ROS containers that don't have access privileges directly associated with them, the user must possess the requisite privileges on the associated schema and table objects instead. For example, to see whether a table has any data in the WOS, you need to have USAGE on the table schema and at least SELECT on the table itself. See also Table Privileges and Projection Privileges.</p> |
| User sessions and functions, and system tables related to these sessions | <p>Users can only access information about their own, current sessions.</p> <p>The following functions provide restricted functionality to users:</p> <ul style="list-style-type: none"> • CURRENT_DATABASE • CURRENT_SCHEMA • CURRENT_USER • HAS_TABLE_PRIVILEGE • SESSION_USER (same as CURRENT_USER) <p>The system table, SESSIONS, provides restricted functionality to users.</p> |
| Storage locations | <p>Users require READ permissions to copy data from storage locations.</p> <p>Only a superuser can add or retire storage locations.</p> |

I/O Privileges

Users need no special permissions to connect to and disconnect from an HP Vertica database.

To [EXPORT TO](#) and [COPY FROM](#) HP Vertica, the user must have:

- SELECT privileges on the source table
- USAGE privilege on source table schema
- INSERT privileges for the destination table in target database
- USAGE privilege on destination table schema

To COPY FROM STDIN and use local COPY a user must have INSERT privileges on the table and USAGE privilege on schema.

Note: Only a **superuser** can COPY from *file*.

Comment Privileges

A comment lets you add, revise, or remove a textual message to a database object. You must be an object owner or superuser in order to COMMENT ON one of the following objects:

- COLUMN
- CONSTRAINT
- FUNCTION (including AGGREGATE and ANALYTIC)
- LIBRARY
- NODE
- PROJECTION
- SCHEMA
- SEQUENCE
- TABLE
- TRANSFORM FUNCTION
- VIEW

Other users must have VIEW privileges on an object to view its comments.

Transaction Privileges

No special permissions are required for the following database operations:

- [COMMIT](#)
- [ROLLBACK](#)
- [RELEASE SAVEPOINT](#)
- [SAVEPOINT](#)

Session Privileges

No special permissions are required for users to use the SHOW statement or any of the SET statements.

Tuning Privileges

In order to [PROFILE](#) a single SQL statement or returns a query plan's execution strategy to standard output using the [EXPLAIN](#) command, users must have the same privileges that are required for them to run the same query without the PROFILE or EXPLAIN keyword.

Granting and Revoking Privileges

To grant or revoke a privilege using one of the SQL GRANT or REVOKE statements, the user must have the following permissions for the GRANT/REVOKE statement to succeed:

- **Superuser** or privilege WITH GRANT OPTION
- USAGE privilege on the schema
- Appropriate privileges on the object

The syntax for granting and revoking privileges is different for each database object, such as schema, database, table, view, sequence, procedure, function, resource pool, and so on.

Normally, a superuser first [creates a user](#) and then uses GRANT syntax to define the user's privileges or roles or both. For example, the following series of statements creates user Carol and grants Carol access to the apps database in the PUBLIC schema and also lets Carol grant SELECT privileges to other users on the applog table:

```
=> CREATE USER Carol;=> GRANT USAGE ON SCHEMA PUBLIC to Carol;  
=> GRANT ALL ON DATABASE apps TO Carol;  
=> GRANT SELECT ON applog TO Carol WITH GRANT OPTION;
```

See [GRANT Statements](#) and [REVOKE Statements](#) in the SQL Reference Manual.

About Superuser Privileges

A **superuser** (DBADMIN) is the automatically-created database user who has the same name as the Linux database administrator account and who can bypass all GRANT/REVOKE authorization,

as well as supersede any user that has been granted the [PSEUDOSUPERUSER](#) role.

Note: Database superusers are not the same as a Linux superuser with (root) privilege and cannot have Linux superuser privilege.

A superuser can grant privileges on all database object types to other users, as well as grant privileges to **roles**. Users who have been granted the role will then gain the privilege as soon as they [enable it](#).

Superusers may grant or revoke any object privilege on behalf of the object owner, which means a superuser can grant or revoke the object privilege if the object owner could have granted or revoked the same object privilege. A superuser may revoke the privilege that an object owner granted, as well as the reverse.

Since a superuser is acting on behalf of the object owner, the GRANTOR column of [V_CATALOG.GRANTS](#) table displays the object owner rather than the superuser who issued the GRANT statement.

A superuser can also alter ownership of table and sequence objects.

See Also

[DBADMIN Role](#)

About Schema Owner Privileges

By default, the schema owner has privileges to create objects within a schema. Additionally, the schema owner can drop any object in the schema, requiring no additional privilege on the object.

The schema owner is typically the user who creates the schema.

Schema owners cannot access objects in the schema. Access to objects requires the appropriate privilege at the object level.

All other access to the schema and its objects must be explicitly granted to users or roles by a superuser or schema owner to prevent unauthorized users from accessing the schema and its objects.

See [Schema Privileges](#)

About Object Owner Privileges

The database, along with every object in it, has an owner. The object owner is usually the person who created the object, although a **superuser** can alter ownership of objects, such as table and sequence.

Object owners must have appropriate schema privilege to access, alter, rename, move or drop any object it owns without any additional privileges.

An object owner can also:

- **Grant privileges on their own object to other users**

The WITH GRANT OPTION clause specifies that a user can grant the permission to other users. For example, if user Bob creates a table, Bob can grant privileges on that table to users Ted, Alice, and so on.

- **Grant privileges to roles**

Users who are granted the role gain the privilege.

How to Grant Privileges

As described in [Granting and Revoking Privileges](#), specific users grant privileges using the GRANT statement with or without the optional WITH GRANT OPTION, which allows the user to grant the same privileges to other users.

- A **superuser** can grant privileges on all object types to other users.
- A superuser or object owner can grant privileges to **roles**. Users who have been granted the role then gain the privilege.
- An object owner can grant privileges on the object to other users using the optional WITH GRANT OPTION clause.
- The user needs to have USAGE privilege on schema and appropriate privileges on the object.

When a user grants an explicit list of privileges, such as GRANT INSERT, DELETE, REFERENCES ON applog TO Bob:

- The GRANT statement succeeds only if all the roles are granted successfully. If any grant operation fails, the entire statement rolls back.
- HP Vertica will return ERROR if the user does not have grant options for the privileges listed.

When a user grants ALL privileges, such as GRANT ALL ON applog TO Bob, the statement always succeeds. HP Vertica grants all the privileges on which the grantor has the WITH GRANT OPTION and skips those privileges without the optional WITH GRANT OPTION.

For example, if the user Bob has delete privileges with the optional grant option on the applog table, only DELETE privileges are granted to Bob, and the statement succeeds:

```
=> GRANT DELETE ON applog TO Bob WITH GRANT OPTION;GRANT PRIVILEGE
```

For details, see the [GRANT Statements](#) in the SQL Reference Manual.

How to Revoke Privileges

In general, **ONLY** the user who originally granted a privilege can revoke it using a REVOKE statement. That user must have **superuser** privilege or have the optional WITH GRANT OPTION

on the privilege. The user also must have USAGE privilege on the schema and appropriate privileges on the object for the REVOKE statement to succeed.

In order to revoke a privilege, this privilege must have been granted to the specified grantee by this grantor before. If HP Vertica finds that to be the case, the above REVOKE statement removes the privilege (and WITH GRANT OPTION privilege, if supplied) from the grantee. Otherwise, HP Vertica prints a NOTICE that the operation failed, as in the following example.

```
=> REVOKE SELECT ON applog FROM Bob;  
NOTICE 0: Cannot revoke "SELECT" privilege(s) for relation "applog" that you did not grant to "Bob"  
REVOKE PRIVILEGE
```

The above REVOKE statement removes the privilege (and WITH GRANT OPTION privilege, if applicable) from the grantee or it prints a notice that the operation failed.

In order to revoke grant option for a privilege, the grantor must have previously granted the grant option for the privilege to the specified grantee. Otherwise, HP Vertica prints a NOTICE.

The following REVOKE statement removes the GRANT option only but leaves the privilege intact:

```
=> GRANT INSERT on applog TO Bob WITH GRANT OPTION;  
GRANT PRIVILEGE  
=> REVOKE GRANT OPTION FOR INSERT ON applog FROM Bob;  
REVOKE PRIVILEGE
```

When a user revokes an explicit list of privileges, such as GRANT INSERT, DELETE, REFERENCES ON applog TO Bob:

- The REVOKE statement succeeds only if all the roles are revoked successfully. If any revoke operation fails, the entire statement rolls back.
- HP Vertica returns ERROR if the user does not have grant options for the privileges listed.
- HP Vertica returns NOTICE when revoking privileges that this user had not been previously granted.

When a user revokes ALL privileges, such as REVOKE ALL ON applog TO Bob, the statement always succeeds. HP Vertica revokes all the privileges on which the grantor has the optional WITH GRANT OPTION and skips those privileges without the WITH GRANT OPTION.

For example, if the user Bob has delete privileges with the optional grant option on the applog table, only grant option is revoked from Bob, and the statement succeeds without NOTICE:

```
=> REVOKE GRANT OPTION FOR DELETE ON applog FROM Bob;
```

For details, see the [REVOKE Statements](#) in the SQL Reference Manual.

Privilege Ownership Chains

The ability to revoke privileges on objects can cascade throughout an organization. If the grant option was revoked from a user, the privilege that this user granted to other users will also be revoked.

If a privilege was granted to a user or role by multiple grantors, to completely revoke this privilege from the grantee the privilege has to be revoked by each original grantor. The only exception is a superuser may revoke privileges granted by an object owner, with the reverse being true, as well.

In the following example, the SELECT privilege on table t1 is granted through a chain of users, from a superuser through User3.

- A superuser grants User1 CREATE privileges on the schema s1:

```
=> \c - dbadminYou are now connected as user "dbadmin".
=> CREATE USER User1;
CREATE USER
=> CREATE USER User2;
CREATE USER
=> CREATE USER User3;
CREATE USER
=> CREATE SCHEMA s1;
CREATE SCHEMA
=> GRANT USAGE on SCHEMA s1 TO User1, User2, User3;
GRANT PRIVILEGE
=> CREATE ROLE reviewer;
CREATE ROLE
=> GRANT CREATE ON SCHEMA s1 TO User1;
GRANT PRIVILEGE
```

- User1 creates new table t1 within schema s1 and then grants SELECT WITH GRANT OPTION privilege on s1.t1 to User2:

```
=> \c - User1You are now connected as user "User1".
=> CREATE TABLE s1.t1(id int, sourceID VARCHAR(8));
CREATE TABLE
=> GRANT SELECT on s1.t1 to User2 WITH GRANT OPTION;
GRANT PRIVILEGE
```

- User2 grants SELECT WITH GRANT OPTION privilege on s1.t1 to User3:

```
=> \c - User2You are now connected as user "User2".
=> GRANT SELECT on s1.t1 to User3 WITH GRANT OPTION;
GRANT PRIVILEGE
```

- User3 grants SELECT privilege on s1.t1 to the reviewer role:

```
=> \c - User3You are now connected as user "User3".  
=> GRANT SELECT on s1.t1 to reviewer;  
GRANT PRIVILEGE
```

Users cannot revoke privileges upstream in the chain. For example, User2 did not grant privileges on User1, so when User1 runs the following REVOKE command, HP Vertica rolls back the command:

```
=> \c - User2You are now connected as user "User2".  
=> REVOKE CREATE ON SCHEMA s1 FROM User1;  
ROLLBACK 0: "CREATE" privilege(s) for schema "s1" could not be revoked from "User1"
```

Users can revoke privileges indirectly from users who received privileges through a cascading chain, like the one shown in the example above. Here, users can use the CASCADE option to revoke privileges from all users "downstream" in the chain. A superuser or User1 can use the CASCADE option to revoke the SELECT privilege on table s1.t1 from all users. For example, a superuser or User1 can execute the following statement to revoke the SELECT privilege from all users and roles within the chain:

```
=> \c - User1You are now connected as user "User1".  
=> REVOKE SELECT ON s1.t1 FROM User2 CASCADE;  
REVOKE PRIVILEGE
```

When a superuser or User1 executes the above statement, the SELECT privilege on table s1.t1 is revoked from User2, User3, and the reviewer role. The GRANT privilege is also revoked from User2 and User3, which a superuser can verify by querying the [V_CATALOG.GRANTS](#) system table.

```
=> SELECT * FROM grants WHERE object_name = 's1' AND grantee ILIKE 'User%';  
grantor | privileges_description | object_schema | object_name | grantee  
-----+-----+-----+-----+-----  
dbadmin | USAGE                  |               | s1          | User1  
dbadmin | USAGE                  |               | s1          | User2  
dbadmin | USAGE                  |               | s1          | User3  
(3 rows)
```

Modifying Privileges

A **superuser** or object owner can use one of the ALTER statements to modify a privilege, such as changing a sequence owner or table owner. Reassignment to the new owner does not transfer grants from the original owner to the new owner; grants made by the original owner are dropped.

Changing a Table Owner

The ability to change table ownership is useful when moving a table from one schema to another. Ownership reassignment is also useful when a table owner leaves the company or changes job responsibilities. Because you can change the table owner, the tables won't have to be completely rewritten, you can avoid loss in productivity.

The syntax looks like this:

```
ALTER TABLE [[db-name.]schema.]table-name OWNER TO new-owner name
```

In order to alter table ownership, you must be either the table owner or a **superuser**.

A change in table ownership transfers just the owner and not privileges; grants made by the original owner are dropped and all existing privileges on the table are revoked from the previous owner. However, altering the table owner transfers ownership of dependent sequence objects (associated IDENTITY/AUTO-INCREMENT sequences) but does not transfer ownership of other referenced sequences. See [ALTER SEQUENCE](#) for details on transferring sequence ownership.

Notes

- Table privileges are separate from schema privileges; therefore, a table privilege change or table owner change does not result in any schema privilege change.
- Because projections define the physical representation of the table, HP Vertica does not require separate projection owners. The ability to create or drop projections is based on the table privileges on which the projection is anchored.
- During the alter operation HP Vertica updates projections anchored on the table owned by the old owner to reflect the new owner. For **pre-join projection** operations, HP Vertica checks for privileges on the referenced table.

Example

In this example, user Bob connects to the database, looks up the tables, and transfers ownership of table `t33` from himself to user Alice.

```
=> \c - BobYou are now connected as user "Bob".
=> \d
Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
public | applog | table | dbadmin |
```

```
public | t33 | table | Bob |
(2 rows)
=> ALTER TABLE t33 OWNER TO Alice;
ALTER TABLE
```

Notice that when Bob looks up database tables again, he no longer sees table t33.

```
=> \d
List of tables
List of tables
Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
public | applog | table | dbadmin |
(1 row)
```

When user Alice connects to the database and looks up tables, she sees she is the owner of table t33.

```
=> \c - AliceYou are now connected as user "Alice".
=> \d
List of tables
Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
public | t33 | table | Alice |
(2 rows)
```

Either Alice or a superuser can transfer table ownership back to Bob. In the following case a superuser performs the transfer.

```
=> \c - dbadminYou are now connected as user "dbadmin".
=> ALTER TABLE t33 OWNER TO Bob;
ALTER TABLE
=> \d
List of tables
Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
public | applog | table | dbadmin |
public | comments | table | dbadmin |
public | t33 | table | Bob |
s1 | t1 | table | User1 |
(4 rows)
```

You can also query the [V_CATALOG.TABLES](#) system table to view table and owner information. Note that a change in ownership does not change the table ID.

In the below series of commands, the superuser changes table ownership back to Alice and queries the TABLES system table.

```
=> ALTER TABLE t33 OWNER TO Alice;ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM
tables; table_schema_id | table_schema | table_id | table_name | owner_id
| owner_name
-----+-----+-----+-----+-----+-----
```

```

-----
 45035996273704968 | public      | 45035996273713634 | applog      | 45035996273704962 |
 dbadmin
 45035996273704968 | public      | 45035996273724496 | comments    | 45035996273704962 |
 dbadmin
 45035996273730528 | s1          | 45035996273730548 | t1          | 45035996273730516 |
 User1
 45035996273704968 | public      | 45035996273795846 | t33        | 45035996273724576 |
 Alice
 (5 rows)

```

Now the superuser changes table ownership back to Bob and queries the TABLES table again. Nothing changes but the owner_name row, from Alice to Bob.

```

=> ALTER TABLE t33 OWNER TO Bob;ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM
tables;
  table_schema_id | table_schema | table_id | table_name | owner_id |
owner_name-----+-----+-----+-----+-----
-----+-----
 45035996273704968 | public      | 45035996273713634 | applog      | 45035996273704962 |
 dbadmin
 45035996273704968 | public      | 45035996273724496 | comments    | 45035996273704962 |
 dbadmin
 45035996273730528 | s1          | 45035996273730548 | t1          | 45035996273730516 |
 User1
 45035996273704968 | public      | 45035996273793876 | foo         | 45035996273724576 |
 Alice
 45035996273704968 | public      | 45035996273795846 | t33        | 45035996273714428 |
 Bob
 (5 rows)

```

Table Reassignment with Sequences

Altering the table owner transfers ownership of only associated IDENTITY/AUTO-INCREMENT sequences but not other reference sequences. For example, in the below series of commands, ownership on sequence s1 does not change:

```

=> CREATE USER u1;CREATE USER
=> CREATE USER u2;
CREATE USER
=> CREATE SEQUENCE s1 MINVALUE 10 INCREMENT BY 2;
CREATE SEQUENCE
=> CREATE TABLE t1 (a INT, id INT DEFAULT NEXTVAL('s1'));
CREATE TABLE
=> CREATE TABLE t2 (a INT, id INT DEFAULT NEXTVAL('s1'));
CREATE TABLE
=> SELECT sequence_name, owner_name FROM sequences;
 sequence_name | owner_name
-----+-----
 s1            | dbadmin
 (1 row)

```

```
=> ALTER TABLE t1 OWNER TO u1;
ALTER TABLE
=> SELECT sequence_name, owner_name FROM sequences;
  sequence_name | owner_name
-----+-----
s1              | dbadmin
(1 row)
=> ALTER TABLE t2 OWNER TO u2;
ALTER TABLE
=> SELECT sequence_name, owner_name FROM sequences;
  sequence_name | owner_name
-----+-----
s1              | dbadmin
(1 row)
```

See Also

- [Changing a Sequence Owner](#)

Changing a Sequence Owner

The `ALTER SEQUENCE` command lets you change the attributes of an existing sequence. All changes take effect immediately, within the same session. Any parameters not set during an `ALTER SEQUENCE` statement retain their prior settings.

If you need to change sequence ownership, such as if an employee who owns a sequence leaves the company, you can do so with the following `ALTER SEQUENCE` syntax:

```
ALTER SEQUENCE sequence-name OWNER TO new-owner-name;
```

This operation immediately reassigns the sequence from the current owner to the specified new owner.

Only the sequence owner or a **superuser** can change ownership, and reassignment does not transfer grants from the original owner to the new owner; grants made by the original owner are dropped.

Note: Renaming a table owner transfers ownership of dependent sequence objects (associated `IDENTITY/AUTO-INCREMENT` sequences) but does not transfer ownership of other referenced sequences. See [Changing a Table Owner](#).

Example

The following example reassigns sequence ownership from the current owner to user Bob:

```
=> ALTER SEQUENCE sequential OWNER TO Bob;
```

See [ALTER SEQUENCE](#) in the SQL Reference Manual for details.

Viewing Privileges Granted on Objects

HP Vertica logs information about privileges granted on various objects, including the grantor and grantee, in the `V_CATALOG.GRANTS` system table. The order of columns in the table corresponds to the order in which they appear in the GRANT command. An asterisk in the output means the privilege was granted WITH GRANT OPTION.

The following command queries the GRANTS system table:

```
=> SELECT * FROM grants ORDER BY grantor, grantee;
```

| grantor | privileges_description | object_schema | object_name |
|-----------|------------------------------------|---------------|-------------|
| Alice | | | commentor |
| dbadmin | CREATE | | schema2 |
| Bob | | | commentor |
| dbadmin | | | commentor |
| Bob | | | commentor |
| dbadmin | | | logadmin |
| Bob | | | commentor |
| dbadmin | USAGE | | general |
| Bob | | public | applog |
| dbadmin | INSERT, UPDATE, DELETE, REFERENCES | | logadmin |
| Bob | | | logadmin |
| Ted | | | commentor |
| dbadmin | USAGE | | general |
| Ted | | | commentor |
| dbadmin | USAGE | | general |
| Sue | | | commentor |
| dbadmin | CREATE, CREATE TEMP | | vmart |
| Sue | | | commentor |
| dbadmin | USAGE | | public |
| Sue | | public | applog |
| dbadmin | SELECT* | | commentor |
| Sue | | | commentor |
| dbadmin | USAGE | | general |
| Alice | | public | comments |
| commentor | INSERT, SELECT | | commentor |
| dbadmin | INSERT, SELECT | public | applog |
| commentor | | | commentor |
| dbadmin | | | logwriter |
| logadmin | | | commentor |
| dbadmin | | | logreader |
| logadmin | | | commentor |
| dbadmin | DELETE | public | applog |
| logadmin | | | commentor |
| dbadmin | SELECT | public | applog |
| logreader | | | commentor |


```

dbadmin | INSERT | public | applog
| logwriter
dbadmin | USAGE | | v_internal
| public
dbadmin | CREATE TEMP | | vmart
| public
dbadmin | USAGE | | public
| public
dbadmin | USAGE | | v_catalog
| public
dbadmin | USAGE | | v_monitor
| public
dbadmin | CREATE*, CREATE TEMP* | | vmart
| dbadmin
dbadmin | USAGE*, CREATE* | | schema2
| dbadmin
dbadmin | INSERT*, SELECT*, UPDATE*, DELETE*, REFERENCES* | public | comments
| dbadmin
dbadmin | INSERT*, SELECT*, UPDATE*, DELETE*, REFERENCES* | public | applog
| dbadmin
(30 rows)
    
```

To quickly find all of the privileges that have been granted to all users on the schema named myschema, run the following statement:

```

=> SELECT grantee, privileges_description FROM GRANTS WHERE object_name='myschema';
grantee | privileges_description
-----+-----
Bob     | USAGE, CREATE
Alice   | CREATE
(2 rows)
    
```

Note that the vsql commands, \dp and \z, both return similar information to GRANTS:

```

=> \dp
          Access privileges for database "apps" Grantee | Grantor |
          Privileges | Schema | Name
-----+-----+-----
----
public | dbadmin | USAGE | | v_inter
nal
public | dbadmin | USAGE | | v_catal
og
public | dbadmin | USAGE | | v_monit
or
logadmin | dbadmin | | | logread
er
logadmin | dbadmin | | | logwrit
er
Fred | dbadmin | USAGE | | general
Fred | dbadmin | | | logadmi
n
Bob | dbadmin | USAGE | | general
dbadmin | dbadmin | USAGE*, CREATE* | | schema2
Bob | dbadmin | CREATE | | schema2
    
```

```
Sue      | dbadmin | USAGE          |          | general
public  | dbadmin | USAGE          |          | public
Sue     | dbadmin | USAGE          |          | public
public  | dbadmin | CREATE TEMP    |          | appdat
dbadmin | dbadmin | CREATE*, CREATE TEMP* |          | appdat
Sue     | dbadmin | CREATE, CREATE TEMP |          | appdat
dbadmin | dbadmin | INSERT*, SELECT*, UPDATE*, DELETE*, REFERENCES* | public | applog
logreader | dbadmin | SELECT        | public  | applog
logwriter | dbadmin | INSERT        | public  | applog
logadmin | dbadmin | DELETE        | public  | applog
Sue     | dbadmin | SELECT*       | public  | applog
(22 rows)
```

See [GRANT Statements](#) in the SQL Reference Manual.

About Database Roles

To make managing permissions easier, use roles. A role is a collection of privileges that a **superuser** can grant to (or revoke from) one or more users or other roles. Using roles avoids having to manually grant sets of privileges user by user. For example, several users might be assigned to the administrator role. You can grant or revoke privileges to or from the administrator role, and all users with access to that role are affected by the change.

Note: Users must first enable a role before they gain all of the privileges that have been granted to it. See [Enabling Roles](#).

Role Hierarchies

You can also use roles to build hierarchies of roles; for example, you can create an administrator role that has privileges granted non-administrator roles as well as to the privileges granted directly to the administrator role. See also [Role Hierarchy](#).

Roles do not supersede manually-granted privileges, so privileges directly assigned to a user are not altered by roles. Roles just give additional privileges to the user.

Creating and Using a Role

Using a role follows this general flow:

1. A superuser creates a role using the [CREATE ROLE](#) statement.
2. A superuser or object owner grants privileges to the role using one of the GRANT statements.
3. A superuser or users with administrator access to the role grant users and other roles access to the role.
4. Users granted access to the role use the [SET ROLE](#) command to enable that role and gain the role's privileges.

You can do steps 2 and 3 in any order. However, granting access to a role means little until the role has privileges granted to it.

Tip: You can query the V_CATALOG system tables [ROLES](#), [GRANTS](#), and [USERS](#) to see any directly-assigned roles; however, these tables do not indicate whether a role is available to a user when roles could be available through other roles (indirectly). See the [HAS_ROLE\(\)](#) function for additional information.

Roles on Management Console

When users sign in to the Management Console (MC), what they can view or do is governed by MC roles. For details, see [About MC Users](#) and [About MC Privileges and Roles](#).

Types of Database Roles

HP Vertica has three pre-defined roles:

- [PUBLIC](#)
- [PSEUDOSUPERUSER](#)
- [DBADMIN](#)

Note: You might encounter a DBDUSER role in system table output. This role is internal only; you can ignore it.

Predefined roles cannot be dropped or renamed. Other roles may not be granted to (or revoked from) predefined roles except to/from PUBLIC, but predefined roles may be granted to other roles or users or both.

Individual privileges may be granted to/revoked from predefined roles. See the SQL Reference Manual for all of the GRANT and REVOKE statements.

DBADMIN Role

Every database has the special DBADMIN role. A **superuser** (or someone with the [PSEUDOSUPERUSER Role](#)) can grant this role to or revoke this role from any user or role.

Users who enable the DBADMIN role gain these privileges:

- Create or drop users
- Create or drop schemas
- Create or drop roles
- View all system tables
- View and terminate user sessions

The DBADMIN role does NOT allow users to:

- Start and stop a database
- Change DBADMIN privileges
- Set configuration parameters (`set_config_parameter`)

You can assign additional privileges to the DBADMIN role, but you cannot assign any additional roles; for example, the following is not allowed:

```
=> CREATE ROLE appviewer;  
CREATE ROLE  
=> GRANT appviewer TO dbadmin;  
ROLLBACK 2347: Cannot alter predefined role "dbadmin"
```

You can, however, grant the DBADMIN role to other roles to augment a set of privileges. See [Role Hierarchy](#) for more information.

View a List of Database Superusers

To see who is a superuser, run the **vsql \du** meta-command. In this example, only dbadmin is a superuser.

```
=> \du  
List of users  
User name | Is Superuser  
-----+-----  
dbadmin   | t  
Fred      | f  
Bob       | f  
Sue       | f  
Alice     | f  
User1     | f  
User2     | f  
User3     | f  
u1        | f  
u2        | f  
(10 rows)
```

See Also

[DBADMIN User](#)

DBDUSER Role

The special DBDUSER role must be explicitly granted by a **superuser** and is a predefined role. The DBDUSER role allows non-DBADMIN users to access Database Designer using command-line functions. Users with the DBDUSER role cannot access Database Designer using the Administration Tools. Only [DBADMIN](#) users can run Administration Tools.

You cannot assign any additional privileges to the DBDUSER role, but you can grant the DBDUSER role to other roles to augment a set of privileges.

Once you have been granted the DBDUSER role, you must enable it before you can run Database Designer using command-line functions. For more information, see [About Running Database Designer Programmatically](#).

Important: When you create a DBADMIN user or grant the DBDUSER role, make sure to associate a resource pool with that user to manage resources during Database Designer runs.

Multiple users can run Database Designer concurrently without interfering with each other or using up all the cluster resources. When a user runs Database Designer, either using the Administration Tools or programmatically, its execution is mostly contained by the user's resource pool, but may spill over into some system resource pools for less-intensive tasks.

PSEUDOSUPERUSER Role

The special PSEUDOSUPERUSER role is automatically created in each database. A **superuser** (or someone with the PSEUDOSUPERUSER role) can grant this role to another user, or revoke the role from another user. The PSEUDOSUPERUSER cannot revoke or change any superuser privileges.

Users with the PSEUDOSUPERUSER role enabled have all of the privileges of the database superuser, including the ability to:

- Create schemas
- Create and grant privileges to roles
- Bypass all GRANT/REVOKE authorization
- Set user account's passwords
- Lock and unlock user accounts
- Create or drop a UDF library
- Create or drop a UDF function
- Create or drop an external procedure
- Add or edit comments on nodes
- Create or drop password profiles

You can assign additional privileges to the PSEUDOSUPERUSER role, but you cannot assign any additional roles; for example, the following is not allowed:

```
=> CREATE ROLE appviewer;  
CREATE ROLE  
=> GRANT appviewer TO pseudosuperuser;  
ROLLBACK 2347: Cannot alter predefined role "pseudosuperuser"
```

PUBLIC Role

By default, every database has the special PUBLIC role. HP Vertica grants this role to each user automatically, and it is automatically enabled. You grant privileges to this role that every user should have by default. You can also grant access to roles to PUBLIC, which allows any user to access the role using the [SET ROLE](#) statement.

Note: The PUBLIC role can never be dropped, nor can it be revoked from users or roles.

Example

In the following example, if the superuser hadn't granted INSERT privileges on the table publicdata to the PUBLIC group, the INSERT statement executed by user bob would fail:

```
=> CREATE TABLE publicdata (a INT, b VARCHAR);
CREATE TABLE
=> GRANT INSERT, SELECT ON publicdata TO PUBLIC;
GRANT PRIVILEGE
=> CREATE PROJECTION publicdataproj AS (SELECT * FROM publicdata);
CREATE PROJECTION

dbadmin=> \c - bob
You are now connected as user "bob".

=> INSERT INTO publicdata VALUES (10, 'Hello World');
OUTPUT
-----
      1
(1 row)
```

See Also

[PUBLIC User](#)

Default Roles for Database Users

By default, no roles (other than the default [PUBLIC Role](#)) are enabled at the start of a user session.

```
=> SHOW ENABLED_ROLES;
   name      | setting
-----+-----
enabled roles |
(1 row)
```

A **superuser** can set one or more default roles for a user, which are automatically enabled at the start of the user's session. Setting a default role is a good idea if users normally rely on the privileges granted by one or more roles to carry out the majority of their tasks. To set a default role, use the DEFAULT ROLE parameter of the [ALTER USER](#) statement as superuser:

```
=> \c vmart apps
You are now connected to database "apps" as user "dbadmin".
=> ALTER USER Bob DEFAULT ROLE logadmin;
ALTER USER
=> \c - Bob;
You are now connected as user "Bob"
```

```
=> SHOW ENABLED_ROLES;  
   name      | setting  
-----+-----  
 enabled roles | logadmin  
(1 row)
```

Notes

- Only roles that the user already has access to can be made default.
- Unlike granting a role, setting a default role or roles overwrites any previously-set defaults.
- To clear any default roles for a user, use the keyword `NONE` as the role name in the `DEFAULT ROLE` argument.
- Default roles only take effect at the start of a user session. They do not affect the roles enabled in the user's current session.
- Avoid giving users default roles that have administrative or destructive privileges (the [PSEUDOSUPERUSER](#) role or `DROP` privileges, for example). By forcing users to explicitly enable these privileges, you can help prevent accidental data loss.

Using Database Roles

There are several steps to using roles:

1. A **superuser** creates a role using the [CREATE ROLE](#) statement.
2. A superuser or object owner grants privileges to the role.
3. A superuser or users with administrator access to the role grant users and other roles access to the role.
4. Users granted access to the role run the [SET ROLE](#) command to make that role active and gain the role's privileges.

You can do steps 2 and 3 in any order. However, granting access to a role means little until the role has privileges granted to it.

Tip: Query system tables [ROLES](#), [GRANTS](#), and [USERS](#) to see any directly-assigned roles. Because these tables do not indicate whether a role is available to a user when roles could be available through other roles (indirectly), see the [HAS_ROLE\(\)](#) function for additional information.

See Also

- [About MC Privileges and Roles](#)

Role Hierarchy

In addition to granting roles to users, you can also grant roles to other roles. This lets you build hierarchies of roles, with more privileged roles (an administrator, for example) being assigned all of the privileges of lesser-privileged roles (a user of a particular application), in addition to the privileges you assign to it directly. By organizing your roles this way, any privilege you add to the application role (reading or writing to a new table, for example) is automatically made available to the more-privileged administrator role.

Example

The following example creates two roles, assigns them privileges, then assigns them to a new administrative role.

1. Create new table applog:

```
=> CREATE TABLE applog (id int, sourceID VARCHAR(32), data TIMESTAMP, event VARCHAR(256));
```

2. Create a new role called logreader:

```
=> CREATE ROLE logreader;
```

3. Grant the logreader role read-only access on the applog table:

```
=> GRANT SELECT ON applog TO logreader;
```

4. Create a new role called logwriter:

```
=> CREATE ROLE logwriter;
```

5. Grant the logwriter write access on the applog table:

```
=> GRANT INSERT ON applog to logwriter;
```

6. Create a new role called logadmin, which will rule the other two roles:

```
=> CREATE ROLE logadmin;
```

7. Grant the logadmin role privileges to delete data:

```
=> GRANT DELETE ON applog to logadmin;
```

8. Grant the logadmin role privileges to have the same privileges as the logreader and logwriter roles:

```
=> GRANT logreader, logwriter TO logadmin;
```

9. Create new user Bob:

```
=> CREATE USER Bob;
```

10. Give Bob logadmin privileges:

```
=> GRANT logadmin TO Bob;
```

The user Bob can now enable the logadmin role, which also includes the logreader and logwriter roles. Note that Bob cannot enable either the logreader or logwriter role directly. A user can only enable explicitly-granted roles.

Hierarchical roles also works with administrative access to a role:

```
=> GRANT logreader, logwriter TO logadmin WITH ADMIN OPTION;  
GRANT ROLE  
=> GRANT logadmin TO Bob;  
=> \c - bob; -- connect as Bob  
You are now connected as user "Bob".  
=> SET ROLE logadmin; -- Enable logadmin role  
SET  
=> GRANT logreader TO Alice;  
GRANT ROLE
```

Note that the user Bob only has administrative access to the logreader and logwriter roles through the logadmin role. He doesn't have administrative access to the logadmin role, since it wasn't granted to him with the optional WITH ADMIN OPTION argument:

```
=> GRANT logadmin TO Alice;  
WARNING: Some roles were not granted  
GRANT ROLE
```

For Bob to be able to grant the logadmin role, a superuser would have had to explicitly grant him administrative access.

See Also

- [About MC Privileges and Roles](#)

Creating Database Roles

A **superuser** creates a new role using the [CREATE ROLE](#) statement. Only a superuser can create or drop roles.

```
=> CREATE ROLE administrator;  
CREATE ROLE
```

The newly-created role has no privileges assigned to it, and no users or other roles are initially granted access to it. A superuser must [grant privileges](#) and [access](#) to the role.

Deleting Database Roles

A **superuser** can delete a role with the [DROP ROLE](#) statement.

Note that if any user or other role has been assigned the role you are trying to delete, the DROP ROLE statement fails with a dependency message.

```
=> DROP ROLE administrator;  
NOTICE: User Bob depends on Role administrator  
ROLLBACK: DROP ROLE failed due to dependencies  
DETAIL: Cannot drop Role administrator because other objects depend on it  
HINT: Use DROP ROLE ... CASCADE to remove granted roles from the dependent users/roles
```

Supply the optional CASCADE parameter to drop the role and its dependencies.

```
=> DROP ROLE administrator CASCADE;  
DROP ROLE
```

Granting Privileges to Roles

A **superuser** or owner of a schema, table, or other database object can assign privileges to a role, just as they would assign privileges to an individual user by using the GRANT statements described in the [SQL Reference Manual](#) . See [About Database Privileges](#) for information about which privileges can be granted.

Granting a privilege to a role immediately affects active user sessions. When you grant a new privilege, it becomes immediately available to every user with the role active.

Example

The following example creates two roles and assigns them different privileges on a single table called aplog.

1. Create a table called `applog`:

```
=> CREATE TABLE applog (id int, sourceID VARCHAR(32), data TIMESTAMP, event VARCHAR(256));
```

2. Create a new role called `logreader`:

```
=> CREATE ROLE logreader;
```

3. Assign read-only privileges to the `logreader` role on table `applog`:

```
=> GRANT SELECT ON applog TO logreader;
```

4. Create a role called `logwriter`:

```
=> CREATE ROLE logwriter;
```

5. Assign write privileges to the `logwriter` role on table `applog`:

```
=> GRANT INSERT ON applog TO logwriter;
```

See [SQL Reference Manual](#) for the different GRANT statements.

Revoking Privileges From Roles

Use one of the REVOKE statements to revoke a privilege from a role.

```
=> REVOKE INSERT ON applog FROM logwriter;  
REVOKE PRIVILEGE
```

Revoking a privilege immediately affects any user sessions that have the role active. When you revoke a privilege, it is immediately removed from users that rely on the role for the privilege.

See the [SQL Reference Manual](#) for the different REVOKE statements.

Granting Access to Database Roles

A superuser can assign any role to a user or to another role using the GRANT command. The simplest form of this command is:

```
GRANT role [, ...] TO { user | role } [, ...]
```

HP Vertica will return a NOTICE if you grant a role with or without admin option, to a grantee who has already been granted that role. For example:

```
=> GRANT commenter to Bob;  
NOTICE 4622: Role "commenter" was already granted to user "Bob"
```

See [GRANT \(Role\)](#) in the SQL Reference Manual for details.

Example

The following process illustrates how to create a role called commenter and granting user Bob access to that role.

1. Connect to the database as a superuser:

```
\c - dbadmin
```

2. Create a table called comments:

```
=> CREATE TABLE comments (id INT, comment VARCHAR);
```

3. Create a new role called commenter:

```
=> CREATE ROLE commenter;
```

4. Grant privileges to the new role on the comments table:

```
=> GRANT INSERT, SELECT ON comments TO commenter;
```

5. Grant the commenter role to user Bob.

```
=> GRANT commenter TO Bob;
```

Enable the newly-granted role

1. Connect to the database as user Bob

```
=> \c - Bob
```

2. User Bob enables the role:

```
=> SET ROLE commenter;
```

3. Now insert some values into the comments table:

```
=> INSERT INTO comments VALUES (1, 'Hello World');
```

Based on the privileges granted to Bob by the commenter role, Bob can insert and query the comments table.

4. Query the comments table:

```
=> SELECT * FROM comments;
 id |  comment
----+-----
  1 | Hello World
(1 row)
```

5. Commit the transaction:

```
=> COMMIT;
```

Note that Bob does not have proper permissions to drop the table:

```
=> DROP TABLE comments;ROLLBACK 4000: Must be owner of relation comments
```

See Also

- [Granting Database Access to MC Users](#)

Revoking Access From Database Roles

A **superuser** can revoke any role from a user or from another role using the REVOKE command. The simplest form of this command is:

```
REVOKE role [, ...] FROM { user | role | PUBLIC } [, ...]
```

See [REVOKE \(Role\)](#) in the SQL Reference Manual for details.

Example

To revoke access from a role, use the REVOKE (Role) statement:

1. Connect to the database as a superuser:

```
\c - dbadmin
```

2. Revoke the commenter role from user Bob:

```
=> REVOKE commenter FROM bob;
```

Granting Administrative Access to a Role

A **superuser** can assign a user or role administrative access to a role by supplying the optional `WITH ADMIN OPTION` argument to the `GRANT` statement. Administrative access allows the user to grant and revoke access to the role for other users (including granting them administrative access). Giving users the ability to grant roles lets a superuser delegate role administration to other users.

Example

The following example demonstrates granting the user bob administrative access to the commenter role, then connecting as bob and granting a role to another user.

1. Connect to the database as a superuser (or a user with administrative access):

```
=> \c - dbadmin
```

2. Grant administrative options on the commenter role to Bob

```
=> GRANT commenter TO Bob WITH ADMIN OPTION;
```

3. Connect to the database as user Bob

```
=> \c - Bob
```

4. As user Bob, grant the commenter role to Alice:

```
=> GRANT commenter TO Alice;
```

Users with administrative access to a role can also grant other users administrative access:

```
=> GRANT commenter TO alice WITH ADMIN OPTION;  
GRANT ROLE
```

As with all user privilege models, database superusers should be cautious when granting any user a role with administrative privileges. For example, if the database superuser grants two users a role with administrative privileges, both users can revoke the role of the other user. This example shows granting the `commenter` role (with administrative privileges) to users `bob` and `alice`. After each user has been granted the `commenter` role, either user can connect as the other will full privileges.

```
=> GRANT appadmin TO bob, alice WITH ADMIN OPTION;  
GRANT ROLE  
=> \connect - bob  
You are now connected as user "bob".  
=> REVOKE appadmin FROM alice;  
REVOKE ROLE
```

Revoking Administrative Access From a Role

A **superuser** can revoke administrative access from a role using the ADMIN OPTION parameter with the **REVOKE** statement. Giving users the ability to revoke roles lets a superuser delegate role administration to other users.

Example

The following example demonstrates revoking administrative access from Alice for the commenter role.

1. Connect to the database as a superuser (or a user with administrative access)

```
\c - dbadmin
```

2. Issue the REVOKE command with ADMIN OPTION parameters:

```
=> REVOKE ADMIN OPTION FOR commenter FROM alice;
```

Enabling Roles

By default, roles aren't enabled automatically for a user account. (See [Default Roles for Database Users](#) for a way to make roles enabled automatically.) Users must explicitly enable a role using the **SET ROLE** statement. When users enable a role in their session, they gain all of the privileges assigned to that role. Enabling a role does not affect any other roles that the users have active in their sessions. They can have multiple roles enabled simultaneously, gaining the combined privileges of all the roles they have enabled, plus any of the privileges that have been granted to them directly.

```
=> SELECT * FROM applog;  
ERROR: permission denied for relation applog  
  
=> SET ROLE logreader;  
SET  
  
=> SELECT * FROM applog;  
  
id | sourceID | data | event  
-----+-----+-----+-----
```



```
--
 1 | Loader   | 2011-03-31 11:00:38.494226 | Error: Failed to open source file
 2 | Reporter | 2011-03-31 11:00:38.494226 | Warning: Low disk space on volume /scratch-
a
(2 rows)
```

You can enable all of the roles available to your user account using the SET ROLE ALL statement.

```
=> SET ROLE ALL;SET
=> SHOW ENABLED_ROLES;

  name      |      setting
-----+-----
enabled roles | logreader, logwriter
(1 row)
```

See Also

- [Viewing a user's Role](#)

Disabling Roles

To disable all roles, use the SET ROLE NONE statement:

```
=> SET ROLE NONE;SET
=> SHOW ENABLED_ROLES;

  name      |      setting
-----+-----
enabled roles |
(1 row)
```

Viewing Enabled and Available Roles

You can list the roles you have enabled in your session using the [SHOW ENABLED ROLES](#) statement:

```
=> SHOW ENABLED_ROLES;

  name      |      setting
-----+-----
enabled roles | logreader
(1 row)
```

You can find the roles available to your account using the SHOW AVAILABLE ROLES statement:

```
Bob=> SHOW AVAILABLE_ROLES;

  name      |      setting
-----+-----
```

```
available roles | logreader, logwriter  
(1 row)
```

Viewing Named Roles

To view the names of all roles users can access, along with any roles that have been assigned to those roles, query the `V_CATALOG.ROLES` system table.

```
=> SELECT * FROM roles;  
   role_id | name | assigned_roles  
-----+-----+-----  
 45035996273704964 | public |  
 45035996273704966 | dbduser |  
 45035996273704968 | dbadmin | dbduser*  
 45035996273704972 | pseudosuperuser | dbadmin*  
 45035996273704974 | logreader |  
 45035996273704976 | logwriter |  
 45035996273704978 | logadmin | logreader, logwriter  
(7 rows)
```

Note: An asterisk (*) in the output means that role was granted WITH ADMIN OPTION.

Viewing a user's Role

The `HAS_ROLE()` function lets you see if a role has been granted to a user.

Non-superusers can check their own role membership using `HAS_ROLE('role_name')`, but only a **superuser** can look up other users' memberships using the `user_name` parameter. Omitting the `user_name` parameter will return role results for the superuser who is calling the function.

How to View a user's Role

In this example, user Bob wants to see if he's been assigned the logwriter command. The output returns boolean value *t* for true, denoting that Bob is assigned the specified logwriter role:

```
Bob=> SELECT HAS_ROLE('logwriter');  
HAS_ROLE  
-----  
t  
(1 row)
```

In this example, a superuser wants to verify that the logadmin role has been granted to user Ted:

```
dbadmin=> SELECT HAS_ROLE('Ted', 'logadmin');
```

The output returns boolean value *t* for true, denoting that Ted is assigned the specified logadmin role:

```
HAS_ROLE
-----
t
(1 row)
```

Note that if a superuser omits the `user_name` argument, the function looks up that superuser's role. The following output indicates that this superuser is not assigned the `logadmin` role:

```
dbadmin=> SELECT HAS_ROLE('logadmin');
HAS_ROLE
-----
f
(1 row)
```

Output of the function call with user `Alice` indicates that she is not granted the `logadmin` role:

```
dbadmin=> SELECT HAS_ROLE('Alice', 'logadmin');
HAS_ROLE
-----
f
(1 row)
```

To view additional information about users, roles and grants, you can also query the following system tables in the `V_CATALOG` schema to show directly-assigned roles:

- [ROLES](#)
- [GRANTS](#)
- [USERS](#)

Note that the system tables do not indicate whether a role is available to a user when roles could be available through other roles (indirectly). You need to call the `HAS_ROLE()` function for that information.

Users

This command returns all columns from the `USERS` system table:

```
=> SELECT * FROM users;-[ RECORD 1 ]
-----+-----
user_id      | 45035996273704962
user_name    | dbadmin
is_super_user | t
profile_name | default
is_locked    | f
lock_time    |
resource_pool | general
memory_cap_kb | unlimited
temp_space_cap_kb | unlimited
```

```
run_time_cap      | unlimited
all_roles         | dbadmin*, pseudosuperuser*default_roles
                  | dbadmin*, pseudosuperuser*
```

Note: An asterisk (*) in table output for all_roles and default_roles columns indicates a role granted WITH ADMIN OPTION.

Roles

The following command returns all columns from the ROLES system table:

```
=> SELECT * FROM roles;
   name      | assigned_roles
-----+-----
public      |
dbadmin     | dbduser*
pseudosuperuser | dbadmin
dbduser     |
(4 rows)
```

Note: The dbduser role in output above is internal only; you can ignore it.

Grants

The following command returns all columns from the GRANTS system table:

```
=> SELECT * FROM grants;
 grantor | privileges_description | object_schema | object_name | grantee
-----+-----+-----+-----+-----
dbadmin | USAGE                  |                | public      | public
dbadmin | USAGE                  |                | v_internal  | public
dbadmin | USAGE                  |                | v_catalog   | public
dbadmin | USAGE                  |                | v_monitor   | public
(4 rows)
```

Viewing User Roles on Management Console

You can see an MC user's roles and database resources through the **MC Settings > User management** page on the Management Console interface. For more information, see [About MC Privileges and Roles](#).

About MC Privileges and Roles

As introduced in [About MC Users](#), you control user access to Management Console through groups of privileges (also referred to as access levels) that fall into two types, those that apply to MC configuration, and those that apply to MC-managed HP Vertica databases.

MC Permission Groups

- [MC configuration](#) privileges are made up of roles that control what users can configure on the MC, such as modify MC settings, create/import HP Vertica databases, restart MC, create an HP Vertica cluster through the MC interface, and create and manage MC users.
- [MC database](#) privileges are made up of roles that control what users can see or do on an MC-managed HP Vertica database, such as view the database cluster state, query and session activity, monitor database messages and read log files, replace cluster nodes, and stop databases.

Note: When you grant an MC user a database role, that user inherits the privileges assigned to the database user account to which the MC user is mapped. For maximum access, use the dbadmin username and password.

MC database privileges cannot alter or override the HP Vertica database user's privileges and roles. MC user/database user association is described in [Mapping an MC User to a Database user's Privileges](#).

MC's Configuration Privileges and Database Access

The following table shows MC role-based users and summarizes the levels of access they have on the MC interface, as well as to any MC-managed databases.

| User type | MC config permissions | MC database permissions |
|-------------------------------------|--|---|
| MC administrators (SUPER and ADMIN) | Perform all administrative operations on MC, including configure and restart the MC process and add, change, and remove all user accounts. | Automatically inherit the database privileges of the main database user account used to set up one or more databases on the MC interface. By default, MC administrators have access to all MC-managed databases. |
| IT users (IT) | Monitor all MC-managed databases, view MC-level (non database) messages, logs, and alerts, disable or enable user access to MC, and reset non-LDAP user passwords. | Inherit no database privileges. You must grant the IT user access to one or more MC-managed databases, which you do by mapping this user to the database user account. The MC IT user then inherits the privileges assigned to the database user to which he/she is mapped. |

| User type | MC config permissions | MC database permissions |
|---|--|--|
| Database users (NONE) | Perform no administrative operations on MC. View and/or manage databases that you assign them. | Inherit no database privileges. You must grant the database (NONE) user access to one or more MC-managed databases, which you do by mapping this user to the database user account. The database user inherits the privileges assigned to the database user to which he/she is mapped. |
| Described in About MC Users | Described in MC Configuration Privileges | Described in MC Database Privileges |

See Also

- [About MC Users](#)
- [Creating an MC User](#)
- [Mapping an MC User to a Database user's Privileges](#)

MC Configuration Privileges

When you create an MC user, you assign them an MC configuration access level (role). For the most part, MC configuration permissions control a user's ability to create users and manage MC settings on the MC interface. You can grant a maximum of one role to each MC user, choosing from one of the following:

- **ADMIN Role (mc)**—Full access to all MC functionality, including any MC-managed database
- **IT Role (mc)**—Full access to all MC functionality, but database access is assigned
- **NONE Role (mc)**—Database access only, according to the databases an administrator assigns

You grant MC configuration permissions at the same time you create the user's account, through the **MC Settings** page. You can change MC access levels through the same page later, if necessary. See [Creating an MC User](#) for details.

You will also likely grant non-administrators (users with the IT and NONE roles) access to one or more MC-managed databases. See [MC Database Privileges](#) for details.

MC Configuration Privileges By User Role

The following table summarizes MC configuration permissions by role. For details, see each role in the above list.

| MC access privileges | ADMIN | IT | NONE |
|---|-------|-----|------|
| Configure MC settings: <ul style="list-style-type: none"> • Configure storage locations and ports • Upload an HP Vertica license • Upload new SSL certificates • Manage LDAP authentication | Yes | | |
| Create and manage databases and clusters <ul style="list-style-type: none"> • Create a new database or import an existing one • Create a new cluster or import an existing one • Remove database/cluster from the MC interface | Yes | | |
| Configure user settings: <ul style="list-style-type: none"> • Add, edit, delete users • Enable/disable user access to MC • Add, change, delete user permissions • Map users to one or more databases | Yes | | |
| Monitor user activity on MC | Yes | | |
| Reset MC to its original, preconfigured state | Yes | | |
| Restart Management Console | Yes | | |
| Disable or enable user access to MC interface | Yes | Yes | |
| Reset users' (non-LDAP) passwords | Yes | Yes | |
| Monitor all console-managed databases | Yes | Yes | |
| View MC log and non-database MC alerts | Yes | Yes | |

See Also

- [About MC Users](#)
- [About MC Privileges and Roles](#)

- [MC Database Privileges](#)
- [Creating an MC User](#)
- [Granting Database Access to MC Users](#)
- [Mapping an MC User to a Database user's Privileges](#)

SUPER Role (mc)

The default superuser administrator, called **Super** on the MC UI, is a Linux user account that gets created when you install and [configure MC](#). During the configuration process, you can assign the Super any name you like; it need not be dbadmin.

The MC SUPER role, a superset of the [ADMIN Role \(mc\)](#), has the following privileges:

- Oversees the entire Management Console, including all MC-managed database clusters

Note: This user inherits the privileges/roles of the user name supplied when importing an HP Vertica database into MC. HP recommends that you use the database administrator's credentials.

- Creates the first MC user accounts and assigns them an MC configuration role
- Grants MC users access to one or more MC-managed HP Vertica databases by assigning [MC Database Privileges](#) to each user

The MC super administrator account is unique. Unlike other MC users you create, including other MC administrators, the MC super account cannot be altered or dropped, and you cannot grant the SUPER role to other MC users. The only property you can change for the MC super is the password. Otherwise the SUPER role has the same privileges on MC as the [ADMIN Role \(mc\)](#).

On MC-managed HP Vertica databases, SUPER has the same privileges as [ADMIN Role \(db\)](#).

The MC super account does not exist within the LDAP server. This account is also different from the special dbadmin account that gets created during an HP Vertica installation, whose privileges are governed by the [DBADMIN Role](#). The HP Vertica-created dbadmin is a Linux account that owns the database catalog and storage locations and can bypass database authorization rules, such as creating or dropping schemas, roles, and users. The MC super does not have the same privileges as dbadmin.

See Also

- [Configuring MC](#)
- [About MC Privileges and Roles](#)

- [Creating an MC User](#)
- [Granting Database Access to MC Users](#)
- [Adding Multiple Users to MC-managed Databases](#)
- [Mapping an MC User to a Database user's Privileges](#)
- [Managing MC Users](#)

ADMIN Role (mc)

This user account is the user who can perform all administrative operations on Management Console, including configure and restart the MC process and add, change, and remove all user accounts. By default, MC administrators inherit the database privileges of the main database user account used to set up the database on the MC interface. Therefore, MC administrators have access to all MC-managed databases. Grant the ADMIN role to users you want to be MC administrators.

The difference between this ADMIN user and the default Linux account, the MC [SUPER role](#), is you cannot alter or delete the MC SUPER account, and you can't grant the SUPER role to any other MC users. You can, however, change the access level for other MC administrators, and you can delete this user's accounts from the MC interface.

The following list highlights privileges granted to the ADMIN role:

- Modify MC settings, such as storage locations and ports, restart the MC process, and reset MC to its original, unconfigured state
- Audit license activity and install/upgrade an HP Vertica license
- Upload a new SSL certificate
- Use LDAP for user authentication
- View the MC log, alerts and messages
- Add new users and map them to one or more HP Vertica databases by granting an [MC database-level role](#)
- Select a database and add multiple users at once
- Manage user roles and their access to MC
- Remove users from the MC
- Monitor user activity on the MC interface
- Stop and start any MC-managed database

- Create new databases/clusters and and import existing databases/clusters into MC
- Remove databases/clusters from the MC interface
- View all databases/clusters imported into MC

About the MC Database Administrator Role

There is also an MC database administrator (ADMIN) role that controls a user's access to MC-managed databases. The two ADMIN roles are similar, but they are not the same, and you do not need to grant users with the ADMIN (mc) role an ADMIN (db) role because MC ADMIN users automatically inherit all database privileges of the main database user account that was created on or imported into MC.

The following table summarizes the primary difference between the two ADMIN roles, but see [ADMIN Role \(db\)](#) for details specific to MC-managed database administrators.

| MC configuration ADMIN role | MC database ADMIN role |
|---|---|
| Perform all administrative operations on the MC itself, including restarting the MC process. Privileges extend to monitoring all MC-created and imported databases but anything database-related beyond that scope depends on the user's privileges granted on the database through GRANT statements. | Perform database-specific activities, such as stop and start the database, and monitor query and user activity and resources. Other database operations depend on that user's privileges on the specific database. This ADMIN role cannot configure MC. |

See Also

- [About MC Privileges and Roles](#)
- [ADMIN Role \(db\)](#)
- [Creating an MC User](#)
- [Granting Database Access to MC Users](#)
- [Adding Multiple Users to MC-managed Databases](#)
- [Mapping an MC User to a Database user's Privileges](#)
- [Managing MC Users](#)

IT Role (mc)

MC IT users can monitor all MC-managed databases, view MC-level (non database) messages, logs, and alerts, disable or enable user access to MC, and reset non-LDAP user passwords. You can also assign MC IT users specific database privileges, which you do by mapping IT users to a

user on a database. In this way, the MC IT user inherits the privileges assigned to the database user to which he/she is mapped.

About the MC IT (database) Role

There is also an IT database administrator (IT) role that controls a user's access to MC-managed databases. If you grant an MC user both IT roles, it means the user can perform some configuration on MC and also has access to one or more MC-managed databases. The database mapping is not required, but it gives the IT user wider privileges.

The two IT roles are similar, but they are not the same. The following table summarizes the primary difference between them, but see [IT Role \(db\)](#) for details.

| MC configuration IT role | MC database IT role |
|---|---|
| Monitor MC-managed database, view non-database messages, and manage user access | Monitor databases on which the user has privileges, view the database overview and activity pages, monitor the node state view messages and mark them read/unread, view database settings. Can also be mapped to one or more HP Vertica databases. |

See Also

- [About MC Privileges and Roles](#)
- [IT Role \(db\)](#)
- [Mapping an MC User to a Database user's Privileges](#)

NONE Role (mc)

The default role for all newly-created users on MC is NONE, which prevents users granted this role from configuring the MC. When you create MC users with the NONE role, you grant them an [MC database-level role](#). This assignment maps the MC user to a user account on a specific database and specifies that the NONE user inherits the database user's privileges to which he or she is mapped.

Which database-level role you grant this user with NONE privileges—whether ADMIN (db) or IT (db) or USER (db)—depends on the level of access you want the user to have on the MC-managed database. Database roles have no impact on the ADMIN and IT roles at the MC configuration level.

See Also

- [About MC Privileges and Roles](#)
- [About MC Users](#)

- [MC Database Privileges](#)
- [ADMIN Role \(db\)](#)
- [IT Role \(db\)](#)
- [USER Role \(db\)](#)

MC Database Privileges

When you [create MC users](#), you first assign them [MC configuration](#) privileges, which controls what they can do on the MC itself. In the same user-creation operation, you grant access to one or more MC-managed databases. MC database access does not give the MC user privileges directly on HP Vertica; it provides MC users varying levels of access to assigned database functionality through the MC interface.

Assign users an MC database level through one of the following roles:

- [ADMIN Role \(db\)](#)—Full access to all MC-managed databases. Actual privileges ADMINs inherit depend on the database user account used to create or import the HP Vertica database into the MC interface.
- [IT Role \(db\)](#)—Can start and stop a database but cannot remove it from the MC interface or drop it.
- [USER Role \(db\)](#)—Can only view database information through the database Overview and Activities pages but is restricted from viewing more detailed data.

When you assign an MC database level to an MC user, you need to map the MC user account to a database user account. Mapping lets the MC user inherit the privileges assigned to that database user and ensures that the MC user cannot do or see anything that is not allowed by the privileges set up for the user account on the server database.

Privileges assigned to the database user always supersede privileges of the MC user if there is a conflict, such as stopping a database. When the MC user logs in to MC, using his or her MC user name and password, MC privileges for database-related activities are compared to the user privileges on the database itself (the account you mapped the MC user to). Only when the user has both MC privileges and corresponding database privileges will the operations be exposed to that user in the MC interface.

Tip: As a best practice, you should identify, in advance, the appropriate HP Vertica database user account that has privileges and/or roles similar to one of the MC database roles.

See [Creating an MC User](#) and [Mapping an MC User to a Database user's Privileges](#) for more information.

MC Database Privileges By Role

The following tables summarizes MC configuration-level privileges by user role. The first table shows the default privileges, and the second table shows, for the ADMIN role only, which operations are dependent on the database user account's privileges and/or roles itself.

| Default database-level privileges | ADMIN | IT | USER |
|--------------------------------------|-------|-----|------|
| View messages | Yes | Yes | Yes |
| Delete messages and mark read/unread | Yes | Yes | |
| View database Overview page | Yes | Yes | Yes |
| View database Activity page | Yes | Yes | Yes |
| View database grid page | Yes | Yes | Yes |
| Start a database | Yes | | |
| Stop a node | Yes | | |
| View node state | Yes | Yes | |
| View MC settings | Yes | Yes | |

Privileges governed by the HP Vertica database user account:

| Database-specific privileges | ADMIN |
|--|-------|
| Audit license activity | Yes |
| Install new license | Yes |
| View WLA tuning recommendations | Yes |
| View database query page | Yes |
| Stop a database | Yes |
| Rebalance a database | Yes |
| Drop a database | Yes |
| Start, replace, add, remove nodes | Yes |
| Modify database settings | Yes |

See Also

- [About MC Users](#)
- [About MC Privileges and Roles](#)
- [MC Configuration Privileges](#)

ADMIN Role (db)

ADMIN is a **superuser** with full privileges to monitor MC-managed database activity and messages. Other database privileges (such as stop or drop the database) are governed by the user account on the HP Vertica database that this ADMIN (db) user is mapped to. ADMIN is the most permissive role and is a superset of privileges granted to the [IT](#) and [USER](#) roles.

The ADMIN user has the following database privileges by default:

- View and delete database messages
- Mark messages read or unread
- View the database overview (grid) page
- View the database activity page
- Start the database
- View database cluster node state
- View database settings

The following MC-managed database operations depend on the database user's role that you mapped this ADMIN user to:

- View license information
- Install a new license
- View **Workload Analyzer** tuning recommendations
- View query activity and loads
- Stop the database
- Rebalance the database
- Add, stop, replace, or remove nodes
- Manage database settings

Note: Database access granted through Management Console never overrides roles granted on a specific HP Vertica database.

About the ADMIN (MC configuration) Role

There is also an MC configuration administrator role that defines what the user can change on the MC itself. The two ADMIN roles are similar, but they are not the same. Unlike the MC configuration role of ADMIN, which can manage all MC users and all databases imported into the UI, the MC database ADMIN role has privileges only on the databases you map this user to. The following table summarizes the primary difference between them, but see [ADMIN Role \(mc\)](#) for additional details.

| MC database ADMIN role | MC configuration ADMIN role |
|---|---|
| Perform database-specific activities, such as stop and start the database, and monitor query and user activity and resources. Other database operations depend on that user's privileges on the specific database. This ADMIN role cannot configure MC. | Perform all administrative operations on the MC itself, including restarting the MC process. Privileges extend to monitoring all MC-created and imported databases but anything database-related beyond that scope depends on the user's privileges granted on the database through GRANT statements. |

See Also

- [About MC Privileges and Roles](#)
- [ADMIN Role \(mc\)](#)

IT Role (db)

IT can view most details about an MC-managed database, such as messages (and mark them read/unread), the database overall health and activity/resources, cluster and node state, and MC settings. You grant and manage user role assignments through the **MC Settings > User management** page on the MC.

About the IT (MC configuration) Role

There is also an IT role at the MC configuration access level. The two IT roles are similar, but they are not the same. If you grant an MC user both IT roles, it means the user can perform some configuration on MC and also has access to one or more MC-managed databases. The following table summarizes the primary difference between them, but see [IT Role \(mc\)](#) for additional details.

| MC database IT | MC configuration IT |
|--|--|
| Monitor databases on which the user has privileges, view the database overview and activity pages, monitor the node state view messages and mark them read/unread, view database settings. | Monitor MC-managed database, view non-database messages, and manage user access. |

See Also

- [About MC Privileges and Roles](#)
- [IT Role \(mc\)](#)
- [Mapping an MC User to a Database user's Privileges](#)

USER Role (db)

USER has limited database privileges, such as viewing database cluster health, activity/resources, and messages. MC users granted the USER database role might have higher levels of permission on the MC itself, such as the [IT Role \(mc\)](#). Alternatively, USER users might have no (NONE) privileges to configure MC. How you combine the two levels is up to you.

See Also

- [About MC Privileges and Roles](#)
- [MC Configuration Privileges](#)
- [Mapping an MC User to a Database user's Privileges](#)

Granting Database Access to MC Users

If you did not grant an MC user a [database-level role](#) when you created the user account, this procedure describes how to do so.

Granting the user an MC database-level role associates the MC user with a database user's privileges and ensures that the MC user cannot do or see anything that is not allowed by the privileges set up for the user account on the server database. When that MC user logs in to MC, his or her MC privileges for database-related activities are compared to that user's privileges on the database itself. Only when the user has both MC privileges and corresponding database privileges will the operations be exposed in the MC interface. See [Mapping an MC User to a Database user's Privileges](#) for examples.

Prerequisites

Before you grant database access to an MC user, make sure you have read the prerequisites in [Creating an MC User](#).

Grant a Database-Level Role to an MC user:

1. Log in to Management Console as an administrator and navigate to **MC Settings > User management**.
2. Select an MC user and click **Edit**.
3. Verify the [MC Configuration Privileges](#) are what you want them to be. NONE is the default.
4. Next to the **DB access levels section**, click **Add** and provide the following database access credentials:
 - i. **Choose a database.** Select a database from the list of MC-discovered (databases that were created on or imported into the MC interface).
 - ii. **Database username.** Enter an existing database user name or, if the database is running, click the ellipses [...] to browse for a list of database users, and select a name from the list.
 - iii. **Database password.** Enter the password to the database user account (not this username's password).
 - iv. **Restricted access.** Chose a database level ([ADMIN](#), [IT](#), or [USER](#)) for this user.
 - v. Click **OK** to close the **Add permissions** dialog box.
5. Optionally change the user's **Status** (enabled is the default).
6. Click **Save**.

See [Mapping an MC User to a Database user's Privileges](#) for a graphical illustration of how easy it is to map the two user accounts.

How MC Validates New Users

After you click OK to close the Add permissions dialog box, MC tries to validate the database username and password entered against the selected MC-managed database or against your organization's LDAP directory. If the credentials are found to be invalid, you are asked to re-enter them.

If the database is not available at the time you create the new user, MC saves the username/password and prompts for validation when the user accesses the Database and Clusters page later.

See Also

- [About MC Users](#)
- [About MC Privileges and Roles](#)
- [Creating an MC User](#)
- [Creating a Database User](#)
- [Adding Multiple Users to MC-managed Databases](#)

Mapping an MC User to a Database user's Privileges

Database **mapping** occurs when you link one or more MC user accounts to a database user account. After you map users, the MC user inherits privileges granted to the database user, up to the limitations of the user's database access level on MC.

This topic presents the same mapping information as in [Granting Database Access to MC Users](#) but with graphics. See also [MC Database Privileges](#) for an introduction to database mapping through the MC interface and details about the different database access roles you can grant to an MC user.

How to Map an MC User to a Database User

The following series of images shows you how easy it is to map an MC user to a database user account from the **MC Settings > User management** page.

You view the list of MC users so you can see who has what privileges. You notice that user alice has no database privileges, which would appear under the Resources column.

| Status | MC username | MC access level | Resources |
|--------|-------------|-----------------|-----------------------------------|
| ● | alice | None | |
| ● | bob | Admin | db1, DB4, DB2, db3, mcdb, KellyDB |
| ● | carol | IT | |
| ● | dbadmin | Super | db1, DB4, DB2, db3, mcdb, KellyDB |
| ● | ed | None | |

MC user alice currently has no db access

To give alice database privileges, click to highlight her MC username, click **Edit**, and the **Edit existing user** page displays with no resources (databases) assigned to MC user alice.

Edit existing user

MC username:

MC password:

Email address:

MC configuration permissions:

DB access levels:

| Database | Username | Restrict |
|----------------------------|----------|----------|
| No data available in table | | |

Status:

Click **Add**, and when the **Add permissions** dialog box opens, choose a database from the menu.

Edit existing user

MC username:

MC password:

Email address:

MC configuration permissions:

DB access levels:

Add permissions

Choose a database:

Database username:

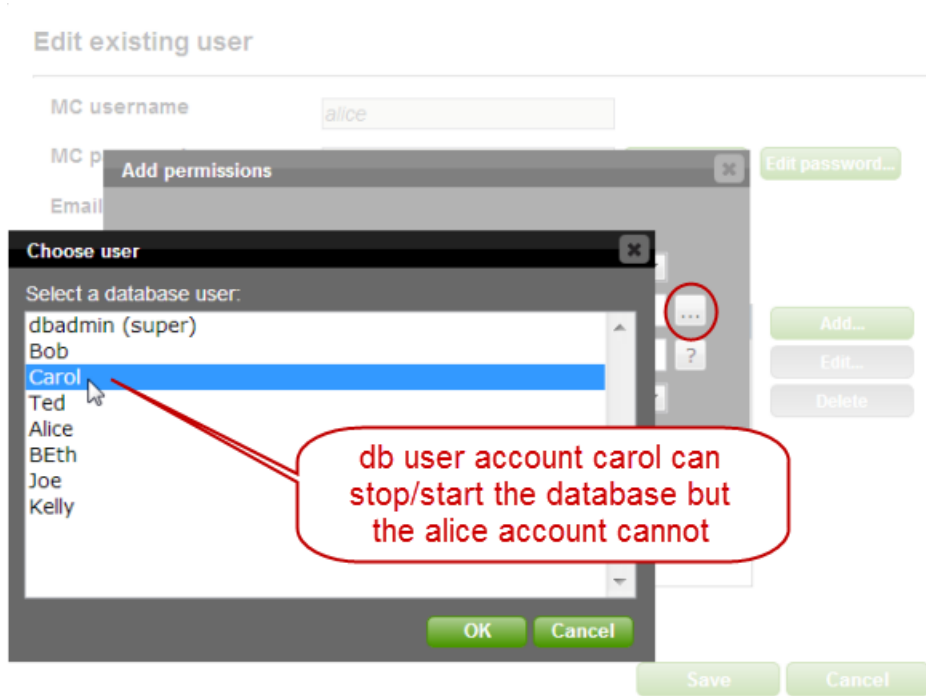
Database password:

Restrict access:

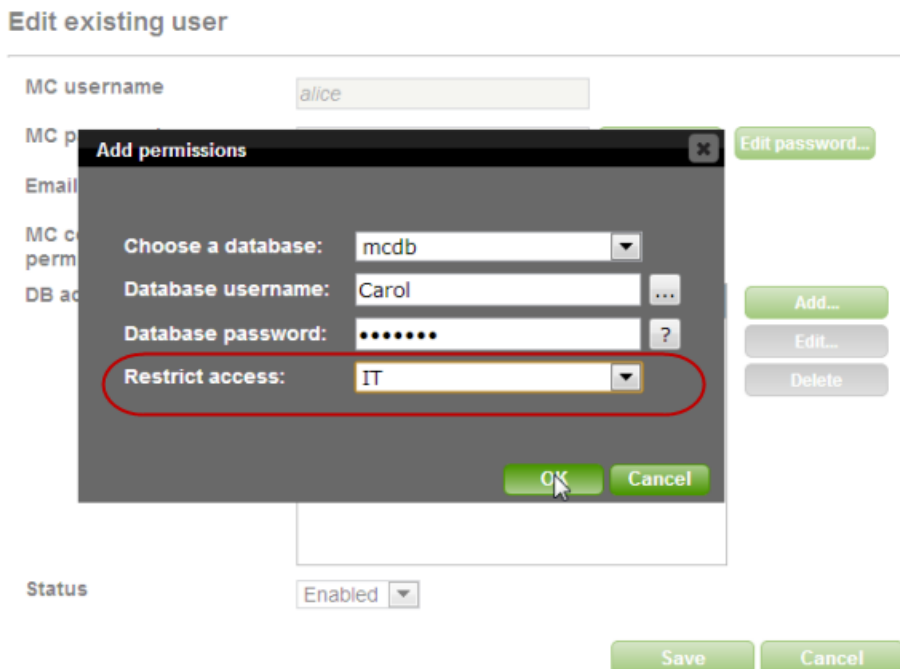
Status:

Select the database you want alice to access

In the same **Add permissions** dialog box, after you select a database, you need to enter the user name of the database user account that you want to map alice to. To see a list of database user names, click the ellipses [...] and select a name from the list. In this example, you already know that database user carol has privileges to stop and start the database, but the alice database account can only view certain tables. On MC, you want alice to have similar privileges to carol, so you map MC alice to database carol.



After you click OK, remember to assign MC user alice an MC database level. In this case, choose IT, a role that has permissions to start and stop the selected database.



Enter the database password, click **OK**, close the confirmation dialog box, and click **Save**.

That's it.

What If You Map the Wrong Permissions

In the following mapping example, if you had granted Alice MC database access level of ADMIN but mapped her to a database account with only USER-type privileges, Alice's access to that database would be limited to USER privileges. This is by design. When Alice logs in to MC using her own user name and password, MC privileges for her ADMIN-assigned role are compared to the user privileges on the database itself. Only when the user has both MC privileges and corresponding database privileges will the appropriate operations be exposed in the MC interface.

Edit existing user MC user Alice

MC username:

MC password: Generate new Edit password...

Email address:

MC configuration permissions: None Alice cannot configure MC, which means she has database access only

DB access levels:

| Database | Username | Access level |
|----------|----------|--------------|
| KellyDB | KoKo | User |
| mcdb | Alice | Admin |

Alice mapped to database user KoKo's privileges on KellyDB, which grants Alice USER database privileges on KellyDB through MC interface

Alice mapped to her own database user account on mcdb, which grants her ADMIN database privileges on mcdb through MC

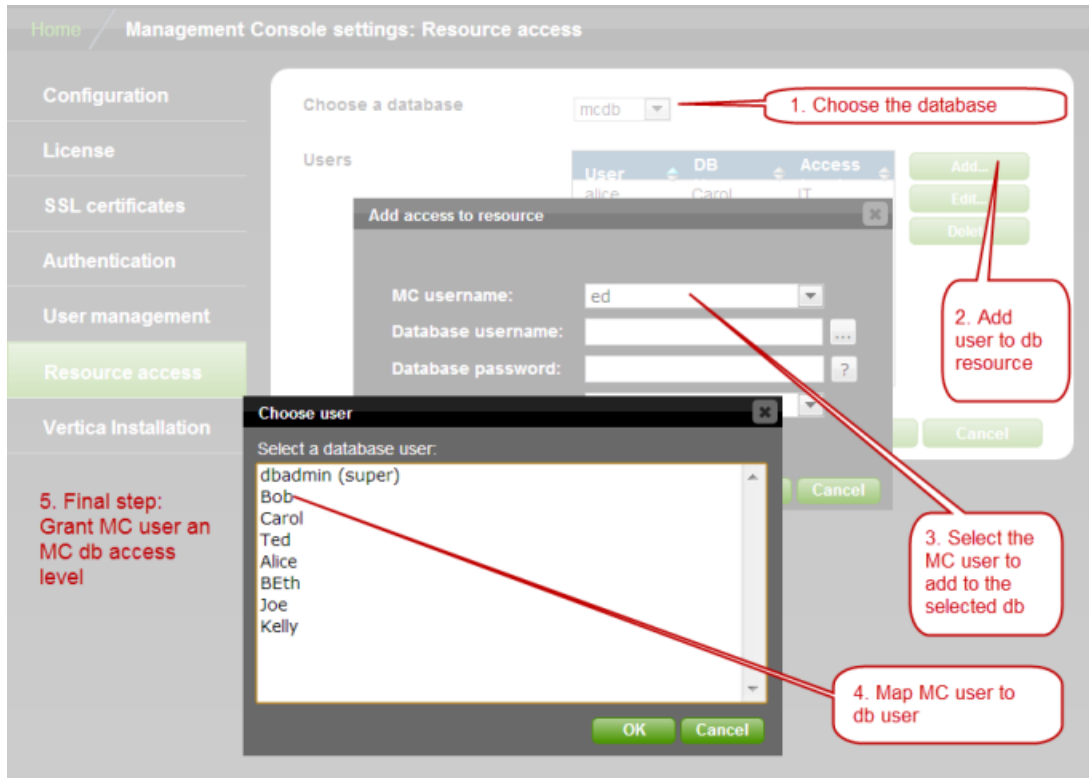
Enabled Save Cancel

Adding Multiple MC Users to a Database

In addition to creating or editing MC users and mapping them to a selected database, you can also select a database and add users to that database on the **MC Settings > Resource access** page.

Choose a database from the list, click **Add**, and select an MC user name, one at a time. Map the MC user to the database user account, and then grant each MC user the database level you want him or her to have.

It is possible you will grant the same database access to several MC users.



See [Granting Database Access to MC Users](#) and [Mapping an MC User to a Database user's Privileges](#) for details.

How to Find Out an MC user's Database Role

On the User management page, the **Resources** column lists all of the databases a user is mapped to. It does not, however, display the user's [database access level](#) (role).

| Status | MC username | MC access level | Resources |
|-------------------------------------|-------------|-----------------|---------------|
| <input checked="" type="checkbox"/> | Alice | None | mcdB, KellyDB |
| <input checked="" type="checkbox"/> | Bob | None | mcdB |
| <input checked="" type="checkbox"/> | Carol | IT | mcdB |
| <input checked="" type="checkbox"/> | dbadmin | Super | mcdB, KellyDB |
| <input checked="" type="checkbox"/> | Joe | Admin | mcdB, KellyDB |
| <input checked="" type="checkbox"/> | Kelly | None | mcdB, KellyDB |
| <input checked="" type="checkbox"/> | Koko | None | |

Shows databases mapped to each MC user but not roles on that

Show enabled Show disabled

You can retrieve that information by highlighting a user and clicking **Edit**. In the dialog box that opens (shown in example below), Bob's role on the mcdb database is ADMIN. You can change Bob's role from this dialog box by clicking Edit and assigning a different database-access role.

Edit existing user

MC username

MC password

Email address

MC configuration permissions

DB access levels

| Database | Username | Access level |
|----------|----------|--------------|
| mcdb | Bob | Admin |

Status

Adding Multiple Users to MC-managed Databases

If you are administering one or more MC-managed databases, and several MC users need access to it, you have two options on the **MC Settings** page:

- From the **User management** option, select each user and grant database access, one user at a time
- From the **Resource access** option, select a database first and add users to it

This procedure describes how to add several users to one database at once. If you want to add users one at a time, see [Creating an MC User](#).

Before You Start

Read the prerequisites in [Creating an MC User](#).

How to Add Multiple Users to a Database

1. Log in to MC as an administrator and navigate to **MC Settings > Resource access**.
2. **Choose a database** from the list of discovered databases. Selecting the database populates a table with users who already have privileges on the selected database.
3. To add new users, click **Add** and select the **MC username** you want to add to the database from the drop-down list.
4. Enter an existing **Database username** on the selected database or click the ellipses button [...] to browse for names. (This is the database account you want to map the selected user to.)
5. Enter the **database password** (not this username's password).

Note: The database password is generally the dbadmin superuser's password.

6. Choose a **database-access** role (**ADMIN** or **IT** or **USER**) for this user.
7. Click **OK** to close the Add access to resource dialog box.
8. Perform steps 3-7 for each user you want to add to the selected database, and then click **Save**.

See Also

- [About MC Users](#)
- [About MC Privileges and Roles](#)
- [Mapping an MC User to a Database user's Privileges](#)

MC Mapping Matrix

The following table shows the three different [MC configuration](#) roles, ADMIN, IT, and NONE, combined with the type of privileges a user granted that role inherits when mapped to a specific [database-level](#) role.

| MC configuration level | MC database level | The combination lets the user ... |
|------------------------|-------------------|--|
| ADMIN Role (mc) | All (implicit) | <ul style="list-style-type: none"> Perform all administrative operations on Management Console, including configure and restart the MC process. Maximum access to all databases created and/or imported into the MC interface—governed by the privileges associated with the database user account used to set up the database on the MC. |
| IT Role (mc) | ADMIN Role (db) | <ul style="list-style-type: none"> Monitor MC-managed database activity. View non-database messages. Manage user access (enable/disable). Monitor MC-managed database activity and messages. Other database privileges (such as stop or drop the database) are governed by the mapped user account on the database itself. Automatically inherits all privileges granted to the NONE:IT combination. |
| IT Role (mc) | IT Role (db) | <ul style="list-style-type: none"> Monitor MC-managed database activity. View non-database messages. Manage user access (edit/enable/disable). On databases where granted privileges, monitor database overview and activity, monitor node state, view messages and mark them read/unread, view database settings Automatically inherits all privileges granted to the IT:USER combination. |
| IT Role (mc) | USER Role (db) | <ul style="list-style-type: none"> Monitor MC-managed database activity View non-database messages. Manage user access (enable/disable). Viewing database cluster health, activity/resources, and messages and alerts. |

| MC configuration level | MC database level | The combination lets the user ... |
|------------------------|-------------------|---|
| NONE Role (mc) | ADMIN Role (db) | <ul style="list-style-type: none"> • No privileges to monitor/modify anything related to the MC itself. • Monitor MC-managed database activity, node state, and messages. • Other database privileges (such as stop or drop the database) are governed by the mapped user account on the database itself. • Automatically inherits all privileges granted to the NONE:IT combination. |
| NONE Role (mc) | IT Role (db) | <ul style="list-style-type: none"> • No privileges to monitor/modify anything related to the MC itself • Monitor MC-managed database activity, node state, and settings. • View the database overview and activity pages. • View messages and mark them read/unread. • Automatically inherits all privileges granted to the NONE:USER combination. |
| NONE Role (mc) | USER Role (db) | <ul style="list-style-type: none"> • No privileges to monitor/modify anything related to the MC itself. • View database cluster health, activity/resources, and messages and alerts. |

Using the Administration Tools

HP Vertica provides a set of tools that allows you to perform administrative tasks quickly and easily. Most of the database administration tasks in HP Vertica can be done using the Administration Tools.

Always run the Administration Tools using the **Database Administrator** account on the **Administration host**, if possible. Make sure that no other Administration Tools processes are running.

If the Administration host is unresponsive, run the Administration Tools on a different node in the cluster. That node permanently takes over the role of Administration host.

A man page is available for admintools. If you are running as the dbadmin user, simply type: `man admintools`. If you are running as a different user, type: `man -M /opt/vertica/man admintools`.

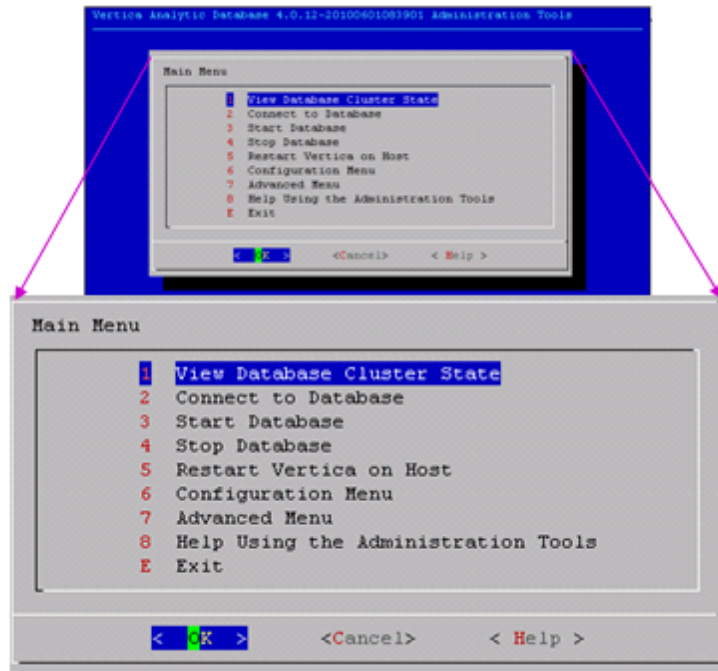
Running the Administration Tools

At the Linux command line:

```
$ /opt/vertica/bin/admintools [ -t | --tool ] toolname [ options ]
```

| | | |
|----------|---|---|
| toolname | Is one of the tools described in the Administration Tools Reference . | |
| options | -h--help | Shows a brief help message and exits. |
| | -a--help_all | Lists all command-line subcommands and options as described in Writing Administration Tools Scripts . |

If you omit toolname and options parameters, the Main Menu dialog box appears inside your console or terminal window with a dark blue background and a title on top. The screen captures used in this documentation set are cropped down to the dialog box itself, as shown below.



If you are unfamiliar with this type of interface, read [Using the Administration Tools Interface](#) before you do anything else.

First Time Only

The first time you log in as the **Database Administrator** and run the Administration Tools, the user interface displays.

1. In the EULA (end-user license agreement) window, type **accept** to proceed.

A window displays, requesting the location of the license key file you downloaded from the HP Web site. The default path is `/tmp/vlicense.dat`.

2. Type the absolute path to your license key (for example, `/tmp/vlicense.dat`) and click **OK**.

Between Dialogs

While the Administration Tools are working, you see the command line processing in a window similar to the one shown below. Do not interrupt the processing.

```
*** Creating database: Stock_Multi ***
Running integrity checks
chroot: changing permissions of '/opt/vertica/config/share/partinfo.dat': Operation
not permitted
Will create database on port 5435
Checking that nodes are defined and installed
stock_multi_node_0 OK [vertica111.2.0112007006230200001]
stock_multi_node_1 OK [vertica111.2.0112007006230200001]
stock_multi_node_2 OK [vertica111.2.0112007006230200001]
Checking full connectivity
Checking/updating replication layer
Spreading daemons processing
Verifying spread has been enabled on all nodes
Minimised initiator node stock_multi_node_0
Creating catalog and data directories
Starting DB in multi-node mode
Creating database Stock_Multi
Participating hosts:
  qa0
  qa1
  qa2
processing host qa0
processing host qa1
processing host qa2
Node Status: stock_multi_node_0: (UP)
Creating database nodes
Node Status: stock_multi_node_0: (UP) stock_multi_node_1: (UP) stock_multi_node_2: (UP)
Multi-node start returns: 1
Multi-node DB create completed
Replicating configuration to all nodes
```

Using the Administration Tools Interface

The HP Vertica Administration Tools are implemented using **Dialog**, a graphical user interface that works in terminal (character-cell) windows. The interface responds to mouse clicks in some terminal windows, particularly local Linux windows, but you might find that it responds only to keystrokes. Thus, this section describes how to use the Administration Tools using only keystrokes.

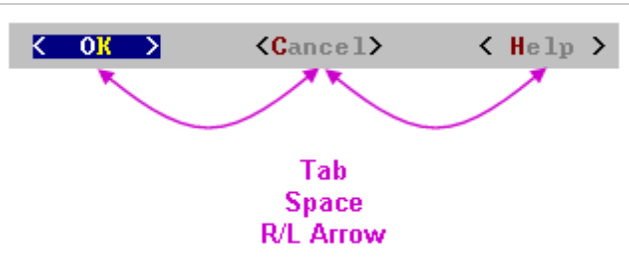
Note: This section does not describe every possible combination of keystrokes you can use to accomplish a particular task. Feel free to experiment and to use whatever keystrokes you prefer.

Enter [Return]

In all dialogs, when you are ready to run a command, select a file, or cancel the dialog, press the **Enter** key. The command descriptions in this section do not explicitly instruct you to press Enter.

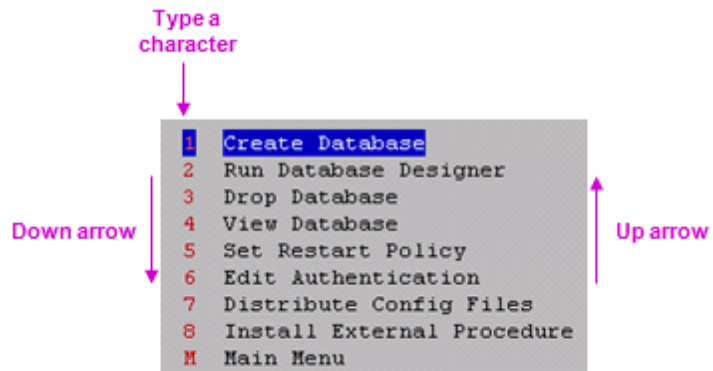
OK - Cancel - Help

The OK, Cancel, and Help buttons are present on virtually all dialogs. Use the tab, space bar, or right and left arrow keys to select an option and then press Enter. The same keystrokes apply to dialogs that present a choice of Yes or No.



Menu Dialogs

Some dialogs require that you choose one command from a menu. Type the alphanumeric character shown or use the up and down arrow keys to select a command and then press Enter.



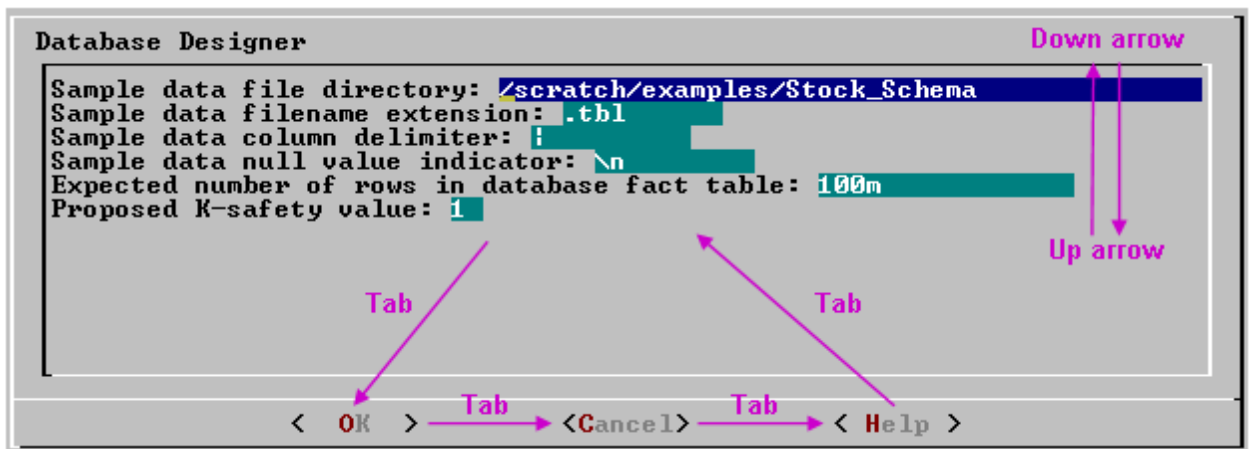
List Dialogs

In a list dialog, use the up and down arrow keys to highlight items, then use the space bar to select the items (which marks them with an X). Some list dialogs allow you to select multiple items. When you have finished selecting items, press Enter.



Form Dialogs

In a form dialog (also referred to as a dialog box), use the tab key to cycle between **OK**, **Cancel**, **Help**, and the form field area. Once the cursor is in the form field area, use the up and down arrow keys to select an individual field (highlighted) and enter information. When you have finished entering information in all fields, press Enter.



Help Buttons

Online help is provided in the form of text dialogs. If you have trouble viewing the help, see [Notes for Remote Terminal Users](#) in this document.

K-Safety Support in Administration Tools

The Administration Tools allow certain operations on a **K-Safe** database, even if some nodes are unresponsive.

The database must have been marked as K-Safe using the [MARK_DESIGN_KSAFE](#) function.

The following management functions within the Administration Tools are operational when some nodes are unresponsive.

Note: HP Vertica users can perform much of the below functionality using the Management Console interface. See [Management Console and Administration Tools](#) for details.

- View database cluster state
- Connect to database
- Start database (including manual recovery)
- Stop database
- Replace node (assuming node that is down is the one being replaced)
- View database parameters
- Upgrade license key

The following operations work with unresponsive nodes; however, you might have to repeat the operation on the failed nodes after they are back in operation:

- Edit authentication
- Distribute config files
- Install external procedure
- (Setting) database parameters

The following management functions within the Administration Tools require that all nodes be UP in order to be operational:

- Create database
- Run the Database Designer

- Drop database
- Set restart policy
- Roll back database to **Last Good Epoch**

Notes for Remote Terminal Users

The appearance of the graphical interface depends on the color and font settings used by your terminal window. The screen captures in this document were made using the default color and font settings in a PuTTY terminal application running on a Windows platform.

Note: If you are using a remote terminal application, such as PuTTY or a Cygwin bash shell, make sure your window is at least 81 characters wide and 23 characters high.

If you are using PuTTY, you can make the Administration Tools look like the screen captures in this document:

1. In a PuTTY window, right click the title area and select Change Settings.
2. Create or load a saved session.
3. In the Category dialog, click Window > Appearance.
4. In the Font settings, click the Change... button.
5. Select Font: Courier New: Regular Size: 10
6. Click Apply.

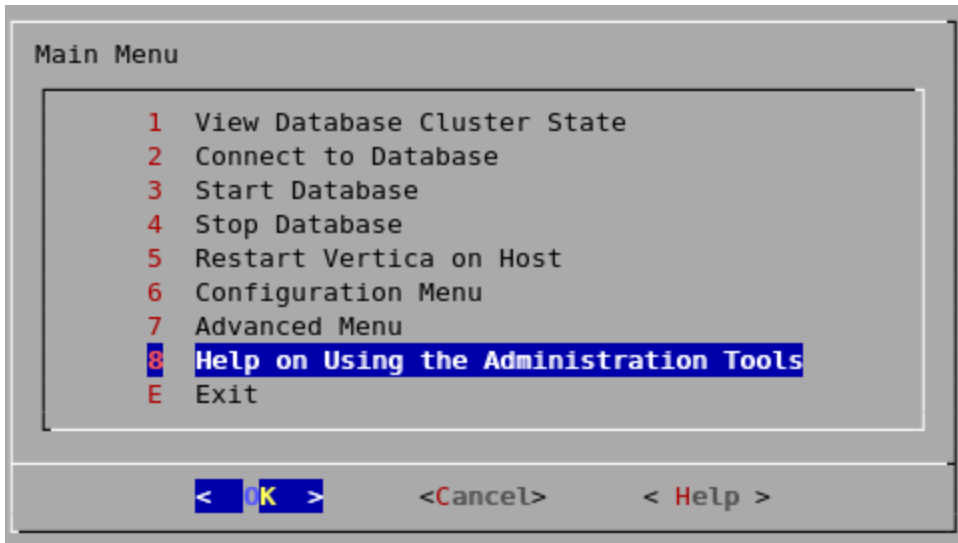
Repeat these steps for each existing session that you use to run the Administration Tools.

You can also change the translation to support UTF-8:

1. In a PuTTY window, right click the title area and select Change Settings.
2. Create or load a saved session.
3. In the Category dialog, click Window > Translation.
4. In the "Received data assumed to be in which character set" drop-down menu, select UTF-8.
5. Click Apply.

Using the Administration Tools Help

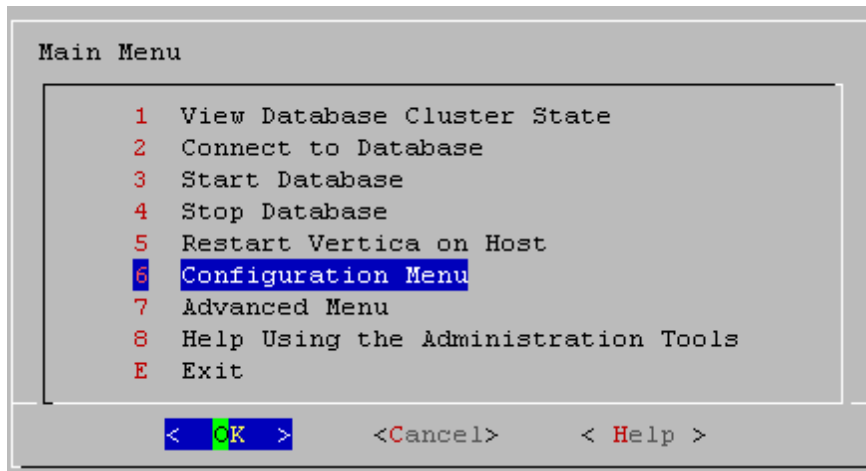
The **Help on Using the Administration Tools** command displays a help screen about using the Administration Tools.



Most of the online help in the Administration Tools is context-sensitive. For example, if you use up/down arrows to select a command, press tab to move to the Help button, and press return, you get help on the selected command.

In a Menu Dialog

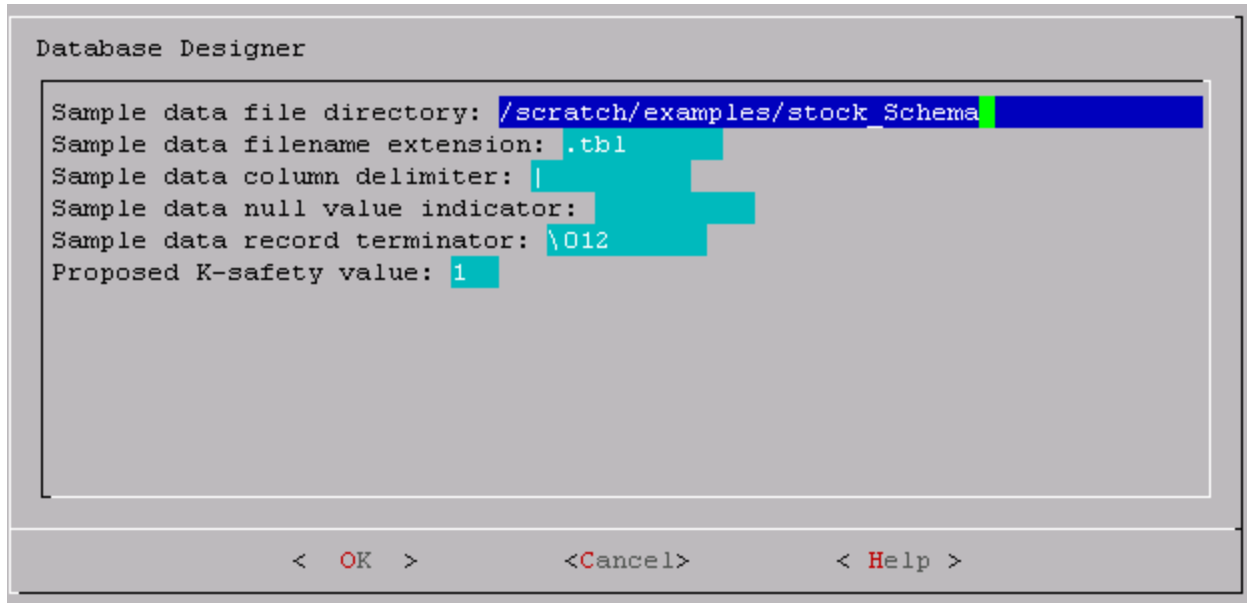
1. Use the up and down arrow keys to choose the command for which you want help.



2. Use the Tab key to move the cursor to the Help button.
3. Press Enter (Return).

In a Dialog Box

1. Use the up and down arrow keys to choose the field on which you want help.



2. Use the Tab key to move the cursor to the Help button.
3. Press Enter (Return).

Scrolling

Some help files are too long for a single screen. Use the up and down arrow keys to scroll through the text.

Password Authentication

When you create a new user with the [CREATE USER](#) command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. You can change a user's password using the [ALTER USER](#) command.

See [Implementing Security](#) for more information about controlling database authorization through passwords.

Tip: Unless the database is used solely for evaluation purposes, HP recommends that all database users have encrypted passwords.

Distributing Changes Made to the Administration Tools Metadata

Administration Tools-specific metadata for a failed node will fall out of synchronization with other cluster nodes if you make the following changes:

- Modify the restart policy
- Add one or more nodes
- Drop one or more nodes.

When you restore the node to the database cluster, you can use the Administration Tools to update the node with the latest Administration Tools metadata:

1. Log on to a host that contains the metadata you want to transfer and start the Administration Tools. (See [Using the Administration Tools](#).)
2. On the **Main Menu** in the Administration Tools, select **Configuration Menu** and click **OK**.
3. On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.
4. Select **AdminTools Meta-Data**.

The Administration Tools metadata is distributed to every host in the cluster.

5. [Restart the database](#).

Administration Tools and Management Console

You can perform most database administration tasks using the Administration Tools, but you have the additional option of using the more visual and dynamic **Management Console**.

The following table compares the functionality available in both interfaces. Continue to use Administration Tools and the command line to perform actions not yet supported by Management Console.

| HP Vertica Functionality | Management Console | Administration Tools |
|--|--------------------|----------------------|
| Use a Web interface for the administration of HP Vertica | Yes | No |
| Manage/monitor one or more databases and clusters through a UI | Yes | No |
| Manage multiple databases on different clusters | Yes | Yes |

| HP Vertica Functionality | Management Console | Administration Tools |
|---|--------------------|----------------------|
| View database cluster state | Yes | Yes |
| View multiple cluster states | Yes | No |
| Connect to the database | Yes | Yes |
| Start/stop an existing database | Yes | Yes |
| Stop/restart HP Vertica on host | Yes | Yes |
| Kill an HP Vertica process on host | No | Yes |
| Create one or more databases | Yes | Yes |
| View databases | Yes | Yes |
| Remove a database from view | Yes | No |
| Drop a database | Yes | Yes |
| Create a physical schema design (Database Designer) | No | Yes |
| Modify a physical schema design (Database Designer) | No | Yes |
| Set the restart policy | No | Yes |
| Roll back database to the Last Good Epoch | No | Yes |
| Manage clusters (add, replace, remove hosts) | Yes | Yes |
| Rebalance data across nodes in the database | Yes | Yes |
| Configure database parameters dynamically | Yes | No |
| View database activity in relation to physical resource usage | Yes | No |
| View alerts and messages dynamically | Yes | No |
| View current database size usage statistics | Yes | No |
| View database size usage statistics over time | Yes | No |
| Upload/upgrade a license file | Yes | Yes |
| Warn users about license violation on login | Yes | Yes |
| Create, edit, manage, and delete users/user information | Yes | No |

| HP Vertica Functionality | Management Console | Administration Tools |
|--|---------------------------|-----------------------------|
| Use LDAP to authenticate users with company credentials | Yes | Yes |
| Manage user access to MC through roles | Yes | No |
| Map Management Console users to an HP Vertica database | Yes | No |
| Enable and disable user access to MC and/or the database | Yes | No |
| Audit user activity on database | Yes | No |
| Hide features unavailable to a user through roles | Yes | No |
| Generate new user (non-LDAP) passwords | Yes | No |

Management Console Provides some, but Not All of the Functionality Provided By the Administration Tools. MC Also Provides Functionality Not Available in the Administration Tools.

See Also

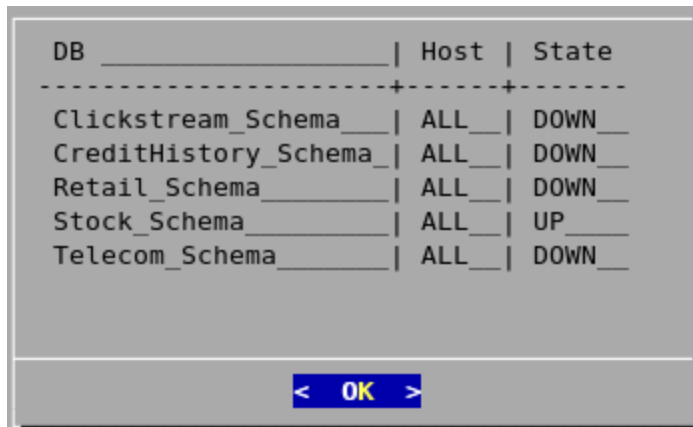
- [Monitoring HP Vertica Using Management Console](#)

Administration Tools Reference

Viewing Database Cluster State

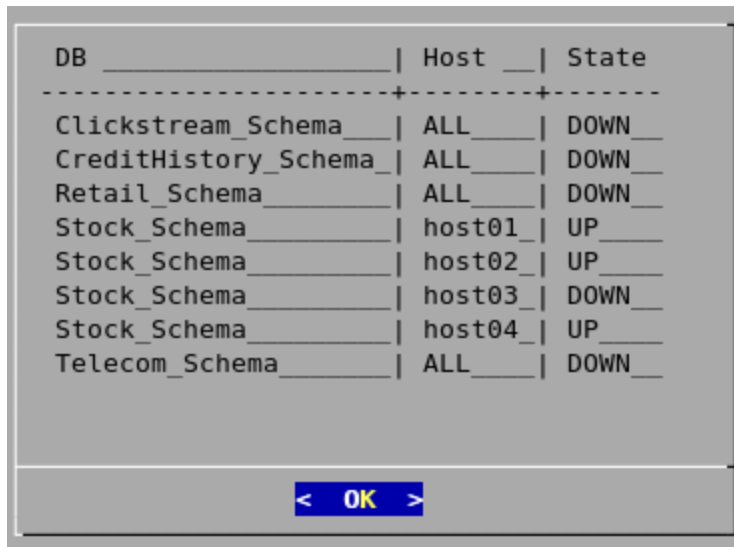
This tool shows the current state of the nodes in the database.

1. On the Main Menu, select **View Database Cluster State**, and click **OK**.
The normal state of a running database is ALL UP. The normal state of a stopped database is ALL DOWN.



| DB | Host | State |
|----------------------|------|-------|
| Clickstream_Schema | ALL | DOWN |
| CreditHistory_Schema | ALL | DOWN |
| Retail_Schema | ALL | DOWN |
| Stock_Schema | ALL | UP |
| Telecom_Schema | ALL | DOWN |

2. If some hosts are UP and some DOWN, restart the specific host that is down using **Restart HP Vertica on Host** from the Administration Tools, or you can start the database as described in [Starting and Stopping the Database](#) (unless you have a known node failure and want to continue in that state.)



| DB | Host | State |
|----------------------|--------|-------|
| Clickstream_Schema | ALL | DOWN |
| CreditHistory_Schema | ALL | DOWN |
| Retail_Schema | ALL | DOWN |
| Stock_Schema | host01 | UP |
| Stock_Schema | host02 | UP |
| Stock_Schema | host03 | DOWN |
| Stock_Schema | host04 | UP |
| Telecom_Schema | ALL | DOWN |

Nodes shown as INITIALIZING or RECOVERING indicate that [Failure Recovery](#) is in progress.

Nodes in other states (such as `NEEDS_CATCHUP`) are transitional and can be ignored unless they persist.

See Also

- [Advanced Menu Options](#)
- [Startup Problems](#)
- [Shutdown Problems](#)

Connecting to the Database

This tool connects to a running **database** with **vsql**. You can use the Administration Tools to connect to a database from any node within the database while logged in to any user account with access privileges. You cannot use the Administration Tools to connect from a host that is not a database node. To connect from other hosts, run `vsql` as described in [Connecting From the Command Line](#) in the Programmer's Guide.

1. On the Main Menu, click **Connect to Database**, and then click **OK**.
2. Supply the database password if asked:

```
Password:
```

When you create a new user with the [CREATE USER](#) command, you can configure the password or leave it empty. You cannot bypass the password if the user was created with a password configured. You can change a user's password using the [ALTER USER](#) command.

The Administration Tools connect to the database and transfer control to **vsql**.

```
Welcome to vsql, the Vertica Analytic Database interactive terminal.  
Type: \h or \? for help with vsql commands  
      \g or terminate with semicolon to execute query  
      \q to quit  
  
=>
```

See [Using vsql](#) for more information.

Note: After entering your password, you may be prompted to change your password if it has expired. See [Implementing Client Authentication](#) for details of password security.

See Also

- [CREATE USER](#)
- [ALTER USER](#)

Starting the Database

Starting a **K-safe** database is supported when up to K nodes are down or unavailable. See [Failure Recovery](#) for a discussion on various scenarios encountered during database shutdown, startup and recovery.

You can start a database using any of these methods:

- The Management Console
- The Administration Tools interface
- The command line

Starting the Database Using MC

On MC's Databases and Clusters page, click a database to select it, and click **Start** within the dialog box that displays.

Starting the Database Using the Administration Tools

1. Open the Administration Tools and select [View Database Cluster State](#) to make sure that all nodes are down and that no other database is running. If all nodes are not down, see Shutdown Problems.
2. Open the Administration Tools. See [Using the Administration Tools](#) for information about accessing the Administration Tools.
3. On the **Main Menu**, select **Start Database**, and then select **OK**.
4. Select the database to start, and then click **OK**.

Caution: HP strongly recommends that you start only one database at a time. If you start more than one database at any time, the results are unpredictable. Users could encounter resource conflicts or perform operations in the wrong database.

5. Enter the database password, and then click **OK**.
6. When prompted that the database started successfully, click **OK**.
7. Check the log files to make sure that no startup problems occurred.

If the database does not start successfully, see [Startup Problems](#).

Starting the Database At the Command Line

If you use the [admintools command line option](#), `start_db()`, to start a database, the `-p` password argument is only required during database creation, when you install a new license.

As long as the license is valid, the `-p` argument is not required to start the database and is silently ignored, even if you introduce a typo or prematurely press the enter key. This is by design, as the database can only be started by the user who (as part of the `verticadba` UNIX user group) initially created the database or who has `root` or `su` privileges.

If the license were to become invalid, HP Vertica would use the `-p` password argument to attempt to upgrade the license with the license file stored in `/opt/vertica/config/share/license.key`.

Following is an example of using `start_db` on a standalone node:

```
[dbadmin@localhost ~]$ /opt/vertica/bin/admintools -t start_db -d VMartInfo: no password
specified, using none
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (UP)
Database VMart started successfully
```

Stopping a Database

This Administration Tool stops a running **database**.

1. Use [View Database Cluster State](#) to make sure that all nodes are up. If all nodes are not up, see [Restarting HP Vertica on Host](#).
2. On the **Main Menu**, select **Stop Database**, and click **OK**.
3. Select the database you want to stop, and click **OK**.
4. Enter the password if asked, and click **OK**.
5. A message confirms that the database has been successfully stopped. Click **OK**.

Error

If users are connected during shutdown operations, the Administration Tools displays a message similar to the following:

```
Database Stock_Schema did not appear to stop in the allotted time. NOTICE: Cannot shut
down while users are connected
shutdown
-----
Shutdown: aborting shutdown
(1 row)
If you need to force a database shutdown, use the
'Stop HP Vertica on Node' command in the Advanced menu,
selecting the appropriate nodes to stop.
```

Description

The message indicates that there are active user connections (sessions). See [Managing Sessions](#) in the Administrator's Guide for more information.

Resolution

The following examples were taken from a different database.

1. To see which users are connected, connect to the database and query the SESSIONS system table described in the SQL Reference Manual. For example:

```
=> \pset expandedExpanded display is on.
=> SELECT * FROM SESSIONS;
```

```
-[ RECORD 1 ]  node_name          | site01
user_name      | dbadmin
client_hostname | 127.0.0.1:57141
login_timestamp | 2009-06-07 14:41:26
session_id     | rhel5-1-30361:0xd7e3e:994462853
transaction_start | 2009-06-07 14:48:54
transaction_id  | 45035996273741092
transaction_description | user dbadmin (select * from session;)
statement_start | 2009-06-07 14:53:31
statement_id    | 0
last_statement_duration | 1
current_statement | select * from sessions;
ssl_state      | None
authentication_method | Trust
-[ RECORD 2 ]
node_name      | site01
user_name      | dbadmin
client_hostname | 127.0.0.1:57142
login_timestamp | 2009-06-07 14:52:55
session_id     | rhel5-1-30361:0xd83ac:1017578618
transaction_start | 2009-06-07 14:53:26
transaction_id  | 45035996273741096
transaction_description | user dbadmin (COPY ClickStream_Fact FROM '/data/clickstream/
1g/ClickStream_Fact.tbl' DELIMITER '|' NULL '\n' DIRECT;)
statement_start | 2009-06-07 14:53:26
statement_id    | 17179869528
```

```

last_statement_duration | 0
current_statement       | COPY ClickStream_Fact FROM '/data/clickstream/1g/ClickStream_Fact.tbl' DELIMITER '|' NULL '\\n' DIRECT;
ssl_state               | None
authentication_method  | Trust

```

The current statement column of Record 1 shows that session is the one you are using to query the system table. Record 2 shows the session that must end before the database can be shut down.

1. If a statement is running in a session, that session must be closed. Use the function `CLOSE_SESSION` or `CLOSE_ALL_SESSIONS` described in the SQL Reference Manual.

Note: `CLOSE_ALL_SESSIONS` is the more common command because it forcefully disconnects all user sessions.

```

=> SELECT * FROM SESSIONS;  -[ RECORD 1 ]
node_name                  | site01
user_name                  | dbadmin
client_hostname            | 127.0.0.1:57141
client_pid                 | 17838
login_timestamp            | 2009-06-07 14:41:26
session_id                 | rhel5-1-30361:0xd7e3e:994462853
client_label               |
transaction_start         | 2009-06-07 14:48:54
transaction_id             | 45035996273741092
transaction_description    | user dbadmin (select * from sessions;)
statement_start           | 2009-06-07 14:53:31
statement_id               | 0
last_statement_duration_us | 1
current_statement         | select * from sessions;
ssl_state                  | None
authentication_method     | Trust
-[ RECORD 2 ]
node_name                  | site01
user_name                  | dbadmin
client_hostname            | 127.0.0.1:57142
client_pid                 | 17839
login_timestamp            | 2009-06-07 14:52:55
session_id                 | rhel5-1-30361:0xd83ac:1017578618
client_label               |
transaction_start         | 2009-06-07 14:53:26
transaction_id             | 45035996273741096
transaction_description    | user dbadmin (COPY ClickStream_Fact FROM
'/data/clickstream/1g/ClickStream_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;)
statement_start           | 2009-06-07 14:53:26
statement_id               | 17179869528
last_statement_duration_us | 0
current_statement         | COPY ClickStream_Fact FROM
'/data/clickstream/1g/ClickStream_Fact.tbl'
DELIMITER '|' NULL '\\n' DIRECT;
ssl_state                  | None

```

```
authentication_method | Trust
=> SELECT CLOSE_SESSION('rhe15-1-30361:0xd83ac:1017578618');
-[ RECORD 1 ]
close_session | Session close command sent. Check sessions for progress.
```

```
=> SELECT * FROM SESSIONS;    -[ RECORD 1 ]
node_name                    | site01
user_name                    | dbadmin
client_hostname              | 127.0.0.1:57141
client_pid                   | 17838
login_timestamp              | 2009-06-07 14:41:26
session_id                   | rhe15-1-30361:0xd7e3e:994462853
client_label                 |
transaction_start           | 2009-06-07 14:48:54
transaction_id               | 45035996273741092
transaction_description      | user dbadmin (select * from sessions;)
statement_start              | 2009-06-07 14:54:11
statement_id                 | 0
last_statement_duration_us   | 98
current_statement            | select * from sessions;
ssl_state                    | None
authentication_method        | Trust
```

2. Query the SESSIONS table again. For example, two columns have changed:

- stmtid is now 0, indicating that no statement is in progress.
- stmt_duration now indicates how long the statement ran in milliseconds before being interrupted.

The SELECT statements that call these functions return when the interrupt or close message has been delivered to all nodes, not after the interrupt or close has completed.

3. Query the SESSIONS table again. When the session no longer appears in the SESSION table, disconnect and run the [Stop Database](#) command.

Controlling Sessions

The database administrator must be able to disallow new incoming connections in order to shut down the database. On a busy system, database shutdown is prevented if new sessions connect after the CLOSE_SESSION or CLOSE_ALL_SESSIONS() command is invoked—and before the database actually shuts down.

One option is for the administrator to issue the SHUTDOWN('true') command, which forces the database to shut down and disallow new connections. See [SHUTDOWN](#) in the SQL Reference Manual.

Another option is to modify the MaxClientSessions parameter from its original value to 0, in order to prevent new non-dbadmin users from connecting to the database.

1. Determine the original value for the MaxClientSessions parameter by querying the V_MONITOR.CONFIGURATIONS_PARAMETERS system table:

```
=> SELECT CURRENT_VALUE FROM CONFIGURATION_PARAMETERS WHERE parameter_name='MaxClient
Sessions';
CURRENT_VALUE
-----
50
(1 row)
```

2. Set the MaxClientSessions parameter to 0 to prevent new non-dbadmin connections:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

Note: The previous command allows up to five administrators to log in.

3. Issue the CLOSE_ALL_SESSIONS() command to remove existing sessions:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

4. Query the SESSIONS table:

```
=> SELECT * FROM SESSIONS;
```

When the session no longer appears in the SESSIONS table, disconnect and run the [Stop Database](#) command.

5. Restart the database.
6. Restore the MaxClientSessions parameter to its original value:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 50);
```

Notes

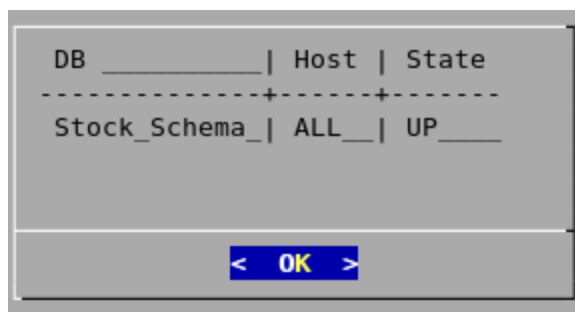
If the database does not stop successfully, see Shutdown Problems.

You cannot stop databases if your password has expired. The **Administration Tools** displays an error message if you attempt to do so. You need to change your expired password using vsql before you can shut down a database.

Restarting HP Vertica on Host

This tool restarts the HP Vertica process one or more nodes in a running database. Use this tool when a cluster host reboots while the database is running. The spread daemon starts automatically but the HP Vertica process does not, thus the node does not automatically rejoin the cluster.

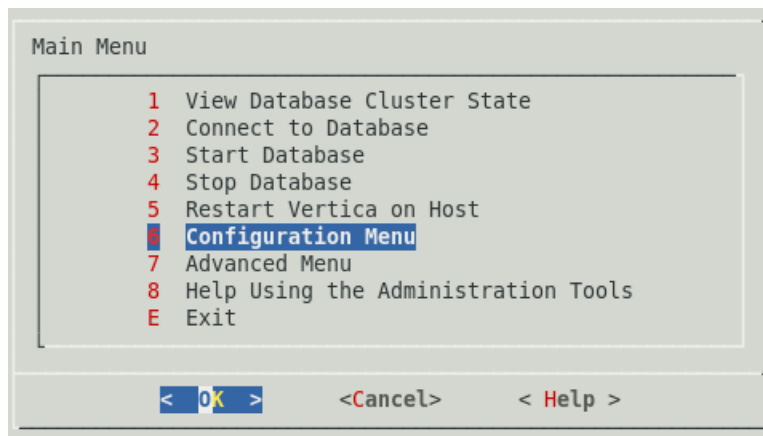
1. On the Main Menu, select **View Database Cluster State**, and click **OK**.
2. If one or more nodes are down, select **Restart HP Vertica on Host**, and click **OK**.
3. Select the database that contains the host that you want to restart, and click **OK**.
4. Select the Host that you want to restart, and click **OK**.
5. Select **View Database Cluster State** again to make sure that all nodes are up.



Configuration Menu Item

The Configuration Menu allows you to:

- Create, drop, and view databases
 - Use the Database Designer to create or modify a physical schema design
1. On the Main Menu, click **Configuration Menu**, and then click **OK**.



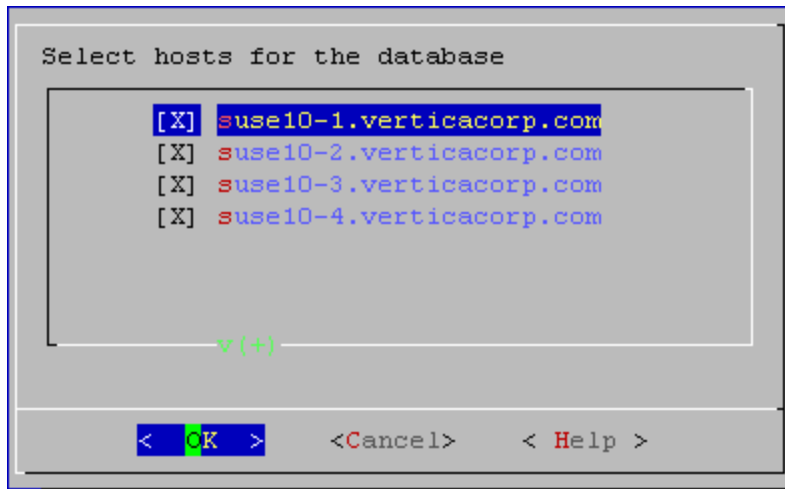
Creating a Database

1. On the **Configuration Menu**, click **Create Database** and then click **OK**.
2. Enter the name of the database and an optional comment. Click **OK**.
3. Enter a password.

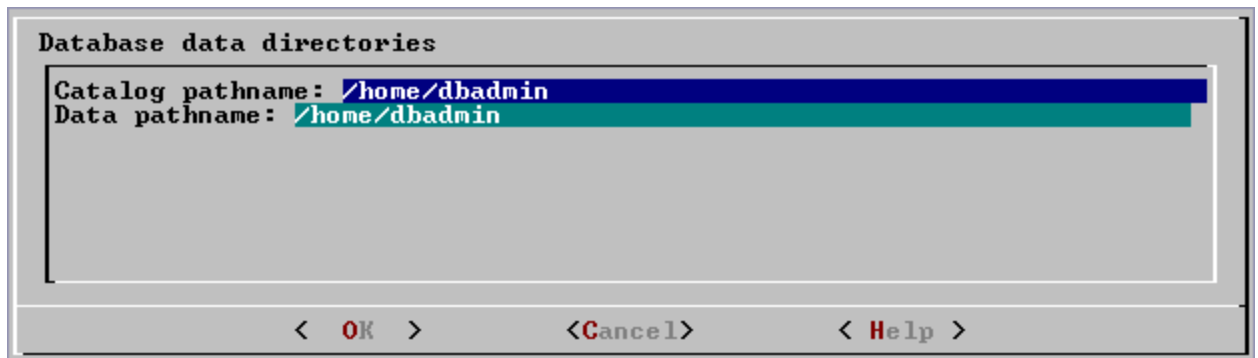
If you do not enter a password, you are prompted to indicate whether you want to enter a password. Click **Yes** to enter a password or **No** to create a database without a superuser password.

Caution: If you do not enter a password at this point, superuser password is set to empty. Unless the database is for evaluation or academic purposes, HP strongly recommends that you enter a superuser password.

4. If you entered a password, enter the password again.
5. Select the hosts to include in the database. The hosts in this list are the ones that were specified at installation time (`install_vertica -s`).



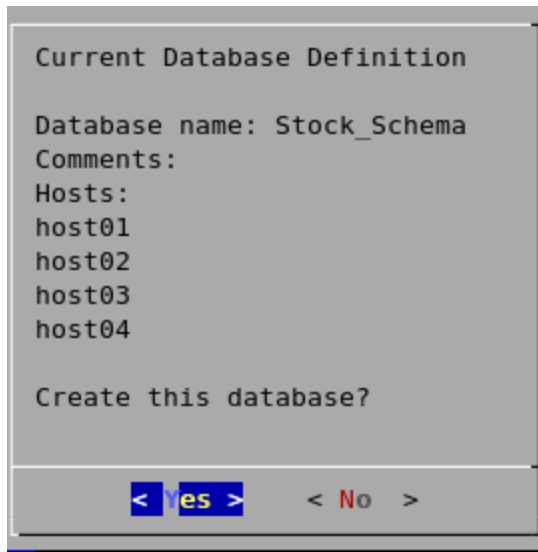
- Specify the directories in which to store the catalog and data files.



Note: Catalog and data paths must contain only alphanumeric characters and cannot have leading space characters. Failure to comply with these restrictions could result in database creation failure.

Note: : Do not use a shared directory for more than one node. Data and catalog directories must be distinct for each node. Multiple nodes must not be allowed to write to the same data or catalog directory.

- Check the current database definition for correctness, and click **Yes** to proceed.



8. A message indicates that you have successfully created a database. Click **OK**.

Note: If you get an error message, see Startup Problems

Dropping a Database

This tool drops an existing **database**. Only the **Database Administrator** is allowed to drop a database.

1. Stop the database as described in [Stopping a Database](#).
2. On the **Configuration Menu**, click **Drop Database** and then click **OK**.
3. Select the database to drop and click **OK**.
4. Click **Yes** to confirm that you want to drop the database.
5. Type **yes** and click **OK** to reconfirm that you really want to drop the database.
6. A message indicates that you have successfully dropped the database. Click **OK**.

Notes

In addition to dropping the database, HP Vertica automatically drops the node definitions that refer to the database unless:

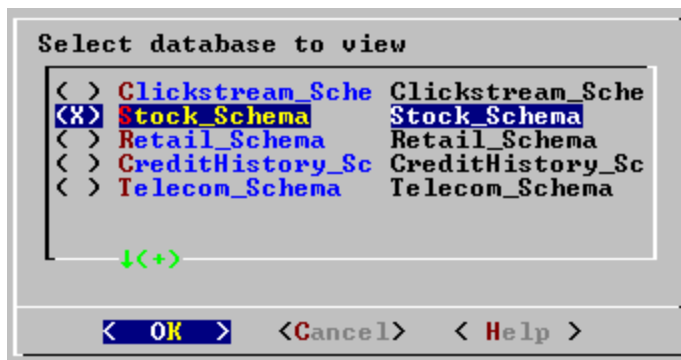
- Another database uses a node definition. If another database refers to any of these node definitions, none of the node definitions are dropped.

- A node definition is the only node defined for the host. (HP Vertica uses node definitions to locate hosts that are available for database creation, so removing the only node defined for a host would make the host unavailable for new databases.)

Viewing a Database

This tool displays the characteristics of an existing **database**.

1. On the **Configuration Menu**, select **View Database** and click **OK**.
2. Select the database to view.



3. HP Vertica displays the following information about the database:

- The name of the database.
- The name and location of the log file for the database.
- The hosts within the database cluster.
- The value of the restart policy setting.

Note: This setting determines whether nodes within a K-Safe database are restarted when they are rebooted. See [Setting the Restart Policy](#).

- The database port.
- The name and location of the catalog directory.

Setting the Restart Policy

The Restart Policy enables you to determine whether or not nodes in a K-Safe **database** are automatically restarted when they are rebooted. Since this feature does not automatically restart nodes if the entire database is DOWN, it is not useful for databases that are not K-Safe.

To set the Restart Policy for a database:

1. Open the Administration Tools.
2. On the Main Menu, select **Configuration Menu**, and click **OK**.
3. In the Configuration Menu, select **Set Restart Policy**, and click **OK**.
4. Select the database for which you want to set the Restart Policy, and click **OK**.
5. Select one of the following policies for the database:
 - **Never** — Nodes are never restarted automatically.
 - **K-Safe** — Nodes are automatically restarted if the database cluster is still UP. This is the default setting.
 - **Always** - Node on a single node database is restarted automatically
6. Click **OK**.

Best Practice for Restoring Failed Hardware

Following this procedure will prevent HP Vertica from misdiagnosing missing disk or bad mounts as data corruptions, which would result in a time-consuming, full-node recovery.

If a server fails due to hardware issues, for example a bad disk or a failed controller, upon repairing the hardware:

1. Reboot the machine into runlevel 1, which is a root and console-only mode.

Runlevel 1 prevents network connectivity and keeps HP Vertica from attempting to reconnect to the cluster.
2. In runlevel 1, validate that the hardware has been repaired, the controllers are online, and any RAID recover is able to proceed.

Note: You do not need to initialize RAID recover in runlevel 1; simply validate that it can recover.

3. Once the hardware is confirmed consistent, only then reboot to runlevel 3 or higher.

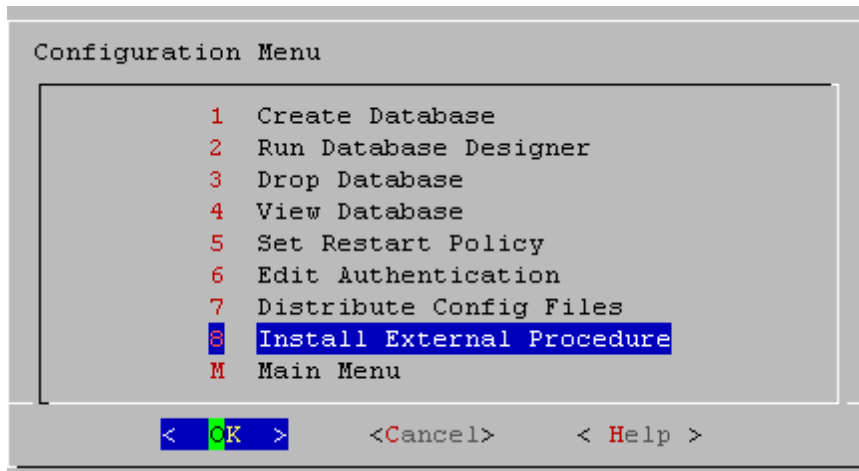
At this point, the network activates, and HP Vertica rejoins the cluster and automatically recovers any missing data. Note that, on a single-node database, if any files that were associated with a projection have been deleted or corrupted, HP Vertica will delete all files associated with that projection, which could result in data loss.

Installing External Procedure Executable Files

1. Run the **Administration Tools**.

```
$ /opt/vertica/bin/adminTools
```

2. On the AdminTools **Main Menu**, click **Configuration Menu**, and then click **OK**.
3. On the **Configuration Menu**, click **Install External Procedure** and then click **OK**.



4. Select the database on which you want to install the external procedure.
5. Either select the file to install or manually type the complete file path, and then click **OK**.
6. If you are not the superuser, you are prompted to enter your password and click **OK**.

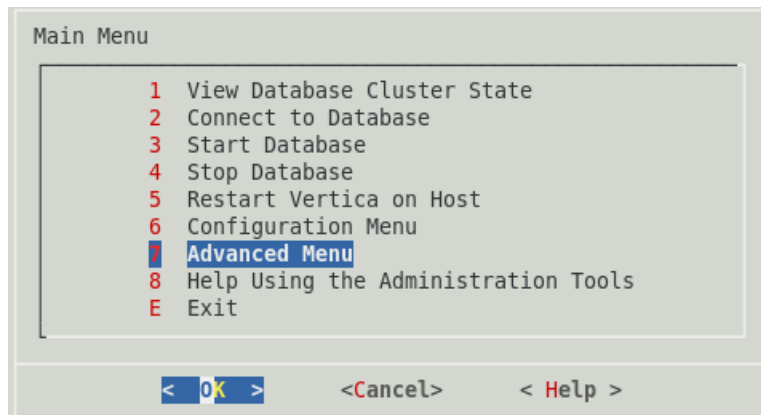
The Administration Tools automatically create the <database_catalog_path>/procedures directory on each node in the database and installs the external procedure in these directories for you.

7. Click **OK** in the dialog that indicates that the installation was successful.

Advanced Menu Options

This Advanced Menu provides interactive recovery and repair commands.

1. On the Main Menu, click Advanced Menu and then OK.



Rolling Back Database to the Last Good Epoch

HP Vertica provides the ability to roll the entire database back to a specific **epoch** primarily to assist in the correction of human errors during data loads or other accidental corruptions. For example, suppose that you have been performing a bulk load and the cluster went down during a particular **COPY** command. You might want to discard all epochs back to the point at which the previous **COPY** command committed and run the one that did not finish again. You can determine that point by examining the log files (see [Monitoring the Log Files](#)).

1. On the Advanced Menu, select **Roll Back Database to Last Good Epoch**.
2. Select the database to roll back. The database must be stopped.
3. Accept the suggested restart epoch or specify a different one.
4. Confirm that you want to discard the changes after the specified epoch.

The database restarts successfully.

Important note:

In HP Vertica 4.1, the default for the `HistoryRetentionTime` configuration parameter changed to 0, which means that HP Vertica only keeps historical data when nodes are down. This new setting effectively prevents the use of the **Administration Tools** 'Roll Back Database to Last Good Epoch' option because the **AHM** remains close to the current epoch and a rollback is not permitted to an epoch prior to the AHM. If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window for removing loaded data; for example:

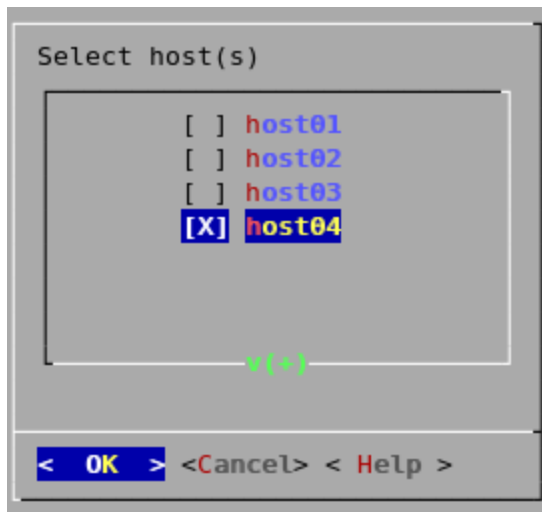
```
=> SELECT SET_CONFIG_PARAMETER ('HistoryRetentionTime', '86400');
```

Stopping HP Vertica on Host

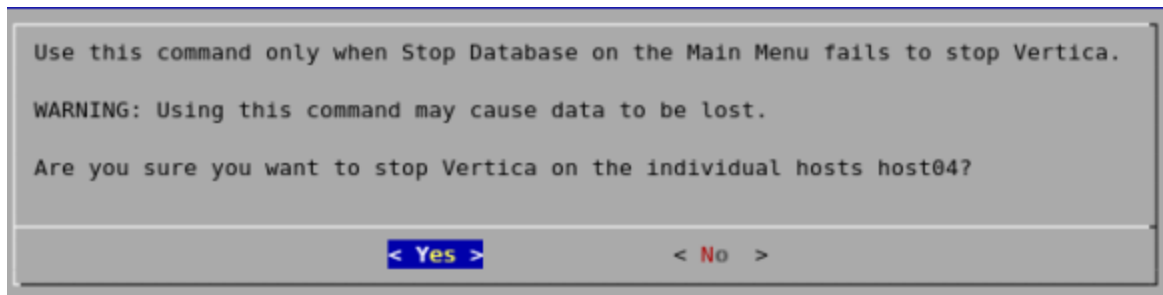
This command attempts to gracefully shut down the HP Vertica process on a single node.

Caution: Do not use this command if you are intending to shut down the entire cluster. Use [Stop Database](#) instead, which performs a clean shutdown to minimize data loss.

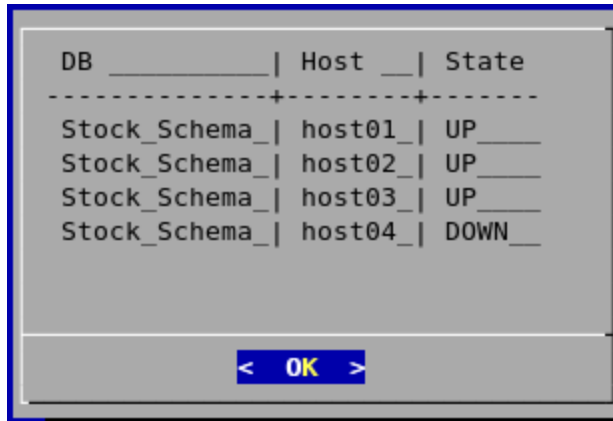
1. On the Advanced Menu, select **Stop HP Vertica on Host** and click **OK**.
2. Select the hosts to stop.



3. Confirm that you want to stop the hosts.



If the command succeeds [View Database Cluster State](#) shows that the selected hosts are DOWN.



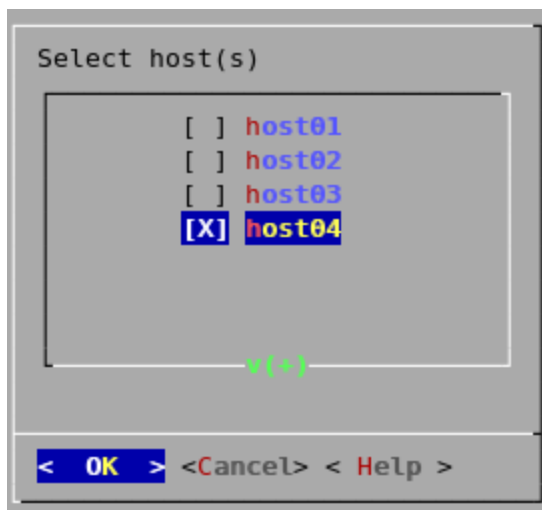
If the command fails to stop any selected nodes, proceed to [Killing HP Vertica Process on Host](#).

Killing the HP Vertica Process on Host

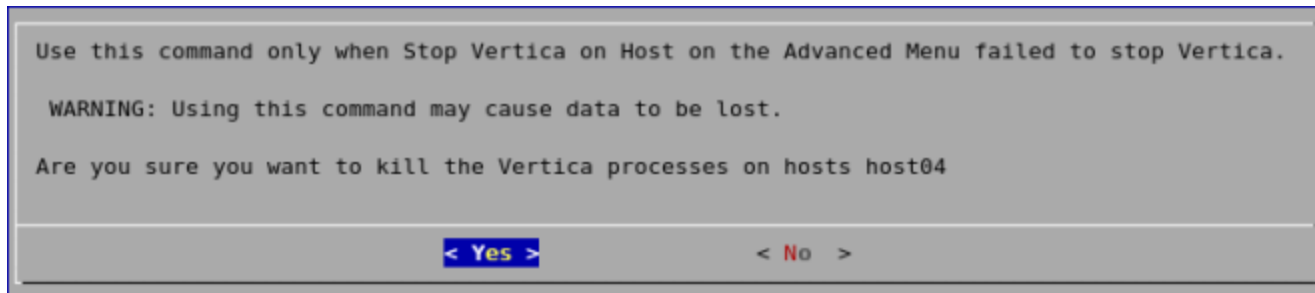
This command sends a kill signal to the HP Vertica process on a node.

Caution: Do not use this command unless you have already tried [Stop Database](#) and [Stop HP Vertica on Node](#) and both were unsuccessful.

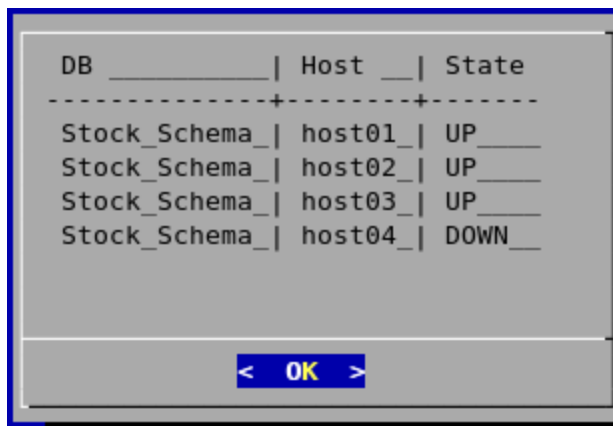
1. On the Advanced menu, select **Kill HP Vertica Process on Host** and click **OK**.
2. Select the hosts on which to kills the HP Vertica process.



3. Confirm that you want to stop the processes.



4. If the command succeeds [View Database Cluster State](#) shows that the selected hosts are DOWN.



5. If the command fails to stop any selected processes, see Shutdown Problems.

Upgrading an Enterprise or Evaluation License Key

The following steps are for HP Vertica Enterprise Edition or evaluation licensed users only. This command copies a license key file into the database. See [Managing Licenses](#) for more information.

1. On the Advanced menu select **Upgrade License Key** and click **OK**.
2. Select the database for which to upgrade the license key.
3. Enter the absolute pathname of your downloaded license key file (for example, /tmp/vlicense.dat) and click **OK**.
4. Click OK when you see a message that indicates that the upgrade succeeded.

Note: If you are using HP Vertica Community Edition, follow the instructions in [HP Vertica License Renewals or Upgrades](#) for instructions to upgrade to an HP Vertica Enterprise Edition or evaluation license key.

Managing Clusters

Cluster Management lets you add, replace, or remove hosts from a database cluster. These processes are usually part of a larger process of [adding](#), [removing](#), or [replacing](#) a database node.

Note: View the database state to verify that it is running. See [View Database Cluster State](#). If the database isn't running, restart it. See [Starting the Database](#).

Using Cluster Management

To use Cluster Management:

1. From the **Main Menu**, select Advanced Menu, and then click **OK**.
2. In the Advanced Menu, select **Cluster Management**, and then click **OK**.
3. Select one of the following, and then click **OK**.
 - **Add Hosts to Database:** See [Adding Hosts to a Database](#).
 - **Re-balance Data:** See [Rebalancing Data](#).
 - **Replace Host:** See [Replacing Hosts](#).
 - **Remove Host from Database:** See [Removing Hosts from a Database](#).

Using the Administration Tools

The **Help Using the Administration Tools** command displays a help screen about using the Administration Tools.

Most of the online help in the Administration Tools is context-sensitive. For example, if you use up/down arrows to select a command, press tab to move to the Help button, and press return, you get help on the selected command.

Administration Tools Metadata

The Administration Tools configuration data (metadata) contains information that databases need to start, such as the hostname/IP address of each participating host in the database cluster.

To facilitate hostname resolution within the Administration Tools, at the command line, and inside the installation utility, HP Vertica enforces all hostnames you provide through the Administration Tools to use IP addresses:

- **During installation**

HP Vertica immediately converts any hostname you provide through command line options `--hosts`, `-add-hosts` or `--remove-hosts` to its IP address equivalent.

- If you provide a hostname during installation that resolves to multiple IP addresses (such as in **multi-homed** systems), the installer prompts you to choose one IP address.
- HP Vertica retains the name you give for messages and prompts only; internally it stores these hostnames as IP addresses.

- **Within the Administration Tools**

All hosts are in IP form to allow for direct comparisons (for example `db = database = database.verticacorp.com`).

- **At the command line**

HP Vertica converts any hostname value to an IP address that it uses to look up the host in the configuration metadata. If a host has multiple IP addresses that are resolved, HP Vertica tests each IP address to see if it resides in the metadata, choosing the first match. No match indicates that the host is not part of the database cluster.

Metadata is more portable because HP Vertica does not require the names of the hosts in the cluster to be exactly the same when you install or upgrade your database.

Writing Administration Tools Scripts

You can invoke most of the Administration Tools from the command line or a shell script.

Syntax

```
> /opt/vertica/bin/admintools [ -t | --tool ] toolname [ options ]
```

Note: For convenience, you can add `/opt/vertica/bin` to your search path.

Parameters

| | | |
|---|---|---|
| [<code>--tool</code> <code>-t</code>] | Instructs the Administration Tools to run the specified tool. Note: If you use the <code>--no-log</code> option to run the Administration Tools silently, <code>--no-log</code> must appear <i>before</i> the <code>--tool</code> option. | |
| <i>toolname</i> | Name of one of the tools described in the help output below. | |
| [<i>options</i>] | <code>-h--help</code> | Shows a brief help message and exits. |
| | <code>-a--help_all</code> | Lists all command-line subcommands and options as shown in the Tools section below. |

Tools

To return a description of the tools you can access, issue the following command at a command prompt:

```
$ admintools -a
```

```
Usage:  
  adminTools [-t | --tool] toolName [options]
```

```
Valid tools are:  
  command_host  
  config_nodes  
  connect_db  
  create_db  
  database_parameters  
  db_add_node  
  db_remove_node  
  db_replace_node  
  db_status  
  drop_db  
  edit_auth  
  host_to_node  
  install_package  
  install_procedure  
  kill_host  
  kill_node  
  list_allnodes  
  list_db  
  list_host  
  list_node  
  list_packages  
  logrotate  
  node_map  
  rebalance_data  
  restart_db  
  restart_node  
  return_epoch  
  set_restart_policy  
  show_active_db  
  start_db  
  stop_db  
  stop_host  
  stop_node  
  uninstall_package  
  upgrade_license_key  
  view_cluster
```

```
-----  
Usage: command_host [options]
```

```
Options:
```

```
-h, --help          show this help message and exit
```

```
-c CMD, --command=CMD
Command to run
-----
Usage: config_nodes [options]
Options:
-h, --help          show this help message and exit
-f NODEHOSTFILE, --file=NODEHOSTFILE
File containing list of nodes, hostnames, catalog
path, and datapath (node<whitespace>host<whitespace>ca
talogPath<whitespace>dataPath one per line)
-c, --check          Check all nodes to make sure they can interconnect
-s SKIPANALYZENODE, --skipanalyze=SKIPANALYZENODE
skipanalyze
-----
Usage: connect_db [options]
Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database to connect
-p DBPASSWORD, --password=DBPASSWORD
Database password in single quotes
-----
Usage: create_db [options]
Options:
-h, --help          show this help message and exit
-s NODES, --hosts=NODES
comma-separated list of hosts to participate in
database
-d DB, --database=DB Name of database to be created
-c CATALOG, --catalog_path=CATALOG
Path of catalog directory[optional] if not using
compat21
-D DATA, --data_path=DATA
Path of data directory[optional] if not using compat21
-p DBPASSWORD, --password=DBPASSWORD
Database password in single quotes [optional]
-l LICENSEFILE, --license=LICENSEFILE
Database license [optional]
-P POLICY, --policy=POLICY
Database restart policy [optional]
```

```
--compat21          Use Vertica 2.1 method using node names instead of  
hostnames
```

```
-----  
Usage: database_parameters [options]
```

```
Options:
```

```
-h, --help          show this help message and exit
```

```
-d DB, --database=DB  Name of database
```

```
-P PARAMETER, --parameter=PARAMETER
```

```
Database parameter
```

```
-c COMPONENT, --component=COMPONENT
```

```
Component[optional]
```

```
-s SUBCOMPONENT, --subcomponent=SUBCOMPONENT
```

```
Sub Component[optional]
```

```
-p PASSWORD, --password=PASSWORD
```

```
Database password[optional]
```

```
-----  
Usage: db_add_node [options]
```

```
Options:
```

```
-h, --help          show this help message and exit
```

```
-d DB, --database=DB  Name of database to be restarted
```

```
-s HOSTS, --hosts=HOSTS
```

```
Comma separated list of hosts to add to database
```

```
-p DBPASSWORD, --password=DBPASSWORD
```

```
Database password in single quotes
```

```
-a AHOSTS, --add=AHOSTS
```

```
Comma separated list of hosts to add to database
```

```
-i, --noprompts      do not stop and wait for user input(default false)
```

```
--compat21          Use Vertica 2.1 method using node names instead of  
hostnames
```

```
-----  
Usage: db_remove_node [options]
```

```
Options:
```

```
-h, --help          show this help message and exit
```

```
-d DB, --database=DB  Name of database to be modified
```

```
-s HOSTS, --hosts=HOSTS
```

```
Name of the host to remove from the db
```

```
-p DBPASSWORD, --password=DBPASSWORD
```

```
Database password in single quotes
```

```
-i, --noprompts      do not stop and wait for user input(default false)
```

```
--compat21          Use Vertica 2.1 method using node names instead of  
hostnames
```

```
-----  
Usage: db_replace_node [options]
```

```
Options:
```

```
-h, --help          show this help message and exit  
-d DB, --database=DB  Name of database to be restarted  
-o ORIGINAL, --original=ORIGINAL
```

```
Name of host you wish to replace
```

```
-n NEWHOST, --new=NEWHOST
```

```
Name of the replacement host
```

```
-p DBPASSWORD, --password=DBPASSWORD
```

```
Database password in single quotes
```

```
-i, --noprompts     do not stop and wait for user input(default false)
```

```
-----  
Usage: db_status [options]
```

```
Options:
```

```
-h, --help          show this help message and exit  
-s STATUS, --status=STATUS
```

```
Database status UP,DOWN or ALL(list running dbs -  
UP,list down dbs - DOWN list all dbs - ALL
```

```
-----  
Usage: drop_db [options]
```

```
Options:
```

```
-h, --help          show this help message and exit  
-d DB, --database=DB  Database to be dropped
```

```
-----  
Usage: edit_auth [options]
```

```
Options:
```

```
-h, --help          show this help message and exit  
-d DATABASE, --database=DATABASE
```

```
database to edit authentication parameters for
```

```
-----  
Usage: host_to_node [options]
```

```
Options:
```

```
-h, --help          show this help message and exit  
-s HOST, --host=HOST  comma separated list of hostnames which is to be  
converted into its corresponding nodenames
```

```
-d DB, --database=DB  show only node/host mapping for this database.
```

Usage: install_package [options]

Options:

-h, --help show this help message and exit

-d DBNAME, --dbname=DBNAME

database name

-p PASSWORD, --password=PASSWORD

database admin password

-P PACKAGE, --package=PACKAGE

specify package or 'all' or 'default'

Usage: install_procedure [options]

Options:

-h, --help show this help message and exit

-d DBNAME, --database=DBNAME

Name of database for installed procedure

-f PROCPATH, --file=PROCPATH

Path of procedure file to install

-p OWNERPASSWORD, --password=OWNERPASSWORD

Password of procedure file owner

Usage: kill_host [options]

Options:

-h, --help show this help message and exit

-s HOSTS, --hosts=HOSTS

comma-separated list of hosts on which the vertica
process is to be killed using a SIGKILL signal

--compat21 Use Vertica 2.1 method using node names instead of
hostnames

Usage: kill_node [options]

Options:

-h, --help show this help message and exit

-s HOSTS, --hosts=HOSTS

comma-separated list of hosts on which the vertica
process is to be killed using a SIGKILL signal

--compat21 Use Vertica 2.1 method using node names instead of
hostnames

```
Usage: list_allnodes [options]
Options:
-h, --help show this help message and exit
-----
Usage: list_db [options]
Options:
-h, --help show this help message and exit
-d DB, --database=DB Name of database to be listed
-----
Usage: list_host [options]
Options:
-h, --help show this help message and exit
-----
Usage: list_node [options]
Options:
-h, --help show this help message and exit
-n NODENAME, --node=NODENAME
Name of the node to be listed
-----
Usage: list_packages [options]
Options:
-h, --help show this help message and exit
-d DBNAME, --dbname=DBNAME
database name
-p PASSWORD, --password=PASSWORD
database admin password
-P PACKAGE, --package=PACKAGE
specify package or 'all' or 'default'
-----
Usage: logrotateconfig [options]
Options:
-h, --help show this help message and exit
-d DBNAME, --dbname=DBNAME
database name
-r ROTATION, --rotation=ROTATION
set how often the log is rotated.[
daily|weekly|monthly ]
-s MAXLOGSZ, --maxsize=MAXLOGSZ
set maximum log size before rotation is forced.
```



```
-k KEEP, --keep=KEEP set # of old logs to keep
-----
Usage: node_map [options]
Options:
-h, --help show this help message and exit
-d DB, --database=DB List only data for this database.
-----
Usage: rebalance_data [options]
Options:
-h, --help show this help message and exit
-d DBNAME, --dbname=DBNAME
database name
-k KSAFETY, --ksafety=KSAFETY
specify the new k value to use
-p PASSWORD, --password=PASSWORD
--script Don't re-balance the data, just provide a script for
later use.
-----
Usage: restart_db [options]
Options:
-h, --help show this help message and exit
-d DB, --database=DB Name of database to be restarted
-e EPOCH, --epoch=EPOCH
Epoch at which the database is to be restarted. If
'last' is given as argument the db is restarted from
the last good epoch.
-p DBPASSWORD, --password=DBPASSWORD
Database password in single quotes
-i, --noprompts do not stop and wait for user input(default false)
-----
Usage: restart_node [options]
Options:
-h, --help show this help message and exit
-s NODES, --hosts=NODES
comma-separated list of hosts to be restarted
-d DB, --database=DB Name of database whose node is to be restarted
-p DBPASSWORD, --password=DBPASSWORD
Database password in single quotes
-i, --noprompts do not stop and wait for user input(default false)
```

```
-F, --force          force the node to start and auto recover if necessary
--compat21          Use Vertica 2.1 method using node names instead of
hostnames

-----
Usage: return_epoch [options]
Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database

-----
Usage: set_restart_policy [options]
Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database for which to set policy
-p POLICY, --policy=POLICY
Restart policy: ('never', 'ksafe', 'always')

-----
Usage: show_active_db [options]
Options:
-h, --help          show this help message and exit

-----
Usage: start_db [options]
Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database to be started
-p DBPASSWORD, --password=DBPASSWORD
Database password in single quotes
-i, --noprompts     do not stop and wait for user input(default false)
-F, --force          force the database to start at an epoch before data
consistency problems were detected.

-----
Usage: stop_db [options]
Options:
-h, --help          show this help message and exit
-d DB, --database=DB Name of database to be stopped
-p DBPASSWORD, --password=DBPASSWORD
Database password in single quotes
-F, --force          Force the databases to shutdown, even if users are
connected.
-i, --noprompts     do not stop and wait for user input(default false)
```

Usage: stop_host [options]

Options:

-h, --help show this help message and exit

-s HOSTS, --hosts=HOSTS

comma-separated list of hosts on which the vertica
process is to be killed using a SIGTERM signal

--compat21 Use Vertica 2.1 method using node names instead of
hostnames

Usage: stop_node [options]

Options:

-h, --help show this help message and exit

-s HOSTS, --hosts=HOSTS

comma-separated list of hosts on which the vertica
process is to be killed using a SIGTERM signal

--compat21 Use Vertica 2.1 method using node names instead of
hostnames

Usage: uninstall_package [options]

Options:

-h, --help show this help message and exit

-d DBNAME, --dbname=DBNAME

database name

-p PASSWORD, --password=PASSWORD

database admin password

-P PACKAGE, --package=PACKAGE

specify package or 'all' or 'default'

Usage: upgrade_license_key [options]

Options:

-h, --help show this help message and exit

-d DB, --database=DB Name of database [required if databases exist]

-l LICENSE, --license=LICENSE

Database license

-i INSTALL, --install=INSTALL

argument '-i install' to Install license else without

'-i install' Upgrade license

-p PASSWORD, --password=PASSWORD

```
Database password[optional]
```

```
-----
```

```
Usage: view_cluster [options]
```

```
Options:
```

```
-h, --help          show this help message and exit  
-x, --xpannd        show the full cluster state, node by node  
-d DB, --database=DB filter the output for a single database
```

Using Management Console

Most of the information you need to use MC is available on the MC interface. The topics in this section augment some areas of the MC interface and provide examples. For an introduction to MC functionality, architecture, and security, see [Management Console](#) in the Concepts Guide.

Management Console provides some, but not all of the functionality that the **Administration Tools** provides. In addition, MC provides extended functionality not available in the Administration Tools, such as a graphical view of your HP Vertica database and detailed monitoring charts and graphs, described in [Monitoring HP Vertica Using MC](#). See [Administration Tools and Management Console](#) in the Administrator's Guide.

If you have not yet installed MC, see [Installing and Configuring Management Console](#) in the Installation Guide.

Connecting to MC

To connect to Management Console:

1. Open an HTML-5 compliant browser.
2. Enter the IP address or host name of the host on which you installed MC (or any cluster node if you installed HP Vertica first), followed by the MC port you assigned when you [configured MC](#) (default 5450).

For example, enter one of:

```
https://00.00.00.00:5450/
```

or

```
https://hostname:5450/
```

3. When the MC logon dialog appears, enter your MC username and password and click **Log in**.

Note: When MC users log in to the MC interface, MC checks their privileges on HP Vertica **Data Collector** (DC) tables on MC-monitored databases. Based on DC table privileges, along with the role assigned the MC user, each user's access to the MC's Overview, Activity and Node details pages could be limited. See [About MC Privileges and Roles](#) for more information.

If you do not have an MC username/password, contact your MC administrator.

Managing Client Connections on MC

Each client session on MC uses a connection from `MaxClientSessions`, a database configuration parameter that determines the maximum number of sessions that can run on a single database cluster node. If multiple MC users are mapped to the same database account and are concurrently monitoring the Overview and Activity pages, graphs could be slow to update while MC waits for a connection from the pool.

Tip: You can increase the value for `MaxClientSessions` on an MC-monitored database to take extra sessions into account. See [Managing Sessions](#) for details.

See Also

- [Monitoring HP Vertica Using MC](#)

Managing Database Clusters on MC

To perform database/cluster-specific tasks on one or more MC-managed clusters, navigate to the **Databases and Clusters** page.

MC administrators see the Import/Create Database Cluster options, while non-administrative MC users see only the databases on which they have been assigned the appropriate [access levels](#). Depending on your access level, the database-related operations you can perform on the MC interface include:

- [Create a new database/cluster](#).
- [Import an existing database/cluster](#) into the MC interface.
- Start the database, unless it is already running (green).
- Stop the database, but only if no users are connected.
- Remove the database from the MC interface.

Note: Remove does not drop the database; it leaves it in the cluster, hidden from the UI. To add the database back to the MC interface, import it using the IP address of any cluster node. A Remove operation also stops metrics gathering on that database, but statistics gathering automatically resumes after you re-import.

- Drop the database after you ensure no users are connected. Drop is a permanent action that drops the database from the cluster.
- View Database to open the Overview page, a layout that provides a dashboard view into the health of your database cluster (node state, storage, performance, CPU/memory, and query concurrency). From this page you can drill down into more detailed database-specific information by clicking data points in the graphs.
- View Cluster to open the Manage page, which shows all nodes in the cluster, as well as each node's state. You can also see a list of monitored databases on the selected cluster and its state; for example, a green arrow indicates a database in an UP state. For node-specific information, click any node to open the Node Details page.

For more information about what users can see and do on MC, see the following topics:

See Also

- [About MC Users](#)
- [About MC Privileges and Roles](#)

Create an Empty Database Using MC

You can create a new database on an existing HP Vertica cluster through the **Management Console** interface.

Database creation can be a long-running process, lasting from minutes to hours, depending on the size of the target database. You can close the web browser during the process and sign back in to MC later; the creation process continues unless an unexpected error occurs. See the **Notes** section below the procedure on this page.

You currently need to use command line scripts to define the database schema and load data. Refer to the topics in [Configuration Procedure](#). You should also run the **Database Designer**, which you access through the **Administration Tools**, to create either a comprehensive or incremental design. Consider using the [Tutorial](#) in the Getting Started Guide to create a sample database you can start monitoring immediately.

How to Create an Empty Database on an MC-managed Cluster

1. If you are already on the **Databases and Clusters** page, skip to the next step; otherwise:
 - a. [Connect](#) to MC and sign in as an MC administrator.
 - b. On the [Home page](#), click the **Databases and Clusters** task.
2. If no databases exist on the cluster, continue to the next step; otherwise:
 - a. If a database is running on the cluster on which you want to add a new database, select the database and click **Stop**.
 - b. Wait for the running database to have a status of *Stopped*.
3. Click the cluster on which you want to create the new database and click **Create Database**.
4. The Create Database wizard opens. Provide the following information:
 - Database name and password. See [Creating a Database Name and Password](#) for rules.
 - Optionally click **Advanced** to open the advanced settings and change the port and catalog, data, and temporary data paths. By default the MC application/web server port is 5450 and paths are `/home/dbadmin`, or whatever you defined for the paths when you ran the Cluster Creation Wizard or the `install_vertica` script. Do not use the default agent port 5444 as a new setting for the MC port. See **MC Settings > Configuration** for port values.
5. Click **Continue**.
6. Select nodes to include in the database.

The Database Configuration window opens with the options you provided and a graphical representation of the nodes appears on the page. By default, all nodes are selected to be part of this database (denoted by a green check mark). You can optionally click each node and clear **Include host in new database** to exclude that node from the database. Excluded nodes are gray. If you change your mind, click the node and select the **Include** check box.

7. Click **Create** in the **Database Configuration** window to create the database on the nodes.

The creation process takes a few moments, after which the database starts and a **Success** message appears on the interface.

8. Click **OK** to close the success message.

MC's Manage page opens and displays the database nodes. Nodes not included in the database are colored gray, which means they are standby nodes you can include later. To add nodes to or remove nodes from your HP Vertica cluster, which are not shown in standby mode, you must run the `install_vertica` script.

Notes

- If warnings occur during database creation, nodes will be marked on the UI with an Alert icon and a message.
 - Warnings do not prevent the database from being created, but you should address warnings after the database creation process completes by viewing the database **Message Center** from the MC Home page.
 - Failure messages display on the database **Manage** page with a link to more detailed information and a hint with an actionable task that you must complete before you can continue. Problem nodes are colored red for quick identification.
 - To view more detailed information about a node in the cluster, double-click the node from the Manage page, which opens the **Node Details** page.
- To create MC users and grant them access to an MC-managed database, see [About MC Users](#) and [Creating an MC User](#).

See Also

- [Creating a Cluster Using MC](#)
- [Troubleshooting Management Console](#)
- [Restarting MC](#)

Import an Existing Database Into MC

If you have already upgraded your database to the current version of HP Vertica, MC automatically discovers the cluster and any databases installed on it, regardless of whether those databases are currently running or are down.

Note: If you haven't created a database and want to create one through the MC, see [Create an Empty Database Using MC](#).

How to Import an Existing Database on the Cluster

The following procedure describes how to import an MC-discovered existing database into the MC interface so you can monitor it.

1. [Connect](#) to Management Console and sign in as an MC administrator.
2. On the MC [Home page](#), click **Databases and Clusters**.
3. On the Databases and Clusters page, click the cluster cube and click **View** in the dialog box that opens.
4. On the left side of the page, look under the Databases heading and click **Import Discovered**.

Tip: A running MC-discovered databases appears as Monitored, and any non-running databases appear as Discovered. MC supports only one running database on a single cluster at a time. In the image above, if you want to monitor the MYDB database, you would need to shut down the DATABASE2 database first.

5. In the **Import Database** dialog box:
 - a. Select the database you want to import.
 - b. Optionally clear auto-discovered databases you don't want to import.
 - c. Supply the database username and password and click **Import**.

After Management Console connects to the database it opens the **Manage** page, which provides a view of the cluster nodes. See [Monitoring Cluster Status](#) for more information.

You perform the import process once per existing database. Next time you connect to Management Console, you'll see your database under the Recent Databases section on the Home page, as well as on the Databases and Clusters page.

Using MC on an AWS Cluster

If you are running an Amazon Web Services (AWS) cluster on HP Vertica 6.1.2, you can install and run MC to monitor and manage your database. You cannot, however, use the MC interface to create or import an HP Vertica cluster.

Managing MC Settings

The **MC Settings** page allows you to configure properties specific to Management Console. You can:

- Change the MC and agent default port assignments
- Upload a new SSL certificate
- Use LDAP for user authentication
- Create new MC users and map them to an MC-managed database using user credentials on the HP Vertica server
- Install HP Vertica on a cluster of hosts through the MC interface
- Customize the look and feel of MC with themes

Modifying Database-Specific Settings

To inspect or modify settings related to an MC-managed database, go to the Databases and Clusters page. On this page, view a running database, and access that database's Settings page from a tab at the bottom at the page.

Changing MC or Agent Ports

When you configure MC, the Configuration Wizard sets up the following default ports:

- 5450—Used to connect a web browser session to MC and allows communication from HP Vertica cluster nodes to the MC application/web server
- 5444—Provides MC-to-node and node-to-node (agent) communications for database create/import and monitoring activities

If You Need to Change the MC Default Ports

A scenario might arise where you need to change the default port assignments for MC or its agents. For example, perhaps one of the default ports is not available on your HP Vertica cluster, or you encounter connection problems between MC and the agents. The following topics describe how to change port assignments for MC or its agents.

See Also

- [Ensure Ports Are Available](#)

How to Change the Agent Port

Changing the agent port takes place in two steps: at the command line, where you modify the `config.py` file and through a browser, where you modify MC settings.

Change the Agent Port in config.py

1. Log in as root on any cluster node and change to the agent directory:

```
# cd /opt/vertica/agent
```

2. Use any text editor to open `config.py`.
3. Scroll down to the `agent_port = 5444` entry and replace 5444 with a different port number.
4. Save and close the file.
5. Copy `config.py` to the `/opt/vertica/agent` directory on all nodes in the cluster.
6. Restart the agent process by running the following command:

```
# /etc/init.d/vertica_agent restart
```

7. Repeat (as root) Step 6 on each cluster node where you copied the `config.py` file.

Change the Agent Port on MC

1. Open a web browser and [connect to MC](#) as a user with [MC ADMIN](#) privileges.
2. Navigate to **MC Settings > Configuration**.
3. Change Default HP Vertica agent port from 5444 to the new value you specified in the `config.py` file.
4. Click **Apply** and click **Done**.
5. [Restart MC](#) so MC can connect to the agent at its new port.

How to Change the MC Port

Use this procedure to change the default port for MC's application server from 5450 to a different value.

1. Open a web browser and [connect to MC](#) as a user with [MC ADMIN](#) privileges.
2. On the MC Home page, navigate to **MC Settings > Configuration** and change the *Application server running port* value from 5450 to a new value.
3. In the change-port dialog, click **OK**.
4. [Restart MC](#).
5. Reconnect your browser session using the new port. For example, if you changed the port from 5450 to 5555, use one of the following formats:

```
https://00.00.00.00:5555/
```

OR

```
https://hostname:5555/
```

Backing Up MC

Before you [upgrade MC](#), HP recommends that you back up your MC metadata (configuration and user settings) on a storage location external to the server on which you installed MC.

1. On the target server (where you want to store MC metadata), log on as root or a user with sudo privileges.
2. Create a backup directory; for example:

```
# mkdir /backups/mc/mc-backup-20130425
```

3. Copy the `/opt/vconsole` directory to the new backup folder:

```
# cp -r /opt/vconsole /backups/mc/mc-backup-20130425
```

Troubleshooting Management Console

The Management Console **Diagnostics** page, which you access from the Home page, helps you resolve issues within the MC process, not the database.

What You Can diagnose:

- View Management Console logs, which you can sort by column headings, such as type, component, or message).
- [Search](#) within messages for key words or phrases and search for log entries within a specific time frame.
- [Export](#) database messages to a file.
- Reset console parameters to their original configuration.

Caution: Reset removes all data (monitoring and configuration information) from storage and forces you to [reconfigure MC](#) as if it were the first time.

- [Restart the Management Console process](#). When the process completes, you are directed back to the login page.

Viewing the MC Log

If you want to browse MC logs (not database logs), navigate to the **Diagnostics > MC Log** page.

This page provides a tabular view of the contents at `/opt/vconsole/log/mc/mconsole.log`, letting you more easily identify and troubleshoot issues related to MC.

You can sort log entries by clicking the column header and search within messages for key words, phrases, and log entries within a specific time frame. You can also export log messages to a file.

| Time | Type | Component | Message |
|----------------------|------|---------------------------|---|
| 03 Dec 2013 19:39:36 | INFO | VuiUtils | ExecuteCommand-> Command: Grep "03 Dec 2013 19:39\03 Dec 2013 19:38\03 Dec 2013 19:37\03 Dec 2013 19:36\03 Dec 2013 19:35\03 Dec 2013 19:34\03 Dec 2013 19:33\03 Dec 2013 19:32\03 Dec 2013 19:31\03 Dec 2013 19:30\03 Dec 2013 19:29\03 Dec 2013 19:28\03 Dec 2013 19:27\03 Dec 2013 19:26\03 Dec 2013 19:25\03 Dec 2013 19:24\03 Dec 2013 19:23\03 Dec 2013 19:22\03 Dec 2013 19:21\03 Dec 2013 19:20\03 Dec 2013 19:19\03 Dec 2013 19:18\03 Dec 2013 19:17\03 Dec 2013 19:16\03 Dec 2013 19:15\03 Dec 2013 19:14\03 Dec 2013 19:13\03 Dec 2013 19:12\03 Dec 2013 19:11\03 Dec 2013 19:10\03 Dec 2013 19:09\03 Dec 2013 19:08\03 Dec 2013 19:07\03 Dec 2013 19:06\03 Dec 2013 19:05\03 Dec 2013 19:04\03 Dec 2013 19:03\03 Dec 2013 19:02\03 Dec 2013 19:01\03 Dec 2013 19:00\03 Dec 2013 18:59\03 Dec 2013 18:58\03 Dec 2013 18:57\03 Dec 2013 18:56\03 Dec 2013 18:55\03 Dec 2013 18:54\03 Dec 2013 18:53\03 Dec 2013 18:52\03 Dec 2013 18:51\03 Dec 2013 18:50\03 Dec 2013 18:49\03 Dec 2013 18:48\03 Dec 2013 18:47\03 Dec 2013 18:46\03 Dec 2013 18:45\03 Dec 2013 18:44\03 Dec 2013 18:43\03 Dec 2013 18:42\03 Dec 2013 18:41\03 Dec 2013 18:40" -N ./Log/Mc/Mconsole.Log Tail -1 |
| 03 Dec 2013 19:39:36 | INFO | VuiUtils | HeapMaxSize: 2075918336, HeapFreeSize: 938560592, TotalHeapSize: 1038286848 |
| 03 Dec 2013 19:39:36 | INFO | LoggingServiceImpl | Time Taken To Execute Parse Methods: 0 |
| 03 Dec 2013 19:39:36 | INFO | TroubleshootingController | ViewMLog-> PageNum: 1, NumLinesPerPage: 1000 |
| 03 Dec 2013 19:39:36 | INFO | VuiUtils | HeapMaxSize: 2075918336, HeapFreeSize: 939169952, TotalHeapSize: 1038286848 |
| 03 Dec 2013 19:39:35 | INFO | VuiWebSocketServlet | Origin: Https://192.168.32.180:5450 |
| 03 Dec 2013 19:39:12 | INFO | AgentCommands | ExecuteCommand-> StatusCode: 200, Raw AgentResponse: [{"Status": "UP", "Hostname": "127.0.0.1", "Nodename": "V_vmartdb_node0001", "Ts": "2013-12-03T19:39:12.740090"}] |
| 03 Dec 2013 19:39:12 | INFO | AgentCommands | TheCommand = Https://127.0.0.1:5444/Databases/Vmartdb/Hosts HttpMethod:GET |
| 03 Dec 2013 19:39:12 | INFO | AgentCommands | ExecuteCommand-> StatusCode: 200, Raw AgentResponse: [{"Total_memory": 1879, "Cpu_info": {"Number_of_cpus": 1, "Cpu_type": "Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz"}, "Nics": [{"Ipaddr": "192.168.32.180", "Name": "Eth0"}, {"Ipaddr": "127.0.0.1", "Name": "Lo"}], "Hostname": "Localhost.Localdomain", "Vertica": {"Release": "20131026", "Brand": "Vertica", "Version": "7.0.0", "Arch": "X86_64", "Host_id": "127.0.0.1"}] |
| 03 Dec 2013 19:39:12 | INFO | VuiWebSocketServlet | Origin: Https://192.168.32.180:5450 |

See Also

- [Exporting MC-managed Database Messages and Logs](#)

Exporting the User Audit Log

When an MC user makes changes on Management Console, whether to an MC-managed database or to the MC itself, their action generates a log entry that contains data you can export to a file.

If you perform an MC factory reset (restore MC to its pre-configured state), you automatically have the opportunity to export audit records before the reset occurs.

To Manually Export MC User Activity

1. From the MC Home page, click **Diagnostics** and then click **Audit Log**.
2. On the Audit log viewer page, click **Export** and save the file to a location on the server.

To see what types of user operations the audit logger records, see [Monitoring MC User Activity](#).

Restarting MC

You might need to restart the MC web/application server for a number of reasons, such as after you change port assignments, use the MC interface to import a new SSL certificate, or if the MC interface or HP Vertica-related tasks become unresponsive.

Restarting MC requires [ADMIN Role \(mc\)](#) or [SUPER Role \(mc\)](#) privileges.

How to Restart MC Through the MC Interface (using Your browser)

1. Open a web browser and [connect to MC](#) as an administrator.
2. On MC's Home page, click **Diagnostics**.
3. Click **Restart Console** and then click OK to continue or Cancel to return to the Diagnostics page..

The MC process shuts down for a few seconds and automatically restarts. After the process completes, you are directed back to the sign-in page.

How to Restart MC At the Command Line

If you are unable to connect to MC through a web browser for any reason, such as if the MC interface or HP Vertica-related tasks become unresponsive, you can run the `vertica-consoled` script with `start`, `stop`, or `restart` arguments.

Follow these steps to start, stop, or restart MC.

1. As root, open a terminal window on the server on which MC is installed.
2. Run the `vertica-consoled` script:

```
# /etc/init.d/vertica-consoled { stop | start | restart }
```

| | |
|----------------------|---|
| <code>stop</code> | Stops the MC application/web server. |
| <code>start</code> | Starts the MC application/web server. Caution: Use <code>start</code> only if you are certain MC is not already running. As a best practice, stop MC before you issue the <code>start</code> command. |
| <code>restart</code> | Restarts the MC application/web server. This process will report that the <code>stop</code> didn't work if MC is not already running. |

Starting over

If you need to return MC to its original state (a "factory reset"), see [Resetting MC to Pre-Configured State](#).

Resetting MC to Pre-Configured State

If you decide to reset MC to its original, preconfigured state, you can do so on the **Diagnostics** page by clicking **Factory Reset**.

Tip: Consider trying one of the options described in [Restarting MC](#) first.

A factory reset removes all metadata (about a week's worth of database monitoring/configuration information and MC users) from storage and forces you to reconfigure MC again, as described in [Configuring MC](#) in the Installation Guide.

After you click Factory Reset, you have the chance to export audit records to a file by clicking Yes. If you click No (do not export audit records), the process begins. There is no undo.

Keep the following in mind concerning user accounts and the MC.

- When you first configure MC, during the configuration process you create an MC super user (a Linux account). Issuing a Factory Reset on the MC does not create a new MC super user, nor does it delete the existing MC super user. When initializing after a Factory Reset, you must logon using the original MC super user account.
- Note that, once MC is configured, you can add users that are specific to MC. Users created through the MC interface are MC specific. When you subsequently change a password through the MC, you only change the password for the specific MC user. Passwords external to MC (i.e., system Linux users and HP Vertica database passwords) remain unchanged.

For information on MC users, refer to the sections, [Creating an MC User](#) and [MC configuration privileges](#).

Avoiding MC Self-Signed Certificate Expiration

When you [connect to MC](#) through a client browser, HP Vertica assigns each HTTPS request a self-signed certificate, which includes a timestamp. To increase security and protect against password replay attacks, the timestamp is valid for several seconds only, after which it expires.

To avoid being blocked out of MC, synchronize time on the hosts in your HP Vertica cluster, as well as on the MC host if it resides on a dedicated server. To recover from loss or lack of synchronization, resync system time and the Network Time Protocol. See [Set Up Time Synchronization](#) in the Installation Guide. If you want to generate your own certificates and keys for MC, see [Generating Certifications and Keys for MC](#).

Operating the Database

Starting and Stopping the Database

This section describes how to start and stop the HP Vertica database using the Administration Tools, Management Console, or from the command line.

Starting the Database

Starting a **K-safe** database is supported when up to K nodes are down or unavailable. See [Failure Recovery](#) for a discussion on various scenarios encountered during database shutdown, startup and recovery.

You can start a database using any of these methods:

- The Management Console
- The Administration Tools interface
- The command line

Starting the Database Using MC

On MC's Databases and Clusters page, click a database to select it, and click **Start** within the dialog box that displays.

Starting the Database Using the Administration Tools

1. Open the Administration Tools and select [View Database Cluster State](#) to make sure that all nodes are down and that no other database is running. If all nodes are not down, see Shutdown Problems.
2. Open the Administration Tools. See [Using the Administration Tools](#) for information about accessing the Administration Tools.
3. On the **Main Menu**, select **Start Database**, and then select **OK**.
4. Select the database to start, and then click **OK**.

Caution: HP strongly recommends that you start only one database at a time. If you start more than one database at any time, the results are unpredictable. Users could encounter resource conflicts or perform operations in the wrong database.

5. Enter the database password, and then click **OK**.

6. When prompted that the database started successfully, click **OK**.
7. Check the log files to make sure that no startup problems occurred.

If the database does not start successfully, see [Startup Problems](#).

Starting the Database At the Command Line

If you use the [admintools command line option](#), `start_db()`, to start a database, the `-p` password argument is only required during database creation, when you install a new license.

As long as the license is valid, the `-p` argument is not required to start the database and is silently ignored, even if you introduce a typo or prematurely press the enter key. This is by design, as the database can only be started by the user who (as part of the `verticadba` UNIX user group) initially created the database or who has root or su privileges.

If the license were to become invalid, HP Vertica would use the `-p` password argument to attempt to upgrade the license with the license file stored in `/opt/vertica/config/share/license.key`.

Following is an example of using `start_db` on a standalone node:

```
[dbadmin@localhost ~]$ /opt/vertica/bin/admintools -t start_db -d VMartInfo: no password
specified, using none
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (DOWN)
Node Status: v_vmart_node0001: (UP)
Database VMart started successfully
```

Stopping the Database

Stopping a **K-safe** database is supported when up to K nodes are down or unavailable. See [Failure Recovery](#) for a discussion on various scenarios encountered during database shutdown, startup and recovery.

You can stop a running database using either of these methods:

- The Management Console
- The Administration Tools interface

Stopping a Running Database Using MC

1. Log in to MC as an MC administrator and navigate to the Manage page to make sure all nodes are up. If a node is down, click that node and select **Start node** in the Node list dialog box.
2. Inform all users that have open connections that the database is going to shut down and instruct them to close their sessions.

Tip: To check for open sessions, query the [V_MONITOR.SESSIONS](#) table. The client_label column returns a value of MC for users who are connected to MC.

3. Still on the Manage page, click **Stop** in the toolbar.

Stopping a Running Database Using the Administration Tools

1. Use [View Database Cluster State](#) to make sure that all nodes are up. If all nodes are not up, see [Restarting a Node](#).
2. Inform all users that have open connections that the database is going to shut down and instruct them to close their sessions.

Tip: A simple way to prevent new client sessions from being opened while you are shutting down the database is to set the [MaxClientSessions](#) configuration parameter to 0. Be sure to restore the parameter to its original setting once you've restarted the database.

```
=> SELECT SET_CONFIG_PARAMETER ('MaxClientSessions', 0);
```

3. Close any remaining user sessions. (Use the [CLOSE_SESSION](#) and [CLOSE_ALL_SESSIONS](#) functions.)
4. Open the Administration Tools. See [Using the Administration Tools](#) for information about accessing the Administration Tools.
5. On the **Main Menu**, select **Stop Database**, and then click **OK**.
6. Select the database you want to stop, and click **OK**.
7. Enter the password if asked, and click **OK**.
8. When prompted that the database has been successfully stopped, click **OK**.

Stopping a Running Database At the Command Line

If you use the [admintools command line option](#), `stop_db()`, to stop a database as follows:

```
[dbadmin@localhost ~]$ /opt/vertica/bin/admintools -t stop_db -d VMartInfo: no password s  
pecified, using none  
    Issuing shutdown command to database  
Database VMart stopped successfully
```

As long as the license is valid, the `-p` argument is not required to stop the database and is silently ignored, even if you introduce a typo or press the enter key prematurely. This is by design, as the database can only be stopped by the user who (as part of the `verticadba` UNIX user group) initially created the database or who has root or su privileges.

If the license were to become invalid, HP Vertica would use the `-p password` argument to attempt to upgrade the license with the license file stored in `/opt/vertica/config/share/license.key`.

Working with the HP Vertica Index Tool

As of HP Vertica 6.0, there are three Index tool options:

- Reindex
- CheckCRC
- Checksort

Note: The Checksort option is available as of Version 6.0.1.

You use the HP Vertica Reindex option only if you have upgraded HP Vertica 6.0 from an earlier version. Following an upgrade to 6.0, any new ROSEs (including those that the TM generates) will use the new index format. New installations use the improved index and maintain CRC automatically.

You can run each of the HP Vertica Index tool options when the database cluster is down. You can run the CheckCRC (-v) and Checksort (-I) options with the cluster up or down, as follows:

| Index Option | Cluster down (per node) | Cluster up (per node or all nodes) |
|--------------|---|--|
| Reindex | <code>vertica -D catalog_path -i</code> | N/A. Node cluster must be down to reindex. |
| CheckCRC | <code>vertica -D catalog_path -v</code> | <code>select run_index_tool ('checkcrc')</code> <code>select run_index_tool ('checkcrc', 'true')</code> |
| Checksort | <code>vertica -D catalog_path -I</code> | <code>select run_index_tool ('checksort')</code> <code>select run_index_tool ('checksort', 'true')</code> |

The HP Vertica Index tool options are accessed through the `vertica` binary, located in the `/opt/vertica/bin` directory on most installations.

Note: Running the Index tool options from the command line outputs progress about tool activity, but not its results. To review the detailed messages after running the Index tool options, see the `indextool.log` file, located in the database catalog directory as described below.

Syntax

```
/opt/vertica/bin/vertica -D catalog_path [-i | -I | -v]
```

Parameters

| Parameter | Description |
|------------------------|--|
| -D <i>catalog_path</i> | Specifies the catalog directory (-D) on which to run each option. |
| -i | Reindexes the ROSEs. |
| -I | Checks the node's ROSEs for correct sort order. |
| -v | Calculates a per-block cyclic redundancy check (CRC) on existing data storage. |

Note: You must run the reindex option on each cluster node, with the cluster down. You can run the Index tool in parallel on different nodes.

Permissions

You must be a superuser to run the Index tool with any option.

Controlling Expression Analysis

The expression analysis that occurs as part of the HP Vertica database indexing techniques improves overall performance. You can turn off such analysis, but doing so is not recommended.

To control expression analysis:

1. Use `add_vertica_options` to turn off EE expression analysis:

```
select add_vertica_options('EE', 'DISABLE_EXPR_ANALYSIS');
```

2. Display the current value by selecting it as follows:

```
select show_current_vertica_options();
```

3. Use `clr_vertica_options` to enable the expression analysis option:

```
select clr_vertica_options('EE', 'DISABLE_EXPR_ANALYSIS');
```

Performance and CRC

HP Vertica recognizes that CRC can affect overall database performance. You can turn off the CRC facility, but doing so is not recommended.

To control CRC:

1. Change the value of the configuration parameter to zero (0):

```
select set_config_parameter('CheckCRC', '0');
```

2. Display the value:

```
select * from configuration_parameters;
```

3. To enable CRC. set the parameter to one (1):

```
select set_config_parameter('CheckCRC', '1');
```

The following sections describe each of the HP Vertica Index tool options and how and when to use them.

Running the Reindex Option

Run the HP Vertica Reindex option to update each ROS index after upgrading to 6.0. Using this option scans all local storage and reindexes the data in all ROSES on the node from which you invoke the tool, adding several new fields to the ROS data blocks. These fields include the data block's minimum and maximum values (`min_value` and `max_value`), and the total number of nulls stored in the block (`null_count`). This option also calculates cyclic redundancy check (CRC) values, and populates the corresponding data block field with the CRC value. The new data block fields are required before using the CheckCRC or Checksort options. Once ROS data has been reindexed, you can use either of the other HP Vertica Index tool options.

To reindex ROSES with the database cluster DOWN:

1. From the command line, start the Index tool with `-D` to specify the catalog directory path, and `-i` to reindex:

```
[dbadmin@localhost bin]$ /opt/vertica/bin/vertica -D /home/dbadmin/VMart/v_vmart_node  
0001_catalog -i
```

2. The Index tool outputs some general data to the terminal, and writes detailed information to the `indextool.log` file:

```
Setting up logger and sessions...Loading catalog...  
Collecting storage containers...  
Scanning data on disk...  
Storages 219/219, bytes 302134347/302134347 100%  
Committing catalog and SAL changes...  
[dbadmin@localhost bin]$
```

3. The `indextool.log` file is located in the database catalog directory:

```
/home/dbadmin/VMart/v_vmart_node0001_catalog/indextool.log
```

Running the CheckCRC Option

The CheckCRC option initially calculates a cyclic redundancy check (CRC) on each block of the existing data storage. You can run this option only after the ROS has been reindexed. Using this Index tool option populates the corresponding ROS data block field with a CRC value. Running CheckCRC after its first invocation checks for data corruption.

To run CheckCRC when the database cluster is down:

1. From the command line, use the Index tool with `-D` to specify the catalog directory path, and `-v` to specify the CheckCRC option.
2. CheckCRC outputs general data such as the following to the terminal, and writes detailed information to the `indextool.log` file:

```
dbadmin@localhost bin]$ /opt/vertica/bin/vertica -D /home/dbadmin/VMart/v_vmart_node0001_catalog -vSetting up logger and sessions...
Loading catalog...
Collecting storage containers...
Scanning data on disk...
Storages 272/272, bytes 302135743/302135743 100%
[dbadmin@localhost bin]$
```

3. The `indextool.log` file is located in the database catalog directory:

```
/home/dbadmin/VMart/v_vmart_node0001_catalog/indextool.log
```

To run CheckCRC when the database is running:

1. From `vsq`, enter this query to run the check on the initiator node:

```
select run_index_tool ('checkcrc');
```

-or-

```
select run_index_tool ('checkcrc', 'false');
```

2. Enter this query to run the check on all nodes:

```
select run_index_tool ('checkcrc', 'true');
```

Handling CheckCRC Errors

Once CRC values exist in each ROS data block, HP Vertica calculates and compares the existing CRC each time data is fetched from disk as part of query processing. If CRC errors occur while fetching data, the following information is written to the `vertica.log` file:

```
CRC Check Failure Details:File Name:  
File Offset:  
Compressed size in file:  
Memory Address of Read Buffer:  
Pointer to Compressed Data:  
Memory Contents:
```

The Event Manager is also notified upon CRC errors, so you can use an SNMP trap to capture CRC errors:

```
"CRC mismatch detected on file <file_path>. File may be corrupted. Please check hardware  
and drivers."
```

If you are running a query from vsql, ODBC, or JDBC, the query returns a `FileColumnReader ERROR`, indicating that a specific block's CRC does not match a given record, with the following hint:

```
hint: Data file may be corrupt. Ensure that all hardware (disk and memory) is working pr  
operly. Possible solutions are to delete the file <pathname> while the node is down, and  
then allow the node to recover, or truncate the table data.code: ERRCODE_DATA_CORRUPTED
```

Running the Checksort Option

If ROS data is not sorted correctly in the projection's order, queries that rely on sorted data will be incorrect. Use the Checksort option to check the ROS sort order if you suspect or detect incorrect queries. The Index tool Checksort option (`-I`) evaluates each ROS row to determine if the row is sorted correctly. If the check locates a row that is not in order, it writes an error message to the log file indicating the row number and contents of the unsorted row.

Note: Running Checksort from the command line does not report any defects that the tool discovers, only the amount of scanned data.

The Checksort option checks only the ROSes of the host from which you initiate the Index tool. For a comprehensive check of all ROSes in the HP Vertica cluster, run check sort on each cluster node to ensure that all ROS data is sorted.

To run Checksort when the database cluster is down:

1. From the command line, start the Index tool with `-D` to specify the catalog directory path, and `-I` to check the sort order:

```
[dbadmin@localhost bin]$ /opt/vertica/bin/vertica -D /home/dbadmin/VMart/v_vmart_node0001_catalog -I
```

2. The Index tool outputs some general data to the terminal, and detailed information to the `indextool.log` file:

```
Setting up logger and sessions... Loading catalog...  
Collecting storage containers...  
Scanning data on disk...  
Storages 17/17, bytes 1739030582/1739030582 100%
```

3. The `indextool.log` file is located in the database catalog directory:

```
/home/dbadmin/VMart/v_vmart_node0001_catalog/indextool.log
```

To run Checksort when the database is running:

1. From `vsql`, enter this query to check the ROS sort order on the initiator node:

```
select run_index_tool ('checksort');
```

-or-

```
select run_index_tool ('checksort', 'false');
```

2. Enter this query to run the sort check on all nodes:

```
select run_index_tool ('checksort', 'true');
```

Viewing Details of Index Tool Results

When running the HP Vertica Index tool options from the command line, the tool writes minimal output to STDOUT, and detailed information to the `indextool.log` file in the database catalog directory. When running CheckCRC and Checksort from `vsql`, results are written to the `vertica.log` file on the node from which you run the query.

To view the results in the `indextool.log` file:

1. From the command line, navigate to the `indextool.log` file, located in the database catalog directory.

```
[15:07:55][vertica-s1]: cd /my_host/databases/check/v_check_node0001_catalog
```

2. For Checksort, all error messages include an OID number and the string 'Sort Order Violation' as follows:

```
<INFO> ...on oid 45035996273723545: Sort Order Violation:
```

3. You can use `grep` on the `indextool.log` file to search for the Sort Order Violation string with a command such as this, which returns the line before each string (-B1), and the four lines that follow (-A4):

```
[15:07:55][vertica-s1]: grep -B1 -A4 'Sort Order Violation:' /my_host/databases/check/v_check_node0001_catalog/indextool.log 2012-06-14 14:07:13.686 unknown:0x7fe1da7a1950 [EE] <INFO> An error occurred when running index tool thread on oid 45035996273723537:  
Sort Order Violation:  
Row Position: 624  
Column Index: 0  
Last Row: 2576000  
This Row: 2575000  
--  
012-06-14 14:07:13.687 unknown:0x7fe1dafa2950 [EE] <INFO> An error occurred when running index tool thread on oid 45035996273723545:  
Sort Order Violation:  
Row Position: 3  
Column Index: 0  
Last Row: 4  
This Row: 2  
--
```

To find the relevant projection where the sort violation was found:

1. Query the `storage_containers` system table using a `storage_oid` equal to the OID value listed in the `indextool.log` file.
2. Use a query such as this:

```
=> select * from storage_containers where storage_oid = 45035996273723545;
```


Working with Tables

Creating Base Tables

The `CREATE TABLE` statement creates a table in the HP Vertica **logical schema**. The example database described in the [Getting Started Guide](#) includes sample SQL scripts that demonstrate this procedure. For example:

```
CREATE TABLE vendor_dimension ( vendor_key          INTEGER          NOT NULL PRIMARY KEY,
  vendor_name      VARCHAR(64),
  vendor_address   VARCHAR(64),
  vendor_city      VARCHAR(64),
  vendor_state     CHAR(2),
  vendor_region    VARCHAR(32),
  deal_size        INTEGER,
  last_deal_update DATE
);
```

Note: Each table can have a maximum 1600 columns.

Creating Tables Using the `/*+direct*/` Clause

You can use the `/* +direct */` clause to create a table or temporary table, saving the table directly to disk (ROS), bypassing memory (WOS). For example, following is an existing table called `states`:

```
VMart=> select * from states;
 State | Bird   | Tree | Tax
-----+-----+-----+-----
 MA   | Robin | Maple | 5.7
 NH   | Thrush | Elm   | 0
 NY   | Cardinal | Oak  | 7.2
(3 rows)
```

Create a new table, `StateBird`, with the `/*+direct*/` clause in the statement, placing the clause directly before the query (`select State, Bird from states`):

```
VMart=> create table StateBird as /*+direct*/ select State, Bird from states;
CREATE TABLE
VMart=> select * from StateBird;
 State | Bird
-----+-----
 MA   | Robin
 NH   | Thrush
 NY   | Cardinal
(3 rows)
```

The following example creates a temporary table using the `/*+direct*/` clause, along with the `ON COMMIT PRESERVE ROWS` directive:

```
VVMart=> create temp table StateTax ON COMMIT PRESERVE ROWS as /*+direct*/ select State,
Tax from states;
CREATE TABLE
VVMart=> select * from StateTax;
  State | Tax
-----+-----
  MA   | 5.7
  NH   | 0
  NY   | 7.2
(3 rows)
```

Automatic Projection Creation

To get your database up and running quickly, HP Vertica automatically creates a default **projection** for each table created through the [CREATE TABLE](#) and [CREATE TEMPORARY TABLE](#) statements. Each projection created automatically (or manually) includes a base projection name prefix. You must use the projection prefix when altering or dropping a projection ([ALTER PROJECTION RENAME](#), [DROP PROJECTION](#)).

How you use the CREATE TABLE statement determines when the projection is created:

- If you create a table without providing the projection-related clauses, HP Vertica automatically creates a **superprojection** for the table when you use an INSERT INTO or COPY statement to load data into the table for the first time. The projection is created in the same schema as the table. Once HP Vertica has created the projection, it loads the data.
- If you use CREATE TABLE AS SELECT to create a table from the results of a query, the table is created first and a projection is created immediately after, using some of the properties of the underlying SELECT query.
- (Advanced users only) If you use any of the following parameters, the default projection is created immediately upon table creation using the specified properties:
 - [column-definition](#) (ENCODING encoding-type and ACCESSRANK integer)
 - ORDER BY table-column
 - [hash-segmentation-clause](#)
 - UNSEGMENTED { NODE *node* | ALL NODES }
 - KSAFE

Note: Before you define a superprojection in the above manner, read [Creating Custom Designs](#) in the Administrator's Guide.

Characteristics of Default Automatic Projections

A default auto-projection has the following characteristics:

- It is a **superprojection**.
- It uses the default [encoding-type](#) AUTO.
- If created as a result of a CREATE TABLE AS SELECT statement, uses the encoding specified in the query table.
- Auto-projections use hash segmentation.
- The number of table columns used in the segmentation expression can be configured, using the MaxAutoSegColumns configuration parameter. See [General Parameters](#) in the Administrator's Guide. Columns are segmented in this order:
 - Short (<8 bytes) data type columns first
 - Larger (> 8 byte) data type columns
 - Up to 32 columns (default for MaxAutoSegColumns configuration parameter)
 - If segmenting more than 32 columns, use nested hash function

Auto-projections are defined by the table properties and creation methods, as follows:

| If table... | Sort order is... | Segmentation is... |
|--|--|--|
| Is created from input stream (COPY or INSERT INTO) | Same as input stream, if sorted. | On PK column (if any), on all FK columns (if any), on the first 31 configurable columns of the table |
| Is created from CREATE TABLE AS SELECT query | Same as input stream, if sorted. If not sorted, sorted using following rules. | Same segmentation columns if query output is segmented The same as the load, if output of query is unsegmented or unknown |
| Has FK and PK constraints | FK first, then PK columns | PK columns |
| Has FK constraints only (no PK) | FK first, then remaining columns | Small data type (< 8 byte) columns first, then large data type columns |
| Has PK constraints only (no FK) | PK columns | PK columns |
| Has no FK or PK constraints | On all columns | Small data type (< 8 byte) columns first, then large data type columns |

Default automatic projections and segmentation get your database up and running quickly. HP recommends that you start with these projections and then use the **Database Designer** to optimize your database further. The Database Designer creates projections that optimize your database based on the characteristics of the data and, optionally, the queries you use.

See Also

- [Creating External Tables](#)
- [Projection Concepts](#)
- [CREATE TABLE](#)

Creating a Table Like Another

You can create a new table based on an existing table using the `CREATE TABLE` statement with the `LIKE existing_table` clause, optionally including the projections of the existing table. Creating a new table with the `LIKE` option replicates the table definition and any storage policy associated with the existing table. The statement does not copy any data. The main purpose of this function is to create an intermediate table into which you can move partition data, and eventually, archive the data and drop the intermediate table.

Note: Invoking `CREATE TABLE` with its `LIKE` clause before calling the function to move partitions for archiving requires first dropping pre-join-projections or refreshing out-of-date projections.

You can optionally use the `including projections` clause to create a table that will have the existing table's current and non-pre-join projection definitions whenever you populate the table. Replicated projections are named automatically to avoid conflict with any existing objects, and follow the same naming conventions as auto projections. You cannot create a new table like another if the source table has pre-join- or out-of-date-projections. The statement displays a warning message.

Note: HP Vertica does not support using `CREATE TABLE new_t LIKE exist_t INCLUDING PROJECTIONS` if *exist_t* is a temporary table.

Epochs and Node Recovery

The checkpoint epoch (CPE) for both the source and target projections are updated as ROSEs are moved. The start and end epochs of all storage containers, such as ROSEs, are modified to the agreed move epoch. When this occurs, the epochs of all columns without an actual data file rewrite advance the CPE to the move epoch. If any nodes are down during the TM moveout, they will detect that there is storage to recover, and will recover from other nodes with the correct epoch upon rejoining the cluster.

Storage Location and Policies for New Tables

When you use the `CREATE TABLE...LIKE` statement, any storage policy objects associated with the table are also copied. Data added to the new table will use the same labeled storage location as the source table, unless you change the storage policy. For more information, see [Working With Storage Locations](#).

Simple Example

This example shows how to use the statement for a table that already exists, and suggests a naming convention that describes the contents of the new table:

Create a new schema in which to create an intermediate table with projections. This is the table into which you will move partitions. Then, create a table identical to the source table from which to move partitions:

```
VMART=> create schema partn_backup;CREATE SCHEMA
VMART=> create table partn_backup.trades_200801 like prod.trades including projections;
CREATE TABLE
```

Once the schema and table exist, you can move one or more of the existing table partitions to the new intermediate table.

Using CREATE TABLE LIKE

For this example, create a table, states:

```
VMART=> create table statesVMART-> (state char(2) not null,
VMART(> bird varchar(20),
VMART(> flower varchar (20),
VMART(> tree char (20),
VMART(> tax float) partition by state;
CREATE TABLE
```

Populate the table with some data on New England:

```
insert into states values ('MA', 'chickadee', 'american_elm', 5.675, '07-04-1620');insert
into states values ('VT', 'Hermit_Thrasher', 'Sugar_Maple', 6.0, '07-04-1610');
.
.
.
```

Select the states table to see its content:

```
VMART=> select * from states; State |          bird          |          tree          | tax
| stateDate
-----+-----+-----+-----+-----
MA | chickadee          | american_elm          | 5.675 | 1620-07-04
NH | Purple_Finch       | White_Birch          | 0     | 1615-07-04
```

```
VT | Hermit_Thrasher | Sugar_maple | 6 | 1618-07-04
ME | Black_Cap_Chickadee | Pine_Tree | 5 | 1615-07-04
CT | American_Robin | White_Oak | 6.35 | 1618-07-04
RI | Rhode_Island_Red | Red_Maple | 5 | 1619-07-04
(6 rows)
```

View the projections for this table:

```
VMART=> \dj
Schema | Name | Owner | Node | Comment
-----+-----+-----+-----+-----
.
.
.
public | states_b0 | dbadmin | |
public | states_b1 | dbadmin | |
public | states_p_node0001 | dbadmin | v_vmart_node0001 |
public | states_p_node0002 | dbadmin | v_vmart_node0002 |
public | states_p_node0003 | dbadmin | v_vmart_node0003 |
```

Now, create a table like the states table, including projections:

```
VMART=> create table newstates like states including projections;CREATE TABLE
VMART=> select * from newstates;
State | bird | tree | tax | stateDate
-----+-----+-----+-----+-----
(0 rows)
```

See Also

- [Creating Base Tables](#)
- [Creating Temporary Tables](#)
- [Creating External Tables](#)
- [Moving Partitions](#)
- [CREATE TABLE](#)

Creating Temporary Tables

You create temporary tables using the CREATE TEMPORARY TABLE statement, specifying the table as either local or global. You cannot create temporary external tables.

A common use case for a temporary table is to divide complex query processing into multiple steps. Typically, a reporting tool holds intermediate results while reports are generated (for example, first get a result set, then query the result set, and so on). You can also write [subqueries](#).

Note: The default retention when creating temporary tables is `ON COMMIT DELETE ROWS`, which discards data at transaction completion. The non-default value is `ON COMMIT PRESERVE ROWS`, which discards data when the current session ends.

Global Temporary Tables

HP Vertica creates global temporary tables in the public schema, with the data contents private to the transaction or session through which data is inserted.

Global temporary table definitions are accessible to all users and sessions, so that two (or more) users can access the same global table concurrently. However, whenever a user commits or rolls back a transaction, or ends the session, HP Vertica removes the global temporary table data automatically, so users see only data specific to their own transactions or session.

Global temporary table definitions persist in the database catalogs until they are removed explicitly through a [DROP TABLE](#) statement.

Local Temporary Tables

Local temporary tables are created in the `V_TEMP_SCHEMA` namespace and inserted into the user's search path transparently. Each local temporary table is visible only to the user who creates it, and only for the duration of the session in which the table is created.

When the session ends, HP Vertica automatically drops the table definition from the database catalogs. You cannot preserve non-empty, session-scoped temporary tables using the `ON COMMIT PRESERVE ROWS` statement.

Creating local temporary tables is significantly faster than creating regular tables, so you should make use of them whenever possible.

Note: You cannot add projections to non-empty, session-scoped temporary tables if you specify `ON COMMIT PRESERVE ROWS`. Be sure that projections exist before you load data, as described in the section *Automatic Projection Creation* in [CREATE TABLE](#). Also, while you can add projections for tables created with the `ON COMMIT DELETE ROWS` option, be aware that you could save the projection but still lose all the data.

Creating a Temp Table Using the `/*+direct*/` Clause

You can use the `/*+direct*/` clause to create a table or temporary table, saving the table directly to disk (ROS), bypassing memory (WOS). For example, following is an existing table called `states`:

```
VMart=> select * from states;
State | Bird   | Tree | Tax
-----+-----+-----+-----
MA    | Robin | Maple | 5.7
NH    | Thrush| Elm   | 0
NY    | Cardinal| Oak  | 7.2
```

```
(3 rows)
```

Create a new table, `StateBird`, with the `/*+direct*/` clause in the statement, placing the clause directly before the query (`select State, Bird from states`):

```
VMart=> create table StateBird as /*+direct*/ select State, Bird from states;
CREATE TABLE
VMart=> select * from StateBird;
  State | Bird
-----+-----
  MA    | Robin
  NH    | Thrush
  NY    | Cardinal
(3 rows)
```

The following example creates a temporary table using the `/*+direct*/` clause, along with the `ON COMMIT PRESERVE ROWS` directive:

```
VVMart=> create temp table StateTax ON COMMIT PRESERVE ROWS as /*+direct*/ select State,
Tax from states;
CREATE TABLE
VMart=> select * from StateTax;
  State | Tax
-----+-----
  MA    | 5.7
  NH    | 0
  NY    | 7.2
(3 rows)
```

Characteristics of Default Automatic Projections

Once local or global table exists, HP Vertica creates auto-projections for temporary tables whenever you load or insert data.

The default auto-projection for a temporary table has the following characteristics:

- It is a **superprojection**.
- It uses the default encoding-type `AUTO`.
- It is automatically unsegmented on the initiator node, if you do not specify a [hash-segmentation-clause](#).
- The projection is not **pinned**.
- Temp tables are not recoverable, so the superprojection is not K-Safe (`K-SAFE=0`), and you cannot make it so.

Auto-projections are defined by the table properties and creation methods, as follows:

| If table... | Sort order is... | Segmentation is... |
|--|--|---|
| Is created from input stream (COPY or INSERT INTO) | Same as input stream, if sorted. | On PK column (if any), on all FK columns (if any), on the first 31 configurable columns of the table |
| Is created from CREATE TABLE AS SELECT query | Same as input stream, if sorted. If not sorted, sorted using following rules. | Same segmentation columns f query output is segmented The same as the load, if output of query is unsegmented or unknown |
| Has FK and PK constraints | FK first, then PK columns | PK columns |
| Has FK constraints only (no PK) | FK first, then remaining columns | Small data type (< 8 byte) columns first, then large data type columns |
| Has PK constraints only (no FK) | PK columns | PK columns |
| Has no FK or PK constraints | On all columns | Small data type (< 8 byte) columns first, then large data type columns |

Advanced users can modify the default projection created through the CREATE TEMPORARY TABLE statement by defining one or more of the following parameters:

- [column-definition \(temp table\)](#) (ENCODING encoding-type and ACCESSRANK integer)
- ORDER BY table-column
- [hash-segmentation-clause](#)
- UNSEGMENTED { NODE node | ALL NODES }
- NO PROJECTION

Note: Before you define the superprojection in this manner, read [Creating Custom Designs](#) in the Administrator's Guide.

Preserving GLOBAL Temporary Table Data for a Transaction or Session

You can preserve session-scoped rows in a GLOBAL temporary table for the entire session or for the current transaction only.

To preserve a temporary table for the transaction, use the ON COMMIT DELETE ROWS clause:

```
=> CREATE GLOBAL TEMP TABLE temp_table1 (x NUMERIC, y NUMERIC )  
    ON COMMIT DELETE ROWS;
```

To preserve temporary table data until the end of the session, use the ON COMMIT PRESERVE ROWS clause:

```
=> CREATE GLOBAL TEMP TABLE temp_table2 (x NUMERIC, y NUMERIC )  
    ON COMMIT PRESERVE ROWS;
```

Specifying Column Encoding

You can specify the [encoding type](#) to use per column.

The following example specifies that the superprojection created for the temp table use RLE encoding for the y column:

```
=> CREATE LOCAL TEMP TABLE temp_table1 (x NUMERIC, y NUMERIC ENCODING RLE )  
    ON COMMIT DELETE ROWS;
```

The following example specifies that the superprojection created for the temp table use the sort order specified by the ORDER BY clause, rather than the order of columns in the column list.

```
=> CREATE GLOBAL TEMP TABLE temp_table1 (  
    x NUMERIC,  
    y NUMERIC ENCODING RLE,  
    b VARCHAR(8),  
    z VARCHAR(8) )  
    ORDER BY z, x;
```

See Also

- [Projection Concepts](#)
- [CREATE TEMPORARY TABLE](#)
- [CREATE TABLE](#)
- [TRUNCATE TABLE](#)
- [DELETE](#)
- [ANALYZE_STATISTICS](#)

Creating External Tables

You create an external table using the CREATE EXTERNAL TABLE AS COPY statement. You cannot create temporary external tables. For the syntax details to create an external table, see the [CREATE EXTERNAL TABLE](#) statement in the SQL Reference Manual.

Note: Each table can have a maximum of 1600 columns.

Required Permissions for External Tables

You must be a database superuser to create external tables, unless you create a USER-accessible storage location (see [ADD_LOCATION](#)) and grant user privileges to the location, schema, and so on.

Note: Permission requirements for external tables differ from other tables. To gain full access (including SELECT) to an external table that a user has privileges to create, the database superuser must also grant READ access to the USER-accessible storage location, see [GRANT \(Storage Location\)](#).

COPY Statement Definition

When you create an external table, table data is not added to the database, and no projections are created. Instead, HP Vertica performs a syntactic check of the CREATE EXTERNAL TABLE... statement, and stores the table name and COPY statement definition in the catalog. When a SELECT query references an external table, the stored COPY statement is parsed to obtain the referenced data. Successfully returning data from the external table requires that the COPY definition is correct, and that other dependencies, such as files, nodes, and other resources are accessible and available at query-time.

For more information about checking the validity of the external table COPY definition, see [Validating External Tables](#).

Developing User-Defined Load (UDL) Functions for External Tables

You can create external tables with your own load functions. For more information about developing user-defined load functions, see [User Defined Load \(UDL\)](#) and the extended COPY syntax in the SQL Reference Manual.

Examples

Examples of external table definitions:

```
CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat' DELIMITER ',';
CREATE EXTERNAL TABLE ext1 (x integer) AS COPY FROM '/tmp/ext1.dat.bz2' BZIP DELIMITER ',
';
CREATE EXTERNAL TABLE ext1 (x integer, y integer) AS COPY (x as '5', y) FROM '/tmp/ext1.d
at.bz2' BZIP DELIMITER ',';
```

See Also

- [COPY](#)
- [CREATE EXTERNAL TABLE AS COPY](#)

Validating External Tables

When you create an external table, HP Vertica validates the syntax of the `CREATE EXTERNAL TABLE AS COPY FROM` statement. For instance, if you omit a required keyword in the statement (such as `FROM`), creating the external table fails, as in this example:

```
VMart=> create external table ext (ts timestamp,d varchar) as copy '/home/dbadmin/designer.log'; ERROR 2778: COPY requires a data source; either a FROM clause or a WITH SOURCE clause or a user-defined source
```

Checking other aspects of the `COPY` definition (such as path statements and node availability) does not occur until a select query references the external table.

To validate that you have successfully created an external table definition, run a select query referencing the external table. Check that the returned query data is what you expect. If the query does not return data correctly, check the `COPY` exception and rejected data log files.

Since the `COPY` definition determines what occurs when you query an external table, obtaining `COPY` statement errors can help reveal any underlying problems. For more information about `COPY` exceptions and rejections, see [Capturing Load Exceptions and Rejections](#).

Limiting the Maximum Number of Exceptions

Querying external table data with an incorrect `COPY FROM` statement definition can potentially result in many exceptions. To limit the number of saved exceptions, HP Vertica sets the maximum number of reported rejections with the `ExternalTablesExceptionsLimit` configuration parameter. The default value is 100. Setting the `ExternalTablesExceptionsLimit` to -1 disables the limit.

For more information about configuration parameters, see [Configuration Parameters](#), and specifically, [General Parameters](#).

If `COPY` errors reach the maximum number of exceptions, the external table query continues, but `COPY` generates a warning in the `vertica.log`, and does not report subsequent rejections and/or exceptions.

Note: Using the `ExternalTablesExceptionsLimit` configuration parameter differs from the `COPY` statement `REJECTMAX` clause. If `COPY` reaches the number of exceptions defined by `REJECTMAX`, `COPY` aborts execution, and does not generate a `vertica.log` warning.

Working with External Tables

After creating external tables, you access them as any other table.

Managing Resources for External Tables

External tables require minimal additional resources. When you use a select query for an external table, HP Vertica uses a small amount of memory when reading external table data, since the table contents are not part of your database and are parsed each time the external table is used.

Backing Up and Restoring External Tables

Since the data in external tables is managed outside of HP Vertica, only the external table definitions, not the data files, are included in database backups.

Using Sequences and Identity Columns in External Tables

The COPY statement definition for external tables can include identity columns and sequences. Whenever a select statement queries the external table, sequences and identity columns are re-evaluated. This results in changing the external table column values, even if the underlying external table data remains the same.

Viewing External Table Definitions

When you create an external table, HP Vertica stores the COPY definition statement in the `table_definition` column of the `v_catalog.tables` system table.

1. To list all tables, use a `select *` query, as shown:

```
select * from v_catalog.tables where table_definition <> '';
```

2. Use a query such as the following to list the external table definitions (`table_definition`):

```
select table_name, table_definition from v_catalog.tables; table_name |
table_definition
-----+-----
t1          | COPY          FROM 'TMPDIR/external_table.dat' DELIMITER ','
t1_copy     | COPY          FROM 'TMPDIR/external_table.dat' DELIMITER ','
t2          | COPY FROM 'TMPDIR/external_table2.dat' DELIMITER ','
(3 rows)
```

External Table DML Support

Following are examples of supported queries, and others that are not:

| Supported | Unsupported |
|--|--|
| <code>SELECT * FROM external_table;</code> | <code>DELETE FROM external_table WHERE x = 5 ;</code> |
| <code>SELECT * FROM external_table where col1=4;</code> | <code>INSERT INTO external_table SELECT * FROM ext;</code> |
| <code>DELETE FROM ext WHERE id IN (SELECT x FROM external_table);</code> | |
| <code>INSERT INTO ext SELECT * FROM external_table;</code> | <code>SELECT * FROM external_table for update;</code> |

Using External Table Values

Following is a basic example of how you could use the values of an external table.

1. Create and display the contents of a file with some integer values:

```
[dbadmin@localhost ~]$ more ext.dat1
2
3
4
5
6
7
8
10
11
12
```

2. Create an external table pointing at `ext.dat`:

```
VMart=> create external table ext (x integer) as copy from '/home/dbadmin/ext.dat';
CREATE TABLE
```

3. Select the table contents:

```
VMart=> select * from ext; x
----
1
2
3
4
5
6
7
8
10
```

```
11
12
(11 rows)
```

4. Perform evaluation on some external table contents:

```
VMart=> select ext.x, ext.x + ext.x as double_x from ext where x > 5; x | double_x
-----+-----
 6 |          12
 7 |          14
 8 |          16
10 |          20
11 |          22
12 |          24
(6 rows)
```

5. Create a second table (second), also with integer values:

```
VMart=> create table second (y integer);CREATE TABLE
```

6. Populate the table with some values:

```
VMart=> copy second from stdin;Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 1
>> 3
>> 4
>> 5
>> \.
```

7. Join the external table (ext) with the table created in HP Vertica, called second:

```
VMart=> select * from ext join second on x=y; x | y
-----+-----
 1 | 1
 1 | 1
 3 | 3
 4 | 4
 5 | 5
(5 rows)
```

Using External Tables

External tables let you query data stored in files accessible to the HP Vertica database, but not managed by it. Creating external tables supplies read-only access through SELECT queries. You

cannot modify external tables through DML commands, such as INSERT, UPDATE, DELETE, and MERGE.

Using CREATE EXTERNAL TABLE AS COPY Statement

You create external tables with the CREATE EXTERNAL TABLE AS COPY... statement, shown in this basic example:

```
CREATE EXTERNAL TABLE tbl(i INT) AS COPY (i) FROM 'path1' ON node1, 'path2' ON node2;
```

For more details on the supported options to create an external table, see the [CREATE EXTERNAL TABLE](#) statement in the SQL Reference Manual.

The data you specify in the FROM clause of a CREATE EXTERNAL TABLE AS COPY statement can reside in one or more files or directories, and on one or more nodes. After successfully creating an external table, HP Vertica stores the table name and its COPY definition. Each time a select query references the external table, HP Vertica parses the COPY statement definition again to access the data. Here is a sample select statement:

```
SELECT * FROM tbl WHERE i > 10;
```

Storing HP Vertica Data in External Tables

While there are many requirements for you to use external table data, one reason is to store infrequently-accessed HP Vertica data on low-cost external media. If external storage is a goal at your site, the process to accomplish that requires exporting the older data to a text file, creating a bzip or gzip file of the export data, and saving the compressed file on an NFS disk. You can then create an external table to access the data any time it is required.

Using External Tables with User-Defined Load (UDL) Functions

You can also use external tables in conjunction with the UDL functions that you create. For more information about using UDLs, see [User Defined Load \(UDL\)](#) in the Programmer's Guide.

Organizing External Table Data

If the data you store in external tables changes regularly (for instance, each month in the case of storing recent historical data), your COPY definition statement can use wildcards to make parsing the stored COPY statement definition more dynamic. For instance, if you store monthly data on an NFS mount, you could organize monthly files within a top-level directory for a calendar year, such as:

```
/2012/monthly_archived_data/
```

In this case, the external table COPY statement will include a wildcard definition such as the following:

```
CREATE TABLE archive_data (...) AS COPY FROM 'nfs_name/2012/monthly_archived_data/*'
```

Whenever an HP Vertica query references the external table `months`, and HP Vertica parses the COPY statement, all stored data tables in the top-level `monthly_archived_data` directory are made accessible to the query.

Altering Table Definitions

Using **ALTER TABLE** syntax, you can respond to your evolving database schema requirements. The ability to change the definition of existing database objects facilitates ongoing maintenance. Furthermore, most of these options are both fast and efficient for large tables, because they consume fewer resources and less storage than having to stage data in a temporary table.

Here are some of the operations you can perform using the ALTER TABLE statement:

- Rename a table
- Add, drop, and rename columns
- Add and drop constraints
- Add table columns with a default derived expression
- Change a column's data type
- Change a table owner
- Rename a table schema
- Move a table to a new schema
- Change, reorganize, and remove table partitions

External Table Restrictions

Not all ALTER TABLE options are applicable for external tables. For instance, you cannot add a column to an external table, but you can rename the table:

```
=> ALTER TABLE mytable RENAME TO mytable2;  
ALTER TABLE
```

Exclusive ALTER TABLE Clauses

The following clauses are exclusive, which means you cannot combine them with another ALTER TABLE clause:

- ADD COLUMN
- RENAME COLUMN
- SET SCHEMA
- PARTITION BY
- REORGANIZE
- REMOVE PARTITIONING
- RENAME [TO]
- OWNER TO

Note: You can use the ADD constraints and DROP constraints clauses together.

Using Consecutive ALTER TABLE Commands

With the exception of performing a table rename, complete other ALTER TABLE statements consecutively. For example, to add multiple columns to a table, issue consecutive ALTER TABLE ADD COLUMN commands, ending each statement with a semicolon.

For more information about ALTER TABLE syntax and parameters, see the SQL Reference Manual.

Adding Table Columns

When you use the ADD COLUMN syntax as part of altering a table, HP Vertica takes an **O lock** on the table until the operation completes. The lock prevents DELETE, UPDATE, INSERT, and COPY statements from accessing the table, and blocks SELECT statements issued at SERIALIZABLE isolation level, until the operation completes. Each table can have a maximum of 1600 columns. You cannot add columns to a temporary table or to tables that have out-of-date **superprojections** with up-to-date buddies.

The following operations occur as part of adding columns:

- Inserts the default value for existing rows. For example, if the default expression is CURRENT_TIMESTAMP, all rows have the current timestamp.
- Automatically adds the new column with a unique projection column name to all **superprojections** of the table.
- Populates the column according to the **column-constraint** (DEFAULT, for example).

Note: Adding a column to a table does not affect the **K-safety** of the **physical schema** design, and you can add columns when nodes are down.

Updating Associated Table Views

Adding new columns to a table that has an associated view does not update the view's result set, even if the view uses a wildcard (*) to represent all table columns. To incorporate new columns, you must recreate the view. See [CREATE VIEW](#) in the SQL Reference Manual.

Specifying Default Expressions

When you add a new column to a table using `ALTER TABLE ADD COLUMN`, the default expression for the new column can evaluate to a user-defined scalar function, a constant, or a [derived expression](#) involving other columns of the same table.

The default expression of an `ADD COLUMN` statement disallows nested queries, or aggregate functions. Instead, use the `ALTER COLUMN` option, described in [Altering Table Columns](#).

About Using Volatile Functions

You cannot use a **volatile** function in the following two scenarios. Attempting to do so causes a rollback.

- As the default expression for an `ALTER TABLE ADD COLUMN` statement:

```
ALTER TABLE t ADD COLUMN a2 INT DEFAULT my_sequence.nextval;  
ROLLBACK: VOLATILE functions are not supported in a default expression  
ALTER TABLE t ADD COLUMN n2 INT DEFAULT my_sequence.currval;  
ROLLBACK: VOLATILE functions are not supported in a default expression  
ALTER TABLE t ADD COLUMN c2 INT DEFAULT RANDOM() + 1;  
ROLLBACK: VOLATILE functions are not supported in a default expression
```

- As the default expression for an `ALTER TABLE ALTER COLUMN` statement on an external table:

```
ALTER TABLE mytable ADD COLUMN a2 FLOAT DEFAULT RANDOM();  
ROLLBACK 5241: Unsupported access to external table  
ALTER TABLE mytable ALTER COLUMN x SET DEFAULT RANDOM();  
ROLLBACK 5241: Unsupported access to external table
```

You *can* specify a volatile function as a column default expression using the `ALTER TABLE ALTER COLUMN` statement:

```
ALTER TABLE t ALTER COLUMN a2 SET DEFAULT my_sequence.nextval;
```

Updating Associated Table Views

Adding new columns to a table that has an associated view does not update the view's result set, even if the view uses a wildcard (*) to represent all table columns. To incorporate new columns, you must recreate the view. See [CREATE VIEW](#) in the SQL Reference Manual.

Altering Table Columns

Use `ALTER COLUMN` syntax to alter an existing table column to change, drop, or establish a default expression for the column. You can also use `DROP DEFAULT` to remove a default expression.

Any new data that you load after altering a column will conform to the modified table definition. For example:

- After a `DROP COLUMN` operation completes, data backed up from the current epoch onward will recover without the column. Data recovered from a backup prior to the current epoch will re-add the table column. Because drop operations physically purge object storage and catalog definitions (table history) from the table, `AT EPOCH` (historical) queries return nothing for the dropped column.
- If you change a column's data type from `CHAR(8)` to `CHAR(16)` in epoch 10 and you restore the database from epoch 5, the column will be `CHAR(8)` again.

Adding Columns with a Default Derived Expression

You can add one or more columns to a table and set the default value as an expression. The expression can reference another column in the same table, or be calculated with a user-defined function (see [Types of UDFs](#) in the Programmer's Guide). You can alter unstructured tables to use a derived expression as described in [Altering Unstructured Tables](#). HP Vertica computes a default value within a row. This flexibility is useful for adding a column to a large fact table that shows another view on the data without having to `INSERT . . . SELECT` a large data set.

Adding columns requires an [O lock](#) on the table until the add operation completes. This lock prevents `DELETE`, `UPDATE`, `INSERT`, and `COPY` statements from affecting the table, and blocks `SELECT` statements issued at `SERIALIZABLE` isolation level, until the operation completes. Only the *new* data you load after the alter operation completes is derived from the expression.

You cannot include a nested query or an aggregate function as a default expression. The column must use a specific expression that involves other elements in the same row.

You cannot specify a default expression that derives data from another derived column. This means that if you already have a column with a default derived value expression, you cannot add another column whose default references the existing column.

Note: You can add a column when nodes are down.

Add a Default Column Value Derived From Another Column

1. Create a sample table called `t` with `timestamp`, `integer` and `varchar(10)` columns:

```
=> CREATE TABLE t (a TIMESTAMP, b INT, c VARCHAR(10));
```



```
f      |
public | t | b      | int      | 8 |          | f      |
f      |
public | t | c      | varchar(10) | 10 |          | f      |
f      |
public | t | d      | int      | 8 | date_part('month', t.a) | f      |
f      |
(4 rows)
```

6. Drop the sample table t:

```
=> DROP TABLE t;
DROP TABLE
```

Add a Default Column Value Derived From a UDSF

This example shows a user-defined scalar function that adds two integer values. The function is called `add2ints` and takes two arguments.

1. Develop and deploy the function, as described in [Developing and Using User Defined Functions](#).
2. Create a sample table, `t1`, with two integer columns:

```
=> CREATE TABLE t1 ( x int, y int );
CREATE TABLE
```

3. Insert some values into `t1`:

```
=> insert into t1 values (1,2);
OUTPUT
-----
      1
(1 row)
=> insert into t1 values (3,4);
OUTPUT
-----
      1
(1 row)
```

4. Use `ALTER TABLE` to add a column to `t1` with the default column value derived from the UDSF, `add2ints`:

```
alter table t1 add column z int default add2ints(x,y);
ALTER TABLE
```

5. List the new column:

```
select z from t1;
 z
----
 3
 7
(2 rows)
```

Changing a column's Data Type

You can change a table column's data type for any type whose conversion does not require storage reorganization. For example, the following types are the conversions that HP Vertica supports:

- Binary types—expansion and contraction but *cannot* convert between BINARY and VARBINARY types.
- Character types—all conversions allowed, even between CHAR and VARCHAR
- Exact numeric types—INTEGER, INT, BIGINT, TINYINT, INT8, SMALLINT, and all NUMERIC values of scale ≤ 18 and precision 0 are interchangeable. For NUMERIC data types, you cannot alter precision, but you can change the scale in the ranges (0-18), (19-37), and so on.

HP Vertica does not allow data type conversion on types that require storage reorganization:

- Boolean type conversion to other types
- DATE/TIME type conversion
- Approximate numeric type conversions
- Between BINARY and VARBINARY types

You can expand (and shrink) columns within the same class of data type, which is useful if you want to store longer strings in a column. HP Vertica validates the data before it performs the conversion.

For example, if you try to convert a column from varchar(25) to varchar(10) and that column holds a string with 20 characters, the conversion will fail. HP Vertica allows the conversion as long as that column does not have a string larger than 10 characters.

Examples

The following example expands an existing CHAR column from 5 to 10:

```
=> CREATE TABLE t (x CHAR, y CHAR(5));CREATE TABLE
```

```
=> ALTER TABLE t ALTER COLUMN y SET DATA TYPE CHAR(10);  
ALTER TABLE  
=> DROP TABLE t;  
DROP TABLE
```

This example illustrates the behavior of a changed column's type. First you set column y's type to VARCHAR(5) and then insert strings with characters that equal 5 and exceed 5:

```
=> CREATE TABLE t (x VARCHAR, y VARCHAR);CREATE TABLE  
=> ALTER TABLE t ALTER COLUMN y SET DATA TYPE VARCHAR(5);  
ALTER TABLE  
=> INSERT INTO t VALUES ('1232344455','hello');  
OUTPUT  
-----  
1  
(1 row)  
=> INSERT INTO t VALUES ('1232344455','hello1');  
ERROR 4797: String of 6 octets is too long for type Varchar(5)  
=> DROP TABLE t;  
DROP TABLE
```

You can also contract the data type's size, as long as that altered column contains no strings greater than 5:

```
=> CREATE TABLE t (x CHAR, y CHAR(10));CREATE TABLE  
=> ALTER TABLE t ALTER COLUMN y SET DATA TYPE CHAR(5);  
ALTER TABLE  
=> DROP TABLE t;  
DROP TABLE
```

You cannot convert types between binary and varbinary. For example, the table definition below contains two binary columns, so when you try to convert column y to a varbinary type, HP Vertica returns a ROLLBACK message:

```
=> CREATE TABLE t (x BINARY, y BINARY);CREATE TABLE  
=> ALTER TABLE t ALTER COLUMN y SET DATA TYPE VARBINARY;--N  
ROLLBACK 2377: Cannot convert column "y" from "binary(1)" to type "varbinary(80)"  
=> DROP TABLE t;  
DROP TABLE
```

How to Perform an Illegitimate Column Conversion

The SQL standard disallows an **illegitimate** column conversion, but you can work around this restriction if you need to convert data from a non-SQL database. The following example takes you through the process step by step, where you'll manage your own **epochs**.

Given a `sales` table with columns `id` (INT) and `price` (VARCHAR), assume you want to convert the VARCHAR column to a NUMERIC field. You'll do this by adding a temporary column whose default value is derived from the existing price column, rename the column, and then drop the original column.

1. Create the sample table with INTEGER and VARCHAR columns and insert two rows.

```
=> CREATE TABLE sales(id INT, price VARCHAR) UNSEGMENTED ALL NODES;CREATE TABLE
=> INSERT INTO sales VALUES (1, '$50.00');
=> INSERT INTO sales VALUES (2, '$100.00');
```

2. Commit the transaction:

```
=> COMMIT;COMMIT
```

3. Query the sales table:

```
=> SELECT * FROM SALES; id | price
-----+-----
  1 | $50.00
  2 | $100.00
(2 rows)
```

4. Add column temp_price. This is your temporary column.

```
=> ALTER TABLE sales ADD COLUMN temp_price NUMERIC DEFAULT SUBSTR(sales.price, 2)::NUMERIC;ALTER TABLE
```

5. Query the sales table, and you'll see the new temp_price column with its derived NUMERIC values:

```
=> SELECT * FROM SALES; id | price | temp_price
-----+-----+-----
  1 | $50.00 | 50.0000000000000000
  2 | $100.00 | 100.0000000000000000
(2 rows)
```

6. Drop the default expression from that column.

```
=> ALTER TABLE sales ALTER COLUMN temp_price DROP DEFAULT;ALTER TABLE
```

7. Advance the **AHM**:

```
SELECT advance_epoch(1); advance_epoch
-----
New Epoch: 83
(1 row)
```

8. Manage epochs:

```
SELECT manage_epoch();          manage_epoch
-----
Current Epoch=83, AHM Epoch=82
(1 row)
```

9. Drop the original price column.

```
=> ALTER TABLE sales DROP COLUMN price CASCADE;ALTER COLUMN
```

10. Rename the new (temporary) temp_price column back to its original name, price:

```
=> ALTER TABLE sales RENAME COLUMN temp_price to price;ALTER COLUMN
```

11. Query the sales table one last time:

```
=> SELECT * FROM SALES; id |      price
-----
1 | 50.0000000000000000
2 | 100.0000000000000000
(2 rows)
```

12. Clean up (drop table sales):

```
=> DROP TABLE sales;DROP TABLE
```

See [ALTER TABLE](#) in the SQL Reference Manual

Adding Constraints on Columns

To add constraints on a new column:

1. Use the ALTER TABLE ADD COLUMN clause to add a new table column.
2. Use ALTER TABLE ADD CONSTRAINT to define constraints for the new column.

Adding and Removing NOT NULL Constraints

Use the [SET | DROP] NOT NULL clause to add (SET) or remove (DROP) a NOT NULL constraint on the column.

When a column is a primary key and you drop the primary key constraint, the column retains the NOT NULL constraint. If you want to allow that column to now contain NULL values, use [DROP NOT NULL] to remove the NOT NULL constraint.

Examples

```
ALTER TABLE T1 ALTER COLUMN x SET NOT NULL;  
ALTER TABLE T1 ALTER COLUMN x DROP NOT NULL;
```

Note: Using the [SET | DROP] NOT NULL clause does not validate whether the column data conforms to the NOT NULL constraint. Use [ANALYZE_CONSTRAINTS](#) to check for constraint violations in a table.

See Also

- [About Constraints](#)

Dropping a Table Column

When you use the ALTER TABLE ... DROP COLUMN statement to drop a column, HP Vertica drops both the specified column from the table and the **ROS containers** that correspond to the dropped column.

The syntax looks like this:

```
ALTER TABLE [[db-name.]schema.]table-name ... | DROP [ COLUMN ] column-name [ CASCADE | RESTRICT ]
```

Because drop operations physically purge object storage and catalog definitions (table history) from the table, AT EPOCH (historical) queries return nothing for the dropped column.

The altered table has the same object ID.

Note: Drop column operations can be fast because these catalog-level changes do not require data reorganization, letting you quickly reclaim disk storage.

Restrictions

- At the table level, you cannot drop or alter a primary key column or a column participating in the table's partitioning clause.
- At the projection level, you cannot drop the first column in a projection's sort order or columns that participate in the segmentation expression of a projection.
- All nodes must be up for the drop operation to succeed.

Using CASCADE to Force a Drop

You can work around some of the restrictions by using the CASCADE keyword which enforces minimal reorganization of the table's definition in order to drop the column. You can use CASCADE

to drop a column if that column fits into one of the scenarios in the following table. Note that in all cases that use CASCADE, HP Vertica tries to drop the projection(s) and will roll back if **K-safety** is compromised:

| Column to drop ... | DROP column CASCADE behavior |
|---|---|
| Has a constraint of any kind on it | HP Vertica will drop the column with CASCADE specified when a FOREIGN KEY constraint depends on a UNIQUE or PRIMARY KEY constraint on the referenced columns. |
| Participates in the projection's sort order | If you drop a column using the CASCADE keyword, HP Vertica truncates the projection's sort order up to and including the projection that has been dropped without impact on physical storage for other columns and then drops the specified column. For example if a projection's columns are in sort order (a,b,c), dropping column b causes the projection's sort order to be just (a), omitting column (c). |
| Participates in ONE of the following: <ul style="list-style-type: none"> • Is a pre-join projection • Participates in the projection's segmentation expression | <p>In these scenarios, HP Vertica drops any non-critical, dependent projections, maintains the superprojection that contains the data, and drops the specified column. When a pre-join projection contains a column to be dropped with CASCADE, HP Vertica tries to drop the projection.</p> <p>Assume, for example, that you have a table with multiple projections and where one projection has the column you are trying to drop is part of the segmentation clause. When you specify CASCADE, the DROP COLUMN statement tries to implicitly drop the projection that has this column in the segmentation clause. If it succeeds the transaction completes, but if it violates k-safety the transaction rolls back.</p> <p>Although this is a DROP COLUMN ... CASCADE operation (not DROP PROJECTION), HP Vertica could encounter cases when it is not possible to drop a projection's column without reorganizing the projection. In these cases, CASCADE will try to drop the projection itself to maintain data integrity. If K-safety is compromised, the operation rolls back.</p> |

Examples

The following series of commands successfully drops a BYTEA data type column:

```
=> CREATE TABLE t (x BYTEA(65000), y BYTEA, z BYTEA(1));CREATE TABLE
=> ALTER TABLE t DROP COLUMN y;
ALTER TABLE
=> SELECT y FROM t;
ERROR 2624: Column "y" does not exist
=> ALTER TABLE t DROP COLUMN x RESTRICT;
ALTER TABLE
=> SELECT x FROM t;
ERROR 2624: Column "x" does not exist
=> SELECT * FROM t;
z
```

```
---  
(0 rows)  
=> DROP TABLE t CASCADE;  
DROP TABLE
```

The following series of commands tries to drop a FLOAT(8) column and fails because there are not enough projections to maintain k-safety.

```
=> CREATE TABLE t (x FLOAT(8),y FLOAT(08));CREATE TABLE  
=> ALTER TABLE t DROP COLUMN y RESTRICT;  
ALTER TABLE  
=> SELECT y FROM t;  
ERROR 2624: Column "y" does not exist  
=> ALTER TABLE t DROP x CASCADE;  
ROLLBACK 2409: Cannot drop any more columns in t  
=> DROP TABLE t CASCADE;
```

Moving a Table to Another Schema

The ALTER TABLE SET SCHEMA statement moves a table from one schema to another. Moving a table requires that you have CREATE privileges for the destination schema. You can move only one table between schemas at a time. You cannot move temporary tables between schemas.

SET SCHEMA has two options, CASCADE and RESTRICT. CASCADE, which is the default, automatically moves all projections that are anchored on the source table to the destination schema, regardless of the schema in which the projections reside. The RESTRICT option moves only projections that are anchored on the source table and which also reside in the same schema.

If a table of the same name or any of the projections that you want to move already exist in the new schema, the statement rolls back and does not move either the table or any projections. To work around name conflicts:

1. Rename any conflicting table or projections that you want to move.
2. Run the ALTER TABLE SET SCHEMA statement again.

Note: HP Vertica lets you move system tables to system schemas. Moving system tables could be necessary to support designs created through the **Database Designer**.

Changing a Table Owner

The ability to change table ownership is useful when moving a table from one schema to another. Ownership reassignment is also useful when a table owner leaves the company or changes job responsibilities. Because you can change the table owner, the tables won't have to be completely rewritten, you can avoid loss in productivity.

The syntax looks like this:

```
ALTER TABLE [[db-name.]schema.]table-name OWNER TO new-owner name
```

In order to alter table ownership, you must be either the table owner or a **superuser**.

A change in table ownership transfers just the owner and not privileges; grants made by the original owner are dropped and all existing privileges on the table are revoked from the previous owner. However, altering the table owner transfers ownership of dependent sequence objects (associated IDENTITY/AUTO-INCREMENT sequences) but does not transfer ownership of other referenced sequences. See [ALTER SEQUENCE](#) for details on transferring sequence ownership.

Notes

- Table privileges are separate from schema privileges; therefore, a table privilege change or table owner change does not result in any schema privilege change.
- Because projections define the physical representation of the table, HP Vertica does not require separate projection owners. The ability to create or drop projections is based on the table privileges on which the projection is anchored.
- During the alter operation HP Vertica updates projections anchored on the table owned by the old owner to reflect the new owner. For **pre-join projection** operations, HP Vertica checks for privileges on the referenced table.

Example

In this example, user Bob connects to the database, looks up the tables, and transfers ownership of table t33 from himself to user Alice.

```
=> \c - BobYou are now connected as user "Bob".
=> \d
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | applog | table | dbadmin |
 public | t33 | table | Bob |
(2 rows)
=> ALTER TABLE t33 OWNER TO Alice;
ALTER TABLE
```

Notice that when Bob looks up database tables again, he no longer sees table t33.

```
=> \d
          List of tables
          List of tables
 Schema | Name | Kind | Owner | Comment
-----+-----+-----+-----+-----
 public | applog | table | dbadmin |
(1 row)
```

When user Alice connects to the database and looks up tables, she sees she is the owner of table t33.

```
=> \c - AliceYou are now connected as user "Alice".
=> \d
          List of tables
 Schema | Name | Kind | Owner | Comment
```

```
-----+-----+-----+-----+-----
public | t33 | table | Alice |
(2 rows)
```

Either Alice or a superuser can transfer table ownership back to Bob. In the following case a superuser performs the transfer.

```
=> \c - dbadminYou are now connected as user "dbadmin".
=> ALTER TABLE t33 OWNER TO Bob;
ALTER TABLE
=> \d

          List of tables
 Schema |   Name   | Kind | Owner  | Comment
-----+-----+-----+-----+-----
public | applog   | table | dbadmin |
public | comments | table | dbadmin |
public | t33      | table | Bob     |
s1     | t1       | table | User1   |
(4 rows)
```

You can also query the [V_CATALOG.TABLES](#) system table to view table and owner information. Note that a change in ownership does not change the table ID.

In the below series of commands, the superuser changes table ownership back to Alice and queries the TABLES system table.

```
=> ALTER TABLE t33 OWNER TO Alice;ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM
tables; table_schema_id | table_schema | table_id | table_name | owner_id
| owner_name
-----+-----+-----+-----+-----+-----
45035996273704968 | public      | 45035996273713634 | applog     | 45035996273704962 |
dbadmin
45035996273704968 | public      | 45035996273724496 | comments   | 45035996273704962 |
dbadmin
45035996273730528 | s1          | 45035996273730548 | t1         | 45035996273730516 |
User1
45035996273704968 | public      | 45035996273795846 | t33        | 45035996273724576 |
Alice
(5 rows)
```

Now the superuser changes table ownership back to Bob and queries the TABLES table again. Nothing changes but the owner_name row, from Alice to Bob.

```
=> ALTER TABLE t33 OWNER TO Bob;ALTER TABLE
=> SELECT table_schema_id, table_schema, table_id, table_name, owner_id, owner_name FROM
tables;
table_schema_id | table_schema | table_id | table_name | owner_id |
owner_name-----+-----+-----+-----+-----+-----
45035996273704968 | public      | 45035996273713634 | applog     | 45035996273704962 |
dbadmin
```

```
45035996273704968 | public      | 45035996273724496 | comments | 45035996273704962 |
dbadmin
45035996273730528 | s1        | 45035996273730548 | t1       | 45035996273730516 |
User1
45035996273704968 | public      | 45035996273793876 | foo      | 45035996273724576 |
Alice
45035996273704968 | public      | 45035996273795846 | t33     | 45035996273714428 |
Bob
(5 rows)
```

Table Reassignment with Sequences

Altering the table owner transfers ownership of only associated IDENTITY/AUTO-INCREMENT sequences but not other reference sequences. For example, in the below series of commands, ownership on sequence s1 does not change:

```
=> CREATE USER u1;CREATE USER
=> CREATE USER u2;
CREATE USER
=> CREATE SEQUENCE s1 MINVALUE 10 INCREMENT BY 2;
CREATE SEQUENCE
=> CREATE TABLE t1 (a INT, id INT DEFAULT NEXTVAL('s1'));
CREATE TABLE
=> CREATE TABLE t2 (a INT, id INT DEFAULT NEXTVAL('s1'));
CREATE TABLE
=> SELECT sequence_name, owner_name FROM sequences;
sequence_name | owner_name
-----+-----
s1            | dbadmin
(1 row)
=> ALTER TABLE t1 OWNER TO u1;
ALTER TABLE
=> SELECT sequence_name, owner_name FROM sequences;
sequence_name | owner_name
-----+-----
s1            | dbadmin
(1 row)
=> ALTER TABLE t2 OWNER TO u2;
ALTER TABLE
=> SELECT sequence_name, owner_name FROM sequences;
sequence_name | owner_name
-----+-----
s1            | dbadmin
(1 row)
```

See Also

- [Changing a Sequence Owner](#)

Changing a Sequence Owner

The `ALTER SEQUENCE` command lets you change the attributes of an existing sequence. All changes take effect immediately, within the same session. Any parameters not set during an `ALTER SEQUENCE` statement retain their prior settings.

If you need to change sequence ownership, such as if an employee who owns a sequence leaves the company, you can do so with the following `ALTER SEQUENCE` syntax:

```
ALTER SEQUENCE sequence-name OWNER TO new-owner-name;
```

This operation immediately reassigns the sequence from the current owner to the specified new owner.

Only the sequence owner or a **superuser** can change ownership, and reassignment does not transfer grants from the original owner to the new owner; grants made by the original owner are dropped.

Note: Renaming a table owner transfers ownership of dependent sequence objects (associated `IDENTITY/AUTO-INCREMENT` sequences) but does not transfer ownership of other referenced sequences. See [Changing a Table Owner](#).

Example

The following example reassigns sequence ownership from the current owner to user Bob:

```
=> ALTER SEQUENCE sequential OWNER TO Bob;
```

See [ALTER SEQUENCE](#) in the SQL Reference Manual for details.

Renaming Tables

The `ALTER TABLE RENAME TO` statement lets you rename one or more tables. The new table names must not exist already.

Renaming tables does not affect existing pre-join projections because pre-join projections refer to tables by their unique numeric object IDs (OIDs). Renaming tables also does not change the table OID.

To rename two or more tables:

1. List the tables to rename with a comma-delimited list, specifying a schema-name after part of the table specification only before the `RENAME TO` clause:

```
=> ALTER TABLE S1.T1, S1.T2 RENAME TO U1, U2;
```

The statement renames the listed tables to their new table names from left to right, matching them sequentially, in a one-to-one correspondence.

The `RENAME TO` parameter is applied atomically so that all tables are renamed, or none of the tables is renamed. For example, if the number of tables to rename does not match the number of new names, none of the tables is renamed.

2. Do not specify a schema-name as part of the table specification after the `RENAME TO` clause, since the statement applies to only one schema. The following example generates a syntax error:

```
=> ALTER TABLE S1.T1, S1.T2 RENAME TO S1.U1, S1.U2;
```

Note: Renaming a table referenced by a view causes the view to fail, unless you create another table with the previous name to replace the renamed table.

Using Rename to Swap Tables Within a Schema

You can use the `ALTER TABLE RENAME TO` statement to swap tables within a schema without actually moving data. You cannot swap tables across schemas.

To swap tables within a schema (example statement is split to explain steps):

1. Enter the names of the tables to swap, followed by a new temporary table placeholder (`temps`):

```
=> ALTER TABLE T1, T2, temps
```

2. Use the `RENAME TO` clause to swap the tables: `T1` to `temps`, `T2` to `T1`, and `temps` to `T2`:

```
RENAME TO temps, T1, T2;
```


Using Named Sequences

Named sequences are database objects that generate unique numbers in ascending or descending sequential order. They are most often used when an application requires a unique identifier in a table or an expression. Once a named sequence returns a value, it never returns that same value again. Named sequences are independent objects, and while you can use their values in tables, they are not subordinate to them.

Types of Incrementing Value Objects

In addition to named sequences, HP Vertica supports two other kinds of sequence objects, which also increment values:

- **Auto-increment column value:** a sequence available only for numeric column types. Auto-increment sequences automatically assign the next incremental sequence value for that column when a new row is added to the table.
- **Identity column:** a sequence available only for numeric column types.

Auto-increment and Identity sequences are defined through [column constraints](#) in the [CREATE TABLE](#) statement and are incremented each time a row is added to the table. Both of these object types are table-dependent and do not persist independently. The identity value is never rolled back even if the transaction that tries to insert a value into the table is not committed. The [LAST_INSERT_ID](#) function returns the last value generated for an auto-increment or identity column.

Each type of sequence object has a set of properties and controls. A named sequence has the most controls, and an Auto-increment sequence the least. The following table lists the major differences between the three sequence objects:

| Behavior | Named Sequence | Identity | Auto-increment |
|------------------------------|----------------|----------|----------------|
| Default cache value 250K | X | X | X |
| Set initial cache | X | X | |
| Define start value | X | X | |
| Specify increment unit | X | X | |
| Create as standalone object | X | | |
| Create as column constraint | | X | X |
| Exists only as part of table | | X | X |
| Requires name | X | | |
| Use in expressions | X | | |
| Unique across tables | X | | |

| Behavior | Named Sequence | Identity | Auto-increment |
|-------------------------------|----------------|----------|----------------|
| Change parameters | X | | |
| Move to different schema | X | | |
| Set to increment or decrement | X | | |
| Grant privileges to object | X | | |
| Specify minimum value | X | | |
| Specify maximum value | X | | |
| Always starts at 1 | | | X |

While sequence object values are guaranteed to be unique, they are not guaranteed to be contiguous. Since sequences are not necessarily contiguous, you may interpret the returned values as missing. For example, two nodes can increment a sequence at different rates. The node with a heavier processing load increments the sequence, but the values are not contiguous with those being incremented on the other node.

Using a Sequence with an Auto_Increment or Identity Column

Each table can contain only one `auto_increment` or `identity` column. A table can contain both an `auto_increment` or `identity` column, and a named sequence, as in the next example, illustrating a table with both types of sequences:

```
VMart=> CREATE TABLE test2 (id INTEGER NOT NULL UNIQUE,
    middle INTEGER DEFAULT NEXTVAL('my_seq'),
    next INT, last auto_increment);
CREATE TABLE
```

Named Sequence Functions

When you create a named sequence object, you can also specify the increment or decrement value. The default is 1. Use these functions with named sequences:

- **NEXTVAL** — Advances the sequence and returns the next sequence value. This value is incremented for ascending sequences and decremented for descending sequences. The first time you call `NEXTVAL` after creating a sequence, the function sets up the cache in which to store the sequence values, and returns either the default sequence value, or the start number you specified with `CREATE SEQUENCE`.
- **CURRVAL** — Returns the LAST value across all nodes returned by a previous invocation of `NEXTVAL` in the same session. If there were no calls to `NEXTVAL` after creating a sequence, the `CURRVAL` function returns an error:

```
dbt=> create sequence seq2;  
CREATE SEQUENCE  
dbt=> select currval('seq2');  
ERROR 4700: Sequence seq2 has not been accessed in the session
```

You can use the NEXTVAL and CURRVAL functions in INSERT and COPY expressions.

Using DDL Commands and Functions With Named Sequences

For details, see the following related statements and functions in the SQL Reference Manual:

| Use this statement... | To... |
|---------------------------------|---|
| CREATE SEQUENCE | Create a named sequence object. |
| ALTER SEQUENCE | Alter named sequence parameters, rename a sequence within a schema, or move a named sequence between schemas. |
| DROP SEQUENCE | Remove a named sequence object. |
| GRANT SEQUENCE | Grant user privileges to a named sequence object. See also Sequence Privileges . |

Creating Sequences

Create a sequence using the [CREATE SEQUENCE](#) statement. All of the parameters (besides a sequence name) are optional.

The following example creates an ascending sequence called `my_seq`, starting at 100:

```
dbt=> create sequence my_seq START 100;  
CREATE SEQUENCE
```

After creating a sequence, you must call the [NEXTVAL](#) function at least once in a session to create a cache for the sequence and its initial value. Subsequently, use NEXTVAL to increment the sequence. Use the [CURRVAL](#) function to get the current value.

The following NEXTVAL function instantiates the newly-created `my_seq` sequence and sets its first number:

```
=> SELECT NEXTVAL('my_seq');  
nextval  
-----  
100
```

```
(1 row)
```

If you call CURRVAL before NEXTVAL, the system returns an error:

```
dbt=> SELECT CURRVAL('my_seq');  
ERROR 4700: Sequence my_seq has not been accessed in the session
```

The following command returns the current value of this sequence. Since no other operations have been performed on the newly-created sequence, the function returns the expected value of 100:

```
=> SELECT CURRVAL('my_seq');  
currval  
-----  
100  
(1 row)
```

The following command increments the sequence value:

```
=> SELECT NEXTVAL('my_seq');  
nextval  
-----  
101  
(1 row)
```

Calling the CURRVAL again function returns only the current value:

```
=> SELECT CURRVAL('my_seq');  
currval  
-----  
101  
(1 row)
```

The following example shows how to use the my_seq sequence in an INSERT statement.

```
=> CREATE TABLE customer (  
    lname VARCHAR(25),  
    fname VARCHAR(25),  
    membership_card INTEGER,  
    id INTEGER  
);  
=> INSERT INTO customer VALUES ('Hawkins', 'John', 072753, NEXTVAL('my_seq'));
```

Now query the table you just created to confirm that the ID column has been incremented to 102:

```
=> SELECT * FROM customer;  
  lname | fname | membership_card | id  
-----+-----+-----+-----  
Hawkins | John  |           72753 | 102  
(1 row)
```

The following example shows how to use a sequence as the default value for an INSERT command:

```
=> CREATE TABLE customer2(  
    id INTEGER DEFAULT NEXTVAL('my_seq'),  
    lname VARCHAR(25),  
    fname VARCHAR(25),  
    membership_card INTEGER  
);  
=> INSERT INTO customer2 VALUES (default, 'Carr', 'Mary', 87432);
```

Now query the table you just created. The ID column has been incremented again to 103:

```
=> SELECT * FROM customer2;  
 id | lname | fname | membership_card  
-----+-----+-----+-----  
 103 | Carr  | Mary  |           87432  
(1 row)
```

The following example shows how to use NEXTVAL in a SELECT statement:

```
=> SELECT NEXTVAL('my_seq'), lname FROM customer2;  
NEXTVAL | lname  
-----+-----  
    104 | Carr  
(1 row)
```

As you can see, each time you call NEXTVAL(), the value increments.

The CURRVAL() function returns the current value.

Altering Sequences

The [ALTER SEQUENCE](#) statement lets you change the attributes of a previously-defined sequence. Changes take effect in the next database session. Any parameters not specifically set in the ALTER SEQUENCE command retain their previous settings.

The ALTER SEQUENCE statement lets you rename an existing sequence, or the schema of a sequence, but you cannot combine either of these changes with any other optional parameters.

Note: Using ALTER SEQUENCE to set a START value below the CURRVAL can result in duplicate keys.

Examples

The following example modifies an ascending sequence called my_seq to start at 105:

```
ALTER SEQUENCE my_seq RESTART WITH 105;
```

The following example moves a sequence from one schema to another:

```
ALTER SEQUENCE [public.]my_seq SET SCHEMA vmart;
```

The following example renames a sequence in the Vmart schema:

```
ALTER SEQUENCE [vmart.]my_seq RENAME TO serial;
```

Remember that changes occur only after you start a new database session. For example, if you create a sequence named `my_sequence` and start the value at 10, each time you call the `NEXTVAL` function, you increment by 1, as in the following series of commands:

```
CREATE SEQUENCE my_sequence START 10;SELECT NEXTVAL('my_sequence');
nextval
-----
      10
(1 row)
SELECT NEXTVAL('my_sequence');
nextval
-----
      11
(1 row)
```

Now issue the `ALTER SEQUENCE` statement to assign a new value starting at 50:

```
ALTER SEQUENCE my_sequence RESTART WITH 50;
```

When you call the `NEXTVAL` function, the sequence is incremented again by 1 value:

```
SELECT NEXTVAL('my_sequence'); NEXTVAL
-----
      12
(1 row)
```

The sequence starts at 50 only after restarting the database session:

```
SELECT NEXTVAL('my_sequence'); NEXTVAL
-----
      50
(1 row)
```

Distributed Sequences

When you create a sequence object, the `CACHE` parameter controls the sequence efficiency, by determining the number of sequence values each node maintains during a session. The default cache value is 250K, meaning that each node reserves 250,000 values for each sequence per session.

HP Vertica distributes a session across all nodes. After you create a sequence, the first time a node executes a `NEXTVAL()` function as part of a SQL statement, the node reserves its own cache of sequence values. The node then maintains that set of values for the current session. Other

nodes executing a NEXTVAL() function will also create and maintain their own cache of sequence values cache.

Note: If any node consumes all of its sequence values, HP Vertica must perform a catalog lock to obtain a new set of values. A catalog lock can be costly in terms of database performance, since certain activities, such as data inserts, cannot occur until HP Vertica releases the lock.

During a session, one node can use its allocated set of sequence values slowly, while another node uses its values more quickly. Therefore, the value returned from NEXTVAL in one statement can differ from the values returned in another statement executed on another node.

Regardless of the number of calls to NEXTVAL and CURRVAL, HP Vertica increments a sequence only once per row. If multiple calls to NEXTVAL occur within the same row, the function returns the same value. If sequences are used in join statements, HP Vertica increments a sequence once for each composite row output by the join.

The current value of a sequence is calculated as follows:

- At the end of every statement, the state of all sequences used in the session is returned to the initiator node.
- The initiator node calculates the maximum CURRVAL of each sequence across all states on all nodes.
- This maximum value is used as CURRVAL in subsequent statements until another NEXTVAL is invoked.

Sequence values in cache can be lost in the following situations:

- If a statement fails after NEXTVAL is called (thereby consuming a sequence value from the cache), the value is lost.
- If a disconnect occurs (for example, dropped session), any remaining values in the cache that have not been returned through NEXTVAL (unused) are lost.

To recover lost sequence values, you can run an [ALTER SEQUENCE](#) command to define a new sequence number generator, which resets the counter to the correct value in the next session.

Note: An elastic projection (a segmented projection created when Elastic Cluster is enabled) created with a modularhash segmentation expression uses hash instead.

The behavior of sequences across HP Vertica nodes is explained in the following examples.

Note: IDENTITY and AUTO_INCREMENT columns behave in a similar manner.

Example 1: The following example, which illustrates sequence distribution, assumes a 3-node cluster with node01 as the initiator node.

First create a simple table called dist:

```
CREATE TABLE dist (i INT, j VARCHAR);
```

Create a projection called oneNode and segment by column i on node01:

```
CREATE PROJECTION oneNode AS SELECT * FROM dist SEGMENTED BY i NODES node01;
```

Create a second projection called twoNodes and segment column x by modularhash on node02 and node03:

```
CREATE PROJECTION twoNodes AS SELECT * FROM dist SEGMENTED BY MODULARHASH(i) NODES node02, node03;
```

Create a third projection called threeNodes and segment column i by modularhash on all nodes (1-3):

```
CREATE PROJECTION threeNodes as SELECT * FROM dist SEGMENTED BY MODULARHASH(i) ALL NODES;
```

Insert some values:

```
COPY dist FROM STDIN;1|ONE
2|TWO
3|THREE
4|FOUR
5|FIVE
6|SIX
\.
```

Query the [STORAGE_CONTAINERS](#) table to return the projections on each node:

```
SELECT node_name, projection_name, total_row_count FROM storage_containers;
node_name | projection_name | total_row_count-----+-----+-----
--
node0001 | oneNode        |                6 --Contains rows with i=(1,2,3,4,5,6)
node0001 | threeNodes     |                2 --Contains rows with i=(3,6)
node0002 | twoNodes       |                3 --Contains rows with i=(2,4,6)
node0002 | threeNodes     |                2 --Contains rows with i=(1,4)
node0003 | twoNodes       |                3 --Contains rows with i=(1,3,5)
node0003 | threeNodes     |                2 --Contains rows with i=(2,5)
(6 rows)
```

The following table shows the segmentation of rows for projection oneNode:

| | | | | |
|---|-------|---------|-----|--------|
| 1 | ONE | Node012 | TWO | Node01 |
| 3 | THREE | Node01 | | |
| 4 | FOUR | Node01 | | |
| 5 | FIVE | Node01 | | |
| 6 | SIX | Node01 | | |

The following table shows the segmentation of rows for projection twoNodes:

| | | | | |
|---|-------|---------|-----|--------|
| 1 | ONE | Node032 | TWO | Node02 |
| 3 | THREE | Node03 | | |
| 4 | FOUR | Node02 | | |
| 5 | FIVE | Node03 | | |
| 6 | SIX | Node02 | | |

The following table shows the segmentation of rows for projection threeNodes:

| | | | | |
|---|-------|---------|-----|--------|
| 1 | ONE | Node022 | TWO | Node03 |
| 3 | THREE | Node01 | | |
| 4 | FOUR | Node02 | | |
| 5 | FIVE | Node03 | | |
| 6 | SIX | Node01 | | |

Create a sequence and specify a cache of 10. The sequence will cache up to 10 values in memory for performance. As per the [CREATE SEQUENCE](#) statement, the minimum value is 1 (only one value can be generated at a time, for example, no cache).

Example 2: Create a sequence named s1 and specify a cache of 10:

```
CREATE SEQUENCE s1 cache 10;SELECT s1.nextval, s1.currval, s1.nextval, s1.currval, j FROM
oneNode;
  nextval | currval | nextval | currval |  j
-----+-----+-----+-----+--
        1 |        1 |        1 |        1 |  ONE
        2 |        2 |        2 |        2 |  TWO
        3 |        3 |        3 |        3 | THREE
        4 |        4 |        4 |        4 |  FOUR
        5 |        5 |        5 |        5 |  FIVE
        6 |        6 |        6 |        6 |  SIX
(6 rows)
```

The following table illustrates the current state of the sequence for that session. It holds the current value, values remaining (the difference between the current value (6) and the cache (10)), and cache activity. There is no cache activity on node02 or node03.

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|----------|----------|
| Current value | 6 | NO CACHE | NO CACHE |
| Remainder | 4 | NO CACHE | NO CACHE |

Example 3: Return the current values from twoNodes:

```
SELECT s1.currval, j FROM twoNodes; currval |  j
-----+-----
        6 |  ONE
        6 | THREE
        6 |  FIVE
        6 |  TWO
        6 |  FOUR
        6 |  SIX
```

(6 rows)

Example 4: Now call NEXTVAL from threeNodes. The assumption is that node02 holds the cache before node03:

```
SELECT s1.nextval, j from threeNodes; nextval | j
-----+-----
    101 | ONE
    201 | TWO
     7  | THREE
    102 | FOUR
    202 | FIVE
     8  | SIX
(6 rows)
```

The following table illustrates the sequence cache state with values on node01, node02, and node03:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|--------|
| Current value | 8 | 102 | 202 |
| Left | 2 | 8 | 8 |

Example 5: Insert values from twoNodes into the destination table:

```
SELECT s1.currval, j FROM twoNodes; nextval | j
-----+-----
    202 | ONE
    202 | TWO
    202 | THREE
    202 | FOUR
    202 | FIVE
    202 | SIX
(6 rows)
```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|--------|
| Current value | 6 | 102 | 202 |
| Left | 4 | 8 | 8 |

Example 6: The following command runs on node02 only:

```
SELECT s1.nextval, j FROM twoNodes WHERE i = 2; nextval | j
-----+-----
    103 | TWO
(1 row)
```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|--------|
| Current value | 6 | 103 | 202 |
| Left | 4 | 7 | 8 |

Example 7: The following command calls the current value from twoNodes:

```
SELECT s1.currval, j FROM twoNodes; currval | j
-----+-----
 103 | ONE
 103 | TWO
 103 | THREE
 103 | FOUR
 103 | FIVE
 103 | SIX
(6 rows)
```

Example 8: This example assume that node02 holds the cache before node03:

```
SELECT s1.nextval, j FROM twoNodes; nextval | j
-----+-----
 203 | ONE
 104 | TWO
 204 | THREE
 105 | FOUR
 205 | FIVE
 106 | SIX
(6 rows)
```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|--------|
| Current value | 6 | 106 | 205 |
| Left | 4 | 6 | 5 |

Example 9: The following command calls the current value from oneNode:

```
SELECT s1.currval, j FROM twoNodes; currval | j
-----+-----
 205 | ONE
 205 | TWO
 205 | THREE
 205 | FOUR
 205 | FIVE
 205 | SIX
(6 rows)
```

Example 10: This example calls the NEXTVAL function on oneNode:

```
SELECT s1.nextval, j FROM oneNode; nextval | j
```

```

-----+-----
      7 | ONE
      8 | TWO
      9 | THREE
     10 | FOUR
     301 | FIVE
     302 | SIX
(6 rows)

```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|--------|
| Current value | 302 | 106 | 205 |
| Left | 8 | 4 | 5 |

Example 11: In this example, twoNodes is the outer table and threeNodes is the inner table to a merge join. threeNodes is resegmented as per twoNodes.

```
SELECT s1.nextval, j FROM twoNodes JOIN threeNodes ON twoNodes.i = threeNodes.i;
```

```

SELECT s1.nextval, j FROM oneNode; nextval |  j
-----+-----
     206 | ONE
     107 | TWO
     207 | THREE
     108 | FOUR
     208 | FIVE
     109 | SIX
(6 rows)

```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|--------|
| Current value | 302 | 109 | 208 |
| Left | 8 | 1 | 2 |

Example 12: This next example shows how sequences work with buddy projections.

```

--Same session
DROP TABLE t CASCADE;
CREATE TABLE t (i INT, j varchar(20));
CREATE PROJECTION threeNodes AS SELECT * FROM t
SEGMENTED BY MODULARHASH(i) ALL NODES KSAFE 1;
COPY t FROM STDIN;
1|ONE
2|TWO
3|THREE
4|FOUR

```

```

5|FIVE
6|SIX
\.
SELECT node_name, projection_name, total_row_count FROM storage_containers;
 node_name | projection_name | total_row_count
-----+-----
 node01    | threeNodes_b0  |                2
 node03    | threeNodes_b0  |                2
 node02    | threeNodes_b0  |                2
 node02    | threeNodes_b1  |                2
 node01    | threeNodes_b1  |                2
 node03    | threeNodes_b1  |                2
(6 rows)

```

The following function call assumes that node02 is down. It is the same session. Node03 takes up the work of node02:

```

SELECT s1.nextval, j FROM t; nextval | j
-----+-----
 401 | ONE
 402 | TWO
 305 | THREE
 403 | FOUR
 404 | FIVE
 306 | SIX
(6 rows)

```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|--------|
| Current value | 306 | 110 | 404 |
| Left | 4 | 0 | 6 |

Example 13: This example starts a new session.

```

DROP TABLE t CASCADE;CREATE TABLE t (i INT, j VARCHAR);
CREATE PROJECTION oneNode AS SELECT * FROM t SEGMENTED BY i NODES node01;
CREATE PROJECTION twoNodes AS SELECT * FROM t SEGMENTED BY MODULARHASH(i) NODES node02, node03;
CREATE PROJECTION threeNodes AS SELECT * FROM t SEGMENTED BY MODULARHASH(i) ALL NODES;
INSERT INTO t values (nextval('s1'), 'ONE');
SELECT * FROM t;
 i | j
-----+-----
501 | ONE
(1 rows)

```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|----------|----------|
| Current value | 501 | NO CACHE | NO CACHE |
| Left | 9 | 0 | 0 |

Example 14:

```
INSERT INTO t SELECT s1.nextval, 'TWO' FROM twoNodes;
      SELECT * FROM t; i | j
-----+-----
 501 | ONE  --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
 601 | TWO  --stored in node01 for oneNode, node03 for twoNodes, node01 for threeNodes
(2 rows)
```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|----------|
| Current value | 501 | 601 | NO CACHE |
| Left | 9 | 9 | 0 |

Example 15:

```
INSERT INTO t select s1.nextval, 'TRE' from threeNodes;SELECT * FROM t;
i | j
-----+-----
 501 | ONE  --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
 601 | TWO  --stored in node01 for oneNode, node03 for twoNodes, node01 for threeNodes
 502 | TRE  --stored in node01 for oneNode, node03 for twoNodes, node03 for threeNodes
 602 | TRE  --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
(4 rows)
```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|----------|
| Current value | 502 | 602 | NO CACHE |
| Left | 9 | 9 | 0 |

Example 16:

```
INSERT INTO t SELECT s1.currval, j FROM threeNodes WHERE i != 502;
      SELECT * FROM t; i | j
-----+-----
 501 | ONE  --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
 601 | TWO  --stored in node01 for oneNode, node03 for twoNodes, node01 for threeNodes
 502 | TRE  --stored in node01 for oneNode, node03 for twoNodes, node03 for threeNodes
 602 | TRE  --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
 602 | ONE  --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
 502 | TWO  --stored in node01 for oneNode, node03 for twoNodes, node03 for threeNodes
```

```
602 | TRE --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
(7 rows)
```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|----------|
| Current value | 502 | 602 | NO CACHE |
| Left | 9 | 9 | 0 |

Example 17:

```
INSERT INTO t VALUES (s1.currval + 1, 'QUA');
SELECT * FROM t;
 i | j
-----+-----
501 | ONE --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
601 | TWO --stored in node01 for oneNode, node03 for twoNodes, node01 for threeNodes
502 | TRE --stored in node01 for oneNode, node03 for twoNodes, node03 for threeNodes
602 | TRE --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
602 | ONE --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
502 | TWO --stored in node01 for oneNode, node03 for twoNodes, node03 for threeNodes
602 | TRE --stored in node01 for oneNode, node02 for twoNodes, node02 for threeNodes
603 | QUA
(8 rows)
```

The following table illustrates the sequence cache state:

| Sequence Cache State | Node01 | Node02 | Node03 |
|----------------------|--------|--------|----------|
| Current value | 502 | 602 | NO CACHE |
| Left | 9 | 9 | 0 |

See Also

- [Sequence Privileges](#)
- [ALTER SEQUENCE](#)
- [CREATE TABLE](#)
- [Column-Constraint](#)
- [CURRVAL](#)
- [DROP SEQUENCE](#)

- [GRANT \(Sequence\)](#)
- [NEXTVAL](#)

Loading Sequences

You can use a sequence as part of creating a table. The sequence must already exist, and have been instantiated using the NEXTVAL statement.

Creating and Instantiating a Sequence

The following example creates an ascending sequence called `my_seq`, starting at 100:

```
dbt=> create sequence my_seq START 100;  
CREATE SEQUENCE
```

After creating a sequence, you must call the [NEXTVAL](#) function at least once in a session to create a cache for the sequence and its initial value. Subsequently, use NEXTVAL to increment the sequence. Use the [CURRVAL](#) function to get the current value.

The following NEXTVAL function instantiates the newly-created `my_seq` sequence and sets its first number:

```
=> SELECT NEXTVAL('my_seq');  
nextval  
-----  
      100  
(1 row)
```

If you call CURRVAL before NEXTVAL, the system returns an error:

```
dbt=> SELECT CURRVAL('my_seq');  
ERROR 4700: Sequence my_seq has not been accessed in the session
```

Using a Sequence in an INSERT Command

Update sequence number values by calling the NEXTVAL function, which increments/decrements the current sequence and returns the *next* value. Use CURRVAL to return the *current* value. These functions can also be used in INSERT and COPY expressions.

The following example shows how to use a sequence as the default value for an INSERT command:

```
CREATE TABLE customer2( ID INTEGER DEFAULT NEXTVAL('my_seq'),  
  lname VARCHAR(25),  
  fname VARCHAR(25),  
  membership_card INTEGER
```



```
);  
INSERT INTO customer2 VALUES (default, 'Carr', 'Mary', 87432);
```

Now query the table you just created. The column named ID has been incremented by (1) again to 104:

```
SELECT * FROM customer2; ID | lname | fname | membership_card  
-----+-----+-----+-----  
104 | Carr | Mary | 87432  
(1 row)
```

Dropping Sequences

Use the [DROP SEQUENCE](#) function to remove a sequence. You cannot drop a sequence:

- If other objects depend on the sequence. The CASCADE keyword is not supported.
- That is used in the default expression of a column until all references to the sequence are removed from the default expression.

Example

The following command drops the sequence named `my_sequence`:

```
=> DROP SEQUENCE my_sequence;
```

Synchronizing Table Data with MERGE

The most convenient way to update, delete, and/or insert table data is using the [MERGE](#) statement, where you can combine multiple operations in a single command. When you write a MERGE statement, you specify the following:

- A target table—The master table that contains existing rows you want to update or insert into using rows from another (source) table.
- A source table—The table that contains the new and/or changed rows you'll use to update the target table.
- A search condition—The merge join columns, specified in the ON clause, that HP Vertica uses to evaluate each row in the source table to update, delete, and/or insert rows into the target table.
- Additional filters that instruct HP Vertica what to do when the search condition is or is not met. For example, when you use:
 - `WHEN MATCHED THEN UPDATE` clause— HP Vertica updates and/or deletes existing rows in the target table with data from the source table.

Note: HP Vertica assumes the values in the merge join column are unique. If more than one matching value is present in either the target or source table's join column, the MERGE statement could fail with a run-time error. See [Optimized Versus Non-optimized MERGE](#) for more information.

- `WHEN NOT MATCHED THEN INSERT` clause—HP Vertica inserts into the target table all rows from the source table that do not match any rows in the target table.

Optimized Versus Non-Optimized MERGE

By default, HP Vertica prepares an optimized query plan to improve merge performance when the MERGE statement and its tables meet certain criteria. If the criteria are not met, MERGE could run without optimization or return a run-time error. This section describes scenarios for both optimized and non-optimized MERGE.

Conditions for an Optimized MERGE

HP Vertica prepares an optimized query plan when *all* of the following conditions are true:

- The **target** table's join column has a unique or primary key constraint
- UPDATE and INSERT clauses include every column in the target table
- UPDATE and INSERT clause column attributes are identical

When the above conditions are not met, HP Vertica prepares a non-optimized query plan, and MERGE runs with the same performance as in HP Vertica 6.1 and prior.

Note: The **source** table's join column does not require a unique or primary key constraint to be eligible for an optimized query plan. Also, the source table can contain more columns than the target table, as long as the UPDATE and INSERT clauses use the same columns and the column attributes are the same.

How to determine if a MERGE statement is eligible for optimization

To determine whether a MERGE statement is eligible for optimization, prefix MERGE with the [EXPLAIN](#) keyword and examine the plan's textual output. (See [Viewing the MERGE Query Plan](#) for examples.) A `Semi` path indicates the statement is eligible for optimization, whereas a `Right Outer` path indicates the statement is ineligible and will run with the same performance as MERGE in previous releases unless a duplicate merge join key is encountered at query run time.

About duplicate matching values in the join column

Even if the MERGE statement and its tables meet the required criteria for optimization, MERGE could fail with a run-time error if there are duplicate values in the join column.

When HP Vertica prepares an optimized query plan for a merge operation, it enforces strict requirements for unique and primary key constraints in the MERGE statement's join columns. If you haven't enforced constraints, MERGE fails under the following scenarios:

- **Duplicates in the source table.** If HP Vertica finds more than one matching value in the source join column for a corresponding value in the target table, MERGE fails with a run-time error.
- **Duplicates in the target table.** If HP Vertica finds more than one matching value in target join column for a corresponding value in the source table, *and* the target join column has a unique or primary key constraint, MERGE fails with a run-time error. If the target join column has no such constraint, the statement runs without error and without optimization.

Be aware that if you run MERGE multiple times using the same target and source table, each statement run has the potential to introduce duplicate values into the join columns, such as if you use constants in the UPDATE/INSERT clauses. These duplicates could cause a run-time error the next time you run MERGE.

To avoid duplicate key errors, be sure to enforce constraints you declare to assure unique values in the merge join column.

Examples

The examples that follow use a simple schema to illustrate some of the conditions under which HP Vertica prepares or does not prepare an optimized query plan for MERGE:

```
CREATE TABLE target(a INT PRIMARY KEY, b INT, c INT) ORDER BY b,a;
CREATE TABLE source(a INT, b INT, c INT) ORDER BY b,a;
INSERT INTO target VALUES(1,2,3);
INSERT INTO target VALUES(2,4,7);
INSERT INTO source VALUES(3,4,5);
INSERT INTO source VALUES(4,6,9);
COMMIT;
```

Example of an optimized MERGE statement

HP Vertica can prepare an optimized query plan for the following MERGE statement because:

- The target table's join column (ON t.a=s.a) has a primary key constraint
- All columns in the target table (a,b,c) are included in the UPDATE and INSERT clauses
- Columns attributes specified in the UPDATE and INSERT clauses are identical

```
MERGE INTO target t USING source s ON t.a = s.a
  WHEN MATCHED THEN UPDATE SET a=s.a, b=s.b, c=s.c
  WHEN NOT MATCHED THEN INSERT(a,b,c) VALUES(s.a,s.b,s.c);
```

```
OUTPUT
-----
2
(1 row)
```

The output value of 2 indicates success and denotes the number of rows updated/inserted from the source into the target.

Example of a non-optimized MERGE statement

In the next example, the MERGE statement runs without optimization because the column attributes in the UPDATE/INSERT clauses are not identical. Specifically, the UPDATE clause includes constants for columns `s.a` and `s.c` and the INSERT clause does not:

```
MERGE INTO target t USING source s ON t.a = s.a
  WHEN MATCHED THEN UPDATE SET a=s.a + 1, b=s.b, c=s.c - 1
  WHEN NOT MATCHED THEN INSERT(a,b,c) VALUES(s.a,s.b,s.c);
```

To make the previous MERGE statement eligible for optimization, rewrite the statement as follows, so the attributes in the UPDATE and INSERT clauses are identical:

```
MERGE INTO target t USING source s ON t.a = s.a
  WHEN MATCHED THEN UPDATE SET a=s.a + 1, b=s.b, c=s.c - 1
  WHEN NOT MATCHED THEN INSERT(a,b,c)
    VALUES(s.a + 1, s.b, s.c - 1);
```

Troubleshooting the MERGE Statement

Here are a few things to consider so that HP Vertica will prepare an optimized query plan for MERGE, or to check if you encounter run-time errors after you run the MERGE statement.

MERGE performance considerations

You can help improve the performance of MERGE operations by ensuring projections are designed for optimal use. See [Projection Design for Merge Operations](#).

You can also improve the chances that HP Vertica prepares an optimized query plan for a MERGE statement by making sure the statement and its tables meet certain requirements. See the following topics for more information:

- [Optimized Versus Non-optimized MERGE](#)
- [Viewing the MERGE Plan](#)

Duplicate values in the merge join key

HP Vertica assumes that the data you want to merge conforms with constraints you declare. To avoid duplicate key errors, be sure to enforce declared constraints to assure unique values in the merge join column. If the MERGE statement fails with a duplicate key error, you must correct your data.

Also, be aware that if you run MERGE multiple times with the same target and source tables, you could introduce duplicate values into the join columns, such as if you use constants in the UPDATE/INSERT clauses. These duplicates could cause a run-time error.

Using MERGE with sequences

If you are using named [sequences](#), HP Vertica can perform a MERGE operation if you omit the sequence from the query.

You cannot run MERGE on identity/auto-increment columns or on columns that have primary key or foreign key referential integrity constraints, as defined in CREATE TABLE [column-constraint](#) syntax.

Dropping and Truncating Tables

HP Vertica provides two statements to manage tables: DROP TABLE and TRUNCATE TABLE. You cannot truncate an external table.

Dropping Tables

Dropping a table removes its definition from the HP Vertica database. For the syntax details of this statement, see [DROP TABLE](#) in the SQL Reference Manual.

To drop a table, use the statement as follows:

```
=> DROP TABLE IF EXISTS mytable;DROP TABLE
=> DROP TABLE IF EXISTS mytable; -- Doesn't exist
NOTICE: Nothing was dropped
DROP TABLE
```

You cannot use the CASCADE option to drop an external table, since the table is read-only, you cannot remove any of its associated files.

Truncating Tables

Truncating a table removes all storage associated with the table, but preserves the table definitions. Use TRUNCATE TABLE for testing purposes to remove all table data without having to recreate projections when you reload table data. For the syntax details of this statement, see [TRUNCATE TABLE](#) in the SQL Reference Manual. You cannot truncate an external table.

The TRUNCATE TABLE statement commits the entire transaction after statement execution, even if truncating the table fails. You cannot roll back a TRUNCATE statement.

If the truncated table is a large single (fact) table containing pre-join projections, the projections show zero (0) rows after the transaction completes and the table is ready for data reload.

If the table to truncate is a dimension table, drop the pre-join projections before executing the TRUNCATE TABLE statement. Otherwise, the statement returns the following error:

```
Cannot truncate a dimension table with pre-joined projections
```

If the truncated table has out-of-date projections, those projections are cleared and marked up-to-date after the TRUNCATE TABLE operation completes.

TRUNCATE TABLE takes an O (Owner) lock on the table until the truncation process completes, and the **savepoint** is then released.

About Constraints

Constraints specify rules on data that can go into a column. Some examples of constraints are:

- Primary or foreign key
- Uniqueness
- Not NULL
- Default values
- Automatically incremented values
- Values that are generated by the database

Use constraints when you want to ensure the integrity of your data in one or more columns, but be aware that it is your responsibility to ensure data integrity. HP Vertica can use constraints to perform optimizations (such as the optimized MERGE) that assume the data is consistent. Do not define constraints on columns unless you expect to keep the data consistent.

Adding Constraints

Add constraints on one or more table columns using the following SQL commands:

- [CREATE TABLE](#)—Add a constraint on one or more columns.
- [ALTER TABLE](#)—Add or drop a constraint on one or more columns.

There are two syntax definitions you can use to add or change a constraint:

- [column-constraint](#)—Use this syntax when you add a constraint on a column definition in a CREATE TABLE statement.
- [table-constraint](#)—Use this syntax when you add a constraint after a column definition in a CREATE TABLE statement, or when you add, alter, or drop a constraint on a column using ALTER TABLE.

HP Vertica recommends naming a constraint but it is optional; if you specify the CONSTRAINT keyword, you must give a name for the constraint.

The examples that follow illustrate several ways of adding constraints. For additional details, see:

- [Primary Key Constraints](#)
- [Foreign Key Constraints](#)
- [Unique Constraints](#)
- [Not NULL Constraints](#)

Adding Column Constraints with CREATE TABLE

There are several ways to add a constraint on a column using CREATE TABLE:

- On the column definition using the CONSTRAINT keyword, which requires that you assign a constraint name, in this example, dim1PK:

```
CREATE TABLE dim1 ( c1 INTEGER CONSTRAINT dim1PK PRIMARY KEY,  
                    c2 INTEGER  
                    );
```

- On the column definition, omitting the CONSTRAINT keyword. When you omit the CONSTRAINT keyword, you cannot specify a constraint name:

```
CREATE TABLE dim1 ( c1 INTEGER PRIMARY KEY,  
                    c2 INTEGER
```



```
);
```

- After the column definition, using the CONSTRAINT keyword and assigning a name, in this example, dim1PK:

```
CREATE TABLE dim1 ( c1 INTEGER,  
                    c2 INTEGER,  
                    CONSTRAINT dim1pk PRIMARY KEY(c1)  
);
```

- After the column definition, omitting the CONSTRAINT keyword:

```
CREATE TABLE dim1 ( c1 INTEGER,  
                    c2 INTEGER,  
                    PRIMARY KEY(c1)  
);
```

Adding Two Constraints on a Column

To add more than one constraint on a column, specify the constraints one after another when you create the table column. For example, the following statement enforces both not NULL and unique constraints on the `customer_key` column, indicating that the column values cannot be NULL and must be unique:

```
CREATE TABLE test1 ( id INTEGER NOT NULL UNIQUE,  
                    ...  
);
```

Adding a Foreign Key Constraint on a Column

There are four ways to add a foreign key constraint on a column using CREATE TABLE. The FOREIGN KEY keywords are not valid *on* the column definition, only *after* the column definition:

- On the column definition, use the CONSTRAINT and REFERENCES keywords and name the constraint, in this example, fact1dim1PK. This example creates a column with a named foreign key constraint referencing the table (`dim1`) with the primary key (`c1`):

```
CREATE TABLE fact1 ( c1 INTEGER CONSTRAINT fact1dim1FK REFERENCES dim1(c1),  
                    c2 INTEGER  
);
```

- On the column definition, omit the CONSTRAINT keyword and use the REFERENCES keyword with the table name and column:

```
CREATE TABLE fact1 ( c1 INTEGER REFERENCES dim1(c1),
c2 INTEGER
);
```

- After the column definition, use the CONSTRAINT, FOREIGN KEY, and REFERENCES keywords and name the constraint:

```
CREATE TABLE fact1 ( c1 INTEGER,
c2 INTEGER,
CONSTRAINT fk1 FOREIGN KEY(c1) REFERENCES dim1(c1)
);
```

- After the column definition, omitting the CONSTRAINT keyword:

```
CREATE TABLE fact1 ( c1 INTEGER,
c2 INTEGER,
FOREIGN KEY(c1) REFERENCES dim1(c1)
);
```

Each of the following ALTER TABLE statements adds a foreign key constraint on an existing column, with and without using the CONSTRAINT keyword:

```
ALTER TABLE fact2
ADD CONSTRAINT fk1 FOREIGN KEY (c1) REFERENCES dim2(c1);
```

or

```
ALTER TABLE fact2 ADD FOREIGN KEY (c1) REFERENCES dim2(c1);
```

For additional details, see [Foreign Key Constraints](#).

Adding Multicolumn Constraints

The following example defines a primary key constraint on multiple columns by first defining the table columns (c1 and c2), and then specifying both columns in a PRIMARY KEY clause:

```
CREATE TABLE dim ( c1 INTEGER,
c2 INTEGER,
PRIMARY KEY (c1, c2)
);
```

To specify multicolumn (compound) primary keys, the following example uses CREATE TABLE to define the columns. After creating the table, ALTER TABLE defines the compound primary key and names it dim2PK:

```
CREATE TABLE dim2 ( c1 INTEGER,
```

```
c2 INTEGER,  
c3 INTEGER NOT NULL,  
c4 INTEGER UNIQUE  
);  
ALTER TABLE dim2  
  ADD CONSTRAINT dim2PK PRIMARY KEY (c1, c2);
```

In the next example, you define a compound primary key as part of the CREATE TABLE statement. Then you specify the matching foreign key constraint to table dim2 using CREATE TABLE and ALTER TABLE:

```
CREATE TABLE dim2 ( c1 INTEGER,  
  c2 INTEGER,  
  c3 INTEGER NOT NULL,  
  c4 INTEGER UNIQUE,  
  PRIMARY KEY (c1, c2)  
);  
CREATE TABLE fact2 (  
  c1 INTEGER,  
  c2 INTEGER,  
  c3 INTEGER NOT NULL,  
  c4 INTEGER UNIQUE  
);  
ALTER TABLE fact2  
  ADD CONSTRAINT fact2FK FOREIGN KEY (c1, c2) REFERENCES dim2(c1, c2);
```

Specify a foreign key constraint using a reference to the table that contains the primary key. In the ADD CONSTRAINT clause, the REFERENCES column names are optional. The following ALTER TABLE statement is equivalent to the previous ALTER TABLE statement:

```
ALTER TABLE fact2 ADD CONSTRAINT fact2FK FOREIGN KEY (c1, c2) REFERENCES dim2;
```

Adding Constraints on Tables with Existing Data

When you add a constraint on a column with existing data, HP Vertica does not check to ensure that the column does not contain invalid values. If your data does not conform to the declared constraints, your queries could yield unexpected results.

Use [ANALYZE_CONSTRAINTS](#) to check for constraint violations in your column. If you find violations, use the ALTER COLUMN SET/DROP parameters of the [ALTER TABLE](#) statement to apply or remove a constraint on an existing column.

Adding and Changing Constraints on Columns Using ALTER TABLE

The following example uses ALTER TABLE to add a column (b) with not NULL and default 5 constraints to a table (test6):

```
CREATE TABLE test6 (a INT);ALTER TABLE test6 ADD COLUMN b INT DEFAULT 5 NOT NULL;
```

Use ALTER TABLE with the ALTER COLUMN and SET NOT NULL clauses to add the constraint on column a in table test6:

```
ALTER TABLE test6 ALTER COLUMN a SET NOT NULL;
```

Adding and Dropping NOT NULL Column Constraints

Use the SET NOT NULL or DROP NOT NULL clause to add or remove a not NULL column constraint. Use these clauses to ensure that the column has the proper constraints when you have added or removed a primary key constraint on a column, or any time you want to add or remove the not NULL constraint.

Note: A PRIMARY KEY constraint includes a not NULL constraint, but if you drop the PRIMARY KEY constraint on a column, the not NULL constraint remains on that column.

Examples

```
ALTER TABLE T1 ALTER COLUMN x SET NOT NULL;  
ALTER TABLE T1 ALTER COLUMN x DROP NOT NULL;
```

For more information, see [Altering Table Definitions](#).

Enforcing Constraints

To maximize query performance, HP Vertica checks for primary key and foreign key violations when loading into the fact table of a **pre-join projection**. For more details, see [Enforcing Primary Key and Foreign Key Constraints](#).

HP Vertica checks for not NULL constraint violations when loading data, but it does not check for unique constraint violations.

To enforce constraints, load data without committing it using the **COPY** with NO COMMIT option and then perform a post-load check using the **ANALYZE_CONSTRAINTS** function. If constraint violations are found, you can roll back the load because you have not committed it. For more details, see [Detecting Constraint Violations](#).

See Also

- [ALTER TABLE](#)
- [CREATE TABLE](#)
- [COPY](#)
- [ANALYZE_CONSTRAINTS](#)

Primary Key Constraints

A primary key (PK) is a single column or combination of columns (called a *compound key*) that uniquely identifies each row in a table. A primary key constraint contains unique, non-null values.

When you apply the primary key constraint, the not NULL and unique constraints are added implicitly. You do not need to specify them when you create the column. However, if you remove the primary key constraint, the not NULL constraint continues to apply to the column. To remove the not NULL constraint after removing the primary key constraint, use the ALTER COLUMN DROP NOT NULL parameter of the [ALTER TABLE](#) statement (see [Dropping Constraints](#)).

The following statement adds a primary key constraint on the `employee_id` field:

```
CREATE TABLE employees (  employee_id INTEGER PRIMARY KEY
);
```

Alternatively, you can add a primary key constraint after the column is created:

```
CREATE TABLE employees (  employee_id INTEGER
);
ALTER TABLE employees
  ADD PRIMARY KEY (employee_id);
```

Note: If you specify a primary key constraint using ALTER TABLE, the system returns the following message, which is informational only. The primary key constraint is added to the designated column.

Note: WARNING 2623: Column "employee_id" definition changed to NOT NULL

Primary keys can also constrain more than one column:

```
CREATE TABLE employees (  employee_id INTEGER,
  employee_gender CHAR(1),
  PRIMARY KEY (employee_id, employee_gender)
);
```

Foreign Key Constraints

A foreign key (FK) is a column that is used to join a table to other tables to ensure **referential integrity** of the data. A foreign key constraint requires that a column contain only values from the primary key column on a specific dimension table.

A column with a foreign key constraint can contain NULL values if it does not also have a [not NULL](#) constraint, even though the NULL value does not appear in the PRIMARY KEY column of the dimension table. This allows rows to be inserted into the table even if the foreign key is not yet known.

In HP Vertica, the fact table's join columns are required to have foreign key constraints in order to participate in **pre-join projections**. If the fact table join column has a foreign key constraint, outer join queries produce the same result set as inner join queries.

You can add a FOREIGN KEY constraint solely by referencing the table that contains the primary key. The columns in the referenced table do not need to be specified explicitly.

Examples

Create a table called inventory to store inventory data:

```
CREATE TABLE inventory (
    date_key INTEGER NOT NULL,
    product_key INTEGER NOT NULL,
    warehouse_key INTEGER NOT NULL,
    ...
);
```

Create a table called warehouse to store warehouse information:

```
CREATE TABLE warehouse (
    warehouse_key INTEGER NOT NULL PRIMARY KEY,
    warehouse_name VARCHAR(20),
    ...
);
```

To ensure referential integrity between the inventory and warehouse tables, define a foreign key constraint called `fk_inventory_warehouse` on the `inventory` table that references the `warehouse` table:

```
ALTER TABLE inventory ADD CONSTRAINT fk_inventory_warehouse FOREIGN KEY(warehouse_key)
REFERENCES warehouse(warehouse_key);
```

In this example, the `inventory` table is the *referencing* table and the `warehouse` table is the *referenced* table.

You can also create the foreign key constraint in the CREATE TABLE statement that creates the warehouse table, eliminating the need for the ALTER TABLE statement. If you do not specify one or more columns, the PRIMARY KEY of the referenced table is used:

```
CREATE TABLE warehouse (warehouse_key INTEGER NOT NULL PRIMARY KEY REFERENCES warehouse,
    warehouse_name VARCHAR(20),
    ...);
```

A foreign key can also constrain and reference multiple columns. The following example uses CREATE TABLE to add a foreign key constraint to a pair of columns:

```
CREATE TABLE t1 ( c1 INTEGER PRIMARY KEY,
    c2 INTEGER,
    c3 INTEGER,
    FOREIGN KEY (c2, c3) REFERENCES other_table (c1, c2)
);
```

The following two examples use ALTER TABLE to add a foreign key constraint to a pair of columns. When you use the CONSTRAINT keyword, you must specify a constraint name:

```
ALTER TABLE t
  ADD FOREIGN KEY (a, b) REFERENCES other_table(c, d);
ALTER TABLE t
  ADD CONSTRAINT fk_cname FOREIGN KEY (a, b) REFERENCES other_table(c, d);
```

Note: The FOREIGN KEY keywords are valid only *after* the column definition, not *on* the column definition.

Unique Constraints

Unique constraints ensure that the data contained in a column or a group of columns is unique with respect to all rows in the table.

Note: If you add a unique constraint to a column and then insert data into that column that is not unique with respect to other values in that column, HP Vertica inserts the data anyway. If your data does not conform to the declared constraints, your queries could yield unexpected results. Use [ANALYZE_CONSTRAINTS](#) to check for constraint violations.

There are several ways to add a unique constraint on a column. If you use the CONSTRAINT keyword, you must specify a constraint name. The following example adds a UNIQUE constraint on the product_key column and names it product_key_UK:

```
CREATE TABLE product (  product_key INTEGER NOT NULL CONSTRAINT product_key_UK UNIQUE,
  ...
);
```

HP Vertica recommends naming constraints, but it is optional:

```
CREATE TABLE product (  product_key INTEGER NOT NULL UNIQUE,
  ...
);
```

You can specify the constraint after the column definition, with and without naming it:

```
CREATE TABLE product (  product_key INTEGER NOT NULL,
  ...
  CONSTRAINT product_key_uk UNIQUE (product_key)
);
CREATE TABLE product (
  product_key INTEGER NOT NULL,
  ...
  UNIQUE (product_key)
);
```

You can also use ALTER TABLE to specify a unique constraint. This example names the constraint product_key_UK:

```
ALTER TABLE product ADD CONSTRAINT product_key_UK UNIQUE (product_key);
```

You can use CREATE TABLE and ALTER TABLE to specify unique constraints on multiple columns. If a unique constraint refers to a group of columns, separate the column names using commas. The column listing specifies that the combination of values in the indicated columns is unique across the whole table, though any one of the columns need not be (and ordinarily isn't) unique:

```
CREATE TABLE dim1 ( c1 INTEGER,  
                    c2 INTEGER,  
                    c3 INTEGER,  
                    UNIQUE (c1, c2)  
);
```

Not NULL Constraints

A not NULL constraint specifies that a column *cannot* contain a null value. This means that new rows cannot be inserted or updated unless you specify a value for this column.

You can apply the not NULL constraint when you create a column using the CREATE TABLE statement. You can also add or drop the not NULL constraint to an existing column using, respectively:

- ALTER TABLE t ALTER COLUMN x SET NOT NULL
- ALTER TABLE t ALTER COLUMN x DROP NOT NULL

The not NULL constraint is implicitly applied to a column when you add the **PRIMARY KEY** (PK) constraint. When you designate a column as a primary key, you do not need to specify the not NULL constraint.

However, if you remove the primary key constraint, the not NULL constraint still applies to the column. Use the ALTER COLUMN x DROP NOT NULL parameter of the [ALTER TABLE](#) statement to drop the not NULL constraint after dropping the primary key constraint.

The following statement enforces a not NULL constraint on the customer_key column, specifying that the column cannot accept NULL values.

```
CREATE TABLE customer ( customer_key INTEGER NOT NULL,  
                        ...  
);
```


Dropping Constraints

To drop named constraints, use the [ALTER TABLE](#) command.

The following example drops the constraint factfk2:

```
=> ALTER TABLE fact2 DROP CONSTRAINT fact2fk;
```

To drop constraints that you did not assign a name to, query the system table [TABLE_CONSTRAINTS](#), which returns both system-generated and user-named constraint names:

```
=> SELECT * FROM TABLE_CONSTRAINTS;
```

If you do not specify a constraint name, HP Vertica assigns a constraint name that is unique to that table. In the following output, note the system-generated constraint name `C_PRIMARY` and the user-defined constraint name `fk_inventory_date`:

```
-[ RECORD 1 ]-----+-----constraint_id | 45035996273707984
constraint_name | C_PRIMARY
constraint_schema_id | 45035996273704966
constraint_key_count | 1
foreign_key_count | 0
table_id | 45035996273707982
foreign_table_id | 0
constraint_type | p
-[ ... ]-----+-----
-[ RECORD 9 ]-----+-----
constraint_id | 45035996273708016
constraint_name | fk_inventory_date
constraint_schema_id | 0
constraint_key_count | 1
foreign_key_count | 1
table_id | 45035996273708014
foreign_table_id | 45035996273707994
constraint_type | f
```

Once you know the name of the constraint, you can then drop it using the [ALTER TABLE](#) command. (If you do not know the table name, use `table_id` to retrieve `table_name` from the [ALL_TABLES](#) table.)

Notes

- Primary key constraints cannot be dropped if there is another table with a foreign key constraint that references the primary key.
- A foreign key constraint cannot be dropped if there are any pre-join projections on the table.
- Dropping a primary or foreign key constraint does not automatically drop the not NULL constraint on a column. You need to manually drop this constraint if you no longer want it.

See Also

- [ALTER TABLE](#)

Enforcing Primary Key and Foreign Key Constraints

Enforcing Primary Key Constraints

HP Vertica does not enforce the uniqueness of primary keys when they are loaded into a table. However, when data is loaded into a table with a pre-joined dimension, or when the table is joined to a dimension table during a query, a key enforcement error could result if there is not exactly one dimension row that matches each foreign key value.

Note: : Consider using sequences or auto-incrementing columns for primary key columns, which guarantees uniqueness and avoids the constraint enforcement problem and associated overhead. For more information, see [Using Sequences](#).

Enforcing Foreign Key Constraints

A table's foreign key constraints are enforced during data load only if there is a pre-join projection that has that table as its anchor table. If there no such pre-join projection exists, then it is possible to load data that causes a constraint violation. Subsequently, a constraint violation error can happen when:

- An inner join query is processed.
- An outer join is treated as an inner join due to the presence of foreign key.
- A new pre-join projection anchored on the table with the foreign key constraint is **refreshed**.

Detecting Constraint Violations Before You Commit Data

To detect constraint violations, you can load data without committing it using the [COPY](#) statement with the NO COMMIT option, and then perform a post-load check using the [ANALYZE_CONSTRAINTS](#) function. If constraint violations exist, you can roll back the load because you have not committed it. For more details, see [Detecting Constraint Violations](#).

Detecting Constraint Violations

The `ANALYZE_CONSTRAINTS()` function analyzes and reports on constraint violations within the current schema search path. To check for constraint violations:

- Pass an empty argument to check for violations on all tables within the current schema
- Pass a single table argument to check for violations on the specified table
- Pass two arguments, a table name and a column or list of columns, to check for violations in those columns

Given the following inputs, HP Vertica returns one row, indicating one violation, because the same primary key value (10) was inserted into table t1 twice:

```
CREATE TABLE t1(c1 INT);
ALTER TABLE t1 ADD CONSTRAINT pk_t1 PRIMARY KEY (c1);
CREATE PROJECTION t1_p (c1) AS SELECT *
  FROM t1 UNSEGMENTED ALL NODES;
INSERT INTO t1 values (10);
INSERT INTO t1 values (10); --Duplicate primary key value

\x
Expanded display is on.

SELECT ANALYZE_CONSTRAINTS('t1');
-[ RECORD 1 ]-----
Schema Name      | public
Table Name       | t1
Column Names     | c1
Constraint Name  | pk_t1
Constraint Type  | PRIMARY
Column Values    | ('10')
```

If the second `INSERT` statement above had contained any different value, the result would have been 0 rows (no violations).

In the following example, create a table that contains three integer columns, one a unique key and one a primary key:

```
CREATE TABLE table_1(
  a INTEGER,
  b_UK INTEGER UNIQUE,
  c_PK INTEGER PRIMARY KEY
);
```

Issue a command that refers to a nonexistent table and column:

```
SELECT ANALYZE_CONSTRAINTS('a_BB');
ERROR: 'a_BB' is not a table name in the current search path
```

Issue a command that refers to a nonexistent column:

```
SELECT ANALYZE_CONSTRAINTS('table_1','x');  
ERROR 41614: Nonexistent columns: 'x '
```

Insert some values into table `table_1` and commit the changes:

```
INSERT INTO table_1 values (1, 1, 1);  
COMMIT;
```

Run `ANALYZE_CONSTRAINTS` on table `table_1`. No constraint violations are reported:

```
SELECT ANALYZE_CONSTRAINTS('table_1');  
(No rows)
```

Insert duplicate unique and primary key values and run `ANALYZE_CONSTRAINTS` on table `table_1` again. The system shows two violations: one against the primary key and one against the unique key:

```
INSERT INTO table_1 VALUES (1, 1, 1);  
COMMIT;  
SELECT ANALYZE_CONSTRAINTS('table_1');
```

```
-[ RECORD 1 ]----+-----  
Schema Name      | public  
Table Name       | table_1  
Column Names     | b_UK  
Constraint Name  | C_UNIQUE  
Constraint Type  | UNIQUE  
Column Values    | ('1')  
-[ RECORD 2 ]----+-----  
Schema Name      | public  
Table Name       | table_1  
Column Names     | c_PK  
Constraint Name  | C_PRIMARY  
Constraint Type  | PRIMARY  
Column Values    | ('1')
```

The following command looks for constraint validations on only the unique key in the table `table_1`, qualified with its schema name:

```
=> SELECT ANALYZE_CONSTRAINTS('public.table_1', 'b_UK');
```

```
-[ RECORD 1 ]----+-----  
Schema Name      | public  
Table Name       | table_1  
Column Names     | b_UK  
Constraint Name  | C_UNIQUE  
Constraint Type  | UNIQUE  
Column Values    | ('1')
```

```
(1 row)
```

The following example shows that you can specify the same column more than once; `ANALYZE_CONSTRAINTS`, however, returns the violation only once:

```
SELECT ANALYZE_CONSTRAINTS('table_1', 'c_PK, C_PK');
-[ RECORD 1 ]----+-----
Schema Name      | public
Table Name       | table_1
Column Names     | c_PK
Constraint Name  | C_PRIMARY
Constraint Type  | PRIMARY
Column Values    | ('1')
```

The following example creates a new table, `table_2`, and inserts a foreign key and different (character) data types:

```
CREATE TABLE table_2 (
  x VARCHAR(3),
  y_PK VARCHAR(4),
  z_FK INTEGER REFERENCES table_1(c_PK));
```

Alter the table to create a multicolumn unique key and multicolumn foreign key and create superprojections:

```
ALTER TABLE table_2
  ADD CONSTRAINT table_2_multiuk PRIMARY KEY (x, y_PK);
WARNING 2623: Column "x" definition changed to NOT NULL
WARNING 2623: Column "y_PK" definition changed to NOT NULL
```

The following command inserts a missing foreign key (0) into table `dim_1` and commits the changes:

```
INSERT INTO table_2 VALUES ('r1', 'Xpk1', 0);
COMMIT;
```

Checking for constraints on the table `table_2` in the public schema detects a foreign key violation:

```
=> SELECT ANALYZE_CONSTRAINTS('public.table_2');
-[ RECORD 1 ]----+-----
Schema Name      | public
Table Name       | table_2
Column Names     | z_FK
Constraint Name  | C_FOREIGN
Constraint Type  | FOREIGN
Column Values    | ('0')
```

Now add a duplicate value into the unique key and commit the changes:

```
INSERT INTO table_2 VALUES ('r2', 'Xpk1', 1);
INSERT INTO table_2 VALUES ('r1', 'Xpk1', 1);
```

```
COMMIT;
```

Checking for constraint violations on table `table_2` detects the duplicate unique key error:

```
SELECT ANALYZE_CONSTRAINTS('table_2');
-[ RECORD 1 ]-----+-----
Schema Name      | public
Table Name       | table_2
Column Names     | z_FK
Constraint Name  | C_FOREIGN
Constraint Type  | FOREIGN
Column Values    | ('0')
-[ RECORD 2 ]-----+-----
Schema Name      | public
Table Name       | table_2
Column Names     | x, y_PK
Constraint Name  | table_2_multiuk
Constraint Type  | PRIMARY
Column Values    | ('r1', 'Xpk1')
```

Create a table with multicolumn foreign key and create the superprojections:

```
CREATE TABLE table_3(
  z_fk1 VARCHAR(3),
  z_fk2 VARCHAR(4));
ALTER TABLE table_3
  ADD CONSTRAINT table_3_multifk FOREIGN KEY (z_fk1, z_fk2)
  REFERENCES table_2(x, y_PK);
```

Insert a foreign key that matches a foreign key in table `table_2` and commit the changes:

```
INSERT INTO table_3 VALUES ('r1', 'Xpk1');
COMMIT;
```

Checking for constraints on table `table_3` detects no violations:

```
SELECT ANALYZE_CONSTRAINTS('table_3');
(No rows)
```

Add a value that does not match and commit the change:

```
INSERT INTO table_3 VALUES ('r1', 'NONE');
COMMIT;
```

Checking for constraints on table `dim_2` detects a foreign key violation:

```
SELECT ANALYZE_CONSTRAINTS('table_3');
-[ RECORD 1 ]-----+-----
Schema Name      | public
Table Name       | table_3
```

```
Column Names      | z_fk1, z_fk2
Constraint Name   | table_3_multifk
Constraint Type   | FOREIGN
Column Values     | ('r1', 'NONE')
```

Analyze all constraints on all tables:

```
SELECT ANALYZE_CONSTRAINTS('');
-[ RECORD 1 ]-----+-----
Schema Name      | public
Table Name       | table_3
Column Names     | z_fk1, z_fk2
Constraint Name   | table_3_multifk
Constraint Type   | FOREIGN
Column Values    | ('r1', 'NONE')
-[ RECORD 2 ]-----+-----
Schema Name      | public
Table Name       | table_2
Column Names     | x, y_PK
Constraint Name   | table_2_multiuk
Constraint Type   | PRIMARY
Column Values    | ('r1', 'Xpk1')
-[ RECORD 3 ]-----+-----
Schema Name      | public
Table Name       | table_2

Column Names     | z_FK
Constraint Name   | C_FOREIGN
Constraint Type   | FOREIGN
Column Values    | ('0')
-[ RECORD 4 ]-----+-----
Schema Name      | public
Table Name       | t1
Column Names     | c1
Constraint Name   | pk_t1
Constraint Type   | PRIMARY
Column Values    | ('10')
-[ RECORD 5 ]-----+-----
Schema Name      | public
Table Name       | table_1
Column Names     | b_UK
Constraint Name   | C_UNIQUE
Constraint Type   | UNIQUE
Column Values    | ('1')
-[ RECORD 6 ]-----+-----
Schema Name      | public
Table Name       | table_1
Column Names     | c_PK
Constraint Name   | C_PRIMARY
Constraint Type   | PRIMARY
Column Values    | ('1')
-[ RECORD 7 ]-----+-----
Schema Name      | public
Table Name       | target
Column Names     | a
Constraint Name   | C_PRIMARY
```



```
Constraint Type | PRIMARY  
Column Values  | ('1')  
(5 rows)
```

To quickly clean up your database, issue the following command:

```
DROP TABLE table_1 CASCADE;  
DROP TABLE table_2 CASCADE;  
DROP TABLE table_3 CASCADE;
```

Fixing Constraint Violations

When HP Vertica finds duplicate primary key or unique values at run time, use the [DISABLE_DUPLICATE_KEY_ERROR](#) function to suppress error messaging. Queries execute as though no constraints are defined on the schema and the effects are session scoped.

Caution: When called, `DISABLE_DUPLICATE_KEY_ERROR` suppresses data integrity checking and can lead to incorrect query results. Use this function only after you insert duplicate primary keys into a dimension table in the presence of a pre-join projection. Correct the violations and reenables integrity checking with [REENABLE_DUPLICATE_KEY_ERROR](#).

The following series of commands create a table named `dim` and the corresponding projection:

```
CREATE TABLE dim (pk INTEGER PRIMARY KEY, x INTEGER);  
CREATE PROJECTION dim_p (pk, x) AS SELECT * FROM dim ORDER BY x UNSEGMENTED ALL NODES;
```

The next two statements create a table named `fact` and the pre-join projection that joins `fact` to `dim`.

```
CREATE TABLE fact(fk INTEGER REFERENCES dim(pk));  
CREATE PROJECTION prejoin_p (fk, pk, x) AS SELECT * FROM fact, dim WHERE pk=fk ORDER BY x  
;
```

The following statements load values into table `dim`. The last statement inserts a duplicate primary key value of 1:

```
INSERT INTO dim values (1,1);INSERT INTO dim values (2,2);  
INSERT INTO dim values (1,2); --Constraint violation  
COMMIT;
```

Table `dim` now contains duplicate primary key values, but you cannot delete the violating row because of the presence of the pre-join projection. Any attempt to delete the record results in the following error message:

```
ROLLBACK: Duplicate primary key detected in FK-PK join Hash-Join (x dim_p), value 1
```

In order to remove the constraint violation ($pk=1$), use the following sequence of commands, which puts the database back into the state just before the duplicate primary key was added.

To remove the violation:

1. Save the original `dim` rows that match the duplicated primary key:

```
CREATE TEMP TABLE dim_temp(pk integer, x integer);  
INSERT INTO dim_temp SELECT * FROM dim WHERE pk=1 AND x=1; -- original dim row
```

2. Temporarily disable error messaging on duplicate constraint values:

```
SELECT DISABLE_DUPLICATE_KEY_ERROR();
```

Caution: Remember that running the `DISABLE_DUPLICATE_KEY_ERROR` function suppresses the enforcement of data integrity checking.

3. Remove the original row that contains duplicate values:

```
DELETE FROM dim WHERE pk=1;
```

4. Allow the database to resume data integrity checking:

```
SELECT REENABLE_DUPLICATE_KEY_ERROR();
```

5. Reinsert the original values back into the dimension table:

```
INSERT INTO dim SELECT * from dim_temp;COMMIT;
```

6. Validate your dimension and fact tables.

If you receive the following error message, it means that the duplicate records you want to delete are not identical. That is, the records contain values that differ in at least one column that is not a primary key; for example, (1,1) and (1,2).

```
ROLLBACK: Delete: could not find a data row to delete (data integrity violation?)
```

The difference between this message and the rollback message in the previous example is that a fact row contains a foreign key that matches the duplicated primary key, which has been inserted. A row with values from the fact and dimension table is now in the pre-join projection. In order for the `DELETE` statement (Step 3 in the following example) to complete successfully, extra predicates are required to identify the original dimension table values (the values that are in the pre-join).

This example is nearly identical to the previous example, except that an additional INSERT statement joins the fact table to the dimension table by a primary key value of 1:

```
INSERT INTO dim values (1,1);INSERT INTO dim values (2,2);
INSERT INTO fact values (1); -- New insert statement joins fact with dim on primary
key value=1
INSERT INTO dim values (1,2); -- Duplicate primary key value=1
COMMIT;
```

To remove the violation:

1. Save the original dim and fact rows that match the duplicated primary key:

```
CREATE TEMP TABLE dim_temp(pk integer, x integer);CREATE TEMP TABLE fact_temp(fk inte
ger);
INSERT INTO dim_temp SELECT * FROM dim WHERE pk=1 AND x=1; -- original dim row
INSERT INTO fact_temp SELECT * FROM fact WHERE fk=1;
```

2. Temporarily suppresses the enforcement of data integrity checking:

```
SELECT DISABLE_DUPLICATE_KEY_ERROR();
```

3. Remove the duplicate primary keys. These steps also implicitly remove all fact rows with the matching foreign key.
4. Remove the original row that contains duplicate values:

```
DELETE FROM dim WHERE pk=1 AND x=1;
```

Note: The extra predicate ($x=1$) specifies removal of the original (1,1) row, rather than the newly inserted (1,2) values that caused the violation.

5. Remove all remaining rows:

```
DELETE FROM dim WHERE pk=1;
```

6. Reenable integrity checking:

```
SELECT REENABLE_DUPLICATE_KEY_ERROR();
```

7. Reinsert the original values back into the fact and dimension table:

```
INSERT INTO dim SELECT * from dim_temp;  
INSERT INTO fact SELECT * from fact_temp;  
COMMIT;
```

8. Validate your dimension and fact tables.

Reenabling Error Reporting

If you ran [DISABLE_DUPLICATE_KEY_ERROR](#) to suppress error reporting while fixing duplicate key violations, you can get incorrect query results going forward. As soon as you fix the violations, run the [REENABLE_DUPLICATE_KEY_ERROR](#) function to restore the default behavior of error reporting.

The effects of this function are session scoped.

Working with Table Partitions

HP Vertica supports data partitioning at the table level, which divides one large table into smaller pieces. Partitions are a table property that apply to all **projections** for a given table.

A common use for partitions is to split data by time. For instance, if a table contains decades of data, you can partition it by year, or by month, if the table has a year of data.

Partitions can improve parallelism during query execution and enable some other optimizations. Partitions segregate data on each **node** to facilitate dropping partitions. You can drop older data partitions to make room for newer data.

Tip: When a storage container has data for a single partition, you can discard that storage location ([DROP_LOCATION](#)) after dropping the partition using the `DROP_PARTITION()` function.

Differences Between Partitioning and Segmentation

There is a distinction between partitioning at the table level and segmenting a projection (**hash** or **range**):

- **Partitioning**—defined by the table for fast data purges and query performance. Table partitioning segregates data on each node. You can drop partitions.
- **Segmentation**—defined by the projection for distributed computing. Segmenting distributes projection data across multiple nodes in a cluster. Different projections for the same table have identical partitioning, but can have different segmentation clauses. See [Projection Segmentation](#) in the Concepts Guide.

Both methods of storing and organizing data provide opportunities for parallelism during query processing. See also [Partitioning and Segmenting Data](#).

Partition Operations

The basic operations for working with partitions are as follow:

- [Defining Partitions](#)
- [Bulk Loading Data](#), and engaging in other normal operations
- Forcing data partitioning, if needed
- [Moving partitions](#) to another table as part of archiving historical data
- [Dropping Partitions](#) to drop existing partitions

- Displaying partition metadata with the [PARTITIONS](#) system table, to display one row per partition key, per ROS container.

HP Vertica provides the following functions that let you manage your partitions and obtain additional information about them. See the [Partition Management Functions](#) in the SQL Reference Manual.

See Also

- [Partitioning, repartitioning, and Reorganizing Tables](#)

Defining Partitions

The first step in defining data partitions is to establish the relationship between the data and partitions. To illustrate, consider the following table called `trade`, which contains unpartitioned data for the trade date (`tdate`), ticker symbol (`tsymbol`), and time (`ttime`).

Table 1: Unpartitioned data

| tdate | tsymbol | ttime |
|------------|---------|----------|
| 2008-01-02 | AAA | 13:00:00 |
| 2009-02-04 | BBB | 14:30:00 |
| 2010-09-18 | AAA | 09:55:00 |
| 2009-05-06 | AAA | 11:14:30 |
| 2008-12-22 | BBB | 15:30:00 |

(5 rows)

If you want to discard data once a year, a logical choice is to partition the table by year. The partition expression `PARTITION BY EXTRACT(year FROM tdate)` creates the partitions shown in Table 2:

Table 2: Data partitioned by year

| 2008 | | | 2009 | | | 2010 | | |
|----------|---------|----------|----------|---------|----------|----------|---------|----------|
| tdate | tsymbol | ttime | tdate | tsymbol | ttime | tdate | tsymbol | ttime |
| 01/02/08 | AAA | 13:00:00 | 02/04/09 | BBB | 14:30:00 | 09/18/10 | AAA | 09:55:00 |
| 12/22/08 | BBB | 15:30:00 | 05/06/09 | AAA | 11:14:30 | | | |

Unlike some databases, which require you to explicitly define partition boundaries in the [CREATE TABLE](#) statement, HP Vertica selects a partition for each row based on the result of a partitioning expression provided in the `CREATE TABLE` statement. Partitions do not have explicit names associated with them. Internally, HP Vertica creates a partition for each distinct value in the `PARTITION BY` expression.

After you specify a partition expression, HP Vertica processes the data by applying the partition expression to each row and then assigning partitions.

The following syntax generates the partitions for this example, with the results shown in Table 3. It creates a table called `trade`, partitioned by year. For additional information, see [CREATE TABLE](#) in the SQL Reference Manual.

```
CREATE TABLE trade (
  tdate DATE NOT NULL,
  tsymbol VARCHAR(8) NOT NULL,
  ttime TIME)
PARTITION BY EXTRACT (year FROM tdate);
CREATE PROJECTION trade_p (tdate, tsymbol, ttime) AS
SELECT * FROM trade
ORDER BY tdate, tsymbol, ttime UNSEGMENTED ALL NODES;
INSERT INTO trade VALUES ('01/02/08' , 'AAA' , '13:00:00');
INSERT INTO trade VALUES ('02/04/09' , 'BBB' , '14:30:00');
INSERT INTO trade VALUES ('09/18/10' , 'AAA' , '09:55:00');
INSERT INTO trade VALUES ('05/06/09' , 'AAA' , '11:14:30');
INSERT INTO trade VALUES ('12/22/08' , 'BBB' , '15:30:00');
```

Table 3: Partitioning Expression and Results

| tdate | tsymbol | ttime | EXTRACT (year FROM tdate) | tdate | tsymbol | ttime |
|----------|---------|-------------|---------------------------|----------|---------|-------------|
| 01/02/06 | AAA | 13:00:00.00 | 2006 | 01/02/06 | AAA | 13:00:00.00 |
| 02/04/07 | BBB | 14:30:00.00 | 2007 | 12/22/06 | BBB | 15:30:00.00 |
| 09/18/08 | AAA | 09:55:00.00 | 2008 | 05/06/07 | AAA | 11:14:30.00 |
| 05/06/07 | AAA | 11:14:30.00 | 2007 | 02/04/07 | BBB | 14:30:00.00 |
| 12/22/06 | BBB | 15:30:00.00 | 2006 | 09/18/08 | AAA | 09:55:00.00 |

Partitioning By Year and Month

To partition by both year and month, you need a partition expression that pads the month out to two digits so the partition keys appear as:

```
201101 201102
201103
...
201111
201112
```

You can use the following partition expression to partition the table using the year and month:

```
PARTITION BY EXTRACT(year FROM tdate)*100 + EXTRACT(month FROM tdate)
```

Restrictions on Partitioning Expressions

- The partitioning expression can reference one or more columns from the table.
- The partitioning expression cannot evaluate to NULL for any row, so do not include columns that allow a NULL value in the `CREATE TABLE..PARTITION BY` expression.
- Any SQL functions in the partitioning expression must be **immutable**, meaning that they return the exact same value regardless of when they are invoked, and independently of session or environment settings, such as LOCALE. For example, you cannot use the TO_CHAR function

in a partition expression, because it depends on locale settings, or the `RANDOM` function, since it produces different values at each invocation.

- [HP Vertica meta-functions](#) cannot be used in partitioning expressions.
- All projections anchored on a table must include all columns referenced in the `PARTITION BY` expression; this allows the partition to be calculated.
- You cannot modify partition expressions once a partitioned table is created. If you want modified partition expressions, create a new table with a new `PARTITION BY` clause, and then `INSERT...SELECT` from the old table to the new table. Once your data is partitioned the way you want it, you can drop the old table.

Best Practices for Partitioning

- While HP Vertica supports a maximum of 1024 partitions, few, if any, organizations will need to approach that maximum. Fewer partitions are likely to meet your business needs, while also ensuring maximum performance. Many customers, for example, partition their data by month, bringing their partition count to 12. HP Vertica recommends you keep the number of partitions between 10 and 20 to achieve excellent performance.
- Do not apply partitioning to tables used as dimension tables in pre-join projections. You can apply partitioning to tables used as large single (fact) tables in pre-join projections.
- For maximum performance, do not partition projections on `LONG VARBINARY` and `LONG VARCHAR` columns.

Dropping Partitions

Use the [DROP_PARTITION](#) function to drop a partition. Normally, this is a fast operation that discards all **ROS containers** that contain data for the partition.

Occasionally, a ROS container contains rows that belong to more than one partition. For example, this can happen after a `MERGE_PARTITIONS` operation. In this case, HP Vertica performs a split operation to avoid discarding too much data. HP Vertica tries to keep data from different partitions segregated into different ROS containers, but there are a small number of exceptions. For instance, the following operations can result in a ROS container with mixed partitions:

- [MERGE_PARTITIONS](#), which merges ROS containers that have data belonging to partitions in a specified partition key range
- **Refresh** and **recovery** operations can generate ROS containers with mixed partitions under some conditions. See [Auto Partitioning](#).

The number of partitions that contain data is restricted by the number of ROS containers that can comfortably exist in the system.

In general, if a ROS container has data that belongs to $n+1$ partitions and you want to drop a specific partition, the `DROP_PARTITION` operation:

1. Forces the partition of data into two containers where
 - one container holds the data that belongs to the partition that is to be dropped
 - another container holds the remaining n partitions
2. Drops the specified partition.

DROP_PARTITION forces a **moveout** if there is data in the **WOS** (WOS is not partition aware).

DROP_PARTITION acquires an exclusive lock on the table to prevent DELETE | UPDATE | INSERT | COPY statements from affecting the table, as well as any SELECT statements issued at SERIALIZABLE isolation level.

Users must have USAGE privilege on schema that contains the table.

DROP_PARTITION operations cannot be performed on tables with projections that are not up to date (have not been **refreshed**).

DROP_PARTITION fails if you do not set the optional third parameter to *true* and it encounters ROS's that do not have partition keys.

Examples

Using the example schema in [Defining Partitions](#), the following command explicitly drops the 2009 partition key from table `trade`:

```
SELECT DROP_PARTITION('trade', 2009);
DROP_PARTITION
-----
Partition dropped
(1 row)
```

Here, the partition key is specified:

```
SELECT DROP_PARTITION('trade', EXTRACT('year' FROM '2009-01-01'::date));
DROP_PARTITION
-----
Partition dropped
(1 row)
```

The following example creates a table called `dates` and partitions the table by year:

```
CREATE TABLE dates (year INTEGER NOT NULL,
                    month VARCHAR(8) NOT NULL)
PARTITION BY year * 12 + month;
```

The following statement drops the partition using a constant for Oct 2010 ($2010*12 + 10 = 24130$):

```
SELECT DROP_PARTITION('dates', '24130');
```

```
DROP_PARTITION
-----
Partition dropped
(1 row)
```

Alternatively, the expression can be placed in line: `SELECT DROP_PARTITION('dates', 2010*12 + 10);`

The following command first reorganizes the data if it is unpartitioned and then explicitly drops the 2009 partition key from table `trade`:

```
SELECT DROP_PARTITION('trade', 2009, false, true);
DROP_PARTITION
-----
Partition dropped
(1 row)
```

See Also

- [DROP_PARTITION](#)
- [MERGE_PARTITIONS](#)

Partitioning and Segmenting Data

Partitioning and segmentation have completely separate functions in HP Vertica, and opposite goals regarding data localization. Since other databases often use the terms interchangeably, it is important to know the differences.

- **Segmentation** defines how data is spread among cluster nodes. The goal is to distribute data evenly across multiple database nodes so that all nodes can participate in query execution.
- **Partitioning** specifies how data is organized within individual nodes. Partitioning attempts to introduce hot spots within the node, providing a convenient way to drop data and reclaim the disk space.

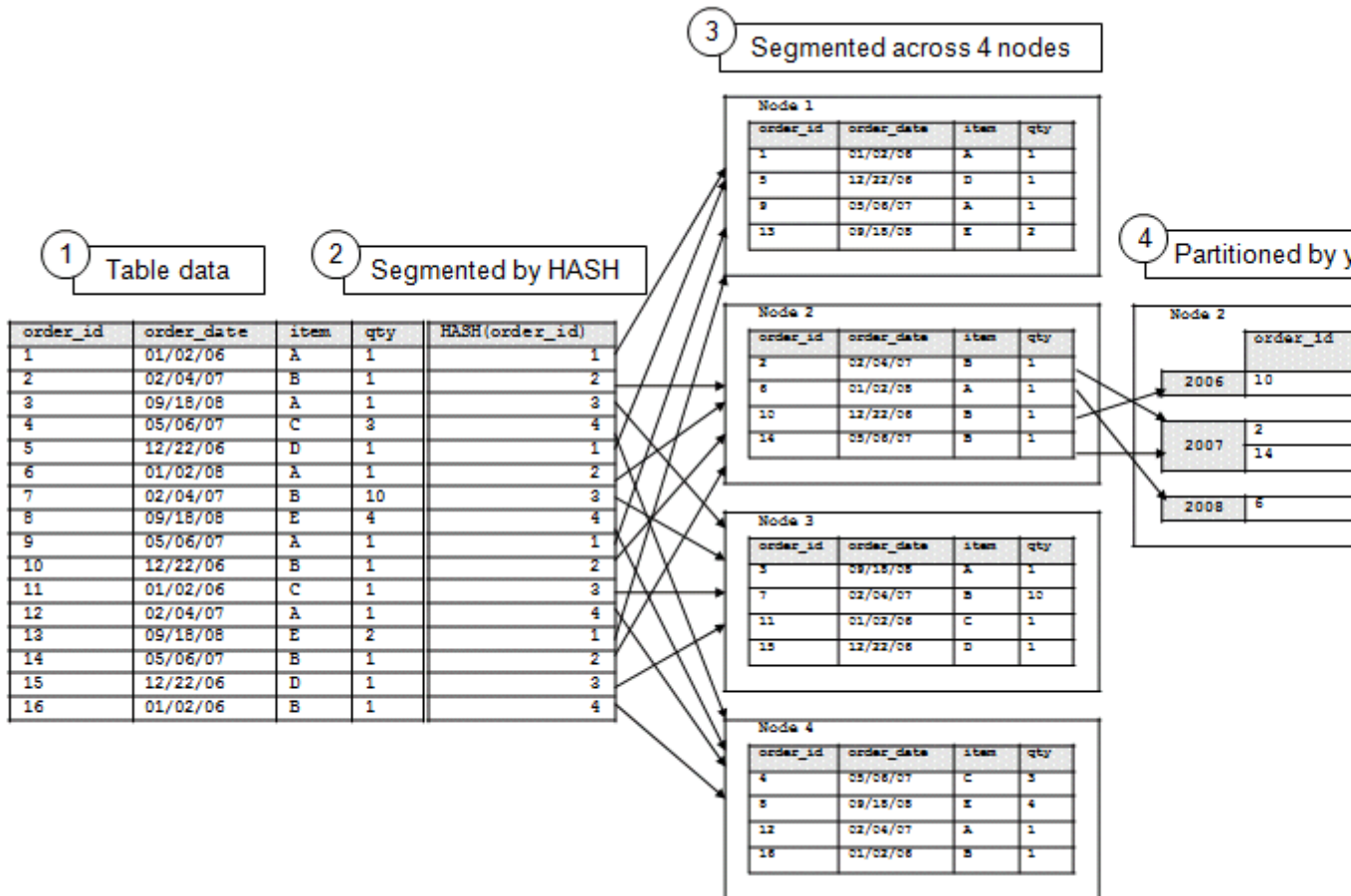
Note: Segmentation is defined by the [CREATE PROJECTION](#) statement, and partitioning is defined by the [CREATE TABLE](#) statement. Logically, the partition clause is applied after the segmentation clause. See the SQL Reference Manual for details.

To further illustrate the differences, partitioning data by year makes sense if you intend to retain and drop data at the granularity of a year. On the other hand, segmenting the data by year would be inefficient, because the node holding data for the current year would likely answer far more queries than the other nodes.

The following diagram illustrates the flow of segmentation and partitioning on a four-node database cluster:

1. Example table data
2. Data segmented by HASH(order_id)
3. Data segmented by hash across four nodes
4. Data partitioned by year on a single node

While partitioning occurs on all four nodes, the illustration shows partitioned data on one node for simplicity.



See Also

- [Reclaiming Disk Space From Deleted Records](#)
-
-
-
-

Partitioning and Data Storage

Partitions and ROS Containers

- Data is automatically split into partitions during load / refresh / recovery operations.
- The **Tuple Mover** maintains physical separation of partitions.
- Each **ROS container** contains data for a single partition, though there can be multiple ROS containers for a single partition.

Partition Pruning

When a query predicate includes one more more columns in the partitioning clause, queries look only at relevant ROS containers. See [Partition Elimination](#) for details.

Managing Partitions

HP Vertica provides various options to let you manage and monitor the partitions you create.

PARTITIONS system table

You can display partition metadata, one row per partition key, per **ROS container**, by querying the [PARTITIONS](#) system table.

Given a projection named p1, with three ROS containers, the PARTITIONS table returns three rows:

```
=> SELECT
    PARTITION_KEY,
    PROJECTION_NAME,
    ROS_ID,
    ROS_SIZE_BYTES,
    ROS_ROW_COUNT,
    NODE_NAME
```

```
FROM partitions;
PARTITION_KEY | PROJECTION_NAME | ROS_ID | ROS_SIZE_BYTES | ROS_ROW_COUNT |
NODE_NAME
-----+-----+-----+-----+-----+
2008 | trade_p_node0001 | 45035996273740461 | 90 | 1 |
node0001
2007 | trade_p_node0001 | 45035996273740477 | 99 | 2 |
node0001
2006 | trade_p_node0001 | 45035996273740493 | 99 | 2 |
node0001
(3 rows)
```

MERGE_PARTITIONS() function

The [MERGE_PARTITIONS\(\)](#) function merges partitions between the specified values to a single ROS container and takes the following form:

```
MERGE_PARTITIONS ( table_name , partition_key_from , partition_key_to )
```

The edge values of the partition key are included in the range, and *partition_key_from* must be less than or equal to *partition_key_to*. Inclusion of partitions in the range is based on the application of less than(<)/greater than(>) operators of the corresponding data type. If *partition_key_from* is the same as *partition_key_to*, all ROS containers of the partition key are merged into one ROS.

Note: No restrictions are placed on a partition key's data type.

Users must have USAGE privilege on schema that contains the table.

The following series of statements show how to merge partitions in a table called T1:

```
=> SELECT MERGE_PARTITIONS('T1', '200', '400');
=> SELECT MERGE_PARTITIONS('T1', '800', '800');
=> SELECT MERGE_PARTITIONS('T1', 'CA', 'MA');
=> SELECT MERGE_PARTITIONS('T1', 'false', 'true');
=> SELECT MERGE_PARTITIONS('T1', '06/06/2008', '06/07/2008');
=> SELECT MERGE_PARTITIONS('T1', '02:01:10', '04:20:40');
=> SELECT MERGE_PARTITIONS('T1', '06/06/2008 02:01:10', '06/07/2008 02:01:10');
=> SELECT MERGE_PARTITIONS('T1', '8 hours', '1 day 4 hours 20 seconds');
```

PARTITION_TABLE() function

The [PARTITION_TABLE\(\)](#) function physically separates partitions into separate containers. Only ROS containers with more than one distinct value participate in the split.

The following example creates a simple table called *states* and partitions data by state.

```
=> CREATE TABLE states (year INTEGER NOT NULL,
```

```
state VARCHAR NOT NULL)  
PARTITION BY state;  
=> CREATE PROJECTION states_p (state, year) AS  
SELECT * FROM states  
ORDER BY state, year UNSEGMENTED ALL NODES;
```

Now call the `PARTITION_TABLE` function to partition table `states`:

```
=> SELECT PARTITION_TABLE('states');  
PARTITION_TABLE  
-----  
partition operation for projection 'states_p_node0004'  
partition operation for projection 'states_p_node0003'  
partition operation for projection 'states_p_node0002'  
partition operation for projection 'states_p_node0001'  
(1 row)
```

Notes

There are just a few more things worth mentioning to help you manage your partitions:

- To prevent too many **ROS containers**, be aware that delete operations must open all the containers; thus, ideally create fewer than 20 partitions and avoid creating more than 50.

You can use the `MERGE_PARTITIONS()` function to merge old partitions to a single ROS container.

- You cannot use non-deterministic functions in a `PARTITION BY` expression. One example is `TIMESTAMP WITH TIME ZONE`, because the value depends on user settings.
- A dimension table in a **pre-join projection** cannot be partitioned.

Partitioning, repartitioning, and Reorganizing Tables

Chris wrote this content in the Reference Manual. I just dragged it into a new topic for the Admin Guide. You probably already reviewed this.

Using the [ALTER TABLE](#) statement with its `PARTITION BY` syntax and the optional `REORGANIZE` keyword partitions or re-partitions a table according to the *partition-clause* that you define in the statement. HP Vertica immediately drops any existing partition keys when you execute the statement.

You can use the `PARTITION BY` and `REORGANIZE` keywords separately or together. However, you cannot use these keywords with any other `ALTER TABLE` clauses.

Partition-clause expressions are limited in the following ways:

- Your partition-clause must calculate a single non-null value for each row. You can reference multiple columns, but each row must return a single value.
- You can specify leaf expressions, functions, and operators in the partition clause expression.
- All leaf expressions in the partition clause must be either constants or columns of the table.
- Aggregate functions and queries are not permitted in the partition-clause expression.
- SQL functions used in the partition-clause expression must be **immutable**.

Partitioning or re-partitioning tables requires USAGE privilege on the schema that contains the table.

Reorganizing Data After Partitioning

Partitioning is not complete until you reorganize the data. The optional REORGANIZE keyword completes table partitioning by assigning partition keys. You can use REORGANIZE with PARTITION BY, or as the only keyword in the ALTER TABLE statement for tables that were previously altered with the PARTITION BY modifier, but were not reorganized with the REORGANIZE keyword.

If you specify the REORGANIZE keyword, data is partitioned immediately to the new schema as a background task.

Tip: As a best practice, HP recommends that you reorganize the data while partitioning the table, using PARTITION BY with the REORGANIZE keyword. If you do not specify REORGANIZE, performance for queries, DROP_PARTITION() operations, and node recovery could be degraded until the data is reorganized. Also, without reorganizing existing data, new data is stored according to the new partition expression, while the existing data storage remains unchanged.

Monitoring Reorganization

When you use the ALTER TABLE ... REORGANIZE, the operation reorganizes the data in the background.

You can monitor details of the reorganization process by polling the following system tables:

- [V_MONITOR.PARTITION_STATUS](#) displays the fraction of each table that is partitioned correctly.
- [V_MONITOR.PARTITION_REORGANIZE_ERRORS](#) logs any errors issued by the background REORGANIZE process.
- [V_MONITOR.PARTITIONS](#) displays NULLS in the partition_key column for any ROS's that have not been reorganized.

Note: The corresponding foreground process to ALTER TABLE ... REORGANIZE is [PARTITION_TABLE\(\)](#).

Auto Partitioning

HP Vertica attempts to keep data from each partition stored separately. Auto partitioning occurs when data is written to disk, such as during COPY DIRECT or **moveout** operations.

Separate storage provides two benefits: Partitions can be dropped quickly, and [partition elimination](#) can omit storage that does not need to need not to participate in a query plan.

Note: If you use INSERT...SELECT in a partitioned table, HP Vertica sorts the data before writing it to disk, even if the source of the SELECT has the same sort order as the destination.

Examples

The examples that follow use this simple schema. First create a table named t1 and partition the data on the c1 column:

```
CREATE TABLE t1 ( c1 INT NOT NULL,  
                 c2 INT NOT NULL)  
SEGMENTED BY c1 ALL NODES  
PARTITION BY c2;
```

Create two identically-segmented buddy projections:

```
CREATE PROJECTION t1_p AS SELECT * FROM t1 SEGMENTED BY HASH(c1) ALL NODES OFFSET 0; CREA  
TE PROJECTION t1_p1 AS SELECT * FROM t1 SEGMENTED BY HASH(c1) ALL NODES OFFSET 1;
```

Now insert some data:

```
INSERT INTO t1 VALUES(10,15);INSERT INTO t1 VALUES(20,25);  
INSERT INTO t1 VALUES(30,35);  
INSERT INTO t1 VALUES(40,45);
```

Query the table to verify the inputs:

```
SELECT * FROM t1; c1 | c2  
-----  
10 | 15  
20 | 25  
30 | 35  
40 | 45  
(4 rows)
```

Now perform a **moveout** operation on the projections in the table:


```
SELECT DO_TM_TASK('moveout','t1');          do_tm_task
-----
moveout for projection 't1_p1'
moveout for projection 't1_p'
(1 row)
```

Query the [PARTITIONS](#) system table, and you'll see that the four partition keys reside on two nodes, each in its own ROS container (see the `ros_id` column). The `PARTITION BY` clause was used on column `c2`, so HP Vertica auto partitioned the input values during the `COPY` operation:

```
SELECT partition_key, projection_name, ros_id, ros_size_bytes, ros_row_count, node_name F
ROM PARTITIONS WHERE projection_name like 't1_p1';
partition_key | projection_name |      ros_id      | ros_size_bytes | ros_row_count |
node_name
-----+-----+-----+-----+-----+
15            | t1_p1          | 49539595901154617 | 78            | 1            | n
ode0002
25            | t1_p1          | 54043195528525081 | 78            | 1            | n
ode0003
35            | t1_p1          | 54043195528525069 | 78            | 1            | n
ode0003
45            | t1_p1          | 49539595901154605 | 79            | 1            | n
ode0002
(4 rows)
```

HP Vertica does not auto partition when you refresh with the same sort order. If you create a new projection, HP Vertica returns a message telling you to refresh the projections; for example:

```
CREATE PROJECTION t1_p2 AS SELECT * FROM t1 SEGMENTED BY HASH(c1) ALL NODES OFFSET 2;
```

```
WARNING: Projection <public.t1_p2> is not available for query processing. Execute the
select start_refresh() function to copy data into this projection.
The projection must have a sufficient number of buddy projections and all nodes
must be up before starting a refresh.
```

Run the [START_REFRESH](#) function:

```
SELECT START_REFRESH();          start_refresh
-----
Starting refresh background process.
(1 row)
```

Query the `PARTITIONS` system table again. The partition keys now reside in two ROS containers, instead of four, which you can tell by looking at the values in the `ros_id` column. The `ros_row_count` column holds the number of rows in the ROS container:

```
SELECT partition_key, projection_name, ros_id, ros_size_bytes, ros_row_count, node_name F
ROM PARTITIONS WHERE projection_name like 't1_p2';
partition_key | projection_name |      ros_id      | ros_size_bytes | ros_row_count |
```

| node_name | | | | | |
|-----------|-------|-------------------|----|---|---|
| 15 | t1_p2 | 54043195528525121 | 80 | 2 | n |
| ode0003 | | | | | |
| 25 | t1_p2 | 58546795155895541 | 77 | 2 | n |
| ode0004 | | | | | |
| 35 | t1_p2 | 58546795155895541 | 77 | 2 | n |
| ode0004 | | | | | |
| 45 | t1_p2 | 54043195528525121 | 80 | 2 | n |
| ode0003 | | | | | |

(4 rows)

The following command more specifically queries ROS information for the partitioned tables. In this example, the query counts two ROS containers each on two different nodes for projection t1_p2:

```
SELECT ros_id, node_name, COUNT(*) FROM PARTITIONS WHERE projection_name LIKE 't1_p2' GROUP BY ros_id, node_name;
```

| ros_id | node_name | COUNT |
|-------------------|-----------|-------|
| 54043195528525121 | node0003 | 2 |
| 58546795155895541 | node0004 | 2 |

(2 rows)

This command returns a result of four ROS containers on two different nodes for projection t1_p1:

```
SELECT ros_id,node_name, COUNT(*) FROM PARTITIONS WHERE projection_name LIKE 't1_p1' GROUP BY ros_id, node_name;
```

| ros_id | node_name | COUNT |
|-------------------|-----------|-------|
| 49539595901154605 | node0002 | 1 |
| 49539595901154617 | node0002 | 1 |
| 54043195528525069 | node0003 | 1 |
| 54043195528525081 | node0003 | 1 |

(4 rows)

See Also

-
-
-

Eliminating Partitions

If the **ROS** containers of partitioned tables are not needed, HP Vertica can eliminate the containers from being processed during query execution. To eliminate ROS containers, HP Vertica compares query predicates to partition-related metadata.

Each ROS partition expression column maintains the minimum and maximum values of data stored in that ROS, and HP Vertica uses those min/max values to potentially eliminate ROS containers from query planning. Partitions that cannot contain matching values are not scanned. For example, if a ROS does not contain data that satisfies a given query predicate, the optimizer eliminates (prunes) that ROS from the query plan. After non-participating ROS containers have been eliminated, queries that use partitioned tables run more quickly.

Note: Partition pruning occurs at query run time and requires a query predicate on the partitioning column.

Assume a table that is partitioned by year (2007, 2008, 2009) into three ROS containers, one for each year. Given the following series of commands, the two ROS containers that contain data for 2007 and 2008 fall outside the boundaries of the requested year (2009) and get eliminated.

```
=> CREATE TABLE ... PARTITION BY EXTRACT(year FROM date);=> SELECT ... WHERE date = '12-2-2009';
```

| date | amount | date | amount | date | amount |
|--------------------------------|--------|--------------------------------|--------|--------------------------------|--------|
| 11/11/09 | 6 | 03/13/08 | 96 | 07/12/07 | 43 |
| 06/05/09 | 12 | 04/21/08 | 17 | 03/02/07 | 45 |
| ... | ... | ... | ... | ... | ... |
| 12/02/09 | 8 | 12/02/08 | 7 | 12/02/07 | 68 |
| Min: 01/01/09 Max: 12/31/09 | | Min: 01/01/08 Max: 12/31/08 | | Min: 01/01/07 Max: 12/31/07 | |

On any database that has been upgraded from version 3.5, or earlier, **ROS** containers are ineligible for partition elimination because they do not contain the minimum/maximum partition key values required. These ROS containers need to be recreated or merged by the **Tuple Mover**.

Making Past Partitions Eligible for Elimination

The following procedure lets you make past partitions eligible for elimination. The easiest way to guarantee that all ROS containers are eligible is to:

1. Create a new fact table with the same projections as the existing table.
2. Use INSERT..SELECT to populate the new table.
3. Drop the original table and rename the new table.

If there is not enough disk space for a second copy of the fact table, an alternative is to:

1. Verify that the Tuple Mover has finished all post-upgrade work; for example, when the following command shows no mergeout activity:

```
=> SELECT * FROM TUPLE_MOVER_OPERATIONS;
```

2. Identify which partitions need to be merged to get the ROS minimum/maximum values by running the following command:

```
=> SELECT DISTINCT table_schema, projection_name, partition_key      FROM partitions p
LEFT OUTER JOIN vs_ros_min_max_values v
  ON p.ros_id = v.delid
WHERE v.min_value IS null;
```

3. Insert a record into each partition that has ineligible ROS containers and commit.
4. Delete each inserted record and commit again.

At this point, the Tuple Mover automatically merges ROS containers from past partitions.

Verifying the ROS Merge

1. Query the TUPLE_MOVER_OPERATIONS table again:

```
=> SELECT * FROM TUPLE_MOVER_OPERATIONS;
```

2. Check again for any partitions that need to be merged:

```
=> SELECT DISTINCT table_schema, projection_name, partition_key      FROM partitions p
LEFT OUTER JOIN vs_ros_min_max_values v
  ON p.ros_id = v.rosid
WHERE v.min_value IS null;
```

Examples

Assume a table that is partitioned by time and will use queries that restrict data on time.

```
CREATE TABLE time (
  tdate DATE NOT NULL,
  tnum INTEGER)
PARTITION BY EXTRACT(year FROM tdate);
CREATE PROJECTION time_p (tdate, tnum) AS
SELECT * FROM time
ORDER BY tdate, tnum UNSEGMENTED ALL NODES;
```

Note: Projection sort order has no effect on partition elimination.

```
INSERT INTO time VALUES ('03/15/04' , 1);      INSERT INTO time VALUES ('03/15/05' , 2);  
INSERT INTO time VALUES ('03/15/06' , 3);  
INSERT INTO time VALUES ('03/15/06' , 4);
```

The data inserted in the previous series of commands would be loaded into three ROS containers, one per year, since that is how the data is partitioned:

```
SELECT * FROM time ORDER BY tnum;  tdate    | tnum  
-----+-----  
2004-03-15 |    1  --ROS1 (min 03/01/04, max 03/15/04)  
2005-03-15 |    2  --ROS2 (min 03/15/05, max 03/15/05)  
2006-03-15 |    3  --ROS3 (min 03/15/06, max 03/15/06)  
2006-03-15 |    4  --ROS3 (min 03/15/06, max 03/15/06)  
(4 rows)
```

Here's what happens when you query the `time` table:

- In this query, HP Vertica can eliminate ROS2 because it is only looking for year 2004:

```
=> SELECT COUNT(*) FROM time WHERE tdate = '05/07/2004';
```

- In the next query, HP Vertica can eliminate both ROS1 and ROS3:

```
=> SELECT COUNT(*) FROM time WHERE tdate = '10/07/2005';
```

- The following query has an additional predicate on the `tnum` column for which no minimum/maximum values are maintained. In addition, the use of logical operator OR is not supported, so no ROS elimination occurs:

```
=> SELECT COUNT(*) FROM time WHERE tdate = '05/07/2004' OR tnum = 7;
```

Moving Partitions

You can move partitions from one table to another using the [MOVE_PARTITIONS_TO_TABLE](#) function. Use this function as part of creating offline archives of older partitions. By moving partitions from one table to an intermediate table, you can then create a backup of the new table, and drop the partition. If you need the historical data later, you can restore the archived partitions, described in [Restoring Archived Partitions](#).

If the target table does not exist, the [MOVE_PARTITIONS_TO_TABLE](#) function creates a table definition using the CREATE TABLE statement with its LIKE clause. Creating a table with the LIKE clause is performed as a DDL operation. HP Vertica does not copy any data from the source table, and the new table is not connected to its source in any way. The CREATE TABLE statement with the LIKE clause does not copy constraints, automatic values (such as a sequences and identity values), or a default values. Corresponding columns will exist in the new table with the same type as the source table, but the columns will not have constraints or automatic values.

Archiving Steps

These are the steps required to archive partitions:

1. Prepare and move the partitions with the `MOVE_PARTITIONS_TO_TABLE` function
2. Create an object-level snapshot of the intermediate table
3. Drop the intermediate table

The next sections describe the archiving steps.

Preparing and Moving Partitions

Before moving partitions to another table, be sure to:

- Create a separate schema for the intermediate table
- Check that the name you plan to use does not conflict with an existing table name
- Use a name that represents the partition values you are moving
- Keep each partition in a different backup table

When you have created a separate schema for the intermediate table, call the [MOVE_PARTITIONS_TO_TABLE](#) function.

If you call `MOVE_PARTITIONS_TO_TABLE` and the destination table does not exist, the function will create the table automatically:

```
VMART=> SELECT MOVE_PARTITIONS_TO_TABLE (  
          'prod_trades',  
          '200801',  
          '200801',  
          'partn_backup.trades_200801');  
          MOVE_PARTITIONS_TO_TABLE  
-----  
  1 distinct partition values moved at epoch 15. Effective move epoch: 14.  
(1 row)
```

Creating a Snapshot of the Intermediate Table

Creating an object-level snapshot of the intermediate table containing the partitions you want to archive requires a `vbr.py` configuration file.

These are the two steps to create an object-level snapshot of an intermediate table so you can then drop the table:

1. As a best practice, HP Vertica recommends that you create a full database snapshot first, since you can only restore object-level snapshots into the original database. However, creating a full snapshot is not a requirement.
2. Create an object-level snapshot of the intermediate table.

For details of setting up backup hosts, creating a configuration file, and taking a snapshot, see [Backing Up and Restoring the Database](#).

Copying the Config File to the Storage Location

When `vbr.py` creates the partition snapshot, it copies it to the archive storage location automatically.

HP Vertica recommends that you also copy the configuration file for the partition snapshot to the storage location. You can do this automatically by entering `y` to the `Backup vertica configurations?` question when creating the configuration file for the snapshot.

Drop the Intermediate Table

You can drop the intermediate table into which you moved partitions to archive, as described in [Dropping and Truncating Tables](#). Dropping the intermediate table maintains database K-safety, keeping a minimum $K+1$ copies of the data, and more if additional projections exist.

See Also

-

Restoring Archived Partitions

You can restore partitions that you previously moved to an intermediate table, archived as an object-level snapshot, and then dropped.

Note: Restoring an archived partition requires that the original table definition has not changed since the partition was archived and dropped. If you have changed the table definition, you can only restore an archived partition using `INSERT/SELECT` statements, which are not described here.

These are the steps to restoring archived partitions:

1. Restore the snapshot of the intermediate table you saved when you moved one or more partitions to archive (see [Moving Partitions](#)).
2. Move the restored partitions from the intermediate table to the original table.
3. Drop the intermediate table.

See Also

-

Bulk Loading Data

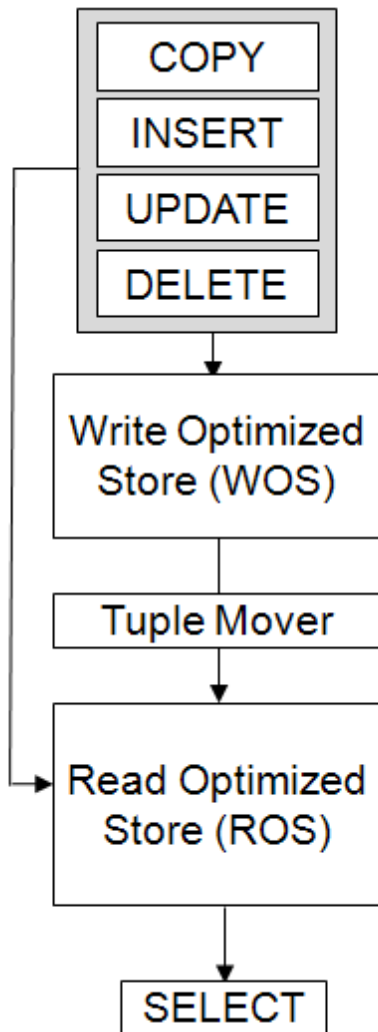
This section describes different methods for bulk loading data into an HP Vertica database using the [COPY statement](#). In its basic form, use COPY as follows:

```
COPY to_table FROM data_source
```

The COPY statement loads data from a file stored on the host or client (or in a data stream) into a database table. You can pass the COPY statement many different parameters to define various options such as:

- The format of the incoming data
- Metadata about the data load
- Which parser COPY should use
- Load data over parallel load streams
- How to transform data as it is loaded
- How to handle errors

HP Vertica's [hybrid storage model](#) provides a great deal of flexibility for loading and managing data.



See the remaining sections here for other options, and the [COPY](#) statement in the SQL Reference Manual for syntax details.

Checking Data Format Before or After Loading

HP Vertica expects all data files being loaded to be in the Unicode **UTF-8** format. You can load ASCII data, which is UTF-8 compatible. Character sets like ISO 8859-1 (Latin1), are incompatible with UTF-8 and are not supported.

Before loading data from text files, you can use several UNIX tools to ensure that your data is in UTF-8 format. The `file` command reports the encoding of any text files.

To check the type of a data file, use the `file` command. For example:

```
$ file Date_Dimension.tblDate_Dimension.tbl: ASCII text
```

The `file` command could indicate ASCII TEXT even though the file contains multibyte characters.

To check for multibyte characters in an ASCII file, use the `wc` command. For example:

```
$ wc Date_Dimension.tbl 1828 5484 221822 Date_Dimension.tbl
```

If the `wc` command returns an error such as `Invalid or incomplete multibyte or wide character`, the data file is using an incompatible character set.

This example describes files that are not UTF-8 data files. Two text files have filenames starting with the string `data`. To check their format, use the `file` command as follows:

```
$ file data* data1.txt: Little-endian UTF-16 Unicode text  
data2.txt: ISO-8859 text
```

The results indicate that neither of the files is in UTF-8 format.

Converting Files Before Loading Data

To convert files before loading them into HP Vertica, use the `iconv` UNIX command. For example, to convert the `data2.txt` file from the previous example, use the `iconv` command as follows:

```
iconv -f ISO88599 -t utf-8 data2.txt > data2-utf8.txt
```

See the man pages for `file` and `iconv` for more information.

Checking UTF-8 Compliance After Loading Data

After loading data, use the `ISUTF8` function to verify that all of the string-based data in the table is in UTF-8 format. For example, if you loaded data into a table named `nametable` that has a `VARCHAR` column named `name`, you can use this statement to verify that all of the strings are UTF-8 encoded:

```
=> SELECT name FROM nametable WHERE ISUTF8(name) = FALSE;
```

If all of the strings are in UTF-8 format, the query should not return any rows.

Performing the Initial Database Load

To perform the initial database load, use `COPY` with its `DIRECT` parameter from `vsqll`.

Tip: HP Vertica supports multiple schema types. If you have a **star schema**, load the smaller tables before you load the largest tables.

Only a **superuser** can use the `COPY` statement to bulk load data. Two exceptions to the superuser requirement are to:

1. Run `COPY` to load from a stream on the host (such as `STDIN`) rather than a file (see [Streaming Data via JDBC](#)).

2. Use the COPY statement with the FROM LOCAL option.

A non-superuser can also perform a standard [batch insert using a prepared statement](#), which invokes COPY to load data as a background task.

Extracting Data From an Existing Database

If possible, export the data in text form to a local file or attached disk. When working with large amounts of load data (> 500GB), HP recommends that you test the load process using smaller load files as described in [Configuration Procedure](#) to avoid compatibility or file formatting issues.

ETL products typically use ODBC or JDBC to extract data, which gives them program-level access to modify load file column values, as needed.

Database systems typically provide a variety of export methods.

Tip: To export data from an Oracle database, run a SELECT query in Oracle's SQL*Plus command line query tool using a specified column delimiter, suppressed headers, and so forth. Redirect the output to a local file.

Smaller tables generally fit into a single load file. Split any large tables into 250-500GB load files. For example, a 10 TB fact table will require 20-40 load files to maintain performance.

Checking for Delimiter Characters in Load Data

The default delimiter for the COPY statement is a vertical bar (|). Before loading your data, make sure that no CHAR(N) or VARCHAR(N) data values include the delimiter character.

To test for the existence of a specific character in a column, use a query such as this:

```
SELECT COUNT(*) FROM T WHERE X LIKE '%|%'
```

If only a few rows contain |, you can eliminate them from the load file using a WHERE clause and load them separately using a different delimiter.

Tip: : For loading data from an Oracle database, use a WHERE clause to avoid problem rows in the main load file, and the negated WHERE clause with REGEX_REPLACE for problem rows.

Moving Data From an Existing Database to HP Vertica Nodes

To move data from an existing database to HP Vertica, consider using:

- USB 2.0 (or possibly SATA) disks
- A fast local network connection

Deliver chunks of data to the different HP Vertica nodes by connecting the transport disk or by writing files from network copy.

Loading From a Local Hard Disk

USB 2.0 disks can deliver data at about 30 MB per second, or 108 GB per hour. USB 2.0 disks are easy to use for transporting data from Linux to Linux. Set up an ext3 filesystem on the disk and write large files there. Linux 2.6 has USB plug-and-play support, so a USB 2.0 disk is instantly usable on various Linux systems.

For other UNIX variants, if there is no common filesystem format available, use the disk without a filesystem to copy a single large file. For example:

```
$ cp bigfile /dev/sdc1
```

Even without a filesystem on the disk, plug-and-play support still works on Linux to provide a device node for the disk. To find out the assigned device, plug in the disk and enter:

```
$ dmesg | tail -40
```

SATA disks are usually internal, but can be external, or unmounted safely if they are internal.

Loading Over the Network

A 1Gbps (gigabits per second) network can deliver about 50 MB/s, or 180GB/hr. HP Vertica can load about 30-50GB/hour/node for a 1-Ksafe projection design. Therefore, you should use a dedicated 1Gbps LAN. Using a LAN with a performance that is < 1Gbps will be proportionally slower. HP Vertica recommends not loading data across an external network, because the delays over distance slow down the TCP protocol to a small fraction of its available bandwidth, even without competing traffic.

Note: The actual load rates you obtain can be higher or lower depending on the properties of the data, number of columns, number of projections, and hardware and network speeds. Load speeds can be further improved by using multiple parallel streams.

Loading From Windows

Use NTFS for loading files directly from Windows to Linux. Although Red Hat Linux as originally installed can read Windows FAT32 file systems, this is not recommended.

Using Load Scripts

You can write and run a load script for the [COPY](#) statement using a simple text-delimited file format. For information about other load formats see [Specifying a COPY Parser](#). HP Vertica recommends that you load the smaller tables before the largest tables. To check data formats before loading, see [Checking Data Format Before or After Loading](#).

Using Absolute Paths in a Load Script

Unless you are using the `COPY FROM LOCAL` statement, using `COPY` on a remote client requires an absolute path for a data file. You cannot use relative paths on a remote client. For a load script, you can use vsql variables to specify the locations of data files relative to your Linux working directory.

To use vsql variables to specify data file locations:

1. Create a vsql variable containing your Linux current directory.

```
\set t_pwd `pwd`
```

2. Create another vsql variable that uses a path relative to the Linux current directory variable for a specific data file.

```
\set input_file `':t_pwd'/Date_Dimension.tbl`
```

3. Use the second variable in the `COPY` statement:

```
COPY Date_Dimension FROM :input_file DELIMITER '|';
```

4. Repeat steps 2 and 3 to load all data files.

Note: `COPY FROM LOCAL` does not require an absolute path for data files. You can use paths that are relative to the client's running directory.

Running a Load Script

You can run a load script on any host, as long as the data files are on that host.

1. Change your Linux working directory to the location of the data files.

```
$ cd /opt/vertica/doc/retail_example_database
```

2. Run the Administration Tools.

```
$ /opt/vertica/bin/admintools
```

3. Connect to the database.
4. Run the load script.

Using COPY and COPY LOCAL

The COPY statement bulk loads data into an HP Vertica database. You can initiate loading one or more files or pipes on a cluster host. You can load directly from a client system, too, using the COPY statement with its FROM LOCAL option.

COPY lets you load *parsed* or *computed* data. Parsed data is from a table or schema using one or more columns, and computed data is calculated with a column expression on one or more column values.

COPY invokes different parsers depending on the format you specify:

- Delimited text (the default parser format, but not specified)
- Native binary (NATIVE) (not supported with COPY LOCAL)
- Native varchar (NATIVE VARCHAR) (not supported with COPY LOCAL)
- Fixed-width data (FIXEDWIDTH)

COPY has many options, which you combine to make importing data flexible. For detailed syntax for the various options see the SQL Reference Manual. For example:

| For this option... | See this section... |
|---|--|
| Read uncompressed data, or data compressed with GZIP or BZIP. | Specifying COPY FROM Options |
| Insert data into the WOS (memory) or directly into the ROS (disk). | Choosing a Load Method |
| Set parameters such as data delimiters and quote characters for the entire load operation or, for specific columns. | Loading UTF-8 Format Data |
| Transform data before inserting it into the database. | Transforming Data During Loads |

Copying Data From an HP Vertica Client

Use COPY LOCAL to load files on a client to the HP Vertica database. For example, to copy a GZIP file from your local client, use a command such as this:

```
=> COPY store.store_dimension FROM LOCAL '/usr/files/my_data/input_file' GZIP;
```

You can use a comma-separated list to load multiple files of the same compression type. COPY local then concatenates the files into a single file, so you cannot combine files with different compression types in the list. When listing multiple files, be sure to specify the type of every input file, such as BZIP, as shown:

```
COPY simple_table FROM LOCAL 'input_file.bz' BZIP, 'input_file.bz' BZIP;
```

You can load on a client (LOCAL) from STDIN, as follows:

```
COPY simple_table FROM LOCAL STDIN;
```

Transforming Data During Loads

To promote a consistent database and reduce the need for scripts to transform data at the source, HP Vertica lets you transform data as part of loading it into the target database. Transforming data during loads is useful for computing values to insert into a target database column from other columns in the source database.

To transform data during load, use the following syntax to specify the target column for which you want to compute values, as an expression:

```
COPY [ [database-name.]schema-name.]table [( [Column as Expression] / column[FORMAT '
format' ]
[ ,...])]
FROM ...
```

Understanding Transformation Requirements

When transforming data during loads, the `COPY` statement must contain at least one parsed column. The parsed column can be a `FILLER` column. (See [Ignoring Columns and Fields in the Load File](#) for more information about using fillers.)

Specify only `RAW` data in the parsed column source data. If you specify nulls in that `RAW` data, the columns are evaluated with the same rules as for SQL statement expressions.

You can intersperse parsed and computed columns in a `COPY` statement.

Loading FLOAT Values

HP Vertica parses floating-point values internally. `COPY` does not require you to cast floats explicitly, unless you need to transform the values for another reason. For more information, see [DOUBLE PRECISION \(FLOAT\)](#).

Using Expressions in COPY Statements

The expression you use in a `COPY` statement can be as simple as a single column or as complex as a case expression for multiple columns. You can specify multiple columns in a `COPY` expression, and have multiple `COPY` expressions refer to the same parsed column. You can specify `COPY` expressions for columns of all supported data types.

`COPY` expressions can use many HP Vertica-supported SQL functions, operators, constants, `NULL`s, and comments, including these functions:

- [Date/time](#)
- [Formatting Functions](#)

- [String](#)
- [Null-handling](#)
- [System information](#)

COPY expressions cannot use SQL meta functions ([HP Vertica-specific](#)), [analytic](#) functions, [aggregate](#) functions, or computed columns.

For computed columns, all parsed columns in the expression must be listed in the COPY statement. Do not specify FORMAT or RAW in the source data for a computed column.

Expressions used in a COPY statement can contain only constants. The return data type of the expression must be coercible to that of the target column. Parsed column parameters are also coerced to match the expression.

Handling Expression Errors

Errors that occur in COPY expressions are treated as SQL errors, not parse errors. When a parse error occurs, COPY rejects the row and adds a copy of the row to the rejected data file. COPY also adds a message to the exceptions file describing why the row was rejected. For example, <DBMS_SHORT> does not implicitly cast data types during parsing. If a type mismatch occurs between the data being loaded and a column type (such as loading a text value for a FLOAT column), COPY rejects the row, but continues processing.

COPY expression errors are treated as SQL errors and cause the entire load to rollback. For example, if the COPY statement has an expression with a transform function, and a syntax error occurs in the function, the entire load is rolled back. The [HP Vertica-specific](#) log file will include the SQL error message, but the reason for the rollback is not obvious without researching the log.

Transformation Example

Following is a small transformation example.

1. Create a table and corresponding projection.

```
CREATE TABLE t (
  year VARCHAR(10),
  month VARCHAR(10),
  day VARCHAR(10),
  k timestamp
);
CREATE PROJECTION tp (
  year,
  month,
  day,
  k)
AS SELECT * from t;
```

2. Use COPY to copy the table, computing values for the year, month, and day columns in the target database, based on the timestamp columns in the source table.
3. Load the parsed column, timestamp, from the source data to the target database.

```
COPY t(year AS TO_CHAR(k, 'YYYY'),
      month AS TO_CHAR(k, 'Month'),
      day AS TO_CHAR(k, 'DD'),
      k FORMAT 'YYYY-MM-DD') FROM STDIN NO COMMIT;
2009-06-17
1979-06-30
2007-11-26
\.
```

4. Select the table contents to see the results:

```
SELECT * FROM t;
 year | month | day | k
-----+-----+-----+-----
 2009 | June  | 17  | 2009-06-17 00:00:00
 1979 | June  | 30  | 1979-06-30 00:00:00
 2007 | November | 26  | 2007-11-26 00:00:00
(3 rows)
```

Deriving Table Columns From Data File Columns

You can use COPY to derive a table column from the data file to load.

The next example illustrates how to use the year, month, and day columns from the source input to derive and load the value for the `TIMESTAMP` column in the target database.

1. Create a table and corresponding projection:

```
=> CREATE TABLE t (k TIMESTAMP);=> CREATE PROJECTION tp (k) AS SELECT * FROM t;
```

2. Use COPY with the `FILLER` keyword to skip the year, month, and day columns from the source file.

```
=> COPY t(year FILLER VARCHAR(10),
      month FILLER VARCHAR(10),
      day FILLER VARCHAR(10),
      k AS TO_DATE(YEAR || MONTH || DAY, 'YYYYMMDD') )
FROM STDIN NO COMMIT;
>> 2009|06|17
>> 1979|06|30
>> 2007|11|26
>> \.
```

3. Select from the copied table to see the results:

```
=> SELECT * FROM t;  
      k  
-----  
2009-06-17 00:00:00  
1979-06-30 00:00:00  
2007-11-26 00:00:00  
(3 rows)
```

See also [Using Sequences](#) for how to generate an auto-incrementing value for columns.

See the [COPY](#) statement in the SQL Reference Manual for further information.

Specifying COPY FROM Options

Each COPY statement requires a FROM option to indicate the location of the file or files being loaded. This syntax snippet shows the available FROM keywords, and their associated file format options:

```
FROM { STDIN ..... [ BZIP | GZIP | UNCOMPRESSED ]  
...| 'pathToData' [ ON nodename | ON ANY NODE ]  
..... [ BZIP | GZIP | UNCOMPRESSED ] [, ...]  
...| LOCAL STDIN | 'pathToData'  
..... [ BZIP | GZIP | UNCOMPRESSED ] [, ...]  
}
```

Each of the FROM keywords lets you optionally specify the format of the load file as UNCOMPRESSED, BZIP, or GZIP.

Note: When using COPY in conjunction with a CREATE EXTERNAL TABLE statement, you cannot use the COPY FROM STDIN or LOCAL options.

Loading From STDIN

Using STDIN for the FROM option lets you load uncompressed data, bzip, or gzip files.

Loading From a Specific Path

Use the 'pathToData' option to indicate the location of the load file, optionally indicating a node name or ON ANY NODE to indicate which node (or nodes) should parse the load file. You can load one or more files in the supported formats: UNCOMPRESSED, BZIP, or GZIP.

Note: Using the ON ANY NODE clause indicates that the source file to load is on all of the nodes, so COPY opens the file and parses it from any node in the cluster. Be sure that the source file you specify is available and accessible on each cluster node.

If *pathToData* resolves to a storage location, and the user invoking COPY is not a superuser, these are the required permissions:

- The storage location must have been created with the USER option (see [ADD_LOCATION](#))
- The user must already have been granted READ access to the storage location where the file(s) exist, as described in [GRANT \(Storage Location\)](#)

Further, if a non-superuser invokes COPY from a storage location to which she has privileges, HP Vertica also checks any symbolic links (symlinks) the user has to ensure no symlink can access an area to which the user has not been granted privileges.

Loading with Wildcards (glob) ON ANY NODE

COPY fully supports the ON ANY NODE clause with a wildcard (glob). You can invoke COPY for a large number of files in a shared directory with a single statement such as this:

```
COPY myTable FROM '/mydirectory/ofmanyfiles/*.dat' ON ANY NODE
```

Using a wildcard with the ON ANY NODE clause expands the file list on the initiator node, and then distributes the individual files among all nodes, evenly distributing the COPY workload across the entire cluster.

Loading From a Local Client

To bulk load data from a client, and without requiring database superuser privileges, use the COPY FROM LOCAL option. You can load from either STDIN, or a specific path, but not from a specific node (or ON ANY NODE), since you are loading from the client. All local files are loaded and parsed serially with each COPY statement, so you cannot perform parallel loads with the LOCAL option. See [Using Parallel Load Streams](#).

You can load one or more files in the supported formats: UNCOMPRESSED, BZIP, or GZIP.

For specific information about saving rejected data and exceptions files when using COPY from LOCAL, see [Capturing Load Exceptions and Rejections](#).

Choosing a Load Method

Depending on what data you are loading, COPY statement has these load method options:

| Load Method | Description and Use |
|-------------|--|
| AUTO | Loads data into WOS . Use the default COPY load method for smaller bulk loads. |
| DIRECT | Loads data directly into ROS containers. Use the DIRECT load method for large bulk loads (100MB or more). |
| TRICKLE | Loads only into WOS. Use for frequent incremental loads, after the initial bulk load is complete. |

Note: COPY ignores any load method you specify as part of creating an external table.

Loading Directly into WOS (AUTO)

This is the default load method. If you do not specify a load option, COPY uses the AUTO method to load data into **WOS** (Write Optimized Store in memory). The default method is good for smaller bulk loads (< 100MB). Once WOS is full, COPY continues loading directly to **ROS** (Read Optimized Store on disk) containers.

Loading Directly to ROS (DIRECT)

Use the DIRECT keyword in the COPY statement to bypass loading data into WOS, and instead, load data directly into ROS containers. The DIRECT option is best suited for loading large amounts of data (100MB or more) at a time. Using DIRECT for many loads of smaller data sets results in many ROS containers, which have to be combined later.

```
COPY a FROM stdin DIRECT;  
COPY b FROM LOCAL STDIN DIRECT;
```

Note: A large initial bulk load can temporarily affect query performance while HP Vertica organizes the data.

Loading Data Incrementally (TRICKLE)

Use the TRICKLE load option to load data incrementally after the initial bulk load is complete. Trickle loading loads data only into the WOS. If the WOS becomes full, an error occurs and the entire data load is rolled back. Use this option only when you have a finely-tuned load and moveout process so that you are sure there is room in the WOS for the data you are loading. This option is more efficient than AUTO when loading data into partitioned tables.

For other details on trickle-loading data and [WOS Overflow](#) into the ROS, see [Trickle Loading](#).

Loading Data Without Committing Results (NO COMMIT)

Use the NO COMMIT option with COPY (unless the tables are temp tables) to perform a bulk load transaction without automatically committing the results. This option is useful for executing multiple COPY commands in a single transaction.

For example, the following set of COPY ... NO COMMIT statements performs several copy statements sequentially, and then commits them all. In this way, all of the copied data is either committed or rolled back as a single transaction.

```
COPY... NO COMMIT;  
COPY... NO COMMIT;
```

```
COPY... NO COMMIT;  
COPY X FROM LOCAL NO COMMIT;  
COMMIT;
```

Using a single transaction for multiple COPY statements also allows HP Vertica to load the data more efficiently since it can combine the larger amounts of data from multiple COPY statements into fewer **ROS** containers.

HP recommends that you COMMIT or ROLLBACK the current transaction before you use COPY.

You can combine NO COMMIT with most other existing COPY options, but not the REJECTED DATA AS TABLE option. The standard transaction semantics apply. If a transaction is in progress that was initiated by a statement other than COPY (such as INSERT), using COPY with NO COMMIT adds rows to the existing transaction, rather than starting a new one. The previous statements are NOT committed.

Note: NO COMMIT is ignored when COPY is part of the CREATE EXTERNAL TABLE FROM COPY statement.

Using NO COMMIT to Detect Constraint Violations

You can use the NO COMMIT option to detect constraint violations as part of the load process.

HP Vertica checks for constraint violations when running a query, but not when loading data. To detect constraint violations, load data with the NO COMMIT keyword and then test the load using [ANALYZE_CONSTRAINTS](#). If you find any constraint violations, you can roll back the load because you have not committed it.

See [Detecting Constraint Violations](#) for detailed instructions.

Using COPY Interactively

HP Vertica recommends using the COPY statement in one or more script files, as described in [Using Load Scripts](#). You can also use commands such as the following interactively by piping a text file to vsql and executing COPY (or COPY FROM LOCAL) statement with the standard input stream as the input file. For example:

```
$ cat fact_table.tbl | vsql -c "COPY FACT_TABLE FROM STDIN DELIMITER '|' DIRECT";  
$ cat fact_table.tbl | vsql -c "COPY FACT_TABLE FROM LOCAL STDIN DELIMITER '|' DIRECT";
```

Canceling a COPY Statement

If you cancel a bulk data load, the COPY statement rolls back all rows that it attempted to load.

Specifying a COPY Parser

By default, COPY uses the DELIMITER parser to load raw data into the database. Raw input data must be in UTF-8, delimited text format. Data is compressed and encoded for efficient storage. If your raw data does not consist primarily of delimited text, specify the parser COPY should use to align most closely with the load data:

- NATIVE
- NATIVE VARCHAR
- FIXEDWIDTH

Note: You do not specify the DELIMITER parser directly; absence of a specific parser indicates the default.

Two other parsers are available specifically to load unstructured data into flex tables, as described in the [Using Flex Table Parsers](#) section of the Flex Tables Guide:

- FJSONPARSER
- FDELIMITEDPARSER

While these two parsers are for use with flex tables, you can also use them with columnar tables, to make loading data more flexible. For instance, you can load JSON data into a columnar table in one load, and delimited data into the same table in another. The Flex Tables Guide describes this use case and presents an example.

Using a different parser for your data can improve load performance. If delimited input data includes binary data types, COPY translates the data on input. See [Using Load Scripts](#) and [Loading Binary \(Native\) Data](#) for examples. You can also load binary data, but only if it adheres to the COPY format requirements, described in [Creating Native Binary Format Files](#).

You cannot mix raw data types that require different parsers (such as NATIVE and FIXEDWIDTH) in a single bulk load COPY statement. To check data formats before (or after) loading, see [Checking Data Format Before or After Loading](#).

Specifying Load Metadata

In addition to choosing a parser option, COPY supports other options to determine how to handle the raw load data. These options are considered load metadata, and you can specify metadata options at different parts of the COPY statement as follows:

| Metadata Option | As a Column or Expression Option | As a COLUMN OPTION | As a FROM Level Option |
|-----------------|----------------------------------|--------------------|------------------------|
| DELIMITER | X | X | X |

| Metadata Option | As a Column or Expression Option | As a COLUMN OPTION | As a FROM Level Option |
|-------------------|----------------------------------|--------------------|------------------------|
| ENCLOSED BY | X | X | X |
| ESCAPE AS | X | X | X |
| NULL | X | X | X |
| TRIM | X | | X |
| RECORD TERMINATOR | | | X |
| SKIP | | | X |
| SKIP BYTES | | | X (Fixed-width only) |
| TRAILING NULLCOLS | | | X |

The following precedence rules apply to all data loads:

- All column-level parameters override statement-level parameters.
- COPY uses the statement-level parameter if you do not specify a column-level parameter.
- COPY uses the default metadata values for the DELIMITER, ENCLOSED BY, ESCAPE AS, and NULL options if you do not specify them at either the statement- or column-level.

When you specify any metadata options, COPY uses the parser to produce the best results and stores the raw data and its corresponding metadata in the following formats:

| Raw data format | Metadata Format | Parser |
|-----------------|-----------------|----------------|
| UTF-8 | UTF-8 | DELIMITER |
| Binary | Binary | NATIVE |
| UTF-8 | Binary | NATIVE VARCHAR |
| UTF-8 | UTF-8 | FIXEDWIDTH |

See Also

- [COPY](#)

Interpreting Last Column End of Row Values

When bulk-loading delimited text data using the default parser (DELIMITED), the last column end of row value can be any of the following:

- Record terminator
- EOF designator
- Delimiter and a record terminator

Note: The FIXEDWIDTH parser always requires exactly a record terminator. No other permutations work.

For example, given a three-column table, the following input rows for a COPY statement using a comma (,) delimiter are each valid:

```
1,1,11,1,1,  
1,1,  
1,1,,
```

The following examples illustrate how COPY can interpret different last column end of data row values.

Using a Single End of Row Definition

To see how COPY interprets a single end of row definition:

1. Create a two-column table `two_col`, specifying column b with a default value of 5:

```
VMart=> create table two_col (a int, b int DEFAULT 5);CREATE TABLE
```

2. COPY the `two_col` table using a comma (,) delimiter, and enter values for only one column (as a single, multi-line entry):

```
VMart=> copy two_col from stdin delimiter ',';Enter data to be copied followed by a n  
ewline.  
End with a backslash and a period on a line by itself.  
>> 1,  
>> 1,  
>> \.
```

The COPY statement complete successfully.

3. Query table `two_col`, to display the two NULL values for column b as blank:

```
VMart=> select * from two_col; a | b
----+----
 1 |
 1 |
(2 rows)
```

Here, COPY expects two values for each column, but gets only one. Each input value is followed by a delimiter (,), and an implicit record terminator (a newline character, \n). You supply a record terminator with the ENTER or RETURN key. This character is not represented on the screen.

In this case, the delimiter (,) and record terminator (\n) are handled independently. COPY interprets the delimiter (,) to indicate the end of one value, and the record terminator (\n) to specify the end of the column row. Since no value follows the delimiter, COPY supplies an empty string before the record terminator. By default, the empty string signifies a NULL, which is a valid column value.

Using a Delimiter and Record Terminator End of Row Definition

To use a delimiter and record terminator together as an end of row definition:

1. Copy column a (a) of the two_col table, using a comma delimiter again, and enter two values:

```
VMart=> copy two_col (a) from stdin delimiter ',';Enter data to be copied followed by
a newline.
End with a backslash and a period on a line by itself.
>> 2,
>> 2,
>> \.
```

The COPY statement again completes successfully.

2. Query table two_col to see that column b now includes two rows with its default value (5):

```
VMart=> select * from two_col; a | b
----+----
 1 |
 1 |
 2 | 5 2 | 5
(4 rows)
```

In this example, COPY expects values for only one column, because of the column (a) directive. As such, COPY interprets the delimiter and record terminator together as a single, valid, last column end of row definition. Before parsing incoming data, COPY populates column b with its default value, because the table definition has two columns and the COPY statement supplies only one. This example populates the second column with its default column list value, while the previous example used the supplied input data.

Loading UTF-8 Format Data

You can specify these parameters at either a statement or column basis:

- ENCLOSED BY
- ESCAPE AS
- NULL
- DELIMITER

Loading Special Characters As Literals

The default COPY statement escape key is a backslash (\). By preceding any *special character* with an escape character, COPY interprets the character that follows literally, and copies it into the database. These are the special characters that you escape to load them as literals:

| Special Character | COPY Statement Usage |
|--------------------------------------|--------------------------------------|
| Vertical bar () | Default COPY ... DELIMITER character |
| Empty string (' ') | Default COPY ... NULL string. |
| Backslash (\) | Default COPY ... ESC character. |
| Newline and other control characters | Various |

To use a special character as a literal, prefix it with an escape character. For example, to include a literal backslash (\) in the loaded data (such as when including a file path), use two backslashes (\\). COPY removes the escape character from the input when it loads escaped characters.

Using a Custom Column Separator (DELIMITER)

The default COPY delimiter is a vertical bar (|). The DELIMITER is a single ASCII character used to separate columns within each record of a file. Between two delimiters, COPY interprets all string data in load files as characters. Do not enclose character strings in quotes, since quote characters are also treated as literals between delimiters.

You can define a different delimiter using any ASCII value in the range E'\000' to E'\177' inclusive. For instance, if you are loading **CSV** data files, and the files use a comma (,) character as a delimiter, you can change the default delimiter to a comma. You cannot use the same character for both the DELIMITER and NULL options.

If the delimiter character is among a string of data values, use the ESCAPE AS character (\ by default) to indicate that the delimiter should be treated as a literal.

The COPY statement accepts empty values (two consecutive delimiters) as valid input data for CHAR and VARCHAR data types. COPY stores empty columns as an empty string (' '). An empty string is not equivalent to a NULL string.

To indicate a non-printing delimiter character (such as a tab), specify the character in extended string syntax (E'...'). If your database has StandardConformingStrings enabled, use a Unicode string literal (U&'...'). For example, use either E'\t' or U&'\0009' to specify tab as the delimiter.

Using a Custom Column Option *DELIMITER*

This example, redefines the default delimiter through the COLUMN OPTION parameter.

1. Create a simple table.

```
=> CREATE TABLE t(      pk INT,
      col1 VARCHAR(10),
      col2 VARCHAR(10),
      col3 VARCHAR(10),
      col4 TIMESTAMP);
```

2. Use the COLUMN OPTION parameter to change the col1 default delimiter to a tilde (~).

```
=> COPY t COLUMN OPTION(col1 DELIMITER '~') FROM STDIN NO COMMIT;>> 1|ee~gg|yy|1999-1
2-12
>> \.
=> SELECT * FROM t;
pk | col1 | col2 | col3 |          col4
-----+-----+-----+-----+-----
 1 | ee   | gg   | yy   | 1999-12-12 00:00:00
(1 row)
```

Defining a Null Value (*NULL*)

The default NULL value for COPY is an empty string (''). You can specify a NULL as any ASCII value in the range E'\001' to E'\177' inclusive (any ASCII character except NUL: E'\000'). You cannot use the same character for both the DELIMITER and NULL options.

When NULL is an empty string (''), use quotes to insert an empty string instead of a NULL. For example, using NULL " ENCLOSED BY ''':

- 1| |3 — Inserts a NULL in the second column.
- 1|""|3 — Inserts an empty string instead of a NULL in the second columns.

To input an empty or literal string, use quotes (ENCLOSED BY); for example:

```
NULL 'NULL 'literal'
```

A NULL is case-insensitive and must be the only value between the data field delimiters. For example, if the null string is NULL and the delimiter is the default vertical bar (|):

|NULL| indicates a null value.

| NULL | does not indicate a null value.

When you use the `COPY` command in a script, you must substitute a double-backslash for each null string that includes a backslash. For example, the scripts used to load the example database contain:

```
COPY ... NULL E'\\n' ...
```

Loading NULL Values

You can specify NULL by entering fields without content into a data file, using a field delimiter.

For example, given the default delimiter (`|`) and default NULL (empty string) definition, `COPY` inserts the following input data:

```
| | 1| 2 | 3  
4 | | 5  
6 | |
```

into the table as follows:

```
(null, null, 1)(null, 2, 3)  
(4, null, 5)  
(6, null, null)
```

If NULL is set as a literal (`'null'`), `COPY` inserts the following inputs:

```
null | null | 1null | 2 | 3  
4 | null | 5  
6 | null | null
```

as follows:

```
(null, null, 1)(null, 2, 3)  
(4, null, 5)  
(6, null, null)
```

Filling Columns with Trailing Nulls (TRAILING NULLCOLS)

Loading data using the `TRAILING NULLCOLS` option inserts NULL values into any columns without data. Before inserting `TRAILING NULLCOLS`, HP Vertica verifies that the column does not have a `NOT NULL` constraint.

To use the `TRAILING NULLCOLS` parameter to handle inserts with fewer values than data columns:

1. Create a table:

```
=> CREATE TABLE z (      a INT,  
                          b INT,  
                          c INT );
```

2. Insert some values into the table:

```
=> INSERT INTO z VALUES (1, 2, 3);
```

3. Query table z to see the inputs:

```
=> SELECT * FROM z; a | b | c  
-----  
1 | 2 | 3  
(1 row)
```

4. Insert two rows of data from STDIN, using TRAILING NULLCOLS:

```
=> COPY z FROM STDIN TRAILING NULLCOLS;>> 4 | 5 | 6  
>> 7 | 8  
>> \.
```

5. Query table z again to see the results. Using TRAILING NULLCOLS, the COPY statement correctly handled the third row of column c, which had no value:

```
=> SELECT * FROM z; a | b | c  
-----  
1 | 2 | 3  
4 | 5 | 6  
7 | 8 |  
(3 rows)
```

Attempting to Fill a NOT NULL Column with TRAILING NULLCOLS

You cannot use TRAILING NULLCOLS on a column that has a NOT NULL constraint. For instance:

1. Create a table n, declaring column b with a NOT NULL constraint:

```
=> CREATE TABLE n (      a INT,
```

```
b INT NOT NULL,  
c INT );
```

2. Insert some table values:

```
=> INSERT INTO n VALUES (1, 2, 3);=> SELECT * FROM n;  
a | b | c  
-----  
1 | 2 | 3  
(1 row)
```

3. Use COPY with TRAILING NULLCOLS on table n to see the COPY error due to the column constraint:

```
=> COPY n FROM STDIN trailing nullcols abort on error;Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 4 | 5 | 6  
>> 7 | 8  
>> 9  
>> \.  
ERROR: COPY: Input record 3 has been rejected (Cannot set trailing column to NULL as column 2 (b) is NOT NULL)
```

4. Query the table to see that the COPY statement values were rejected:

```
=> SELECT * FROM n; a | b | c  
-----  
1 | 2 | 3  
(1 row)
```

Changing the Default Escape Character (ESCAPE AS)

The default escape character is a backslash (\). To change the default to a different character, use the `ESCAPE AS` option. To use an alternative escape character:

```
=> COPY mytable FROM '/data/input.txt' ESCAPE AS E('\001');
```

You can set the escape character to be any ASCII value value in the range `E'\001'` to `E'\177'` inclusive.

Eliminating Escape Character Handling

If you do not want any escape character and want to prevent any characters from being interpreted as escape sequences, use the `NO ESCAPE` option as part of the `COPY` statement.

Delimiting Characters (ENCLOSED BY)

The COPY ENCLOSED BY parameter lets you set an ASCII character to delimit characters to embed in string values. You can use any ASCII value in the range E '\001' to E '\177' inclusive (any ASCII character except NULL: E '\000') for the ENCLOSED BY value. Using double quotation marks (") is the most commonly used quotation character. For instance, the following parameter specifies that input data to the COPY statement is enclosed within double quotes:

```
ENCLOSED BY '"'
```

With the following input (using the default DELIMITER (|) character), specifying:

```
"vertica | value"
```

Results in:

- Column 1 containing "vertica
- Column 2 containing value"

Notice the double quotes (") before vertica and after value.

Using the following sample input data as follows, columns are distributed as shown:

```
"1", "vertica,value", ",", ""
```

```
col1 | col2          | col3 | col4-----+-----+-----+-----  
1    | vertica,value | ,    | '    
(1 row)
```

Alternatively, write the above example using any ASCII character of your choosing:

```
~1~, ~vertica,value~, ~,~, ~'~
```

If you use a single quote ('), rather than double quotes (") as the ENCLOSED BY character, you must escape it using extended string syntax, a Unicode literal string (if StandardConformingStrings is enabled), or by using four single quotes:

```
ENCLOSED BY E'\''ENCLOSED BY U&'\0027'  
ENCLOSED BY ''''
```

Using any of the definitions means the following input is properly parsed:

```
'1', 'vertica,value', ',', '\'
```

See [String Literals \(Character\)](#) for an explanation of the string literal formats you can use to specify the ENCLOSED BY parameter.

Use the `ESCAPE AS` character to embed the `ENCLOSED BY` delimiter within character string values. For example, using the default `ESCAPE AS` character (`\`) and double quote as the `ENCLOSED BY` character, the following input returns "vertica":

```
"\"vertica\""
```

Using *ENCLOSED BY* for a Single Column

The following example uses double quotes to enclose a single column (rather than the entire row). The `COPY` statement also specifies a comma (,) as the delimiter.

```
=> COPY Retail.Dim (Dno, Dname ENCLOSED BY '"', Dstore) FROM '/home/dbadmin/dim3.txt'  
DELIMITER ','  
EXCEPTIONS '/home/dbadmin/exp.txt';
```

This example correctly loads data such as:

```
123,"Smith, John",9832
```

Specifying a Custom End of Record String (*RECORD TERMINATOR*)

To specify the literal character string that indicates the end of a data file record, use the `RECORD TERMINATOR` parameter, followed by the string to use. If you do not specify a value, then HP Vertica attempts to determine the correct line ending, accepting either just a linefeed (`E'\n'`) common on UNIX systems, or a carriage return and linefeed (`E'\r\n'`) common on Windows platforms.

For example, if your file contains comma-separated values terminated by line feeds that you want to maintain, use the `RECORD TERMINATOR` option to specify an alternative value:

```
=> COPY mytable FROM STDIN DELIMITER ',' RECORD TERMINATOR E'\n';
```

To specify the `RECORD TERMINATOR` as non-printing characters, use either the extended string syntax or Unicode string literals. The following table lists some common record terminator characters. See [String Literals](#) for an explanation of the literal string formats.

| Extended String Syntax | Unicode Literal String | Description | ASCII Decimal |
|------------------------|----------------------------|-----------------|---------------|
| <code>E'\b'</code> | <code>U&'\0008'</code> | Backspace | 8 |
| <code>E'\t'</code> | <code>U&'\0009'</code> | Horizontal tab | 9 |
| <code>E'\n'</code> | <code>U&'\000a'</code> | Linefeed | 10 |
| <code>E'\f'</code> | <code>U&'\000c'</code> | Formfeed | 12 |
| <code>E'\r'</code> | <code>U&'\000d'</code> | Carriage return | 13 |
| <code>E'\''</code> | <code>U&'\005c'</code> | Backslash | 92 |

If you use the `RECORD TERMINATOR` option to specify a custom value, be sure the input file matches the value. Otherwise, you may get inconsistent data loads.

Note: The record terminator cannot be the same as `DELIMITER`, `NULL`, `ESCAPE`, or `ENCLOSED BY`.

If using JDBC, HP recommends that you use the following value for the `RECORD TERMINATOR`:

```
System.getProperty("line.separator")
```

Examples

The following examples use a comma (,) as the `DELIMITER` for readability.

```
,1,2,3,,1,2,3  
1,2,3,
```

Leading (,) and trailing (,) delimiters are ignored. Thus, the rows all have three columns.

| | |
|-------------------------------|---|
| <pre>123, '\n', \n, 456</pre> | Using a non-default null string, the row is interpreted as: <pre>123newLine \n 456</pre> |
|-------------------------------|---|

| | |
|--|---|
| <pre>123,this\, that\, or the other,something else,456</pre> | The row would be interpreted as: <pre>123this, that, or the other something else 456</pre> |
|--|---|

Loading Native Varchar Data

Use the `NATIVE VARCHAR` parser option when the raw data consists primarily of `CHAR` or `VARCHAR` data. `COPY` performs the conversion to the actual table data types on the database server. This option with `COPY LOCAL` is not supported.

Using `NATIVE VARCHAR` does not provide the same efficiency as `NATIVE`. However, `NATIVE VARCHAR` precludes the need to use delimiters or to escape special characters, such as quotes, which can make working with client applications easier.

Note: `NATIVE VARCHAR` does not support concatenated BZIP and GZIP files.

Batch data inserts performed through the HP Vertica ODBC and JDBC drivers automatically use either the `NATIVE BINARY` or `NATIVE VARCHAR` formats. ODBC and JDBC use `NATIVE BINARY` if the application data types match the actual table data types exactly (including maximum

lengths of CHAR/VARCHAR and precision/scale of numeric data types), which provide the best possible load performance. If there is any data type mismatch, NATIVE VARCHAR is used.

Loading Binary (Native) Data

You can load binary data using the NATIVE parser option, except with COPY LOCAL, which does not support this option. Since binary-format data does not require the use and processing of delimiters, it precludes the need to convert integers, dates, and timestamps from text to their native storage format, and improves load performance over delimited data. All binary-format files must adhere to the formatting specifications described in [Appendix: Binary File Formats](#).

Native binary format data files are typically larger than their delimited text format counterparts, so use GZIP or BZIP to compress the data before loading it. NATIVE BINARY does not support concatenated BZIP and GZIP files. You can load native (binary) format files when developing plug-ins to ETL applications, and completing batch insert commands with ODBC and JDBC.

Note: The ODBC client driver does not perform any data validation. When loading native (binary) format data with the ODBC client driver, your application should validate the binary data format before loading to confirm that it conforms to the HP Vertica specifications.

There is no copy format to load binary data byte-for-byte because the column and record separators in the data would have to be escaped. Binary data type values are padded and translated on input, and also in the functions, operators, and casts supported.

Loading Hexadecimal, Octal, and Bitstring Data

You can use hexadecimal, octal, and bitstring formats only to load binary columns. To specify these column formats, use the COPY statement's FORMAT options:

- **Hexadecimal**
- **Octal**
- **Bitstring**

The following examples illustrate how to use the FORMAT option.

1. Create a table:

```
=> CREATE TABLE t(
    oct VARBINARY(5),
    hex VARBINARY(5),
    bitstring VARBINARY(5) );
```

2. Create the projection:

```
=> CREATE PROJECTION t_p(oct, hex, bitstring) AS SELECT * FROM t;
```

3. Use the COPY command from STDIN (not a file), specifying each of the formats:

```
=> COPY t (oct FORMAT 'octal',          hex FORMAT 'hex',  
          bitstring FORMAT 'bitstring')  
FROM STDIN DELIMITER ',';
```

4. Enter the data to load, ending the statement with a backslash (\) and a period (.) on a separate line:

```
>> 141142143144145,0x6162636465,0110000101100010011000110110010001100101>> \.
```

5. Use a select query on table t to view the input values results:

```
=> SELECT * FROM t;oct      | hex      | bitstring  
-----+-----+-----  
abcde | abcde | abcde  
(1 row)
```

COPY uses the same default format to load binary data, as used to input binary data. Since the backslash character ('\') is the default escape character, you must escape octal input values. For example, enter the byte '\141' as '\\141'.

Note: If you enter an escape character followed by an invalid octal digit or an escape character being escaped, COPY returns an error.

On input, COPY translates string data as follows:

- Uses the [HEX_TO_BINARY](#) function to translate from hexadecimal representation to binary.
- Uses the [BITSTRING_TO_BINARY](#) function to translate from bitstring representation to binary.

Both functions take a VARCHAR argument and return a VARBINARY value.

You can also use the escape character to represent the (decimal) byte 92 by escaping it twice; for example, '\\\\'. Note that vsql inputs the escaped backslash as four backslashes. Equivalent inputs are hex value '0x5c' and octal value '\134' ($134 = 1 \times 8^2 + 3 \times 8^1 + 4 \times 8^0 = 92$).

You can load a delimiter value if you escape it with a backslash. For example, given delimiter '|', '\\001\\|\\002' is loaded as {1,124,2}, which can also be represented in octal format as '\\001\\174\\002'.

If you insert a value with more bytes than fit into the target column, COPY returns an error. For example, if column c1 is VARBINARY(1):

```
=> INSERT INTO t (c1) values ('ab'); ERROR: 2-byte value too long for type Varbinary(1)
```

If you implicitly or explicitly cast a value with more bytes than fit the target data type, COPY silently truncates the data. For example:

```
=> SELECT 'abcd'::binary(2); binary
-----
ab
(1 row)
```

Hexadecimal Data

The optional '0x' prefix indicates that a value is hexadecimal, not decimal, although not all hexadecimal values use A-F; for example, 5396. COPY ignores the 0x prefix when loading the input data.

If there are an odd number of characters in the hexadecimal value, the first character is treated as the low **nibble** of the first (furthest to the left) byte.

Octal Data

Loading octal format data requires that each byte be represented by a three-digit octal code. The first digit must be in the range [0,3] and the second and third digits must both be in the range [0,7].

If the length of an octal value is not a multiple of three, or if one of the three digits is not in the proper range, the value is invalid and COPY rejects the row in which the value appears. If you supply an invalid octal value, COPY returns an error. For example:

```
SELECT '\\000\\387'::binary(8);ERROR: invalid input syntax for type binary
```

Rows that contain binary values with invalid octal representations are also rejected. For example, COPY rejects '\\008' because '\\ 008' is not a valid octal number.

BitString Data

Loading bitstring data requires that each character must be zero (0) or one (1), in multiples of eight characters. If the bitstring value is not a multiple of eight characters, COPY treats the first *n* characters as the low bits of the first byte (furthest to the left), where *n* is the remainder of the value's length, divided by eight.

Examples

The following example shows VARBINARY HEX_TO_BINARY(VARCHAR) and VARCHAR TO_HEX (VARBINARY) usage.

1. Create table *t* and its projection with binary columns:

```
=> CREATE TABLE t (c BINARY(1));=> CREATE PROJECTION t_p (c) AS SELECT c FROM t;
```

2. Insert minimum and maximum byte values, including an IP address represented as a character string:

```
=> INSERT INTO t values(HEX_TO_BINARY('0x00')); => INSERT INTO t values(HEX_TO_BINARY('0xFF'));  
=> INSERT INTO t values (V6_ATON('2001:DB8::8:800:200C:417A'));
```

Use the TO_HEX function to format binary values in hexadecimal on output:

```
=> SELECT TO_HEX(c) FROM t;to_hex  
-----  
00  
ff  
20  
(3 rows)
```

See Also

- [COPY](#)
- [Binary Data Types](#)
- [Formatting Functions](#)
- [ASCII](#)

Loading Fixed-Width Format Data

Use the FIXEDWIDTH parser option to bulk load fixed-width data. You must specify the COLSIZES option values to specify the number of bytes for each column. The definition of the table you are loading (COPY table f (x, y, z)) determines the number of COLSIZES values to declare.

To load fixed-width data, use the COLSIZES option to specify the number of bytes for each input column. If any records do not have values, COPY inserts one or more null characters to equal the specified number of bytes. The last record in a fixed-width data file must include a record terminator to determine the end of the load data.

Supported Options for Fixed-Width Data Loads

Loading fixed-width data supports the options listed in the [COPY Option Summary](#).

These options are not supported:

- DELIMITER
- ENCLOSED BY

- ESCAPE AS
- TRAILING NULLCOLS

Using Nulls in Fixed-Width Data

The default NULL string for a fixed-width load cannot be an empty string, and instead, consists of all spaces. The number of spaces depends on the column width declared with the COLSIZES (integer, [,...]) option.

For fixed-width loads, the NULL definition depends on whether you specify NULL at the column or statement level, as follows:

- *Statement level*—NULL must be defined as a single-character. The default (or custom) NULL character is repeated for the entire width of the column.
- *Column Level*—NULL must be defined as a string whose length matches the column width.

For fixed-width loads, if the input data column has fewer values than the specified column size, COPY inserts NULL characters. The number of NULLs must match the declared column width. If you specify a NULL string at the column level, COPY matches the string with the column width.

Note: To turn off NULLs, use the NULL AS option and specify NULL AS ''.

Defining a Null Character (Statement Level)

1. Create a two-column table (fw):

```
VMart=> create table fw(co int, ci int);CREATE TABLE
```

2. Copy the table, specifying null as 'N', and enter some data:

```
VMart=> copy fw from STDIN fixedwidth colsizes(2,2) null as 'N' no commit;Enter data
to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> NN12
>> 23NN
>> NNNN
>> nnnn
>> \.
```

3. Select all (*) from the table:

```
VMart=> select * from fw;
co | ci
```

```
----+----  
    | 12  
23  |  
    |  
    |  
    |  
    |  
(5 rows)
```

Defining a Custom Record Terminator

To define a record terminator other than the COPY default when loading fixed-width data, take these steps:

1. Create a table, `fw`, with two columns, `co` and `ci`:

```
VMart=> create table fw(co int, ci int);CREATE TABLE
```

2. Copy table `fw`, specifying two 2-byte column sizes, and specifying a comma (,) as the record terminator:

```
VMart=> copy fw from STDIN fixedwidth colsizes(2,2) record terminator ',';Enter data  
to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> 1234,1444,6666  
>> \.
```

3. Select all (*) from the table:

```
VMart=> select * from fw; co | ci  
----+----  
 12 | 34  
 14 | 44  
(2 rows)
```

The SELECT output indicates only two values. COPY rejected the third value (6666) because it was not followed by a comma (,) record terminator. Fixed-width data requires a trailing record terminator only if you explicitly specify a record terminator explicitly.

Copying Fixed-Width Data

Use COPY FIXEDWIDTH COLSIZES (n [, ...]) to load files into an HP Vertica database. By default, all spaces are NULLs. For example, in the simple case:

```
=> create table mytest(co int, ci int);=> create projection mytest_p1 as select * from my  
test segmented by hash(co) all nodes;
```



```
=> create projection mytest_p2 as select * from mytest segmented by hash(co) all nodes of
fset 1;
=> copy mytest(co,ci) from STDIN fixedwidth colsizes(6,4) no commit;
=> select * from mytest order by co;
  co | ci
-----+-----
(0 rows)
```

Skipping Content in Fixed-Width Data

The COPY statement has two options to skip input data. The SKIP BYTES option is only for fixed-width data loads:

| | |
|-------------------------|---|
| SKIP BYTES <i>total</i> | Skips the <i>total</i> number (integer) of bytes from the input data. |
| SKIP <i>records</i> | Skips the number (integer) of <i>records</i> you specify. |

This example uses SKIP BYTES to skip 10 bytes when loading a fixed-width table with two columns (4 and 6 bytes):

1. Copy a table, using SKIP BYTES to skip 10 bytes of input data:

```
VMart=> copy fw from stdin fixedwidth colsizes (4,6) SKIP BYTES 10;Enter data to be c
opied followed by a newline.
End with a backslash and a period on a line by itself.
>> 2222666666
>> 1111999999
>> 1632641282
>> \.
```

2. Select all (*) from the table:

```
VMart=> select * from fw order by co;  co |  ci
-----+-----
 1111 | 999999
 1632 | 641282
(2 rows)
```

The select output indicates that COPY skipped the first 10 bytes of load data, as directed.

This example uses SKIP when loading a fixed-width (4,6) table to skip one (1) record of input data:

1. Copy a table, using SKIP to skip 1 record of input data:

```
VMart=> copy fw from stdin fixedwidth colsizes (4,6) SKIP 1;Enter data to be copied f
ollowed by a newline.
End with a backslash and a period on a line by itself.
```

```
>> 2222666666  
>> 1111999999  
>> 1632641282  
>> \.
```

2. Select all (*) from the table:

```
VMart=> select * from fw order by co; co | ci  
-----+-----  
1111 | 999999  
1632 | 641282  
(2 rows)
```

The SELECT output indicates that COPY skipped the first record of load data, as directed.

Trimming Characters in Fixed-Width Data Loads

Use the TRIM option to trim a character. TRIM accepts a single-byte character, which is trimmed at the beginning and end of the data. For fixed-width data loads, when you specify a TRIM character, COPY first checks to see if the row is NULL. If the row is not null, COPY trims the character(s). The next example instructs COPY to trim the character A, and shows the results. Only the last two lines entered comply to the specified (4, 6) fixed width:

1. Copy table fw, specifying the TRIM character, A:

```
VMart=> copy fw from stdin fixedwidth colsizes(4,6) TRIM 'A';  
Enter data to be copied followed by a newline.  
End with a backslash and a period on a line by itself.  
>> A2222A444444  
>> 2222A444444  
>> A22A444444  
>> A22AA4444A  
>> \.
```

2. Select all (*) from the table:

```
VMart=> select * from fw order by co;  
co | ci  
-----+-----  
22 | 4444  
22 | 444444  
(2 rows)
```

Using Padding in Fixed-Width Data Loads

By default, the padding character is ' ' (a single space). The padding behavior for fixed-width data loads is similar to how a space is treated in other formats, differing by data type as follows:

| Datatype | Padding |
|--|--|
| Integer | Leading and trailing spaces |
| Bool | Leading and trailing spaces |
| Float | Leading and trailing spaces |
| [var]Binary | None. All characters are significant. |
| [Var]Char | Trailing spaces if string is too large |
| DateInterval Time Timestamp TimestampTZ TimeTZ | None. All characters are significant. The COPY statement uses an internal algorithm to parse these data types. |
| Date (formatted) | Use the COPY FORMAT option string to match the expected column length. |
| Numerics | Leading and trailing spaces |

Ignoring Columns and Fields in the Load File

When bulk loading data, your source data may contain a column that does not exist in the destination table. Use the FILLER option to have COPY ignore an input column and the fields it contains when a corresponding column does not exist in the destination table. You can also use FILLER to transform data through derivation from the source into the destination. Use FILLER for:

- Omitting columns that you do not want to transfer into a table.
- Transforming data from a source column and then loading the transformed data to the destination table, without loading the original, untransformed source column (parsed column). (See [Transforming Data During Loads.](#))

Using the FILLER Parameter

Your COPY statement expressions can contain one or more filler columns. You can use any number of filler columns in the expression. The only restriction is that at least one column must *not* be a filler column. You cannot specify target table columns as filler, regardless of whether they are in the column list.

A data file can consist entirely of filler columns, indicating that all data in a file can be loaded into filler columns and then transformed and loaded into table columns. The filler column must be a parsed column, not a computed column. Also, the name of the filler column must be unique within both the source file and the destination table. You must specify the data type of the filler column as part of the FILLER parameter.

FILLER Parameter Examples

You can specify all parser parameters for filler columns, and all statement level parser parameters apply to filler columns.

To ignore a column, use the COPY statement FILLER parameter, followed by its data type. The next example creates a table with one column, and then copies it using two filler parameters. Since the second filler column is not part of any expression, it is discarded:

```
create table t (k timestamp);copy t(y FILLER date FORMAT 'YYYY-MM-DD', t FILLER varchar(10), k as y) from STDIN no commit;
2009-06-17|2009-06-17
\.
```

The following example derives and loads the value for the timestamp column in the target database from the year, month, and day columns in the source input. The year, month, and day columns are not loaded because the FILLER parameter specifies to skip each of those columns:

```
CREATE TABLE t (k TIMESTAMP);CREATE PROJECTION tp (k) AS SELECT * FROM t;
COPY t(year FILLER VARCHAR(10),
      month FILLER VARCHAR(10),
      day FILLER VARCHAR(10),
      k AS TO_DATE(YEAR || MONTH || DAY, 'YYYYMMDD'))
FROM STDIN NO COMMIT;
2009|06|17
1979|06|30
2007|11|26
\.
```

```
SELECT * FROM t;
      k
-----
2009-06-17 00:00:00
1979-06-30 00:00:00
2007-11-26 00:00:00
(3 rows)
```

See the [COPY](#) statement in the SQL Reference Manual for more information about syntax and usage.

Loading Data into Pre-Join Projections

A pre-join projection stores rows of a fact table joined with rows of dimension tables. Storing pre-join projections improves query performance, since the join does not occur when you query the data, but is already stored.

To insert a row into the fact table of a pre-join projection, the associated values of the dimension table's columns must be looked up. Thus, an insert into a pre-join projection shares some of the qualities of a query. The following sections describe the behaviors associated with loading data into pre-join projections.

Foreign and Primary Key Constraints

To ensure referential integrity, foreign and primary key constraints are enforced on inserts into fact tables of pre-join projections. If a fact row attempts to reference a row that does not exist in the dimension table, the load is automatically rolled back. The load is also rolled back if a fact row references more than one dimension row.

Note: Unless it also has a **NOT NULL** constraint, a column with a **FOREIGN KEY** constraint can contain a NULL value even though the dimension table's primary key column does not contain a NULL value. This allows for records to be inserted into the fact table even though the foreign key in the dimension table is not yet known.

The following tables and SQL examples highlight these concepts.

- **Fact Table:** Employees
- **Dimension Table:** HealthPlans
- **Pre-join Projection:** Joins Employees to HealthPlans using the PlanID column

```
CREATE PROJECTION EMP_HEALTH (EmployeeID, FirstName, LastName, Type)
AS (SELECT EmployeeID, FirstName, LastName,
    Type FROM Employees, HealthPlans
    WHERE Employees.HealthPlanID = HealthPlans.PlanID)
```

Employees (Fact Table)

| EmployeeID(PK) | FirstName | LastName | PlanID(FK) |
|----------------|-----------|----------|------------|
| 1000 | David | Taylor | 01 |
| 1001 | Sunil | Ray | 02 |
| 1002 | Janet | Hildreth | 02 |
| 1003 | Pequan | Lee | 01 |

HealthPlans (Dimension Table)

| PlanID(PK) | Description | Type |
|------------|-------------|------|
| 01 | PlanOne | HMO |
| 02 | PlanTwo | PPO |

The following sequence of commands generate a missing foreign key error that results in a rollback because the reference is to a non-existent dimension row.

```
INSERT INTO Employees (EmployeeID, First, Last, PlanID) VALUES (1004, 'Ben', 'Smith', 0
4);
```

The following sequence of commands generate a foreign key error that results in a rollback because a duplicate row in the HealthPlans dimension table is referenced by an insert in the Employees fact table. The error occurs when the Employees fact table references the HealthPlans dimension table.

```
INSERT INTO HealthPlan VALUES(02, 'MyPlan', 'PPO');
INSERT INTO Employee VALUES(1005, 'Juan', 'Hernandez', 02);
```

Concurrent Loads into Pre-Join Projections

HP Vertica supports concurrent inserts where two transactions can simultaneously insert rows into the same table. A transaction inserting records into a pre-join projection can run concurrently with another transaction inserting records into either the fact table or a dimension table of the pre-join projection. A load into a pre-join projection cannot run concurrently with updates or deletes on either the fact or the dimension tables.

When concurrently loading fact and dimension tables, the state of the dimension tables is fixed at the start of the insert or load into the fact table. Rows that are added to a dimension table after the start of an insert or load into a fact table are not available for joining because they are not visible to the fact table. The client is responsible for ensuring that all values in dimension tables are present before issuing the insert or load statement.

The following examples illustrate these behaviors.

- **Fact Table:** Sales
- **Dimension Table:** Employees
- **Pre-join Projection:** sales join employees on sales.seller=employees.empno

Success

| Session A | Session B | Description |
|--|---|---|
| INSERT INTO EMPLOYEES (EMPNO, NAME) VALUES (1, 'Bob'); | | |
| COPY INTO SALES (AMT, SELLER) 5000 1 3500 1 . . . | | Records loaded by this COPY command all refer to Bob's sales (SELLER = 1) |
| | INSERT INTO EMPLOYEES (EMPNO, NAME)VALUES (2, 'Frank'); | |

| Session A | Session B | Description |
|-----------|---|--|
| 7234 1 | COPY INTO SALES (AMT,SELLER) 50 2 75 2 . . . | Records loaded by this COPY command all refer to Frank's sales (SELLER = 2). |
| COMMIT; | COMMIT; | Both transactions are successful. |

Failure

| Session A | Session B | Description |
|--|--|---|
| INSERT INTO EMPLOYEES (EMPNO, NAME) 1 Bob | | |
| 2 Terry | COPY INTO SALES (AMT,SELLER) 5000 1 | The transaction in Session B fails because the value inserted into the dimension table in Session A was not visible before the COPY into the pre-join in Session B was initiated. |

Using Parallel Load Streams

You can use COPY for multiple parallel load streams to load an HP Vertica database. COPY LOCAL parses files serially, and does not support parallel load streams.

These are the options for parallel load streams:

- Issue multiple separate COPY statements to load different files from different nodes.

This option lets you use vsql, ODBC, ADO.net, or JDBC. You can load server-side files, or client-side files using the COPY from LOCAL statement.
- Issue a single multi-node COPY command that loads different files from different nodes specifying the nodename option for each file.
- Issue a single multi-node COPY command that loads different files from any node, using the ON ANY NODE option.
- Use the COPY x WITH SOURCE PloadDelimitedSource option to parallel load using all cores on the server node on which the file resides.

Files can be of different formats, such as BZIP, GZIP, and others. The multi-node option is not available with the COPY from LOCAL parameter.

The single multi-node COPY options (*nodename* | ON ANY NODE) are possible only using the `vsq1` command, and not all COPY options support this behavior. However, using this method to copy data can result in significantly higher performance and efficient resource usage.

See [COPY](#) in the SQL Reference Manual for syntax details.

While there is no restriction to the number of files you can load, the optimal number of load streams depends on several factors, including the number of nodes, the physical and logical schemas, host processors, memory, disk space, and so forth. Too many load streams can cause systems to run out of memory. See [Best Practices for Managing Workload Resources](#) for advice on configuring load streams.

Monitoring COPY Loads and Metrics

You can check COPY loads using:

- HP Vertica functions
- `LOAD_STREAMS` system table

Using HP Vertica Functions

Two meta-functions return COPY metrics for the number of accepted or rejected rows from a COPY statement:

1. To get the number of accepted rows, use the [GET_NUM_ACCEPTED_ROWS](#) function:

```
VMart=> select get_num_accepted_rows();
get_num_accepted_rows
-----
                        11
(1 row)
```

2. To check the number of rejected rows, use the [GET_NUM_REJECTED_ROWS](#) function:

```
VMart=> select get_num_rejected_rows();
get_num_rejected_rows
-----
                        0
(1 row)
```

Using the LOAD_STREAMS System Table

HP Vertica includes a set of system tables that include monitoring information, as described in [Using System Tables](#). The `LOAD_STREAMS` system table includes information about load stream metrics from COPY and `COPY FROM VERTICA` statements, so you can query table values to get COPY metrics.

To see all table columns:


```
VMart=> select * from load_streams;
```

Using the STREAM NAME Parameter

Using the STREAM NAME parameter as part of the COPY statement labels COPY streams explicitly so they are easier to identify in the LOAD_STREAMS system table.

To use the STREAM NAME parameter:

```
=> COPY mytable FROM myfile DELIMITER '|' DIRECT STREAM NAME 'My stream name';
```

The LOAD_STREAMS system table includes stream names for every COPY statement that takes more than 1-second to run. The 1-second duration includes the time to plan and execute the statement.

HP Vertica maintains system table metrics until they reach a designated size quota (in kilobytes). The quota is set through internal processes and cannot be set or viewed directly.

Other LOAD_STREAMS Columns for COPY Metrics

These LOAD_STREAMS table column values depend on the load status:

- ACCEPTED_ROW_COUNT
- REJECTED_ROW_COUNT
- PARSE_COMPLETE_PERCENT
- SORT_COMPLETE_PERCENT

When a COPY statement using the DIRECT option is in progress, the ACCEPTED_ROW_COUNT field can increase to the maximum number of rows in the input file as the rows are being parsed.

If COPY reads input data from multiple named pipes, the PARSE_COMPLETE_PERCENT field will remain at zero (0) until *all* named pipes return an **EOF**. While COPY awaits an EOF from multiple pipes, it may seem to be hung. Before canceling the COPY statement, however, check your [system CPU and disk accesses](#) to see if any activity is in progress.

In a typical load, PARSE_COMPLETE_PERCENT can either increase slowly to 100%, or jump to 100% quickly if you are loading from named pipes or STDIN, while SORT_COMPLETE_PERCENT is at 0. Once PARSE_COMPLETE_PERCENT reaches 100%, SORT_COMPLETE_PERCENT increases to 100%. Depending on the data sizes, a significant lag can occur between the time PARSE_COMPLETE_PERCENT reaches 100% and the time SORT_COMPLETE_PERCENT begins to increase.

This example sets the VSQL expanded display, and then selects various columns of data from the LOAD_STREAMS system table:

```
=> \pset expandedExpanded display is on.  
=> SELECT stream_name, table_name, load_start, accepted_row_count,
```

```
rejected_row_count, read_bytes, unsorted_row_count, sorted_row_count,  
sort_complete_percent FROM load_streams;  
-[ RECORD 1 ]-----+-----  
stream_name      | fact-13  
table_name       | fact  
load_start       | 2010-12-28 15:07:41.132053  
accepted_row_count | 900  
rejected_row_count | 100  
read_bytes       | 11975  
input_file_size_bytes | 0  
parse_complete_percent | 0  
unsorted_row_count | 3600  
sorted_row_count  | 3600  
sort_complete_percent | 100
```

See the [SQL Reference Manual](#) for other meta-function details.

See the [Programmer's Guide](#) for client-specific documentation.

Capturing Load Exceptions and Rejections

Rejected data occurs during the parsing phase of data loads. A [COPY](#) operation can encounter other problems and failures during different load phases, but rejections consist only of parse errors.

Rejections always have a reason, which you can view as part of the rejection by including both the `REJECTED DATA` and `EXCEPTIONS` clauses with the `COPY` statement (see "[Capturing Load Exceptions and Rejections](#)").

There are many reasons why parsing load data could result in a rejected data row. Following are a few examples of parsing errors that cause a rejected row:

- Unsupported parsing options
- Incorrect data types for the table into which data is being loaded
- Malformed context for the parser in use
- Missing delimiters

The `COPY` statement automatically saves the rejected data (a copy of the row that failed to load), and load exceptions (a text message describing why that row did not load) to individual files. By default, HP Vertica saves both files in the database catalog subdirectory, `CopyErrorLogs`; for example:

```
v_mart_node003_catalog\CopyErrorLogs\trans-STDIN-copy-from-exceptions.1  
v_mart_node003_catalog\CopyErrorLogs\trans-STDIN-copy-from-rejected-data.1
```

You can use the `COPY` parameters `REJECTED DATA` and `EXCEPTIONS` to save one, or both, files to a file location of your choice.

COPY Parameters for Handling Rejections and Exceptions

Several optional parameters let you determine how strictly the COPY statement handles the rejection it encounters when loading data. For example, you can have COPY fail when it encounters a single rejection or permit a specific number of rejected rows before the load fails. Two COPY arguments, REJECTED DATA and EXCEPTIONS, are designed to work together for these purposes.

This section describes the parameters you use to specify the rejections file or table of your choice, and to control load exception handling.

Specifying Where to Save Rejected Data Rows

When you use the REJECTED DATA *path* argument, you specify one or both of the following:

- The path and file in which to save rejected data.

The rejected data file includes only the rejected records themselves. If you want to see the reason each record was rejected, you must also specify the EXCEPTIONS *path* option. The reasons for rejection are written to a separate file.

- The name of the table into which rejected data rows are saved.

Including the AS TABLE clause creates *reject_table* and populates it with both the rejected record and the reason for rejection. You can then query the table to access rejected data information. For more information, see [Saving Load Rejections](#).

For COPY LOCAL operations, the rejected data file must reside on the client.

If *path* resolves to a storage location and the user invoking COPY is not a **superuser**, the following permissions are required:

- The storage location must have been created with the USER usage type (see [ADD_LOCATION](#)).
- The user must already have been granted access to the storage location where the files exist, as described in [GRANT \(Storage Location\)](#).

Saving the Reason for the Rejected Data Row

An EXCEPTIONS file stores the reason each rejected record was rejected during the parsing phase. To save this file, include the EXCEPTIONS clause in the COPY statement and specify the file name or absolute path for the file.

Note: You cannot specify an EXCEPTIONS file if you are using the REJECTED DATA . . AS TABLE clause. See [Saving Load Exceptions \(EXCEPTIONS\)](#).

If you are running COPY LOCAL operations, the file must reside on the client.

If *path* resolves to a storage location and the user invoking COPY is not a superuser, the following permissions are required:

- The storage location must have been created with the `USER` usage type (see [ADD_LOCATION](#)).
- The user must already have been granted access to the storage location where the files exist, as described in [GRANT \(Storage Location\)](#).

Enforcing Truncating or Rejecting Rows (ENFORCELENGTH)

The `ENFORCELENGTH` parameter determines whether `COPY` truncates or rejects rows of data type `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` when they do not fit the target table. By default, `COPY` truncates offending rows of these data types without rejecting them.

For example, if you omit the `ENFORCELENGTH` argument and load 'abc' into a table column specified as `VARCHAR(2)`, `COPY` truncates the value to 'ab' and loads it. If you load the same value with the `ENFORCELENGTH` parameter, `COPY` rejects the 'abc' value, rather than truncating it.

Note: HP Vertica supports `NATIVE` and `NATIVE VARCHAR` values up to 65K. If any value exceeds this limit, `COPY` rejects the row, even when `ENFORCELENGTH` is not in use.

Specifying Maximum Rejections Before a Load Fails (REJECTMAX)

The `REJECTMAX` parameter specifies the maximum number of logical records that can be rejected before a load fails. A rejected row consists of the data that caused an exception and could not be parsed into the corresponding data type during a bulk load. Rejected data does not indicate referential constraints.

When the number of rejected records will be greater than the `REJECTMAX` value (`REJECTMAX+1`), the load fails. If you do not specify a value for `REJECTMAX`, or if the value is 0, `COPY` allows an unlimited number of exceptions to occur.

Note: `COPY` does not accumulate rejected records across files or nodes while data is loading. If one rejected data file exceeds the maximum reject number, the entire load fails.

Aborting Data Loads for Any Error (ABORT ON ERROR)

Using the `ABORT ON ERROR` argument is the most restrictive way to load data, because no exceptions or rejections are allowed. A `COPY` operation stops if any row is rejected. No data is loaded and HP Vertica rolls back the command.

If you use the `ABORT ON ERROR` as part of a `CREATE EXTERNAL TABLE AS COPY FROM` statement, the option is used whenever a query references the external table. The offending error is saved in the `COPY` exceptions or rejected data file.

Understanding Row Rejections and Rollback Errors

Depending on the type of error that COPY encounters, HP Vertica does one of the following:

- Rejects the offending row and loads the other rows into the table
- Rolls back the entire COPY statement without loading any data

Note: If you specify `ABORT ON ERROR` with the `COPY` command, the load automatically rolls back if *any* row causes an exception. The offending row or error is written to the applicable exceptions or rejected data file or to a rejections table, if specified.

The following table summarizes the difference between a rejected row or rollback. See also the example following the table.

| Rejects offending row | Rolls back entire load |
|---|---|
| <p>When HP Vertica encounters an error parsing records in the input file, it rejects the offending row and continues the load.</p> <p>Rows are rejected if they contain any of the following:</p> <ul style="list-style-type: none">• Incompatible data types• Missing fields• Missing delimiters | <p>When HP Vertica rolls back a COPY statement, the command fails without loading data.</p> <p>The following conditions cause a load rollback:</p> <ul style="list-style-type: none">• Server-side errors, such as lack of memory• Violations of primary key or foreign key constraints• Loading NULL data into a NOT NULL column |

Example: Rejection versus Rollback

This example illustrates what happens when HP Vertica can't parse a row to the requested data type. For example, in the following COPY statement, "a::INT + b::INT" is a SQL expression in which "a" and "b" are derived values:

```
=> CREATE TABLE t (i INT);
=> COPY t (a FILLER VARCHAR, b FILLER VARCHAR, i AS a::INT + b::INT)
    FROM STDIN;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> cat|dog
>> \.
```

Vertica Analytics Platform cannot parse the row to the requested data type and rejects the row:

```
ERROR 2827: Could not convert "cat" from column "**FILLER**".a to an int8
```

If "a" were 'cat' and "b" 'dog', the expression 'cat'::INT + 'dog'::INT would return an error and the COPY statement would fail (roll back) without loading any data.

```
=> SELECT 'cat'::INT + 'dog'::INT;  
ERROR 3681: Invalid input syntax for integer: "cat"
```

The following statement would also roll back because Vertica Analytics Platform can't parse the row to the requested data type:

```
=> COPY t (a FILLER VARCHAR, i AS a::INT) FROM STDIN;
```

However, in the following COPY statement, Vertica Analytics Platform rejects only the offending row without rolling back the statement. Instead of evaluating the 'cat' row as a VARCHAR type, it tried to parse 'cat' directly as an INTEGER.

```
=> COPY t (a FILLER INT, i AS a) FROM STDIN;
```

In the case of the above statement, the load parser (unlike the expression evaluator) rejects the row if it contains a field that can't be parsed to the requested data type.

Saving Load Exceptions (EXCEPTIONS)

COPY exceptions consist of informational messages describing why a row of data could not be parsed. The optional EXCEPTIONS parameter lets you specify a file to which COPY writes exceptions. If you do not use this parameter, COPY saves exception files to the following default location:

catalog_dir/CopyErrorLogs/tablename-filename-of-source-copy-from-exceptions

| | |
|--------------------------------------|--|
| <i>catalog_dir</i> | The database catalog files directory |
| <i>tablename-filename- of-source</i> | The names of the table and data file |
| <i>-copy-from-exceptions</i> | The file suffix appended to the table and source file name |

Note: Using the REJECTED DATA parameter with the AS TABLE clause is mutually exclusive with specifying a load Exceptions file. Since the exceptions messages are included in the rejected data table, COPY does not permit both options and displays this error if you try to use them:

```
ERROR 0: Cannot specify both an exceptions file and a rejected table in the same statement
```

The optional EXCEPTIONS parameter lets you specify a file of your choice to which COPY writes load exceptions. The EXCEPTIONS file indicates the input line number and the reason for each data record exception in this format:

```
COPY: Input record number in <pathofinputfile> has been rejected (reason). Please see pathrejectfile, record recordnum for the rejected record.
```

If copying from STDIN, the *filename-of-source* is STDIN.

Note: You can use specific rejected data and exceptions files with one or more of the files you are loading. Separate consecutive rejected data and exception file names with a comma (,) in the COPY statement.

You must specify a filename in the path to load multiple input files. Keep in mind that long table names combined with long data file names can exceed the operating system's maximum length (typically 255 characters). To work around file names exceeding the maximum length, use a path for the exceptions file that differs from the default path; for example, `\tmp\.`

For all data loads (except for COPY LOCAL), COPY behaves as follows:

| No Exceptions file specified... | Exceptions file specified... |
|--|---|
| For one data source file (<i>pathToData</i> or STDIN), all information stored as one file in the default directory. | For one data file, the path is treated as a file, and COPY stores all information in this file. If the path is not a file, COPY returns an error. |
| For multiple data files, all information stored as separate files, one for each data file in default directory. | For multiple data files, the path is treated as a directory and COPY stores all information in separate files, one for each data file. If path is not a directory, COPY returns an error. |
| | Exceptions files are not stored on the initiator node. |
| | You can specify only one path per node. If you specify more than one path per node, COPY returns an error. |

Saving Load Rejections (REJECTED DATA)

COPY rejections are the data rows from the load file that caused a parser exception and did not load.

The optional REJECTED DATA parameter lets you specify either a file or table to which COPY writes rejected data records. If you omit this parameter, COPY saves rejected data files to the following location without saving a table:

`catalog_dir/CopyErrorLogs/tablename-filename-of-source-copy-from-rejections`

| | |
|---|--|
| <code>catalog_dir</code> | The database catalog files directory |
| <code>tablename-filename-of-source</code> | The names of the table and data file |
| <code>-copy-from-rejections</code> | The file suffix appended to the table and source file name |

Once a rejections file exists, you can review its contents to resolve any load problems and reload the data. If you save rejected data to a table, you can query the table to see both exceptions and the rejected data.

If copying from STDIN, the *filename of source* is STDIN.

Note: You can use specific rejected data and exceptions files with one or more of the files you are loading. Separate consecutive rejected data and exception file names with a comma (,) in the COPY statement. Do not use the ON ANY NODE option with rejected data and exceptions files, because ON ANY NODE is applicable only to the load file.

When you load multiple input files, you must specify a file name in the path. Keep in mind that long input file names, combined with rejected data file names, can exceed the operating system's maximum length (typically 255 characters). To work around file names exceeding the maximum length, use a path for the rejected data file that differs from the default path; for example, `\tmp\.`

For all data loads (except for COPY LOCAL), COPY behaves as follows:

| No rejected data file specified... | Rejected data file specified... |
|--|---|
| For one data source file (<i>pathToData</i> or STDIN), all information stored as one file in the default directory. | For one data file, the path is interpreted as a file, and COPY stores all rejected data in this file. If the path is not a file, COPY returns an error. |
| For multiple data files, all information stored as separate files, one for each data file in the default directory. | For multiple data files, the path is treated as a directory and COPY stores all information in separate files, one for each data file. If path is not a directory, COPY returns an error. |
| | Rejected data files are not shipped to the initiator node. |
| | Only one path per node is accepted. If more than one is provided, COPY returns an error. |

Saving Rejected Data to a Table

Using the REJECTED DATA parameter with the AS TABLE clause lets you specify a table in which to save the data. The capability to save rejected data is on by default. If you want to keep saving rejected data to a file, do not use the AS TABLE clause.

When you use the AS TABLE clause, HP Vertica creates a new table if one does not exist. If no parsing rejections occur during a load, the table exists but is empty. The next time you load data, HP Vertica appends any rejected rows to the existing table.

The load rejection tables are a special type of table with the following capabilities and limitations:

- Support SELECT statements
- Can use DROP TABLE
- Cannot be created outside of a COPY statement

- Do not support **DML** and **DDL** activities
- Are not **K-safe**

To make the data in a rejected table K-safe, you can do one of the following:

- Write a `CREATE TABLE . . AS` statement, such as this example:

```
CREATE TABLE new_table AS SELECT * FROM rejected_table;
```

- Create a table to store rejected records, and run `INSERT . . SELECT` operations into the new table

If you use `COPY NO COMMIT`

If you include the `NO COMMIT` and `REJECTED DATA AS TABLE` clauses in your `COPY` statement and the `reject_table` does not already exist, Vertica Analytics Platform saves the rejected-data table as a `LOCAL TEMP` table and returns a message that a `LOCAL TEMP` table is being created.

Rejected-data tables are especially useful for Extract-Load-Transform workflows (where you'll likely use temporary tables more frequently) by letting you quickly load data and identify which records failed to load due to a parsing error. If you load data into a temporary table that you created using the `ON COMMIT DELETE` clause, the `COPY` operation won't commit, even if you specify the `ON COMMIT DELETE` clause.

Rejection Records for Table Files

In the current implementation, the rejected records used to populate a table are also saved to a file in the default rejected data file directory:

```
catalog_dir/CopyErrorLogs/CopyRejectedRecords
```

The file contents differ from the rejected data files that `COPY` saves by default. The rejected records for table files contain rejected data and the reason for the rejection (exceptions), along with other data columns, described next. HP Vertica suggests that you periodically log in to each server and drop the rejections tables that you no longer need.

Querying a Rejection Records Table

To use the `AS TABLE` clause:

1. Create a sample table in which to load data:

```
db=> CREATE TABLE loader(a INT)
CREATE TABLE
```

2. Use the `COPY` statement to load values, and save any rejected rows to a table called `loader_rejects`:

```

dbs=> COPY loader FROM STDIN REJECTED DATA AS table "loader_rejects";
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 2
>> 3
>> 4
>> a
>> b
>> c
>> \.

```

3. Query the loader table after loading data into it directly. Notice only four (4) values exist:

```

dbs=> SELECT * FROM loader;
Value
-----
1
2
3
4
(4 rows)

```

4. Query the loader_rejects table to see its column rows. The following example lists one record from the table (although more exist):

```

dbs=> \x
Expanded display is on.
dbs=> SELECT * FROM loader_rejects;
-[ RECORD 1 ]-----+-----
node_name          | v_dbs_node0001
file_name          |
session_id        | engvmqa2401-12617:0x1cb0
statement_id      | 1
batch_number      | 0
row_number        | 5
rejected_data     | a
rejected_data_orig_length | 1
rejected_reason   | Invalid integer format 'a' for column 1 (a)
-[ RECORD 2 ]-----+-----

```

The rejection data table has the following columns:

| Column | Data Type | Description |
|-----------|-----------|--|
| node_name | VARCHAR | The name of the HP Vertica node on which the input load file was located. |
| file_name | VARCHAR | The name of the file being loaded, which applies if you loaded a file (as opposed to using STDIN). |

| | | |
|---------------------------|--------------|--|
| session_id | VARCHAR | The session ID number in which the COPY statement occurred. |
| transaction_id | INTEGER | Identifier for the transaction within the session, if any; otherwise NULL. |
| statement_id | INTEGER | The unique identification number of the statement within the transaction that included the rejected data. Tip: You can use the session_id, transaction_id, and statement_id columns to create joins with many system tables. For example, if you join against the QUERY_REQUESTS table using those three columns, the QUERY_REQUESTS.REQUEST column contains the actual COPY statement (as a string) used to load this data. |
| batch_number | INTEGER | INTERNAL USE. Represents which batch (chunk) the data comes from. |
| row_number | INTEGER | The rejected row number from the input file. |
| rejected_data | LONG VARCHAR | The actual load data. |
| rejected_data_orig_length | INTEGER | The length of the rejected data. |
| rejected_reason | VARCHAR | The error that caused the rejected row. This column returns the same message that would exist in the load exceptions file, if you saved exceptions to a file, rather than to a table. |

Exporting the Rejected Records Table

You can export the contents of the rejected_data column to a file, correct the rejected rows, and load the corrected rows from the updated file.

To export rejected records:

1. Create a sample table:

```
dbs=> create table t (i int);
CREATE TABLE
```

2. Copy data directly into the table, using a table to store rejected data:

```
dbsh=> copy t from stdin rejected data as table "t_rejects";
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 2
>> 3
>> 4
>> a
>> b
>> c
>> \.
```

3. Show only tuples and set the output format:

```
dbsh=> \t
Showing only tuples.
dbsh=> \a
Output format is unaligned.
```

4. Output to a file (rejected.txt):

```
dbsh=> \o rejected.txt
dbsh=> select rejected_data from t_rejects;
dbsh=> \o
```

5. Use the cat command on the saved file:

```
dbsh=> \! cat rejected.txt
a
b
c
dbsh=>
```

After a file exists, you can fix load errors and use the corrected file as load input to the COPY statement.

COPY Rejected Data and Exception Files

When executing a multi-node COPY statement, each node processes part of the load data. If the load succeeds, all parser rejections that occur during the node's load processing are written to that node's specific rejected data and exceptions files. If the load fails, the file contents can be incomplete, or empty.

Both rejected data and exceptions files are saved and stored on a per-node basis. This example uses multiple files as COPY inputs. Since the statement does not include either the REJECTED DATA or EXCEPTIONS parameters, rejected data and exceptions files are written to the default location, the database catalog subdirectory, CopyErrorLogs, on each node:

```
\set dir `pwd`/data/ \set remote_dir /vertica/test_dev/tmp_ms/  
\set file1 ''':dir'C1_large_tbl.dat''  
\set file2 ''':dir'C2_large_tbl.dat''  
\set file3 ''':remote_dir'C3_large_tbl.dat''  
\set file4 ''':remote_dir'C4_large_tbl.dat''  
  
COPY large_tbl FROM :file1 ON site01,           :file2 ON site01,  
                  :file3 ON site02,  
                  :file4 ON site02  
DELIMITER '|';
```

Note: Always use the COPY statement REJECTED DATA and EXCEPTIONS parameters to save load rejections. Using the RETURNREJECTED parameter is supported only for internal use by the JDBC and ODBC drivers. HP Vertica's internal-use options can change without notice.

Specifying Rejected Data and Exceptions Files

The optional COPY REJECTED DATA and EXCEPTIONS parameters 'path' element lets you specify a non-default path in which to store the files.

If *path* resolves to a storage location, and the user invoking COPY is not a superuser, these are the required permissions:

- The storage location must have been created (or altered) with the USER option (see [ADD_LOCATION](#) and [ALTER_LOCATION_USE](#))
- The user must already have been granted READ access to the storage location where the file(s) exist, as described in [GRANT \(Storage Location\)](#)

Both parameters also have an optional ON *nodename* clause that uses the specified path:

```
...[ EXCEPTIONS 'path' [ ON nodename ] [, ...] ]...[ REJECTED DATA 'path' [ ON nodename ]  
[, ...] ]
```

While '*path*' specifies the location of the rejected data and exceptions files (with their corresponding parameters), the optional ON *nodename* clause moves any existing rejected data and exception files on the node to the specified path on the same node.

Saving Rejected Data and Exceptions Files to a Single Server

The COPY statement does not have a facility to merge exception and rejected data files after COPY processing is complete. To see the contents of exception and rejected data files requires logging on and viewing each node's specific files.

Note: If you want to save all exceptions and rejected data files on a network host, be sure to give each node's files unique names, so different cluster nodes do not overwrite other nodes'

files. For instance, if you set up a server with two directories, such as `/vertica/exceptions` and `/vertica/rejections`, be sure the corresponding file names for each HP Vertica cluster node identify each node, such as `node01_exceptions.txt` and `node02_exceptions.txt`, and so on. In this way, each cluster node's files will reside in one directory, and be easily distinguishable.

Using VSQL Variables for Rejected Data and Exceptions Files

This example uses `vsql` variables to specify the path and file names to use with the exceptions and rejected data parameters (`except_s1` and `reject_s1`). The `COPY` statement specifies a single input file (`large_tbl`) on the initiator node:

```
\set dir `pwd`/data/ \set file1 ''':dir'C1_large_tbl.dat''
\set except_s1 ''':dir'exceptions''
\set reject_s1 ''':dir'rejections''

COPY large_tbl FROM :file1 ON site01DELIMITER '|'
REJECTED DATA :reject_s1 ON site01
EXCEPTIONS :except_s1 ON site01;
```

This example uses variables to specify exception and rejected data files (`except_s2` and `reject_s2`) on a remote node. The `COPY` statement consists of a single input file on a remote node (`site02`):

```
\set remote_dir /vertica/test_dev/tmp_ms/\set except_s2 ''':remote_dir'exceptions''
\set reject_s2 ''':remote_dir'rejections''

COPY large_tbl FROM :file1 ON site02DELIMITER '|'
REJECTED DATA :reject_s2 ON site02
EXCEPTIONS :except_s2 ON site02;
```

This example uses variables to specify that the exception and rejected data files are on a remote node (indicated by `:remote_dir`). The inputs to the `COPY` statement consist of multiple data files on two nodes (`site01` and `site02`). The exceptions and rejected data options use `ON nodename` clause with the `vsql` variables to indicate where the files reside (`site01` and `site02`):

```
\set dir `pwd`/data/ \set remote_dir /vertica/test_dev/tmp_ms/
\set except_s1 ''':dir''''
\set reject_s1 ''':dir''''
\set except_s2 ''':remote_dir''''
\set reject_s2 ''':remote_dir''''

COPY large_tbl FROM :file1 ON site01,           :file2 ON site01,
                  :file3 ON site02,
                  :file4 ON site02
DELIMITER '|'
REJECTED DATA :reject_s1 ON site01, :reject_s2 ON site02
EXCEPTIONS :except_s1 ON site01, :except_s2 ON site02;
```

COPY LOCAL Rejection and Exception Files

Invoking COPY LOCAL (or COPY LOCAL FROM STDIN) does not automatically create rejected data and exceptions files. This behavior differs from using COPY, which saves both files automatically, regardless of whether you use the optional REJECTED DATA and EXCEPTIONS parameters to specify either file explicitly.

Use the REJECTED DATA and EXCEPTIONS parameters with COPY LOCAL and COPY LOCAL FROM STDIN to save the corresponding output files on the client. If you do *not* use these options, rejected data parsing events (and the exceptions that describe them) are not retained, even if they occur.

You can load multiple input files using COPY LOCAL (or COPY LOCAL FROM STDIN). If you also use the REJECTED DATA and EXCEPTIONS options, the statement writes rejected rows and exceptions and to separate files. The respective files contain all rejected rows and corresponding exceptions, respectively, regardless of how many input files were loaded.

Note: Because COPY LOCAL (and COPY LOCAL FROM STDIN) must write any rejected rows and exceptions to the client, you cannot use the [ON nodename] clause with either the rejected data or exceptions options.

Specifying Rejected Data and Exceptions Files

To save any rejected data and their exceptions to files:

1. In the COPY LOCAL (and COPY LOCAL FROM STDIN) statement, use the REJECTED DATA 'path' and the EXCEPTIONS 'path' parameters, respectively.
2. Specify two different file names for the two options. You cannot use one file for both the REJECTED DATA and the EXCEPTIONS.
3. When you invoke COPY LOCAL or COPY LOCAL FROM STDIN, the files you specify need not pre-exist. If they do, COPY LOCAL must be able to overwrite them.

You can specify the path and file names with vsql variables:

```
\set rejected ../except_reject/copyLocal.rejected\set exceptions ../except_reject/copyLocal.exceptions
```

Note: COPY LOCAL does not support storing rejected data in a table, as you can when using the COPY statement.

When you use the COPY LOCAL or COPY LOCAL FROM STDIN statement, specify the variable names for the files with their corresponding parameters:

```
COPY large_tbl FROM LOCAL rejected data :rejected exceptions :exceptions;
```

```
COPY large_tbl FROM LOCAL STDIN rejected data :rejected exceptions :exceptions;
```

Referential Integrity Load Violation

HP Vertica checks for constraint violations when queries are executed, not when loading data.

If you have a pre-joined projection defined on the table being loaded, HP Vertica checks for constraint violations (duplicate primary keys or non-existent foreign keys) during the join operation and reports errors. If there are no pre-joined projections, HP Vertica performs no such checks.

To avoid constraint violations, you can load data without committing it and then use the `ANALYZE_CONSTRAINTS` function to perform a post-load check of your data. If the function finds constraint violations, you can roll back the bulk load because you have not committed it.

See Also

- [Detecting Constraint Violations](#)
- [COPY](#)
- [ANALYZE_CONSTRAINTS](#)

Trickle Loading Data

Once you have a working database and have bulk loaded your initial data, you can use trickle loading to load additional data on an ongoing basis. By default, HP Vertica uses the transaction isolation level of `READ COMMITTED`, which allows users to see the most recently committed data without holding any locks. This allows new data to be loaded while concurrent queries are running.

See [Change Transaction Isolation Levels](#).

Using INSERT, UPDATE, and DELETE

The SQL data manipulation language (DML) commands [INSERT](#), [UPDATE](#), and [DELETE](#) perform the same functions that they do in any ACID compliant database. The `INSERT` statement is typically used to load individual rows into physical memory or load a table using `INSERT AS SELECT`. `UPDATE` and `DELETE` are used to modify the data.

You can intermix the `INSERT`, `UPDATE`, and `DELETE` commands. HP Vertica follows the SQL-92 transaction model. In other words, you do not have to explicitly start a transaction but you must use a [COMMIT](#) or [ROLLBACK](#) command (or [COPY](#)) to end a transaction. Canceling a DML statement causes the effect of the statement to be rolled back.

HP Vertica differs from traditional databases in two ways:

- `DELETE` does not actually delete data from disk storage; it marks rows as deleted so that they can be found by historical queries.
- `UPDATE` writes two rows: one with new data and one marked for deletion.

Like `COPY`, by default, `INSERT`, `UPDATE` and `DELETE` commands write the data to the WOS and on overflow write to the ROS. For large `INSERTS` or `UPDATES`, you can use the `DIRECT` keyword to force HP Vertica to write rows directly to the ROS. Loading large number of rows as single row inserts are not recommended for performance reasons. Use `COPY` instead.

WOS Overflow

The WOS exists to allow HP Vertica to efficiently batch small loads into larger ones for I/O purposes. Loading to the WOS is fast because the work of sorting, encoding, and writing to disk is deferred and performed in the background by the Tuple Mover's moveout process. Since the WOS has a finite amount of available space, it can fill up and force HP Vertica to spill small loads directly to disk. While no data is lost or rejected when the WOS gets full, it can result in wasted I/O bandwidth. Thus, follow the [Tuning the Tuple Mover](#) guidelines to avoid WOS overflow.

Copying and Exporting Data

HP Vertica can easily import data from and export data to other HP Vertica databases. Importing and exporting data is useful for common tasks such as moving data back and forth between a development or test database and a production database, or between databases that have different purposes but need to share data on a regular basis.

Moving Data Directly Between Databases

Two statements move data to and from another HP Vertica database:

- [COPY FROM VERTICA](#)
- [EXPORT TO VERTICA](#)

To execute either of these statements requires first creating a connection to the other HP Vertica database.

Creating SQL Scripts to Export Data

Three functions return a SQL script you can use to export database objects to recreate elsewhere:

- [EXPORT_CATALOG](#)
- [EXPORT_OBJECTS](#)
- [EXPORT_TABLES](#)

While copying and exporting data is similar to [Backing Up and Restoring the Database](#), you should use them for different purposes, outlined below:

| Task | Backup and Restore | COPY and EXPORT Statements |
|--|--------------------|----------------------------|
| Back up or restore an entire database, or incremental changes | YES | NO |
| Manage database objects (a single table or selected table rows) | YES | YES |
| Use external locations to back up and restore your database | YES | NO |
| Use direct connections between two databases | NO | YES |
| Use external shell scripts to back up and restore your database | YES | NO |
| Use SQL commands to incorporate copy and export tasks into DB operations | NO | YES |

The following sections explain how you import and export data between HP Vertica databases.

When importing from or exporting to an HP Vertica database, you can connect only to a database that uses trusted- (username-only) or password-based authentication, as described in [Implementing Security](#). Neither LDAP nor SSL authentication is supported.

Exporting Data

You can export data from an earlier HP Vertica release, as long as the earlier release is a version of the last major release. For instance, for Version 6.x, you can export data from any version of 5.x, but not from 4.x.

You can export a table, specific columns in a table, or the results of a [SELECT](#) statement to another HP Vertica database. The table in the target database receiving the exported data must already exist, and have columns that match (or can be coerced into) the data types of the columns you are exporting.

Exported data is always written in AUTO mode.

When loading data using AUTO mode, HP Vertica inserts the data first into the **WOS**. If the WOS is full, then HP Vertica inserts the data directly into **ROS**. See the [COPY](#) statement for more details.

Exporting data fails if either side of the connection is a single-node cluster installed to localhost or you do not specify a host name or IP address.

Exporting is a three-step process:

1. Use the [CONNECT](#) SQL statement to connect to the target database that will receive your exported data.
2. Use the [EXPORT](#) SQL statement to export the data. If you want to export multiple tables or the results of multiple [SELECT](#) statements, you need to use multiple [EXPORT](#) statements. All statements will use the same connection to the target database.
3. When you are finished exporting data, use the [DISCONNECT](#) SQL statement to disconnect from the target database.

See the entries for [CONNECT](#), [EXPORT](#), and [DISCONNECT](#) statements in the SQL Reference Manual for syntax details.

Exporting Identity Columns

When you use the [EXPORT TO VERTICA](#) statement, HP Vertica exports Identity (and Auto-increment) columns as they are defined in the source data. The Identity column value does not increment automatically, and requires that you use [ALTER SEQUENCE](#) to make updates.

Export Identity (and Auto-increment) columns as follows:

- If source and destination tables have an Identity column, you do not need to list them.
- If source has an Identity column, but not the destination, specify both the source and destination columns.

Note: In earlier releases, Identity columns were ignored. Now, failure to list which Identity columns to export can cause an error, because the Identity column is not ignored and will be interpreted as missing in the destination table.

The default behavior for EXPORT TO VERTICA is to let you export Identity columns by specifying them directly in the source table. To disable this behavior globally, set the CopyFromVerticaWithIdentity configuration parameter, described in [Configuration Parameters](#).

Examples of Exporting Data

The following example demonstrates using the three-step process listed above to export data.

First, open the connection to the other database, then perform a simple export of an entire table to an identical table in the target database.

```
=> CONNECT TO VERTICA testdb USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> EXPORT TO VERTICA testdb.customer_dimension FROM customer_dimension;
Rows Exported
-----
                23416
(1 row)
```

The following statement demonstrates exporting a portion of a table using a simple [SELECT](#) statement.

```
=> EXPORT TO VERTICA testdb.ma_customers AS SELECT customer_key, customer_name, annual_in
come
-> FROM customer_dimension WHERE customer_state = 'MA';
Rows Exported
-----
                3429
(1 row)
```

This statement exports several columns from one table to several different columns in the target database table using column lists. Remember that when supplying both a source and destination column list, the number of columns must match.

```
=> EXPORT TO VERTICA testdb.people (name, gender, age) FROM customer_dimension
-> (customer_name, customer_gender, customer_age);
Rows Exported
-----
                23416
(1 row)
```

You can export tables (or columns) containing Identity and Auto-increment values, but the sequence values are not incremented automatically at their destination.

You can also use the EXPORT TO VERTICA statement with a [SELECT AT EPOCH LATEST](#) expression to include data from the latest committed DML transaction.

Disconnect from the database when the export is complete:

```
=> DISCONNECT testdb;  
DISCONNECT
```

Note: Closing your session also closes the database connection. However, it is a good practice to explicitly close the connection to the other database, both to free up resources and to prevent issues with other SQL scripts you may run in your session. Always closing the connection prevents potential errors if you run a script in the same session that attempts to open a connection to the same database, since each session can only have one connection to a particular database at a time.

Copying Data

You can import a table or specific columns in a table from another HP Vertica database. The table receiving the copied data must already exist, and have columns that match (or can be coerced into) the data types of the columns you are copying from the other database. You can import data from an earlier HP Vertica release, as long as the earlier release is a version of the last major release. For instance, for Version 6.x, you can import data from any version of 5.x, but not from 4.x.

Importing and exporting data fails if either side of the connection is a single-node cluster installed to localhost, or you do not specify a host name or IP address.

Importing is a three-step process:

1. Use the [CONNECT](#) SQL statement to connect to the source database containing the data you want to import.
2. Use the [COPY FROM VERTICA](#) SQL statement to import the data. If you want to import multiple tables, you need to use multiple [COPY FROM VERTICA](#) statements. They all use the same connection to the source database.
3. When you are finished importing data, use the [DISCONNECT](#) SQL statement to disconnect from the source database.

See the entries for [CONNECT](#), [COPY FROM VERTICA](#), and [DISCONNECT](#) statements in the SQL Reference Manual for syntax details.

Importing Identity Columns

You can import Identity (and Auto-increment) columns as follows:

- If source and destination tables have an Identity column, you do not need to list them.
- If source has an Identity column, but not the destination, specify both the source and destination columns.

Note: In earlier releases, Identity columns were ignored. Now, failure to list which Identity columns to export can cause an error, because the Identity column is not ignored and will be

interpreted as missing in the destination table.

After importing the columns, the Identity column values do not increment automatically. Use [ALTER SEQUENCE](#) to make updates.

The default behavior for this statement is to import Identity (and Auto-increment) columns by specifying them directly in the source table. To disable this behavior globally, set the `CopyFromVerticaWithIdentity` configuration parameter, described in [Configuration Parameters](#).

Examples

This example demonstrates connecting to another database, copying the contents of an entire table from the source database to an identically-defined table in the current database directly into **ROS**, and then closing the connection.

```
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> COPY customer_dimension FROM VERTICA vmart.customer_dimension DIRECT;
  Rows Loaded
-----
          500000
(1 row)
=> DISCONNECT vmart;
DISCONNECT
```

This example demonstrates copying several columns from a table in the source database into a table in the local database.

```
=> CONNECT TO VERTICA vmart USER dbadmin PASSWORD '' ON 'VertTest01',5433;
CONNECT
=> COPY people (name, gender, age) FROM VERTICA
-> vmart.customer_dimension (customer_name, customer_gender,
-> customer_age);
  Rows Loaded
-----
          500000
(1 row)
=> DISCONNECT vmart;
DISCONNECT
```

You can copy tables (or columns) containing Identity and Auto-increment values, but the sequence values are not incremented automatically at their destination.

Using Public and Private IP Networks

In many configurations, HP Vertica cluster hosts use two network IP addresses as follows:

- A private address for communication between the cluster hosts.
- A public IP address for communication with for client connections.

By default, importing from and exporting to another HP Vertica database uses the private network.

To use the public network address for copy and export activities, configure the system to use the public network to support exporting to or importing from another HP Vertica cluster:

- [Identify the Public Network to HP Vertica](#)
- [Identify the Database or Nodes Used for Import/Export](#)

Identify the Public Network to HP Vertica

To be able to import to or export from a public network, HP Vertica needs to be aware of the IP addresses of the nodes or clusters on the public network that will be used for import/export activities. Your public network might be configured in either of these ways:

- Public network IP addresses reside on the same subnet (create a subnet)
- Public network IP addresses are on multiple subnets (create a network interface)

To identify public network IP addresses residing on the same subnet:

- Use the [CREATE SUBNET](#) statement provide your subnet with a name and to identify the subnet routing prefix.

To identify public network IP addresses residing on multiple subnets:

- Use the [CREATE NETWORK INTERFACE](#) statement to configure import/export from specific nodes in the HP Vertica cluster.

After you've identified the subnet or network interface to be used for import/export, you must [Identify the Database Or Nodes Used For Import/Export](#).

See Also

- [CREATE SUBNET](#)
- [ALTER SUBNET](#)
- [DROP SUBNET](#)
- [CREATE NETWORK INTERFACE](#)

- [ALTER NETWORK INTERFACE](#)
- [DROP NETWORK INTERFACE](#)

Identify the Database or Nodes Used for Import/Export

Once you've identified the public network to HP Vertica, you can configure databases and nodes to use the public network for import/export. You can configure by:

- Specifying a subnet for the database.
- Specifying a network interface for each node in the database.

To Configure a Database to Import/Export on the Public Network-

- Use the [ALTER DATABASE](#) statement to specify the subnet name of the public network. When you do so, all nodes in the database will automatically use the network interface on the subnet for import/export operations.

To Configure Each Individual Node to Import/Export on a Public Network:

- Use the [ALTER NODE](#) statement to specify the network interface of the public network on each individual node.

See Also

- [ALTER DATABASE](#)
- [CREATE SUBNET](#)
- [CREATE NETWORK INTERFACE](#)
- [V_MONITOR_NETWORK INTERFACES](#)

Using EXPORT Functions

HP Vertica provides several `EXPORT_` functions that let you recreate a database, or specific schemas and tables, in a target database. For example, you can use the `EXPORT_` functions to transfer some or all of the designs and objects you create in a development or test environment to a production database.

The `EXPORT_` functions create SQL scripts that you can run to generate the exported database designs or objects. These functions serve different purposes to the export statements, `COPY FROM VERTICA` (pull data) and `EXPORT TO VERTICA` (push data). These statements transfer data directly from source to target database across a network connection between both. They are dynamic actions and do not generate SQL scripts.

The `EXPORT_` functions appear in the following table. Depending on what you need to export, you can use one or more of the functions. `EXPORT_CATALOG` creates the most comprehensive SQL script, while `EXPORT_TABLES` and `EXPORT_OBJECTS` are subsets of that function to narrow the export scope.

| Use this function... | To recreate... |
|-----------------------------|---|
| <code>EXPORT_CATALOG</code> | These catalog items: <ul style="list-style-type: none">• An existing schema design, tables, projections, constraints, and views• The Database Designer-created schema design, tables, projections, constraints, and views• A design on a different cluster. |
| <code>EXPORT_TABLES</code> | Non-virtual objects up to, and including, the schema of one or more tables. |
| <code>EXPORT_OBJECTS</code> | Catalog objects in order dependency for replication. |

The designs and object definitions that the script creates depend on the `EXPORT_` function scope you specify. The following sections give examples of the commands and output for each function and the scopes it supports.

Saving Scripts for Export Functions

All of the examples in this section were generated using the standard HP Vertica VMART database, with some additional test objects and tables. One output directory was created for all SQL scripts that the functions created:

```
/home/dbadmin/xtest
```

If you specify the destination argument as an empty string (' '), the function writes the export results to STDOUT.

Note: A superuser can export all available database output to a file with the EXPORT_ functions. For a non-superuser, the EXPORT_ functions generate a script containing only the objects to which the user has access.

Exporting the Catalog

Exporting the catalog is useful to quickly move a database design to another cluster. The [EXPORT_CATALOG](#) function generates a SQL script to run on a different cluster to replicate the physical schema design of the source database. You choose what to export by specifying the export scope:

| To export... | Enter this scope... |
|---|--|
| Schemas, tables, constraints, views, and projections. | DESIGN (This is the default scope.) |
| All design objects and system objects created in Database Designer, such as design contexts and their tables. | DESIGN ALL |
| All tables, constraints, and projections. | TABLES |

Function Summary

Here is the function syntax, described in [EXPORT_CATALOG](#) in the SQL Reference Manual:

```
EXPORT_CATALOG ( [ 'destination' ] , [ 'scope' ] )
```

Exporting All Catalog Objects

Use the DESIGN scope to export all design elements of a source database in order dependency. This scope exports all catalog objects in their **OID** (unique object ID) order, including schemas, tables, constraints, views, and projections. This is the most comprehensive export scope, without the Database Designer elements, if they exist.

Note: The result of this function yields the same SQL script as [EXPORT_OBJECTS](#) used with an empty string (' ') as its scope.

```
VMart=> select export_catalog('/home/dbadmin/xtest/sql_cat_design.sql', 'DESIGN');
          export_catalog
-----
Catalog data exported successfully
(1 row)
```

The SQL script includes the following types of statements, each needed to provision a new database:

- CREATE SCHEMA
- CREATE TABLE
- CREATE VIEW
- CREATE SEQUENCE
- CREATE PROJECTION (with ORDER BY and SEGMENTED BY)
- ALTER TABLE (to add constraints)
- PARTITION BY

Projection Considerations

If a projection to export was created with no ORDER BY clause, the SQL script reflects the default behavior for projections. <DB_SHORT> implicitly creates projections using a sort order based on the SELECT columns in the projection definition. The EXPORT_CATALOG script reflects this behavior.

The EXPORT_CATALOG script is portable as long as all projections were generated using UNSEGMENTED ALL NODES or SEGMENTED ALL NODES.

Exporting Database Designer Schema and Designs

Use the DESIGN ALL scope to generate a script to recreate all design elements of a source database and the design and system objects that were created by the Database Designer:

```
VMart=> select export_catalog('/home/dbadmin/xtest/sql_cat_design_all.sql', 'DESIGN_ALL');
          export_catalog
-----
Catalog data exported successfully
(1 row)
```

Exporting Table Objects

Use the TABLES scope to generate a script to recreate all schemas tables, constraints, and sequences:

```
VMart=> select export_catalog('/home/dbadmin/xtest/sql_cat_tables.sql', 'TABLES');
          export_catalog
-----
Catalog data exported successfully
(1 row)
```

The SQL script includes the following types of statements:

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)
- CREATE SEQUENCE

See Also

- [EXPORT_CATALOG](#)
- [EXPORT_OBJECTS](#)
- [EXPORT_TABLES](#)
- [Exporting Tables](#)
- [Exporting Objects](#)

Exporting Tables

Use the EXPORT_TABLES function to recreate one or more tables, and related objects, on a different cluster. Specify one of the following options to determine the scope:

| To export... | Use this scope... |
|--|---|
| All non-virtual objects to which the user has access, including constraints. | An empty string (' ') |
| One or more named objects, such as tables or sequences in one or more schemas. You can optionally qualify the schema with a database prefix, myvertica.myschema.newtable. | A comma-delimited list of items: 'myschema.newtable, yourschema.oldtable' |
| A named database object in the current search path. You can specify a schema, table, or sequence. If the object is a schema, the script includes non-virtual objects to which the user has access. | A single object, 'myschema' |

The SQL script includes only the non-virtual objects to which the current user has access.

Note: You cannot export a view with this function, even if a list includes the view relations. Specifying a view name will not issue a warning, but the view will not exist in the SQL script. Use EXPORT_OBJECTS.

Function Syntax

```
EXPORT_TABLES ( [ 'destination' ] , [ 'scope' ] )
```

For more information, see [EXPORT_TABLES](#) in the SQL Reference Manual.

Exporting All Tables and Related Objects

Specify an empty string (' ') for the scope to export all tables and their related objects.

```
VMart=> select export_tables('/home/dbadmin/xtest/sql_tables_empty.sql','');
          export_tables
-----
Catalog data exported successfully
(1 row)
```

The SQL script includes the following types of statements, depending on what is required to recreate the tables and any related objects (such as sequences):

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)
- CREATE SEQUENCE
- PARTITION BY

Exporting a List Tables

Use EXPORT_TABLE with a comma-separated list of objects, including tables, views, or schemas:

```
VMart=> select export_tables('/home/dbadmin/xtest/sql_tables_del.sql','public.student, pu
          blic.test7');
          export_tables
-----
Catalog data exported successfully
(1 row)
```

The SQL script includes the following types of statements, depending on what is required to create the list of objects:

- CREATE SCHEMA
- CREATE TABLE

- ALTER TABLE (to add constraints)
- CREATE SEQUENCE

Exporting a Single Table or Object

Use the EXPORT_TABLES function to export one or more database table objects.

This example exports a named sequence, my_seq, qualifying the sequence with the schema name (public):

```
VMart=> select export_tables('/home/dbadmin/xtest/export_one_sequence.sql', 'public.my_seq');
          export_tables
-----
Catalog data exported successfully
(1 row)
```

Following are the contents of the export_one_sequence.sql output file using a more command:

```
[dbadmin@node01 xtest]$ more export_one_sequence.sql
CREATE SEQUENCE public.my_seq ;
```

Exporting Objects

Use EXPORT_OBJECTS function to recreate the exported objects. Specify one of the following options to determine the scope:

| To export... | Use this scope... |
|--|---|
| All non-virtual objects to which the user has access, including constraints. | An empty string ('') |
| One or more named objects, such as tables or views in one or more schemas. You can optionally qualify the schema with a database prefix, myvertica.myschema.newtable. | A comma-delimited list of items: 'myschema.newtable, yourschema.oldtable' |
| A named database object in the current search path. You can specify a schema, table, or view. If the object is a schema, the script includes non-virtual objects to which the user has access. | A single object, 'myschema' |

The SQL script includes only the non-virtual objects to which the current user has access.

The EXPORT_OBJECTS function always attempts to recreate projection statements with the KSAFE clauses that existed in the original definitions, or with OFFSET clauses, if they did not.

Function Syntax

```
EXPORT_OBJECTS( [ 'destination' ] , [ 'scope' ] , [ 'ksafe' ] )
```

For more information, see [EXPORT_OBJECTS](#) in the SQL Reference Manual.

Exporting All Objects

Specify an empty string (' ') for the scope to export all non-virtual objects from the source database in order dependency. Running the generated SQL script on another cluster creates all referenced objects and their dependent objects.

By default, this function includes the `KSAFE` argument as `true`, so the script includes the `MARK_DESIGN_KSAFE` statement. Using this function is useful to run the generated SQL script in a new database so it will inherit the K-safety value of the original database.

Note: The result of this function yields the same SQL script as [EXPORT_CATALOG](#) with a `DESIGN` scope.

```
VMart=> select export_objects('/home/dbadmin/xtest/sql_objects_all.sql','','true');
          export_objects
-----
Catalog data exported successfully
(1 row)
```

The SQL script includes the following types of statements:

- CREATE SCHEMA
- CREATE TABLE
- CREATE VIEW
- CREATE SEQUENCE
- CREATE PROJECTION (with ORDER BY and SEGMENTED BY)
- ALTER TABLE (to add constraints)
- PARTITION BY

Here is a snippet from the start of the output SQL file, and the end, showing the `KSAFE` statement:

```
CREATE SCHEMA store;CREATE SCHEMA online_sales;
CREATE SEQUENCE public.my_seq ;
CREATE TABLE public.customer_dimension
(
```



```
customer_key int NOT NULL,  
customer_type varchar(16),  
customer_name varchar(256),  
customer_gender varchar(8),  
title varchar(8),  
household_id int,  
. . .  
);  
. . .  
SELECT MARK_DESIGN_KSAFE(0);
```

Exporting a List of Objects

Use a comma-separated list of objects as the function scope. The list can include one or more tables, sequences, and views in the same, or different schemas, depending on how you qualify the object name. For instance, specify a table from one schema, and a view from another (schema2.view1).

The SQL script includes the following types of statements, depending on what objects you include in the list:

- CREATE SCHEMA
- CREATE TABLE
- ALTER TABLE (to add constraints)
- CREATE VIEW
- CREATE SEQUENCE

If you specify a view without its dependencies, the function displays a WARNING. The SQL script includes a CREATE statement for the dependent object, but will be unable to create it without the necessary relations:

```
VMart=> select export_objects('nameObjectsList', 'test2, tt, my_seq, v2' );  
WARNING 0: View public.v2 depends on other relations  
export_objects  
-----  
Catalog data exported successfully  
(1 row)
```

This example includes the KSAFE argument explicitly:

```
VMart=> select export_objects('/home/dbadmin/xtest/sql_objects_table_view_KSAFE.sql', 'v1,  
test7', 'true');  
export_objects
```

```
-----  
Catalog data exported successfully  
(1 row)
```

Here are the contents of the output file of the example, showing the sample table `test7` and the `v1` view:

```
CREATE TABLE public.test7  
(  
  a int,  
  c int NOT NULL DEFAULT 4,  
  bb int  
);  
CREATE VIEW public.v1 AS  
  SELECT tt.a  
  FROM public.tt;  
SELECT MARK_DESIGN_KSAFE(0);
```

Exporting a Single Object

Specify a single database object as the function scope. The object can be a schema, table, sequence, or view. The function exports all non-virtual objects associated with the one you specify.

```
VMart=> select export_objects('/home/dbadmin/xtest/sql_objects_viewobject_KSAFE.sql', 'v1'  
, 'KSAFE');  
          export_objects  
-----  
Catalog data exported successfully  
(1 row)
```

The output file contains the `v1` view:

```
CREATE VIEW public.v1 AS  
  SELECT tt.a  
  FROM public.tt;  
SELECT MARK_DESIGN_KSAFE(0);
```

Bulk Deleting and Purging Data

HP Vertica provides multiple techniques to remove data from the database in bulk.

| Command | Description |
|-----------------------------------|--|
| DROP TABLE | Permanently removes a table and its definition. Optionally removes associated views and projections as well. |
| DELETE FROM TABLE | Marks rows with delete vectors and stores them so data can be rolled back to a previous epoch. The data must eventually be purged before the database can reclaim disk space. See Purging Deleted Data . |
| TRUNCATE TABLE | Removes all storage and history associated with a table. The table structure is preserved for future use. The results of this command cannot be rolled back. |
| DROP PARTITION | Removes one partition from a partitioned table. Each partition contains a related subset of data in the table. Partitioned data can be dropped efficiently, and provides query performance benefits. See Working with Table Partitions . |

The following table provides a quick reference for the different delete operations you can use. The "Saves History" column indicates whether data can be rolled back to an earlier epoch and queried at a later time.

| Syntax | Performance | Commits Tx | Saves History |
|--|-------------|------------|---------------|
| <code>DELETE FROM base_table</code> | Normal | No | Yes |
| <code>DELETE FROM temp_table</code> | High | No | No |
| <code>DELETE FROM base_table WHERE</code> | Normal | No | Yes |
| <code>DELETE FROM temp_table WHERE</code> | Normal | No | Yes |
| <code>DELETE FROM temp_table WHERE temp_table ON COMMIT PRESERVE ROWS</code> | Normal | No | Yes |
| <code>DELETE FROM temp_table WHERE temp_table ON COMMIT DELETE ROWS</code> | High | Yes | No |
| <code>DROP base_table</code> | High | Yes | No |
| <code>TRUNCATE base_table</code> | High | Yes | No |
| <code>TRUNCATE temp_table</code> | High | Yes | No |
| <code>DROP PARTITION</code> | High | Yes | No |

Choosing the Right Technique for Deleting Data

- To delete both table data and definitions and start from scratch, use the DROP TABLE [CASCADE] command.
- To drop data, while preserving table definitions so that you can quickly and easily reload data, use TRUNCATE TABLE. Note that unlike DELETE, TRUNCATE does not have to mark each row with delete vectors, so it runs much more quickly.
- To perform bulk delete operations on a regular basis, HP Vertica recommends using Partitioning.
- To perform occasional small deletes or updates with the option to roll back or review history, use DELETE FROM TABLE. See [Best Practices for DELETE and UPDATE](#).

For details on syntax and usage, see [DELETE](#), [DROP TABLE](#), [TRUNCATE TABLE](#), [CREATE TABLE](#) and [DROP_PARTITION](#) in the SQL Reference Manual.

Best Practices for DELETE and UPDATE

HP Vertica is optimized for query-intensive workloads, so DELETE and UPDATE queries might not achieve the same level of performance as other queries. DELETE and UPDATE operations go to the **WOS** by default, but if the data is sufficiently large and would not fit in memory, HP Vertica automatically switches to using the **ROS**. See [Using INSERT, UPDATE, and DELETE](#).

The topics that follow discuss best practices when using DELETE and UPDATE operations in HP Vertica.

Performance Considerations for DELETE and UPDATE Queries

To improve the performance of your DELETE and UPDATE queries, consider the following issues:

- **Query performance after large deletes**—A large number of (unpurged) deleted rows can negatively affect query performance.

To eliminate rows that have been deleted from the result, a query must do extra processing. If 10% or more of the total rows in a table have been deleted, the performance of a query on the table degrades. However, your experience may vary depending on the size of the table, the table definition, and the query. If a table has a large number of deleted rows, consider purging those rows to improve performance. For more information on purging, see [Purging Deleted Data](#).

- **Recovery performance**—Recovery is the action required for a cluster to restore K-safety after a crash. Large numbers of deleted records can degrade the performance of a recovery. To improve recovery performance, purge the deleted rows. For more information on purging, see [Purging Deleted Data](#).
- **Concurrency**—DELETE and UPDATE take exclusive locks on the table. Only one DELETE or UPDATE transaction on a table can be in progress at a time and only when no loads (or INSERTs) are in progress. DELETES and UPDATES on different tables can be run concurrently.
- **Pre-join projections**—Avoid pre-joining dimension tables that are frequently updated. DELETE and UPDATE operations on pre-join projections cascade to the fact table, causing large DELETE or UPDATE operations.

For detailed tips about improving DELETE and UPDATE performance, see [Optimizing DELETES and UPDATES for Performance](#).

Caution: HP Vertica does not remove deleted data immediately but keeps it as history for the purposes of **historical query**. A large amount of history can result in slower query performance. For information about how to configure the appropriate amount of history to retain, see [Purging Deleted Data](#).

Optimizing DELETES and UPDATES for Performance

The process of optimizing DELETE and UPDATE queries is the same for both operations. Some simple steps can increase the query performance by tens to hundreds of times. The following sections describe several ways to improve projection design and improve DELETE and UPDATE queries to significantly increase DELETE and UPDATE performance.

Note: For large bulk deletion, HP Vertica recommends using [Partitioned Tables](#) where possible because they provide the best DELETE performance and improve query performance.

Projection Column Requirements for Optimized Deletes

When all columns required by the DELETE or UPDATE predicate are present in a projection, the projection is optimized for DELETES and UPDATES. DELETE and UPDATE operations on such projections are significantly faster than on non-optimized projections. Both simple and pre-join projections can be optimized.

For example, consider the following table and projections:

```
CREATE TABLE t (a INTEGER, b INTEGER, c INTEGER);  
CREATE PROJECTION p1 (a, b, c) AS SELECT * FROM t ORDER BY a;  
CREATE PROJECTION p2 (a, c) AS SELECT a, c FROM t ORDER BY c, a;
```

In the following query, both p1 and p2 are eligible for DELETE and UPDATE optimization because column a is available:

```
DELETE from t WHERE a = 1;
```

In the following example, only projection p1 is eligible for DELETE and UPDATE optimization because the b column is not available in p2:

```
DELETE from t WHERE b = 1;
```

Optimized Deletes in Subqueries

To be eligible for DELETE optimization, all target table columns referenced in a DELETE or UPDATE statement's WHERE clause must be in the projection definition.

For example, the following simple schema has two tables and three projections:

```
CREATE TABLE tb1 (a INT, b INT, c INT, d INT);  
CREATE TABLE tb2 (g INT, h INT, i INT, j INT);
```

The first projection references all columns in tb1 and sorts on column a:

```
CREATE PROJECTION tb1_p AS SELECT a, b, c, d FROM tb1 ORDER BY a;
```

The buddy projection references and sorts on column a in tb1:

```
CREATE PROJECTION tb1_p_2 AS SELECT a FROM tb1 ORDER BY a;
```

This projection references all columns in tb2 and sorts on column i:

```
CREATE PROJECTION tb2_p AS SELECT g, h, i, j FROM tb2 ORDER BY i;
```

Consider the following DML statement, which references tb1.a in its WHERE clause. Since both projections on tb1 contain column a, both are eligible for the optimized DELETE:

```
DELETE FROM tb1 WHERE tb1.a IN (SELECT tb2.i FROM tb2);
```

Restrictions

Optimized DELETES are not supported under the following conditions:

- With pre-join projections on nodes that are down
- With replicated and pre-join projections if subqueries reference the target table. For example, the following syntax is not supported:

```
DELETE FROM tb1 WHERE tb1.a IN (SELECT e FROM tb2, tb2 WHERE tb2.e = tb1.e);
```

- With subqueries that do not return multiple rows. For example, the following syntax is not supported:

```
DELETE FROM tb1 WHERE tb1.a = (SELECT k from tb2);
```

Projection Sort Order for Optimizing Deletes

Design your projections so that frequently-used DELETE or UPDATE predicate columns appear in the sort order of all projections for large DELETES and UPDATES.

For example, suppose most of the DELETE queries you perform on a projection look like the following:

```
DELETE from t where time_key < '1-1-2007'
```

To optimize the DELETES, make time_key appear in the ORDER BY clause of all your projections. This schema design results in better performance of the DELETE operation.

In addition, add additional sort columns to the sort order such that each combination of the sort key values uniquely identifies a row or a small set of rows. For more information, see [Choosing Sort Order: Best Practices](#). To analyze projections for sort order issues, use the `EVALUATE_DELETE_PERFORMANCE` function.

Purging Deleted Data

In HP Vertica, delete operations do not remove rows from physical storage. Unlike most databases, the [DELETE](#) command in HP Vertica marks rows as deleted so that they remain available to **historical queries**. These deleted rows are called **historical data**. Retention of historical data also applies to the [UPDATE](#) command, which is actually a combined DELETE and INSERT operation.

The cost of retaining deleted data in physical storage can be measured in terms of:

- Disk space for the deleted rows and delete markers
- A performance penalty for reading and skipping over deleted data

A purge operation permanently removes deleted data from physical storage so that the disk space can be reused. HP Vertica gives you the ability to control how much deleted data is retained in the physical storage used by your database by performing a purge operation using one of the following techniques:

- [Setting a Purge Policy](#)
- [Manually Purging Data](#)

Both methods set the **Ancient History Mark (AHM)**, which is an epoch that represents the time until which history is retained. History older than the AHM are eligible for purge.

Note: Large delete and purge operations in HP Vertica could take a long time to complete, so use them sparingly. If your application requires deleting data on a regular basis, such as by month or year, HP recommends that you design tables that take advantage of [table partitioning](#). If partitioning tables is not suitable, consider the procedure described in [Rebuilding a Table](#). The [ALTER TABLE..RENAME](#) command lets you build a new table from the old table, drop the old table, and rename the new table in its place.

Setting a Purge Policy

The preferred method for purging data is to establish a policy that determines which deleted data is eligible to be purged. Eligible data is automatically purged when the **Tuple Mover** performs **mergeout** operations.

HP Vertica provides two methods for determining when deleted data is eligible to be purged:

- Specifying the time for which delete data is saved
- Specifying the number of **epochs** that are saved

Specifying the Time for Which Delete Data Is Saved

Specifying the time for which delete data is saved is the preferred method for determining which deleted data can be purged. By default, HP Vertica saves historical data only when nodes are down.

To change the the specified time for saving deleted data, use the `HistoryRetentionTime` configuration parameter:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', '{ <seconds> | -1 }' );
```

In the above syntax:

- `seconds` is the amount of time (in seconds) for which to save deleted data.
- `-1` indicates that you do not want to use the `HistoryRetentionTime` configuration parameter to determine which deleted data is eligible to be purged. Use this setting if you prefer to use the other method (`HistoryRetentionEpochs`) for determining which deleted data can be purged.

The following example sets the history epoch retention level to 240 seconds:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', '240');
```

Specifying the Number of Epochs That Are Saved

Unless you have a reason to limit the number of epochs, HP recommends that you specify the time over which delete data is saved.

To specify the number of historical epoch to save through the `HistoryRetentionEpochs` configuration parameter:

1. Turn off the `HistoryRetentionTime` configuration parameter:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', '-1');
```

2. Set the history epoch retention level through the `HistoryRetentionEpochs` configuration parameter:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionEpochs', '{<num_epochs>|-1}');
```

- `num_epochs` is the number of historical epochs to save.
- `-1` indicates that you do not want to use the `HistoryRetentionEpochs` configuration parameter to trim historical epochs from the epoch map. By default, `HistoryRetentionEpochs` is set to -1.

The following example sets the number of historical epochs to save to 40:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionEpochs', '40');
```

Modifications are immediately implemented across all nodes within the database cluster. You do not need to restart the database.

Note: If both `HistoryRetentionTime` and `HistoryRetentionEpochs` are specified, `HistoryRetentionTime` takes precedence.

See [Epoch Management Parameters](#) for additional details.

Disabling Purge

If you want to preserve all historical data, set the value of both historical epoch retention parameters to -1, as follows:

```
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionTime', '-1');  
=> SELECT SET_CONFIG_PARAMETER('HistoryRetentionEpochs', '-1');
```

Manually Purging Data

Manually purging deleted data consists of the following series of steps:

1. **Determine the point in time** to which you want to purge deleted data.
2. **Set the Ancient History Mark (AHM)** to this point in time using one of the following SQL functions (described in the SQL Reference Manual):
 - `SET_AHM_TIME()` sets the AHM to the **epoch** that includes the specified `TIMESTAMP` value on the **initiator node**.
 - `SET_AHM_EPOCH()` sets the AHM to the specified epoch.
 - `GET_AHM_TIME()` returns a `TIMESTAMP` value representing the AHM.
 - `GET_AHM_EPOCH()` returns the number of the epoch in which the AHM is located.
 - `MAKE_AHM_NOW()` sets the AHM to the greatest allowable value (now), and lets you drop pre-existing projections. This purges all deleted data.

When you use `SET_AHM_TIME` or `GET_AHM_TIME`, keep in mind that the timestamp you specify is mapped to an epoch, which has (by default) a three-minute granularity. Thus, if you specify an AHM time of '2008-01-01 00:00:00.00' the resulting purge could permanently remove as much as the first three minutes of 2008, or could fail to remove the last three minutes of 2007.

Note: The system prevents you from setting the AHM beyond the point at which it would prevent recovery in the event of a node failure.

3. **Manually initiate a purge** using one of the following SQL functions (described in the SQL Reference Manual):
 - `PURGE_PROJECTION()` purges a specified projection.
 - `PURGE_TABLE()` purges all projections on the specified table.
 - `PURGE()` purges all projections in the physical schema.
4. The **Tuple Mover** performs a **mergeout** operation to purge the data.

Note: Manual purge operations can take a long time.

Managing the Database

This section describes how to manage the HP Vertica database, including:

- [Connection Load Balancing](#)
- [Managing Nodes](#)
- [Adding Disk Space to a Node](#)
- [Managing Tuple Mover Operations](#)
- [Managing Workloads](#)

Connection Load Balancing

Each client connection to a host in the HP Vertica cluster requires a small overhead in memory and processor time. If many clients connect to a single host, this overhead can begin to affect the performance of the database. You can attempt to spread the overhead of client connections by dictating that certain clients connect to specific hosts in the cluster. However, this manual balancing becomes difficult as new clients and hosts are added to your environment.

Connection load balancing helps automatically spread the overhead caused by client connections across the cluster by having hosts redirect client connections to other hosts. By redirecting connections, the overhead from client connections is spread across the cluster without having to manually assign particular hosts to individual clients. Clients can connect to a small handful of hosts, and they are naturally redirected to other hosts in the cluster.

There are two ways you can implement load balancing on your HP Vertica cluster:

- Native connection load balancing is a feature built into the HP Vertica server and client libraries that redirect client connections at the application level.
- Internet Protocol Virtual Server (IPVS) is software that can be installed on several hosts in the HP Vertica cluster that provides connection load balancing at the network level.

In most situations, you should use native connection load balancing instead of IPVS. The following sections explain each option in detail and explain their benefits.

Native Connection Load Balancing Overview

Native connection load balancing is a feature built into the Vertica Analytics Platform server and client libraries as well as **vsq1**. Both the server and the client need to enable load balancing for it to function. If connection load balancing is enabled, a host in the database cluster can redirect a client's attempt to it to another currently-active host in the cluster. This redirection is based on a load balancing policy. This redirection can only take place once, so a client is not bounced from one host to another.

Since native connection load balancing is incorporated into the HP Vertica client libraries, any client application that connects to HP Vertica transparently takes advantage of it simply by setting a connection parameter.

For more about native connection load balancing, see [About Native Connection Load Balancing](#).

IPVS Overview

IPVS is a feature of the Linux kernel which lets a single host act as a gateway to an entire cluster of hosts. The load balancing host creates a virtual IP address on the network. When a client connects to the virtual IP address, the IPVS load balancer transparently redirects the connection to one of the hosts in the cluster. For more on IPVS, see [Connection Load Balancing Using IPVS](#).

Choosing Whether to Use Native Connection Load Balancing or IPVS

Native connection load balancing several advantages over IPVS. Native connection load balancing is:

- Easy to set up. All that is required is that the client and server enable connection load balancing. IPVS requires the configuration of several additional software packages and the network.
- Easily overridden in cases where you need to connect to a specific host (for example, when using the [COPY](#) statement to load a file a specific host's filesystem).
- Less at risk of host failures affecting connection load balancing. IPVS is limited to a master and a backup server. All hosts in HP Vertica are capable of redirecting connections for connection load balancing. As long as HP Vertica is running, it can perform connection load balancing.
- Less memory and CPU intensive than running a separate IPVS process on several database hosts.
- IPVS hurts load performance, since there is an additional network hop from the IPVS server to a host in the database. Using native connection load balancing, clients connect directly to hosts in the Vertica Analytics Platform cluster.
- Supported by HP. Currently, IPVS is not being actively supported.
- Is supported on all Linux platforms supported by HP Vertica. HP only supplies IPVS installation packages on Red Hat. IPVS has known compatibility issues with Debian-based Linux distributions.

There are a few cases where IPVS may be more suitable:

- Restrictive firewalls between the clients and the hosts in the database cluster can interfere with native connection load balancing. In order for native connection load balancing to work, clients must be able to access **every** host in the HP Vertica cluster. Any firewall between the hosts and the clients must be configured to allow connections to the HP Vertica database. Otherwise, a

client could be redirected to a host that it cannot access. IPVS can be configured so that the virtual IP address clients use to connect is outside of the firewall while the hosts can be behind the firewall.

- Since native connection load balancing works at the application level, clients must opt-in to have their connections load balanced. Because it works at the network protocol level, IPVS can force connections to be balanced.

How you choose to implement connection load balancing depends on your network environment. Since native load connection balancing is easier to implement, you should use it unless your network configuration requires that clients be separated from the hosts in the HP Vertica database by a firewall.

About Native Connection Load Balancing

Native connection load balancing is a feature built into the HP Vertica server and client libraries that helps spread the CPU and memory overhead caused by client connections across the hosts in the database. It can prevent hosts from becoming burdened by having many client connections while other hosts in the cluster do not.

Native connection load balancing only has an effect when it is enabled by both the server and the client. When both have enabled native connection load balancing, the following process takes place whenever the client attempts to open a connection to HP Vertica::

1. The client connects to a host in the database cluster, with a connection parameter indicating that it is requesting a load-balanced connection.
2. The host chooses a host from the list of currently up hosts in the database based on the current load balancing scheme (see below).
3. The host tells the client which host it has selected to handle the client's connection.
4. If the host chose another host in the database to handle the client connection, the client disconnects from the initial host. Otherwise, the client jumps to step 6.
5. The client establishes a connection to the host that will handle its connection. The client sets this second connection request so that the second host does not interpret the connection as a request for load balancing.
6. The client connection proceeds as usual, (negotiating encryption if the connection has SSL enabled, and proceeding to authenticating the user).

This entire process is transparent to the client application. The client driver automatically disconnects from the initial host and reconnects to the host selected for load balancing.

Notes

- Native connection load balancing works with the ADO.NET driver's connection pooling. The connection the client makes to the initial host and the final connection to the load balanced host used pooled connections if they are available.
- For client applications using the JDBC and ODBC driver in conjunction with third-party connection pooling solutions, the initial connection is not pooled since it is not a full client connection. The final connection is pooled, since it is a standard client connection.
- The client libraries include a failover feature that allow them to connect to backup hosts if the host specified in the connection properties is unreachable. When using native connection load balancing, this failover feature is only used for the initial connection to the database. If the host to which the client was redirected does not respond to the client's connection request, the client does not attempt to connect to a backup host and instead returns a connection error to the user. Since clients are only redirected to hosts that are known to be up, this sort of connection failure should only occur if the targeted host happened to go down at the same moment the client is redirected to it. See [ADO.NET Connection Failover](#), [JDBC Connection Failover](#), and [ODBC Connection Failover](#) in the Programmer's Guide for more information.

Load Balancing Schemes

The load balancing scheme controls how a host selects which host to handle a client connection. There are three available schemes:

- NONE: Disables native connection load balancing. This is the default setting.
- ROUNDROBIN: Chooses the next host from a circular list of currently up hosts in the database (i.e. node #1, node #2, node #3, etc. until it wraps back to node #1 again). Each host in the cluster maintains its own pointer to the next host in the circular list, rather than there being a single cluster-wide state.
- RANDOM: Chooses a host at random from the list of currently up hosts in the cluster.

You set the native connection load balancing scheme using the [SET_LOAD_BALANCE_POLICY](#) function. See [Enabling and Disabling Native Connection Load Balancing](#) for instructions.

Related Tasks

| | |
|---|-----|
| Enabling and Disabling Native Connection Load Balancing | 560 |
| Monitoring Native Connection Load Balancing | 561 |

Enabling and Disabling Native Connection Load Balancing

Only a database **superuser** can enable or disable native connection load balancing. To enable or disable load balancing, use the [SET_LOAD_BALANCE_POLICY](#) function to set the load balance policy. Setting the load balance policy to anything other than 'NONE' enables load balancing on the

server. The following example enables native connection load balancing by setting the load balancing policy to ROUNDROBIN.

```
=> SELECT SET_LOAD_BALANCE_POLICY('ROUNDROBIN');
        SET_LOAD_BALANCE_POLICY
-----
Successfully changed the client initiator load balancing policy to: roundrobin
(1 row)
```

To disable native connection load balancing, use SET_LOAD_BALANCE_POLICY to set the policy to 'NONE':

```
=> SELECT SET_LOAD_BALANCE_POLICY('NONE');
        SET_LOAD_BALANCE_POLICY
-----
Successfully changed the client initiator load balancing policy to: none
(1 row)
```

Note: By default, client connections are not load balanced, even when connection load balancing is enabled on the server. Clients must set a connection parameter to indicate they are willing to have their connection request load balanced. See [Enabling Native Connection Load Balancing in ADO.NET](#), [Enabling Native Connection Load Balancing in JDBC](#) and [Enabling Native Connection Load Balancing in ODBC](#) in the Programmer's Guide for more information.

Resetting the Load Balancing State

When the load balancing policy is ROUNDROBIN, each host in the HP Vertica cluster maintains its own state of which host it will select to handle the next client connection. You can reset this state to its initial value (usually, the host with the lowest-node id) using the [RESET_LOAD_BALANCE_POLICY](#) function:

```
=> SELECT RESET_LOAD_BALANCE_POLICY();
        RESET_LOAD_BALANCE_POLICY
-----
Successfully reset stateful client load balance policies: "roundrobin".
(1 row)
```

Related Information

[About Native Connection Load Balancing](#) 559

Related Tasks

[Monitoring Native Connection Load Balancing](#)..... 561

Monitoring Native Connection Load Balancing

Query the LOAD_BALANCE_POLICY column of the V_MONITOR.DATABASES to determine the state of native connection load balancing on your server:

```
=> SELECT LOAD_BALANCE_POLICY FROM V_CATALOG.DATABASES;  
LOAD_BALANCE_POLICY  
-----  
roundrobin  
(1 row)
```

Determining to which Node a Client Has Connected

A client can determine the node to which is has connected by querying the NODE_NAME column of the V_MONITOR.CURRENT_SESSION table:

```
=> SELECT NODE_NAME FROM V_MONITOR.CURRENT_SESSION;  
NODE_NAME  
-----  
v_vmart_node0002  
(1 row)
```

Related Information

[About Native Connection Load Balancing.....559](#)

Related Tasks

[Enabling and Disabling Native Connection Load Balancing.....560](#)

Connection Load Balancing Using IPVS

The IP Virtual Server (IPVS) provides network-protocol-level connection load balancing. When used with an HP Vertica database cluster, it is installed on two database hosts.

IPVS is made up of the following components:

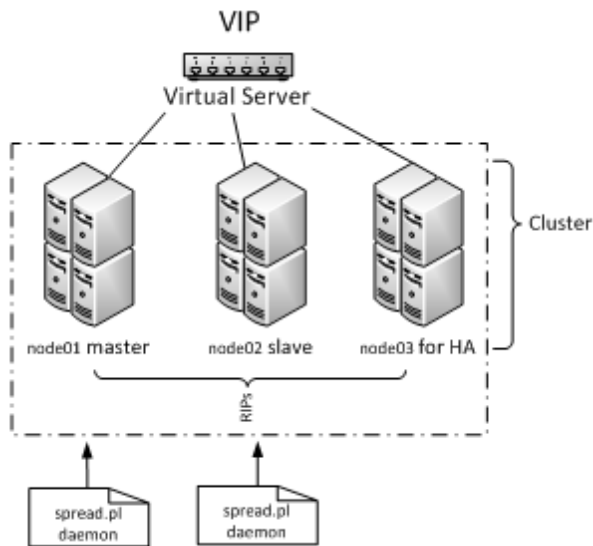
- The Virtual IP (VIP): The IP address that is accessed by all client connections.
- Real server IPs (RIP): The IP addresses of client network interfaces used for connecting database clients to the database engine.
- Cluster: A cluster of real HP Vertica servers (nodes).
- Virtual server: The single point of entry that provides access to a cluster, based on dynamic node selection.

The IPVS load balancer is two node stand-by redundancy only. The IPVS redundancy model is different than that of the HP Vertica Analytic Database. See [Failure Recovery](#) for details on HP Vertica redundancy.

Client connections made through the Virtual IP (VIP) are managed by a primary (master) director node, which is one of the real server nodes (RIP). The master director handles the routing of requests by determining which node has the fewest connections and sending connections to that node. If the director node fails for any reason, a failover (slave) director takes over request routing until the primary (master) director comes back online.

For example, if a user connects to node03 in a three-node cluster and node03 fails, the current transaction rolls back, the client connection fails, and a connection must be reestablished on another node.

The following graphic illustrates a three-node database cluster where all nodes share a single VIP. The cluster contains a master director (node01), a slave director (node02), and an additional host (node03) that together provide the minimum configuration for high availability (**K-safety**). In this setup (and in the configuration and examples that follow), node01 and node02 play dual roles as IPVS directors and HP Vertica nodes.



Notes

- Load balancing on a VIP is supported for Linux Red Hat Enterprise Linux 5 and 6, 64-bit.
- HP Vertica must be installed on each node in the cluster; the database can be installed on any node, but only one database can be running on an HP Vertica cluster at a time.
- Although a 0 K-safety (two-node) design is supported, HP strongly recommends that you create the load-balancing network using a minimum three-node cluster with K-safety set to 1. This way if one node fails, the database stays up. See [Designing for K-Safety](#) for details.
- When K-safety is set to 1, locate the IPVS master and slave on HP Vertica database nodes that comprise a buddy projections pair. This is the best way to ensure high-availability load-balancing. See [High Availability Through Projections](#) for details on buddy projections.
- If the node that is the IPVS master fails completely, the slave IPVS takes over load balancing. However, if the master only partially fails (i.e., it loses some of its processes but the node is still up), you may have to modify IP addresses to direct network traffic to the slave node. Alternatively, you can try to restart the processes on the master.
- Subsequent topics in this section describe how to set up two directors (master and slave), but you can set up more than two directors. See the [Keepalived User Guide](#) for details. See also the [Linux Virtual Server Web site](#).

Configuring HP Vertica Nodes

This section describes how to configure an HP Vertica cluster of nodes for load balancing. You'll set up two directors in a master/slave configuration and include a third node for **K-safety**.

AN HP Vertica cluster designed for load balancing uses the following configuration:

- **Real IP (RIP)** address is the public interface and includes:
 - The master director/node, which handles the routing of requests. The master is collocated with one of the database cluster nodes
 - The slave director/node, which communicates with the master and takes over routing requests in the event of a master node failure. The slave is collocated with another database cluster node

n nodes database cluster, such as at least one failover node to provide the minimum configuration for high availability. (**K-safety**).

- **Virtual IP (VIP)** address (generally assigned to eth0 in Linux) is the public network interface over which database clients connect.

Note: The VIP must be public so that clients outside the cluster can contact it.

Once you have set up an HP Vertica cluster and created a database, you can choose the nodes that will be directors. To achieve the best high-availability load balancing result when K-safety is set to 1, ensure that the IPVS master node and the slave node are located on HP Vertica database nodes with a buddy projections pair. (See [High Availability Through Projections](#) for information on buddy projections.)

The instructions in this section use the following node configuration:

| Pre-configured IP | Node assignment | Public IPs | Private IPs |
|---------------------|------------------------|--------------|--------------|
| VIP | shared among all nodes | 10.10.51.180 | |
| RIP master director | node01 | 10.10.51.55 | 192.168.51.1 |
| RIP slave director | node02 | 10.10.51.56 | 192.168.51.2 |
| RIP failover node | node03 | 10.10.51.57 | 192.168.51.3 |

Notes

- In the above table, the private IPs determine which node to send a request to. They are not the same as the RIPs.
- The VIP must be on the same subnet as the nodes in the HP Vertica cluster.

- Both the master and slave nodes (node01 and node02 in this section) require additional installation and configuration, as described in [Configuring the Directors](#).
- Use the command `$ cat /etc/hosts` to display a list of all hosts in your cluster

The following external web sites might be useful. The links worked at the last date of publication, but HP Vertica does not manage this content.

See Also

- [Linux Virtual Server Web site](#)
- [LVS-HOWTO Page](#)
- [Keepalived.conf\(5\) man page](#)
- [ipvsadm man page](#)

Set Up the Loopback Interface

This procedure sets up the loopback (lo) interface with an alias on each node.

1. Log in as root on the master director (node01):

```
$ su - root
```

2. Use the text editor of your choice to open `ifcfg-lo`:

```
[root@node01]# vi /etc/sysconfig/network-scripts/ifcfg-lo
```

3. Set up the loopback adapter with an alias for the VIP by adding the following block to the end of the file:

```
## vip deviceDEVICE=lo:0  
IPADDR=10.10.51.180  
NETMASK=255.255.255.255  
ONBOOT=yes  
NAME=loopback
```

Note: When you add the above block to your file, be careful not to overwrite the `127.0.0.1` parameter, which is required for proper system operations.

4. Start the device:

```
[root@node01]# ifup lo:0
```

5. Repeat steps 1-4 on each node in the HP Vertica cluster.

Disable Address Resolution Protocol (ARP)

This procedure disables ARP (Address Resolution Protocol) for the VIP.

1. On the master director (node01), log in as root:

```
$ su - root
```

2. Use the text editor of your choice to open the `sysctl` configuration file:

```
[root@node01]# vi /etc/sysctl.conf
```

3. Add the following block to the end of the file:

```
#LVSnet.ipv4.conf.eth0.arp_ignore =1  
net.ipv4.conf.eth0.arp_announce = 2  
# Enables packet forwarding  
net.ipv4.ip_forward =1
```

Note: For additional details, refer to the [LVS-HOWTO Page](#). You might also refer to the [Linux Virtual Server Wiki page](#) for information on using `arp_announce/arp_ignore` to disable the Address Resolution Protocol.

4. Use `ifconfig` to verify that the interface is on the same subnet as the VIP:

```
[root@node01]# /sbin/ifconfig
```

In the following output, the `eth0 inet addr` is the VIP, and subnet 51 matches the private RIP under the `eth1` heading:

```
eth0      Link encap:Ethernet  HWaddr 84:2B:2B:55:4B:BE          inet addr:10.10.51  
.55 Bcast:10.10.51.255 Mask:255.255.255.0  
          inet6 addr: fe80::862b:2bff:fe55:4bbe/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:91694543 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:373212 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:49294294011 (45.9 GiB)  TX bytes:66149943 (63.0 MiB)
```

```
eth1      Interrupt:15 Memory:da000000-da012800
          Link encap:Ethernet HWaddr 84:2B:2B:55:4B:BF
          inet addr:192.168.51.55 Bcast:192.168.51.255 Mask:255.255.255.0
          inet6 addr: fe80::862b:2bff:fe55:4bbf/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:937079543 errors:0 dropped:2780 overruns:0 frame:0
          TX packets:477401433 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:449050544237 (418.2 GiB) TX bytes:46302821625 (43.1 GiB)
          Interrupt:14 Memory:dc000000-dc012800
lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:6604 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6604 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:21956498 (20.9 MiB) TX bytes:21956498 (20.9 MiB)
lo:0     Link encap:Local Loopback
          inet addr:10.10.51.180 Mask:255.255.255.255
          UP LOOPBACK RUNNING MTU:16436 Metric:1
```

5. Use `ifconfig` to verify that the loopback interface is up:

```
[root@node01]# /sbin/ifconfig lo:0
```

You should see output similar to the following:

```
lo:0     Link encap:Local Loopback          inet addr:10.10.51.180 Mask:255.255.255.255
          UP LOOPBACK RUNNING MTU:16436 Metric:1
```

If you do not see `UP LOOPBACK RUNNING`, bring up the loopback interface:

```
[root@node01]# /sbin/ifup lo
```

6. Issue the following command to commit changes to the kernel from the configuration file:

```
[root@node01]# /sbin/sysctl -p
```

7. Repeat steps 1-6 on all nodes in the HP Vertica cluster.

Configuring the Directors

Now you are ready to install the HP Vertica IPVS Load Balancer package and configure the master (node01) and slave (node02) directors.

Install the HP Vertica IPVS Load Balancer Package

The following instructions describe how to download and install the HP Vertica IPVS Load Balancer package for Red Hat Enterprise Linux 5 and Red Hat Enterprise Linux 6.

Note: For illustrative purposes only, this procedure uses `node01` for the master director and `node02` for the slave director.

Before You Begin

Before installing IPVS, you must:

- Install HP Vertica
- Create a database cluster

If You Are Using Red Hat Enterprise Linux 5.x:

Make sure you have downloaded and installed the HP Vertica Analytics Database RPM and the IPVS Load Balancer package for Red Hat Enterprise Linux 5.

1. On the master director (node01) log in as root:

```
$ su - root
```

2. Download the IPVS Load Balancer package for Red Hat Enterprise Linux 5 from the my.vertica.com website to a location on the master server, such as to `/tmp`.
3. Change directory to the location of the downloaded file:

```
# cd /tmp
```

4. Install (or upgrade) the Load Balancer package using the `rpm -Uvh` command.

The following is an example for the Red Hat Linux package only; package names name could change between releases:

```
# rpm -Uvh vertica-ipvs-load-balancer-<current-version>.x86_64.RHEL5.rpm
```

5. Repeat steps 1-4 on the slave director (node02).

If You Are Using Red Hat Enterprise Linux 6.x:

Make sure you have downloaded and installed the HP Vertica Analytics Database RPM and the IPVS Load Balancer package for Red Hat Enterprise Linux 6.

1. On the master director (node01) log in as root:

```
$ su - root
```

2. Download the IPVS Load Balancer package for Red Hat Enterprise Linux 6 from the my.vertica.com website to a location on the master server, such as to /tmp.
3. Change directory to the location of the downloaded file:

```
# cd /tmp
```

4. Run this command as root:

```
/sbin/modprobe ip_vs
```

5. Verify that ip_vs is loaded correctly using this command:

```
lsmod | grep ip_vs
```

6. Install (or upgrade) the Load Balancer package using the `rpm -Uvh` command.

The following is an example for the Red Hat Linux package only; package names name could change between releases:

```
# rpm -Uvh vertica-ipvs-load-balancer-<current-version>.x86_64.RHEL6.rpm
```

7. Repeat steps 1-6 on the slave director (node02).

Configure the HP Vertica IPVS Load Balancer

HP Vertica provides a script called `configure-keepalived.pl` in the IPVS Load Balancer package. The script is located in `/sbin`, and if you run it with no options it prints a usage summary:

```
--ripips      | Comma separated list of HP Vertica nodes; public IPs (e.g., 10.10.50.116, etc.)
--priv_ips    | Comma separated list of HP Vertica nodes; private IPs (e.g., 192.168.51.16, etc.)
--riport      | Port on which HP Vertica runs. Default is 5433
--iface       | Public ethernet interface HP Vertica is configured to use (e.g., eth0)
--emailto     | Address that should get alerts (e.g., user@server.com)
--emailfrom   | Address that mail should come from (e.g., user@server.com)
--mailserver  | E-mail server IP or hostname (e.g., mail.server.com)
--master      | If this director is the master (default), specify --master
--slave       | If this director is the slave, specify --slave
--authpass    | Password for keepalived
--vip         | Virtual IP address (e.g., 10.10.51.180)
--delayloop   | Seconds keepalived waits between healthchecks. Default is 2
--algo        | Sets the algorithm to use: rr, wrr, lc (default), wlc, lb1c, lb1cr, dh, sh, sed, nq
--kind        | Sets the routing method to use. Default is DR.
--priority    | By default, master has priority of 100 and the backup (slave) has priority of 50
```

For details about each of these parameters, refer to the [ipvsadm\(8\) - Linux man page](#).

Public and Private IPs

If your cluster uses private interfaces for spread cluster communication, you need to use the `--priv_ips` switch to enter the private IP addresses that correspond to the public IP addresses (or RIPs). The IPVS keepalive daemon uses these private IPs to determine when a node has left the cluster.

The IP host ID of the RIPs must correspond to the IP host ID of the private interfaces. For example, given the following IP address mappings:

| Public | Private (for spread) |
|--------------|----------------------|
| 10.10.50.116 | 192.168.51.116 |
| 10.10.50.117 | 192.168.51.117 |
| 10.10.50.118 | 192.168.51.118 |

You need to enter the IP addresses in the following order:

```
--ripips 10.10.50.116,10.10.50.117,10.10.50.118
--priv_ips 192.168.51.116,192.168.51.117,192.168.51.118
```

You must use IP addresses, not node names, or the `spread.pl` script could fail.

If you do not specify private interfaces, HP Vertica uses the public RIPs for the MISC check, as shown in step 3 below.

Set up the HP Vertica IPVS Load Balancer Configuration File

1. On the master director (node01) log in as root:

```
$ su - root
```

2. Run the HP-supplied configuration script with the appropriate switches; for example:

```
# /sbin/configure-keepalived.pl --ripips 10.10.50.116,10.10.50.117,10.10.50.118  
--priv_ips 192.168.51.116,192.168.51.117,192.168.51.118  
--riport 5433  
--iface eth0  
--emailto dbadmin@companyname.com  
--emailfrom dbadmin@companyname.com  
--mailserver mail.server.com  
--master  
--authpass password  
--vip 10.10.51.180  
--delayloop 2  
--algo lc  
--kind DR  
--priority 100
```

Caution: The `--authpass` (password) switch must be the same on both the master and slave directors.

3. Check the `keepalived.conf` file to verify private and public IP settings for the `--ripips` and `--priv_ips` switches and make sure the `real_server` IP address is public.

```
# cat /etc/keepalived/keepalived.conf
```

An entry in the `keepalived.conf` file would resemble the following:

```
real_server 10.10.50.116 5433 {  
    MISC_CHECK {  
        misc_path "/etc/keepalived/check.pl 192.168.51.116"  
    }  
}
```

4. Start **spread**:

```
# /etc/init.d/spread.pl start
```

The `spread.pl` script writes to the `check.txt` file, which is rewritten to include only the remaining nodes in the event of a node failure. Thus, the virtual server knows to stop sending `vsq` requests to the failed node.

5. Start `keepalived` on `node01`:

```
# /etc/init.d/keepalived start
```

6. If not already started, start `sendmail` to allow mail messages to be sent by the directors:

```
# /etc/init.d/sendmail start
```

7. Repeat steps 1-5 on the slave director (`node02`), using the same switches, except (**IMPORTANT**) replace the `--master` switch with the `--slave` switch.

Note: Tip: Use a lower priority for the slave `--priority` switch. HP currently suggests 50.

```
# /sbin/configure-keepalived.pl
--ripips 10.10.50.116,10.10.50.117,10.10.50.118
--priv_ips 192.168.51.116,192.168.51.117,192.168.51.118
--riport 5433
--iface eth0
--emailto dbadmin@companyname.com
--emailfrom dbadmin@companyname.com
--mailserver mail.server.com
--slave
--authpass password
--vip 10.10.51.180
--delayloop 2
--algo lc
--kind DR
--priority 100
```

See Also

- [Keepalived.conf\(5\)-Linux man page](#)

Connecting to the Virtual IP (VIP)

To connect to the Virtual IP address using `vsq`, issue a command similar to the following. The IP address, which could also be a DNS address, is the VIP that is shared among all nodes in the HP Vertica cluster.

```
$ /opt/vertica/bin/vsq -h 10.10.51.180 -U dbadmin
```

To verify connection distribution over multiple nodes, repeat the following statement multiple times and observe connection distribution in an `lc` (least amount of connections) fashion.

```
$ vsql -h <VIP> -c "SELECT node_name FROM sessions"
```

Replace `<VIP>` in the above script with the IP address of your virtual server; for example:

```
$ vsql -h 10.10.51.180 -c "SELECT node_name FROM sessions" node_name
-----
v_ipvs_node01
v_ipvs_node02
v_ipvs_node03
(3 rows)
```

Monitoring Shared Node Connections

If you want to monitor which nodes are sharing connections, view the `check.txt` file by issuing the following command at a shell prompt:

```
# watch cat /etc/keepalived/check.txtEvery 2.0s: cat /etc/keepalived/check.txt Wed Nov
 3 10:02:20 2010
N192168051057
N192168051056
N192168051055
```

The `check.txt` is a file located in the `/etc/keepalived/` directory, and it gets updated when you submit changes to the kernel using `sysctl -p`, described in [Disable the Address Resolution Protocol \(ARP\)](#). For example, the `spread.pl` script (see [Configuring the Directors](#)), writes to the `check.txt` file, which is then modified to include only the remaining nodes in the event of a node failure. Thus, the virtual server knows to stop sending `vsql` requests to the failed node.

You can also look for messages by issuing the following command at a shell prompt:

```
# tail -f /var/log/messages
Nov  3 09:21:00 p6 Keepalived: Starting Keepalived v1.1.17 (05/17,2010)Nov  3 09:21:00 p6
Keepalived: Starting Healthcheck child process, pid=32468
Nov  3 09:21:00 p6 Keepalived: Starting VRRP child process, pid=32469
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Using LinkWatch kernel netlink reflector...
Nov  3 09:21:00 p6 Keepalived_vrrp: Using LinkWatch kernel netlink reflector...
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Netlink reflector reports IP 10.10.51.55 ad
ded
Nov  3 09:21:00 p6 Keepalived_vrrp: Netlink reflector reports IP 10.10.51.55 added
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Netlink reflector reports IP 192.168.51.55
added
Nov  3 09:21:00 p6 Keepalived_vrrp: Netlink reflector reports IP 192.168.51.55 added
Nov  3 09:21:00 p6 Keepalived_vrrp: Registering Kernel netlink reflector
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Registering Kernel netlink reflector
Nov  3 09:21:00 p6 Keepalived_vrrp: Registering Kernel netlink command channel
Nov  3 09:21:00 p6 Keepalived_vrrp: Registering gratuitous ARP shared channel
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Registering Kernel netlink command channel
Nov  3 09:21:00 p6 Keepalived_vrrp: Opening file '/etc/keepalived/keepalived.conf'.
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Opening file
```

```
'/etc/keepalived/keepalived.conf'.  
Nov  3 09:21:00 p6 Keepalived_vrrp: Configuration is using : 63730 Bytes  
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Configuration is using : 16211 Bytes  
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Activating healthcheckers for service [10.1  
0.51.55:5433]  
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Activating healthcheckers for service [10.1  
0.51.56:5433]  
Nov  3 09:21:00 p6 Keepalived_healthcheckers: Activating healthcheckers for service [10.1  
0.51.57:5433]  
Nov  3 09:21:00 p6 Keepalived_vrrp: VRRP sockpool: [ifindex(2), proto(112), fd(10,11)]  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: Misc check to [10.10.51.56] for [/etc/keepa  
lived/check.pl 192.168.51.56] failed.  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: Removing service [10.10.51.56:5433] from VS  
[10.10.51.180:5433]  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connecte  
d.  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: Misc check to [10.10.51.55] for [/etc/keepa  
lived/check.pl 192.168.51.55] failed.  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: Removing service [10.10.51.55:5433] from VS  
[10.10.51.180:5433]  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connecte  
d.  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: Misc check to [10.10.51.57] for [/etc/keepa  
lived/check.pl 192.168.51.57] failed.  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: Removing service [10.10.51.57:5433] from VS  
[10.10.51.180:5433]  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connecte  
d.  
Nov  3 09:21:01 p6 Keepalived_healthcheckers: SMTP alert successfully sent.  
Nov  3 09:21:10 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Transition to MASTER STATE  
Nov  3 09:21:20 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Entering MASTER STATE  
Nov  3 09:21:20 p6 Keepalived_vrrp: VRRP_Instance(VI_1) setting protocol VIPs.  
Nov  3 09:21:20 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 f  
or 10.10.51.180  
Nov  3 09:21:20 p6 Keepalived_healthcheckers: Netlink reflector reports IP 10.10.51.180 a  
dded  
Nov  3 09:21:20 p6 Keepalived_vrrp: Remote SMTP server [127.0.0.1:25] connected.  
Nov  3 09:21:20 p6 Keepalived_vrrp: Netlink reflector reports IP 10.10.51.180 added  
Nov  3 09:21:20 p6 Keepalived_vrrp: SMTP alert successfully sent.  
Nov  3 09:21:25 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 f  
or 10.10.51.1
```

Determining Where Connections Are Going

lvsadm is the user code interface to the IP Virtual Server. It is used to set up, maintain, or inspect the virtual server table in the Linux kernel.

If you want to identify where user connections are going, install `lvsadm`.

1. Log in to the master director (node01) as root:

```
$ su - root
```

2. Install ipvsadm:

```
[root@node01]# yum install ipvsadm
Loading "installonlyn" plugin
Setting up Install Process
Setting up repositories
Reading repository metadata in from local files
Parsing package install arguments
Resolving Dependencies
--> Populating transaction set with selected packages. Please wait.
--> Downloading header for ipvsadm to pack into transaction set.
ipvsadm-1.24-10.x86_64.rp 100% |=====| 6.6 kB    00:00
--> Package ipvsadm.x86_64 0:1.24-10 set to be updated
--> Running transaction check
Dependencies Resolved

=====
Package                Arch      Version      Repository      Size
=====
Installing:
ipvsadm                x86_64    1.24-10      base             32 k
Transaction Summary
=====
Install      1 Package(s)
Update      0 Package(s)
Remove      0 Package(s)
Total download size: 32 k
Is this ok [y/N]: y
Downloading Packages:
(1/1): ipvsadm-1.24-10.x86_64 100% |=====| 32 kB    00:00
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing: ipvsadm                ##### [1/1]
Installed: ipvsadm.x86_64 0:1.24-10
Complete!
```

3. Run ipvsadm:

```
[root@node01 ~]# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port          Forward Weight ActiveConn InActConn
TCP vs-wks1.verticacorp.com:pyrr lc
  -> node03.verticacorp.com:pyr Route      1      1      8
  -> node02.verticacorp.com:pyr Route      1      0      8
  -> node01.verticacorp.com:pyr Local      1      0      8
```

See Also

- [ipvsadm man page](#)

Virtual IP Connection Problems

Issue

Users cannot connect to the database through the Virtual IP (VIP) address.

Resolution

1. Check if spread is running:

```
$ ps ax | grep spread
```

```
11895 ?      S<s    4:30 /opt/vertica/spread/sbin/spread -n N192168051055 -c  
/opt/vertica/config/vspread.conf  
29617 pts/3  S+    0:00 grep spread
```

- a. If spread is not running, start spread as root or using sudo:

```
[root@node01]# /etc/init.d/spreadd start
```

- b. If spread is running, restart spread as root or using sudo:

```
[root@node01]# /etc/init.d/spreadd restart
```

- c. Check the spread status as root or using sudo:

```
[root@node01]# /etc/init.d/spreadd status
```

- d. Issue the `ifconfig` command to check the current IP addresses of the hosts, and verify that those IP addresses are listed in `/opt/vertica/config/vspread.conf`.

```
[root@node01]# ifconfig
```

If spread fails to start, examine the following files for problems:

```
/tmp/spread*.log
```

```
/var/log/spreadd.log
```

Permission problems and syntax problems are identified in the log files.

2. Check if keepalived is running:

```
$ ps ax | grep keepalived29622 pts/3 S+ 0:00 grep keepalived
```

- a. If keepalived is not running, start keepalived as root or using sudo:

```
# /etc/init.d/keepalived start
```

- b. If keepalived is running, restart keepalived as root or using sudo:

```
# /etc/init.d/keepalived restart
```

Issue

Users cannot connect to the database.

Resolution

Try to telnet to the VIP and port:

```
# telnet 10.10.51.180 5433
```

If telnet reports no route to host, recheck your `/etc/keepalived/keepalived.conf` file to make sure you entered the correct VIP and RIPs.

Errors and informational messages from the keepalived daemon are written to the `/var/log/messages` file, so check the messages file first:

```
# tail -f /var/log/messagesMay 18 09:04:32 dell02 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 for 10.10.10.100
May 18 09:04:32 dell02 avahi-daemon[3191]: Registering new address record for 10.10.10.100 on eth0.
May 18 09:04:32 dell02 Keepalived_healthcheckers: Netlink reflector reports IP 10.10.10.100 added
```

Expected E-mail Messages From the Keepalived Daemon

- Upon startup:

```
Subject: [node01] VRRP Instance VI_1 - Entering MASTER state=> VRRP Instance is now owning VRRP VIPs <=
```

- When a node fails:

```
Subject:[node01] Realserver 10.10.10.1:5433 - DOWN=> MISC CHECK failed on service <=
```

- When a node comes back up:

```
Subject: [node02] Realserver 10.10.10.1:5433 - UP=> MISC CHECK succeed on service <=
```

Troubleshooting Keepalived Issues

If there are connection or other issues related to the Virtual IP server and Keepalived, try some of the following tips:

- Set `KEEPALIVED_OPTIONS="-D -d"` in the `/etc/sysconfig/keepalived` file to enable both debug mode and dump configuration.
- Monitor the system log in `/var/log/messages`. If `keepalived.conf` is incorrect, the only indication is in the messages log file. For example:

```
$ tail /var/log/messages
```

Errors and informational messages from the keepalived daemon are also written to the `/var/log/messages` files.

- Type `ip addr list` and see the configured VIP addresses for `eth0`. For example:

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue link/loopback 00:00:00:00:00:00
   brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
   inet 10.10.51.180/32 brd 127.255.255.255 scope global lo:0
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
   link/ether 84:2b:2b:55:4b:be brd ff:ff:ff:ff:ff:ff
   inet 10.10.51.55/24 brd 10.10.51.255 scope global eth0
   inet6 fe80::862b:2bff:fe55:4bbe/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
   link/ether 84:2b:2b:55:4b:bf brd ff:ff:ff:ff:ff:ff
   inet 192.168.51.55/24 brd 192.168.51.255 scope global eth1
   inet6 fe80::862b:2bff:fe55:4bbf/64 scope link
       valid_lft forever preferred_lft forever
4: sit0: <NOARP> mtu 1480 qdisc noop
   link/sit 0.0.0.0 brd 0.0.0.0
```

- Check `iptables` and notice the `PREROUTING` rule on the `BACKUP` (slave) director. Even though `ipvsadm` has a complete list of real servers to manage, it does not route anything as the prerouting rule redirects packets to the loopback interface.

```
# /sbin/iptables -t nat -n -LChain PREROUTING (policy ACCEPT)
target      prot opt source                destination
Chain POSTROUTING (policy ACCEPT)
target      prot opt source                destination
Chain OUTPUT (policy ACCEPT)
target      prot opt source                destination
```

Note: On some kernels, the `nat` tables does not show by default without the `-t` parameter, and `-n` is used to avoid long DNS lookups. See the [iptables\(8\) - Linux man page](#) for details.

- During failover, it is normal to expect delay in new connection establishment until the slave node takes control. The delay could be several minutes depending on the load on the cluster. If you cannot connect to the database, try to telnet to the VIP and port:

```
# telnet 10.10.51.180 5433
```

If telnet reports no route to host, recheck the keepalived configuration file (`/etc/keepalived/keepalived.conf`) to make sure you entered the correct VIP and RIPv.

Managing Nodes

HP Vertica provides the ability to [add](#), [remove](#), and [replace](#) nodes on a live cluster that is actively processing queries. This ability lets you scale the database without interrupting users.

Fault Groups

Fault groups provide a way for you to configure Vertica Analytics Platform for your physical cluster layout. Vertica Analytics Platform automatically creates fault groups around control nodes in large cluster arrangements, placing nodes that share a control node into the same fault group.

Consider defining your own fault groups specific to your cluster's physical layout if you want to:

- Influence the placement of control nodes in the cluster
- Reduce the impact of correlated failures inherent in your environment; for example, by describing your rack layout, Vertica Analytics Platform could tolerate a rack failure

Vertica Analytics Platform supports complex, hierarchical fault groups of different shapes and sizes and provides a fault group script (**DDL** generator), SQL statements, system tables, and other monitoring tools. Fault groups are also tightly integrated with [elastic](#) and [large cluster](#) layouts to provide the cluster flexibility and reliability you expect from Vertica Analytics Platform.

See Also

See [High Availability through Fault Groups](#) in the Concepts Guide for a conceptual overview of fault groups with a cluster topology example.

About the Fault Group Script

To help you create fault groups, Vertica Analytics Platform provides a script called `fault_group_ddl_generator.py` in the `/opt/vertica/scripts` directory. This script generates the SQL statements you'll run to create fault groups for your cluster. The script does not create fault groups for you.

Create a fault group input file

Use any text editor to create a fault group input file for the targeted cluster. You'll then pass the input file to the `fault_group_ddl_generator.py` script, which the script will use to output the commands you'll run.

The fault group input file must adhere to the following format:

First line in the template

The first line is for the parent (top-level) fault groups only, delimited by spaces; for example:

```
rack1 rack2 rack3 rack4
```

Remaining lines in template

The format for each subsequent line includes a parent fault group, followed by an equals sign (=) and then includes any combination of one or more nodes, fault groups, virtual machine servers or virtual machine hosts. The objects going into a parent fault group are delimited by spaces; for example:

```
<parent> = <child_1> <child_2> <child...>
```

After the first row of parent fault groups, the order in which you write the group descriptions does not matter; however, all fault groups that you define in this file must refer back to a parent fault group, either directly or by being the child of a fault group that is the child of a parent fault group.

To place, for example, B into A and both C and D into B, the format would be as follows:

```
A = B  
B = C D
```

In the above example, B is a child of A and a parent of both C and D.

Example input file

The following input file is an example only. In your file, the `node*` names will represent nodes in the cluster, such as `v_vmartdb_node0001`, `v_vmartd_node0002`, and so on.

```
rack1 rack2 rack3 rack4  
rack1 = node1 node2 node3 node4  
rack2 = node5 node6 node7 node8  
rack3 = node9 vm3_1  
vm3_1 = node10 node11 node12  
vm4_1 = node13 node14  
vm4_2 = node15 node16  
rack4 = vm4_1 vm4_2
```

Here's the example output returned by the script, which you can also pipe to a file and save for later use:

```
ALTER DATABASE vmartd DROP ALL FAULT GROUP;  
CREATE FAULT GROUP rack1;  
ALTER FAULT GROUP rack1 ADD NODE node1;  
ALTER FAULT GROUP rack1 ADD NODE node2;  
ALTER FAULT GROUP rack1 ADD NODE node3;  
ALTER FAULT GROUP rack1 ADD NODE node4;  
CREATE FAULT GROUP rack2;  
ALTER FAULT GROUP rack2 ADD NODE node5;  
ALTER FAULT GROUP rack2 ADD NODE node6;
```

```
ALTER FAULT GROUP rack2 ADD NODE node7;
ALTER FAULT GROUP rack2 ADD NODE node8;
CREATE FAULT GROUP rack3;
ALTER FAULT GROUP rack3 ADD NODE node9;
CREATE FAULT GROUP vm3_1;
ALTER FAULT GROUP vm3_1 ADD NODE node10;
ALTER FAULT GROUP vm3_1 ADD NODE node11;
ALTER FAULT GROUP vm3_1 ADD NODE node12;
ALTER FAULT GROUP rack3 ADD FAULT GROUP vm3_1;
CREATE FAULT GROUP rack4;
CREATE FAULT GROUP vm4_1;
ALTER FAULT GROUP vm4_1 ADD NODE node13;
ALTER FAULT GROUP vm4_1 ADD NODE node14;
ALTER FAULT GROUP rack4 ADD FAULT GROUP vm4_1;
CREATE FAULT GROUP vm4_2;
ALTER FAULT GROUP vm4_2 ADD NODE node15;
ALTER FAULT GROUP vm4_2 ADD NODE node16;
ALTER FAULT GROUP rack4 ADD FAULT GROUP vm4_2;
```

For more information, see [Creating Fault Groups](#).

About Automatic Fault Groups

If you do not use the `/opt/vertica/scripts/fault_group_ddl_generator.py` helper script and SQL statements to create fault groups, Vertica Analytics Platform creates automatic fault groups for you in order to create the correct control node assignments for the cluster layout.

All nodes that share the same control node reside in the same automatic fault group.

Ephemeral nodes are not included in automatic or user-defined fault groups because they hold no data.

Creating Fault Groups

When you define fault groups, Vertica Analytics Platform distributes data segments across the cluster so the cluster can tolerate large-scale failures inherent in your environment, such as the failure of a rack or a machine that might be hosting multiple virtual machine nodes.

Fault groups prerequisites

- Defining fault groups requires careful and thorough network planning. You must have a solid understanding of your network topology before you define fault groups.
- Vertica Analytics Platform must already be installed or upgraded and you must have already created a database.
- To create fault groups, you must be a **superuser**.

How to create a fault group

The following procedure assumes you have already planned your fault group hierarchy and created an input file to pass to the `fault_group_ddl_generator.py` script, described in [About the Fault Group Script](#).

Tip: Pipe the output from the script to a `*.sql` file so you can run a single file instead of multiple statements. Also consider saving your input file for reuse, in the event you want to modify fault groups, such as after you expand the cluster or change the distribution of control nodes.

1. As the database administrator or, a user with sudo privileges, log in to one of the target hosts in the cluster.
2. Run the `fault_group_ddl_generator.py` script and include the following arguments:
 - The database name
 - The fault group input file name
 - [Optional] The file name of the `.sql` script you want to save the commands to.

Example:

```
$ /opt/vertica/scripts/fault_group_ddl_generator.py  
vmartd faultGroupDescription.txt > faultgroupddl.sql
```

The above command writes the **DDL** instructions (SQL statements) for creating fault groups to the specified file name, `faultgroupddl.sql`.

3. Run **vsq1** and log in to the target database as the database administrator (`dbadmin` by default).
4. Run the `.sql` script you created in Step 2; for example:

```
=> \i faultgroupddl.sql
```

5. If you do not have large cluster enabled (your cluster has fewer than 120 nodes), proceed to the next step; otherwise, you must realign the control nodes by calling the following function:

```
=> SELECT realign_control_nodes();
```

6. Write changes to the spread configuration file:


```
=> SELECT reload_spread(true);
```

7. Use the **Administration Tools** to restart the database.
8. Save changes to the cluster's data layout by calling the following function:

```
=> SELECT rebalance_cluster();
```

See Also

For syntax on each of the fault group statements, see the following topics in the SQL Reference Manual:

- [Cluster Management Functions](#)
- [CREATE FAULT GROUP](#)
- [ALTER FAULT GROUP](#)
- [DROP FAULT GROUP](#)
- [ALTER DATABASE](#)

Modifying Fault Groups

You'll modify fault groups when you want to:

- Add a fault group to another fault group
- Remove a fault group from another fault group
- Add one or more nodes to a fault group
- Remove one or more nodes from a fault group (in which case the node is left without a parent fault group)
- Rename a fault group

How to modify a fault group

Before you modify existing fault groups, carefully plan the new layout and modify the input template file you created for the targeted cluster (see [About the Fault Group Script](#) and [Creating Fault Groups](#)).

1. As a user with sudo privileges, log in to one of the target hosts.
2. Run the [fault groups python script](#) and supply the following:
 - The database name
 - The fault group input file name
 - [Optional] The file name of the .sql script you want to create

Example:

```
$ /opt/vertica/scripts/fault_group_ddl_generator.py  
vmartd ModifiedFaultGroupTemp.txt > faultgroupddl.sql
```

The above command writes the **DDL** instructions (SQL statements) for creating fault groups to the specified file name, `faultgroupddl.sql`.

3. Run **vsq1** and log in to the specified database as the database administrator (default dbadmin).
4. Run the .sql script you created in the step 2; for example:

```
=> \i faultgroupddl.sql
```

5. If you do not have large cluster enabled, skip this step; otherwise, realign the control nodes by calling the following function:

```
=> SELECT realign_control_nodes();
```

6. Restart the spread process to write changes to the spread configuration file:

```
=> SELECT reload_spread(true);
```

7. Use the **Administration Tools** to restart the database.
8. Save changes to the cluster's data layout by calling the following function:

```
=> SELECT rebalance_cluster();
```

See Also

For syntax, see the following topics in the SQL Reference Manual:

- [ALTER FAULT GROUP](#)
- [ALTER DATABASE](#)
- [Cluster Management Functions](#)

Dropping Fault Groups

When you remove a fault group from the cluster, the operation removes the specified fault group and its child fault groups, placing all nodes under the parent of the dropped fault group. To see the current fault group hierarchy in the cluster, query the [FAULT_GROUPS](#) system table.

How to drop a fault group

Use the `DROP FAULT GROUP` statement to remove a fault group from the cluster. The following example drops the `group2` fault group:

```
vmartdb=> DROP FAULT GROUP group2;  
DROP FAULT GROUP
```

How to remove all fault groups

Use the `ALTER DATABASE` statement to drop all fault groups, along with any child fault groups, from the specified database cluster.

The following command drops all fault groups from the `vmartdb` database:

```
vmartdb=> ALTER DATABASE exampledb DROP ALL FAULT GROUP;  
ALTER DATABASE
```

To add nodes back to a fault group

To add a node back to a fault group, you must manually re-assign it to a new or existing fault group using `CREATE FAULT GROUP` and `ALTER FAULT GROUP..ADD NODE` statements.

See Also

- [DROP FAULT GROUP](#)
- [CREATE FAULT GROUP](#)
- [ALTER FAULT GROUP..ADD NODE](#)

Monitoring Fault Groups

You can monitor fault groups using Vertica Analytics Platform system tables or through the Management Console interface.

Monitoring fault groups through system tables

Use the following system tables to observe information about your fault groups as well as cluster vulnerabilities, such as the nodes the cluster cannot lose without the database going down:

- [V_CATALOG.FAULT_GROUPS](#)—View the hierarchy of all fault groups in the cluster.
- [V_CATALOG.CLUSTER_LAYOUT](#)—Observe the *actual* arrangement of the nodes participating in the data business and the fault groups that affect them. Ephemeral nodes are not shown in the cluster layout ring because they hold no data.

Monitoring fault groups through Management Console

An MC administrator can monitor and highlight fault groups of interest by following these steps:

1. Click the running database you want to monitor and click **Manage** in the task bar.
2. Open the **Fault Group View** menu and select the fault groups you want to view.
3. Optionally hide nodes that are not in the selected fault group to focus on fault groups of interest.

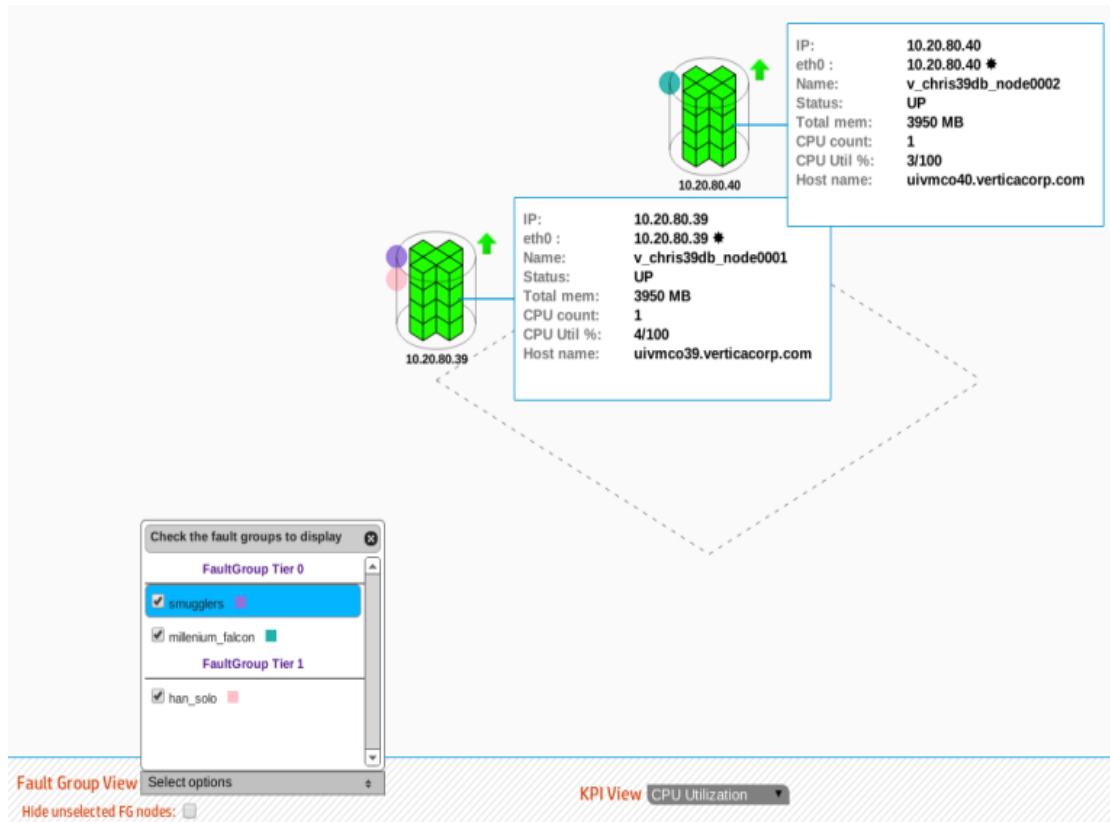
Nodes in a fault group have colored bubble attached to the upper left corner of the node. Each fault group has a unique color (until the number of fault groups exceeds the number of colors available, in which case MC recycles previously-used colors).

Vertica Analytics Platform supports complex, hierarchical fault groups of different shapes and sizes, so MC displays multiple fault group participation as a stack of different-colored bubbles, where the higher bubbles represent a lower-tiered fault group, which means that bubble is closer to the root, or parent, fault group, not the child or grandchild fault group. See [High Availability through Fault Groups](#) in the Concepts Guide for information about fault group hierarchy.

Example: simple fault group hierarchy

The following image shows two tiers of nodes in three fault groups with no nodes hidden. Although fault groups are designed for large clusters, the cluster shown below is intentionally small to make it easier to view details

If information in the node details box is hard to read, increase the zoom level.



Large Cluster

To support scaling of existing clusters into large clusters and improve control message performance, Vertica Analytics Platform delegates control message responsibilities to a subset of Vertica Analytics Platform nodes, called **control nodes**. Control nodes communicate with each other. Other cluster nodes are assigned to a control node, which they use for control message communications.

Note: A large cluster arrangement is tightly integrated with [elastic cluster](#) and [fault groups](#) to provide the cluster flexibility and reliability that you expect from Vertica Analytics Platform. See [High Availability With Fault Groups](#) in the Concepts Guide for details.

Control nodes on large clusters

On clusters of 120 or more nodes, a large cluster layout is necessary and enabled by default. Vertica Analytics Platform makes automatic control node assignments unless you use one of the following options:

| If you want to ... | Do this |
|--|--|
| Install a new cluster before you create a database | Run the Vertica Analytics Platform installation script with the <code>--large-cluster <integer></code> arguments. See the following topics for details: <ul style="list-style-type: none">• Installing HP Vertica with the install_vertica Script in the Installation Guide• Installing a Large Cluster in this guide |
| Expand an existing cluster for pre-existing databases | Use cluster management functions described in Defining and Realigning Control Nodes on an Existing Cluster |
| Change control nodes on an existing cluster | Use cluster management functions described in Defining and Realigning Control Nodes on an Existing Cluster |
| Influence the placement of control nodes in your cluster's physical layout | Define fault groups to configure Vertica Analytics Platform for your physical cluster layout. See Fault Groups for details. |

Control nodes on small clusters

If your cluster has fewer than 120 nodes, large cluster is neither necessary nor automatically applied. As a result, all nodes are control nodes. However, Vertica Analytics Platform lets you define control nodes on any sized cluster. Some environments, such as cloud deployments that might have higher network latency, could benefit from a smaller number of control nodes.

For details, see [Planning a Large Cluster Arrangement](#) and [Installing a Large Cluster](#).

Planning a Large Cluster Arrangement

In a large cluster layout, nodes form a correlated failure group, governed by their control node (the node that runs control messaging/spread). If a control node fails, all nodes in its host group also fail.

Planning the number of control nodes

Configuring a large cluster requires careful and thorough network planning. You must have a solid understanding of your network topology before you configure the cluster.

To assess how many cluster nodes should be control nodes, use the square root of the total number of nodes expected to be in the database cluster to help satisfy both data **K-Safety** and rack fault tolerance for the cluster. Depending on the result, you might need to adjust the number of control nodes to account for your physical hardware/rack count. For example, if you have 121 nodes (with a result of 11), and your nodes will be distributed across 8 racks, you might want to increase the number of control nodes to 16 so you have two control nodes per rack.

Specifying the number of control nodes

Vertica Analytics Platform provides different tools to let you define the number of control nodes, depending on your current configuration, described in [Installing a Large Cluster](#).

Consider the following scenarios in which cluster nodes are spread out among three racks:

- If you have three control nodes and all other nodes are evenly distributed among the three racks, you could set up one control node per rack.
- If you have five control nodes and three racks, you could specify two control nodes on two racks each and one control node on the final rack.
- If you have four control nodes and one rack has twice as many nodes as the other racks, you could specify two control nodes on the larger rack and one control node each on the other two racks, if applicable.

Installing a Large Cluster

Whether you are forming a new large cluster (adding all nodes for the first time) or scaling an existing cluster to a large cluster, Vertica Analytics Platform provides two methods that let you specify the number of control nodes (the nodes that run control messaging). See the following sections:

- [If you want to install a new large cluster](#)
- [If you want to expand an existing cluster](#)

If you want to install a new large cluster

On new cluster formations, where you are adding nodes for the first time by running the `install_vertica` script, you can specify that you want a large-cluster installation by passing the Vertica Analytics Platform installation script the `--large-cluster <integer>` argument. Vertica Analytics Platform selects the first `<integer>` number of hosts from the `--hosts` list (comma separated list of hosts) as control nodes and assigns the other hosts from the list to a control node based on a round-robin model.

Note: The size of the `--hosts` host list determines a large cluster layout, not the number of nodes you include in the database. If you specify 120 or more hosts in the list, and you do not specifically enable large cluster by providing the `--large-cluster` argument, Vertica Analytics Platform enables a large cluster arrangement and configures control nodes for you.

To help ensure that control nodes and their assigned nodes are aligned for the highest possible fault tolerance, you must list the hosts in the `--hosts host_list` in the proper order. For example, if you have four sets of hosts on four racks, the first four nodes in the `--hosts host_list` must be one host from each rack so you have one control node per rack. Then the list must consist of four hosts from each rack in line with the first four hosts. You'll continue to use this

pattern of host listing for all targeted hosts. See [Sample rack-based cluster hosts topology](#) below for examples.

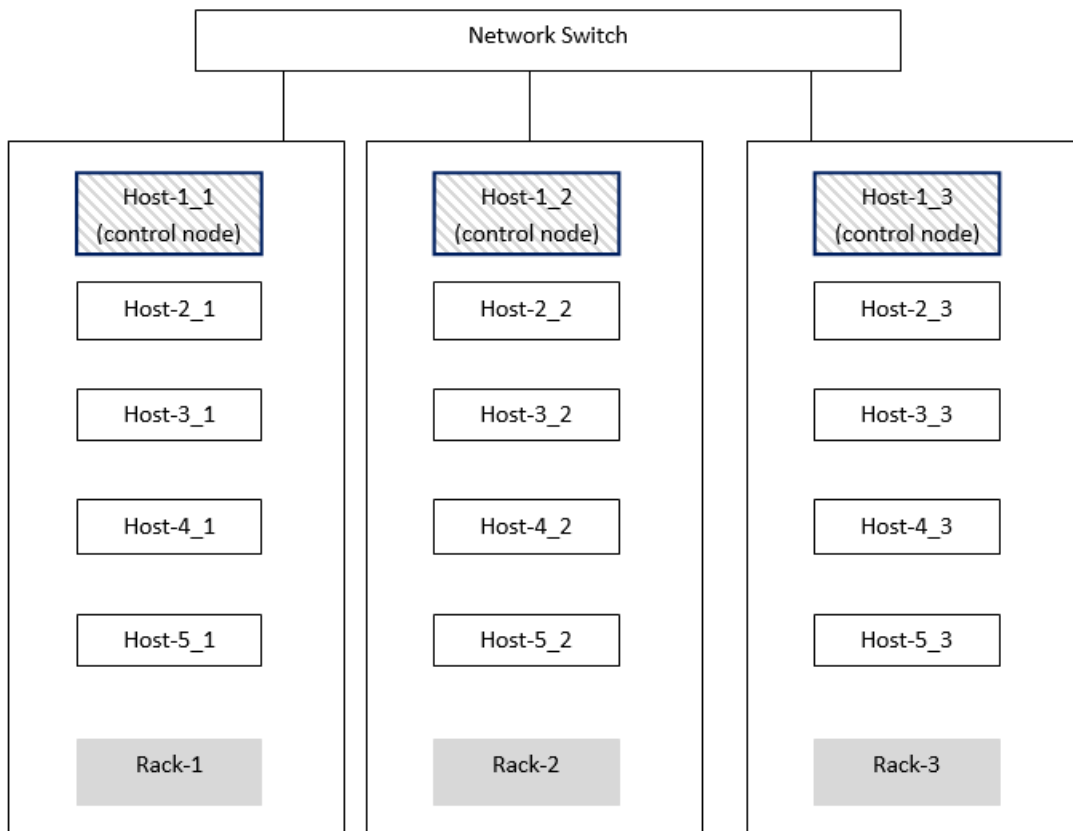
Tip: If you pass the `--large-cluster` argument a DEFAULT value instead of an `<integer>` value, Vertica Analytics Platform calculates a number of control nodes based on the total number of nodes specified in the `host_list` argument.

For more information, see the following topics:

- [Planning a Large Cluster Arrangement](#)
- [Installing HP Vertica with the `install_vertica` Script](#) in the Installation Guide

Sample rack-based cluster hosts topology

This example shows a simple multi-rack cluster layout, in which cluster nodes are evenly distributed across three racks, where each rack has one control node.



In the above example:

- Rack-1, Rack-2, and Rack-3 are managed by a single network switch
- Host-1_1, Host-1_2, and Host-1_3 are control nodes
- All hosts on Rack-1 are assigned to control node Host-1_1
- Hosts on Rack-2 are assigned to control node Host-1_2
- Hosts on Rack-3 are assigned to control node Host-1_3

In this scenario, if control node Host-1_1 or Rack-1 fails, the nodes in Rack-1 can communicate with control node Host-2_1, and the database stays up.

In the following `install_vertica` script fragment, note the order of the hosts in the `--hosts` list argument. The final arguments in the sample specifically enable large cluster and provide the number of control nodes (3):

```
... install_vertica --hosts Host-1-1,Host-1-2,Host-1-3,Host-2-1,Host-2-2,Host-2-3,Host-3-1,Host-3-2,Host-3-3,Host-4-1,Host-4-2,Host-4-3,Host-5-1,Host-5-2,Host-5-3 -rpm <vertica-p  
ackage-name> <other required options> --large-cluster 3
```

After the installation process completes, use the **Administration Tools** to create a database, an operation that generate an Vertica Analytics Platform database cluster environment with three control nodes, along with their respective associated hosts that reside on the same racks as the control node.

If you want to expand an existing cluster

When you add a node to an existing cluster, Vertica Analytics Platform places the new node in an appropriate location within the cluster ring and assigns it to a control node, based on the cluster's current allocations.

To provide you more flexibility and control over which nodes run spread, you can use the `SET_CONTROL_SET_SIZE(integer)` function. This function works like the installation script's `--large-cluster <integer>` option. See [Defining and Realigning Control Nodes on an Existing Cluster](#) for details.

Important: The Vertica Analytics Platform installation script cannot alter the database cluster.

Defining and Realigning Control Nodes on an Existing Cluster

This topic describes how to set up or change control node assignments on an existing cluster using a series of [cluster management functions](#). It assumes you already know how many control nodes the cluster needs for failover safety. See [Planning a Large Cluster Arrangement](#).

Note: If you are adding nodes for the first time, run the Vertica Analytics Platform installation

script using the `--large-cluster <integer>` argument. See [Installing HP Vertica with the install_vertica Script](#) in the Installation Guide.

The procedure below makes the following changes to the cluster:

- Specifies the number of nodes that run control messaging (spread).
- Assigns each non-control cluster node to a control node.
- Saves the new layout to the spread configuration file.
- Redistributes data across the cluster and ensures fault tolerance is realized.

How to set up control nodes on an existing cluster

Perform the following steps when you change the cluster—such as after adding, removing, or swapping nodes—to ensure the cluster maintains adequate control messaging distribution for failover safety. For more details, see "Control node assignment/realignment" in [Large Cluster Best Practices](#).

1. As the database administrator, log in to the **Administration Tools** and connect to the database.
2. Call the `SET_CONTROL_SET_SIZE(integer)` function with an integer argument to specify the number of control nodes; for example, 4:

```
=> SELECT SET_CONTROL_SET_SIZE(4);
```

3. Call the `REALIGN_CONTROL_NODES()` function without arguments; for example:

```
=> SELECT REALIGN_CONTROL_NODES();
```

4. Call the `RELOAD_SPREAD(true)` function to save changes to the spread configuration file:

```
=> SELECT RELOAD_SPREAD(true);
```

5. After the `RELOAD_SPREAD()` operation finishes, log back in to the **Administration Tools**, and restart the database.
6. Call the `REBALANCE_CLUSTER()` function to ensure data is distributed across the cluster:

```
=> SELECT REBALANCE_CLUSTER();
```

Important: You must run `REBALANCE_CLUSTER()` for fault tolerance to be realized. See also [Rebalancing Large Clusters](#).

See Also

- [Cluster Management Functions](#) in the SQL Reference Manual
- [Fault Groups](#)

Rebalancing Large Clusters

A rebalance operation redistributes your database projections' data across all nodes, ensures data is distributed correctly based on [fault groups](#) and [large cluster](#) automatic fault groups, refreshes projections, sets the **Ancient History Mark**, and drops projections that are no longer needed.

When to rebalance the cluster

Rebalancing is useful (or necessary) after you:

- Mark one or more nodes as ephemeral in preparation of removing them from the cluster.
- Add one or more nodes to the cluster, so that HP Vertica can populate the empty nodes with data.
- Remove one or more nodes from the cluster, so Vertica Analytics Platform can redistribute the data among the remaining nodes.
- Change the [scaling factor](#), which determines the number of storage containers used to store a projection across the database.
- Set the control node size or realign control nodes on a [large cluster](#) layout.
- Specify more than 120 nodes in your initial Vertica Analytics Platform cluster configuration.
- Add nodes to or remove nodes from a [fault group](#).

You must be a database administrator to rebalance data across the cluster.

How to rebalance the cluster

You rebalance the cluster using [SQL functions](#), in particular the `REBALANCE_CLUSTER()` function. Call `REBALANCE_CLUSTER()` after you have completed the last add/remove operation.

How long will rebalance take?

A rebalance operation can take some time, depending on the number of projections and the amount of data they contain. HP recommends that you allow the process to complete uninterrupted. If you must cancel the operation, call the [CANCEL_REBALANCE_CLUSTER\(\)](#) function.

See Also

- [Rebalancing Data Using SQL Functions](#)
- [Rebalancing Data Across Nodes](#)
- [Rebalancing Data Using the Administration Tools UI](#)

Expanding the Database for Large Cluster

If you have an existing database cluster that you want to expand to a large cluster (more than 120 nodes), follow these steps:

1. Log in to the **Administration Tools** as the database administrator and stop the database.
2. As root or a user with sudo privileges, open a BASH shell and run the `install_vertica` script with the `--add-hosts` argument, providing a comma-separated list of hosts you want to add to an existing HP Vertica cluster. See [Installing HP Vertica with the install_vertica Script](#) in the Installation Guide.
3. Exit the shell and re-establish a vsql connection as the database administrator.
4. Log in to the Administration Tools and start the database.
5. Use the Administration Tools Advanced Menu > Cluster Management > Add Hosts option to add the standby hosts you created in Step 2.
6. Run `SET_CONTROL_SET_SIZE(integer)` to specify the number of control nodes you want on the cluster. See [Defining and Realigning Control Nodes on an Existing Cluster](#).
7. Optionally create fault groups to further define the layout of the control nodes within the physical cluster. See [Fault Groups](#).

Monitoring Large Clusters

You can monitor large cluster traits by querying the following system tables:

- [V_CATALOG.LARGE_CLUSTER_CONFIGURATION_STATUS](#)—Shows the current spread hosts and the control designations in the Catalog so you can see if they match.
- [V_MONITOR.CRITICAL_HOSTS](#)—Lists the hosts whose failure would cause the database to

become unsafe and force a shutdown.

Tip: The `CRITICAL_HOSTS` view is especially useful for large cluster arrangements. For non-large clusters, query the `CRITICAL_NODES` table.

You might also want to monitor the following tables:

- `V_CATALOG.FAULT_GROUPS`—Provides information about views created on the database cluster
- `V_CATALOG.CLUSTER_LAYOUT`—Shows the relative position of the actual arrangement of the nodes participating in the data business and the fault groups that affect them.

Note: Ephemeral nodes do not appear in this view (or in the cluster layout ring) because they hold no resident data.

Large Cluster Best Practices

Keep the following best practices in mind when planning and managing a large cluster implementation.

Planning the number of control nodes

To assess how many cluster nodes should be control nodes, use the square root of the total number of nodes expected to be in the database cluster to help satisfy both data **K-Safety** and rack fault tolerance for the cluster. Depending on the result, you might need to adjust the number of control nodes to account for your physical hardware/rack count. For example, if you have 121 nodes (with a result of 11), and your nodes will be distributed across 8 racks, you might want to increase the number of control nodes to 16 so you have two control nodes per rack.

See [Planning a Large Cluster Arrangement](#).

Control node assignment/realignment

After you specify the number of control nodes, you must update the control host's (spread) configuration files to reflect the catalog change. Some [cluster management functions](#) might require you to run other functions or restart the database or both.

If, for example, you drop a control node, cluster nodes that point to it are reassigned to another control node. If that node fails, all the nodes assigned to it also fail, so you need to use the Administration Tools to restart the database. In this scenario, you'd call the `REALIGN_CONTROL_NODES()` and `RELOAD_SPREAD(true)` functions, which notify nodes of the changes and realign fault groups. Calling `RELOAD_SPREAD(true)` connects an existing cluster node to a newly-assigned control node. On the other hand, if you run `REALIGN_CONTROL_NODES()` multiple times in a row, the layout does not change beyond the initial setup, so you don't need to restart the database. But if you

add or drop a node and then run `REALIGN_CONTROL_NODES()`, the function call could change many node assignments.

Here's what happens with the control node assignments when you add or drop nodes, whether those nodes are control nodes or non-control nodes:

- **If you add a cluster node**—Vertica Analytics Platform assigns a control node to the newly-added node based on the current cluster configuration. If the new node joins a fault group, it is assigned to a control node from that fault group and requires a database restart to reconnect to that control node.
- **If you drop a non-control node**—Vertica Analytics Platform quietly drops the cluster node. This operation could change the cluster and spread layout, so you should always call `REBALANCE_CLUSTER()` after you drop a node.
- **If you drop a control node**—All nodes assigned to the control node go down. In large cluster implementations, however, the database remains up because the down nodes are not buddies with other cluster nodes.

After you drop a control node, you'll have $(n-1)$ control nodes. You must call `REALIGN_CONTROL_NODES()` to reset the cluster so it has n control nodes, which might or might not be the same number as before you dropped the control node. Remaining nodes are assigned new control nodes. In this operation, Vertica Analytics Platform makes control node assignments based on the cluster layout. When it makes the new assignments, it respects user-defined fault groups, if any, which you can view by querying the `V_CATALOG.CLUSTER_LAYOUT` system table, a view that also lets you see the proposed new layout for nodes in the cluster. If you want to influence the layout of control nodes in the cluster, you should define fault groups.

For more information, see [Defining and Realigning Control Nodes on an Existing Cluster](#) and [Rebalancing Large Clusters](#).

Allocate standby nodes

Have as many standby nodes available as you can, ideally on racks you are already using in the cluster. If a node suffers a non-transient failure, use the **Administration Tools** "Replace Host" utility to swap in a standby node.

Standby node availability is especially important for control nodes. If you are swapping a node that's a control node, all nodes assigned to the control node's host grouping will need to be taken offline while you swap in the standby node. For details on node replacement, see [Replacing Nodes](#).

Plan for cluster growth

If you plan to expand an existing cluster to 120 or more nodes, you can configure the number of control nodes for the cluster after you add the new nodes. See [Defining and Realigning Control Nodes](#).

Write custom fault groups

When you deploy a large cluster, Vertica Analytics Platform automatically creates fault groups around control nodes, placing nodes that share a control node into the same fault group. Alternatively, you can specify which cluster nodes should reside in a particular correlated failure group and share a control node. See [High Availability through Fault Groups](#) in the Concepts Guide.

Use segmented projections

On large cluster setups, minimize the use of unsegmented projections in favor of segmented projections. When you use segmented projections, Vertica Analytics Platform creates **buddy projections** and distributes copies of segmented projections across database nodes. If a node fails, data remains available on the other cluster nodes.

Use the Database Designer

HP recommends that you use the **Database Designer** to create your physical schema. If you choose to design projections manually, you should segment large tables across all database nodes and replicate (unsegment) small table projections on all database nodes.

Elastic Cluster

You can scale your cluster up or down to meet the needs of your database. The most common case is to add nodes to your database cluster to accommodate more data and provide better query performance. However, you can scale down your cluster if you find that it is overprovisioned or if you need to divert hardware for other uses.

You scale your cluster by adding or removing nodes. Nodes can be added or removed without having to shut down or restart the database. After adding a node or before removing a node, HP Vertica begins a rebalancing process that moves data around the cluster to populate the new nodes or move data off of nodes about to be removed from the database. During this process data may also be exchanged between nodes that are not being added or removed to maintain robust intelligent **K-safety**. If HP Vertica determines that the data cannot be rebalanced in a single iteration due to a lack of disk space, then the rebalance is done in multiple iterations.

To help make data rebalancing due to cluster scaling more efficient, HP Vertica locally segments data storage on each node so it can be easily moved to other nodes in the cluster. When a new node is added to the cluster, existing nodes in the cluster give up some of their data segments to populate the new node and exchange segments to keep the number of nodes that any one node depends upon to a minimum. This strategy keeps to a minimum the number of nodes that may become critical when a node fails (see **Critical Nodes/K-safety**). When a node is being removed from the cluster, all of its storage containers are moved to other nodes in the cluster (which also relocates data segments to minimize nodes that may become critical when a node fails). This method of breaking data into portable segments is referred to as elastic cluster, since it makes enlarging or shrinking the cluster easier.

The alternative to elastic cluster is to resegment all of the data in the projection and redistribute it to all of the nodes in the database evenly any time a node is added or removed. This method requires more processing and more disk space, since it requires all of the data in all projections to essentially be dumped and reloaded.

The Elastic Cluster Scaling Factor

In new installs, each node has a "scaling factor" number of local segments. Rebalance efficiently redistributes data by relocating local segments provided that, after nodes are added or removed, there are sufficient local segments in the cluster to redistribute the data evenly (determined by [MAXIMUM_SKEW_PERCENT](#)). For example, if the scaling factor = 8, and there are initially 5 nodes, then there are a total of 40 local segments cluster wide. If two additional nodes are added to bring the total to 7 nodes, relocating local segments would place 5 such segments on 2 nodes and 6 such segments on 5 nodes, which is roughly a 20% skew. Rebalance chooses this course of action only if the resulting skew is less than the allowed threshold, as determined by [MAXIMUM_SKEW_PERCENT](#). Otherwise, segmentation space (and hence data, if uniformly distributed over this space) is evenly distributed among the 7 nodes and new local segment boundaries are drawn for each node, such that each node again has 8 local segments.

Note: By default, the scaling factor only has an effect while HP Vertica rebalances the database. While rebalancing, each node breaks the projection segments it contains into storage containers, which it then moves to other nodes if necessary. After rebalancing, the data is recombined into **ROS** containers. It is possible to have HP Vertica always group data into storage containers. See [Local Data Segmentation](#) for more information.

Enabling and Disabling Elastic Cluster

You enable and disable elastic cluster using functions. See the entries for the [ENABLE_ELASTIC_CLUSTER](#) and [DISABLE_ELASTIC_CLUSTER](#) functions in the SQL Reference Manual.

Note: An elastic projection (a segmented projection created when Elastic Cluster is enabled) created with a modularhash segmentation expression uses hash instead.

Query the [ELASTIC_CLUSTER](#) system table to determine if elastic cluster is enabled:

```
=> select is_enabled from ELASTIC_CLUSTER;
is_enabled
-----
t
(1 row)
```

Scaling Factor Defaults

The default scaling factor is "4" for new installs of HP Vertica and for upgraded install of HP Vertica that had local segments disabled. Versions of HP Vertica prior to 6.0 had local segments disabled by default. The scaling factor is not changed during upgrade on databases upgraded to version 6.0 if local segments were enabled.

Note: Databases created with versions of HP Vertica earlier than version 5.0 have a scaling factor of 0, which disables elastic cluster. This ensures that HP Vertica handles projection segmentation the way it did prior to version 5.0. If you want your older database to have better scaling performance, you need to manually set a scaling factor to enable the new storage segmenting behavior.

Viewing Scaling Factor Settings

To view the scaling factor, query the ELASTIC_CLUSTER table:

```
=> SELECT scaling_factor FROM ELASTIC_CLUSTER;
scaling_factor
-----
                4
(1 row)

=> SELECT SET_SCALING_FACTOR(6);
SET_SCALING_FACTOR
-----
SET
(1 row)

=> SELECT scaling_factor FROM ELASTIC_CLUSTER;
scaling_factor
-----
                6
(1 row)
```

Setting the Scaling Factor

The scaling factor determines the number of storage containers used to store a projection across the database. Use the [SET_SCALING_FACTOR](#) function to change your database's scaling factor. The scaling factor can be an integer between 1 and 32.

Note: Setting the scaling factor value too high can cause nodes to create too many small container files, greatly reducing efficiency and potentially causing a "Too many ROS containers" error (also known as "ROS pushback"). The scaling factor should be set high enough so that rebalance can transfer local segments to satisfy the skew threshold, but small enough that the number of storage containers does not exceed ROS pushback. The number of storage containers should be greater than or equal to the number of partitions multiplied by the number local of segments ($\# \text{ storage containers} \geq \# \text{ partitions} * \# \text{ local segments}$).

```
=> SELECT SET_SCALING_FACTOR(12);
SET_SCALING_FACTOR
-----
SET
(1 row)
```

Local Data Segmentation

By default, the scaling factor only has an effect when HP Vertica rebalances the database. During rebalancing, nodes break the projection segments they contain into storage containers which they can quickly move to other nodes.

This process is more efficient than re-segmenting the entire projection (in particular, less free disk space is required), but it still has significant overhead, since storage containers have to be separated into local segments, some of which are then transferred to other nodes. This overhead is not a problem if you rarely add or remove nodes from your database.

However, if your database is growing rapidly and is constantly busy, you may find the process of adding nodes becomes disruptive. In this case, you can enable local segmentation, which tells HP Vertica to always segment its data based on the scaling factor, so the data is always broken into containers that are easily moved. Having the data segmented in this way dramatically speeds up the process of adding or removing nodes, since the data is always in a state that can be quickly relocated to another node. The rebalancing process that HP Vertica performs after adding or removing a node just has to decide which storage containers to relocate, instead of first having to first break the data into storage containers.

Local data segmentation increases the number of storage containers stored on each node. This is not an issue unless a table contains many partitions. For example, if the table is partitioned by day and contains one or more years. If local data segmentation is enabled, then each of these table partitions is broken into multiple local storage segments, which potentially results in a huge number of files which can lead to ROS "pushback." Consider your table partitions and the effect enabling local data segmentation may have before enabling the feature.

Enabling and Disabling Local Segmentation

To enable local segmentation, use the [ENABLE_LOCAL_SEGMENTS](#) function. To disable local segmentation, use the [DISABLE_LOCAL_SEGMENTATION](#) function:

```
=> SELECT ENABLE_LOCAL_SEGMENTS();
ENABLE_LOCAL_SEGMENTS
-----
ENABLED
(1 row)

=> SELECT is_local_segment_enabled FROM elastic_cluster;
is_enabled
-----
t
(1 row)

=> SELECT DISABLE_LOCAL_SEGMENTS();
DISABLE_LOCAL_SEGMENTS
-----
DISABLED
(1 row)
```

```
=> SELECT is_local_segment_enabled FROM ELASTIC_CLUSTER;  
is_enabled  
-----  
f  
(1 row)
```

Elastic Cluster Best Practices

The following are some best practices with regard to local segmentation and upgrading pre-5.0 databases.

Note: You should always perform a database backup before and after performing any of the operations discussed in this topic. You need to back up before changing any elastic cluster or local segmentation settings to guard against a hardware failure causing the rebalance process to leave the database in an unusable state. You should perform a full backup of the database after the rebalance procedure to avoid having to rebalance the database again if you need to restore from a backup.

When to Enable Local Data Segmentation

[Local Data Segmentation](#) can significantly speed up the process of resizing your cluster. You should enable local data segmentation if

- your database does not contain tables with hundreds partitions.
- the number of nodes in the database cluster is a power of two.
- you plan to expand or contract the size of your cluster.

Local segmentation can result in an excessive number of storage containers with tables that have hundreds of partitions, or in clusters with a non-power-of-two number of nodes. If your database has these two features, take care when enabling local segmentation.

Upgraded Database Consideration

Databases created using a version of HP Vertica earlier than version 5.0 do not have elastic cluster enabled by default. If you expect to expand or contract the database in the future, you may benefit from enabling elastic cluster by setting a scaling factor. There are two strategies you can follow:

- Enable elastic cluster now, and rebalance the database. This may take a significant amount of time to complete, and make consume up to 50% of the free disk space on the nodes in the database, since all of the segmented projections are re-written. However, afterwards, adding and removing nodes will take less time.

- Wait until you need to resize the cluster, then enable elastic cluster just before adding or removing nodes. Changing the setting does not make the resizing of the cluster any faster, but later resize operations will be faster.

Which method you choose depends on your specific circumstances. If you might resize your database on short notice (for example, you may need to load a very large amount of data at once), you can choose to schedule the downtime needed to enable elastic cluster and rebalance the database to enable elastic cluster sooner, so the actual add or remove node process will occur faster.

If you choose to enable elastic cluster for your database, you should consider whether you want to enable local data segmentation at the same time. If you choose to enable local data segmentation at a later time, you will need to rebalance the database again, which is a lengthy process.

Monitoring Elastic Cluster Rebalancing

HP Vertica includes system tables that can be used to monitor the rebalance status of an elastic cluster and gain general insight to the status of elastic cluster on your nodes.

- The [REBALANCE_TABLE_STATUS](#) table provides general information about a rebalance. It shows, for each table, the amount of data that has been separated, the amount that is currently being separated, and the amount to be separated. It also shows the amount of data transferred, the amount that is currently being transferred, and the remaining amount to be transferred (or an estimate if storage is not separated).

Note: If multiple rebalance methods were used for a single table (for example, the table has unsegmented and segmented projections), the table may appear multiple times - once for each rebalance method.

- [REBALANCE_PROJECTION_STATUS](#) can be used to gain more insight into the details for a particular projection that is being rebalanced. It provides the same type of information as above, but in terms of a projection instead of a table.

In each table, *separated_percent* and *transferred_percent* can be used to determine overall progress.

Historical Rebalance Information

Historical information about work completed is retained, so use the predicate "*where is_latest*" to restrict the output to only the most recent or current rebalance activity. The historical data may include information about dropped projections or tables. If a table or projection has been dropped and information about the anchor table is not available, then NULL is displayed for the *table_id* and "<unknown>" is displayed for the *table_name*. Information on dropped tables is still useful, for example, in providing justification for the duration of a task.

Adding Nodes

There are many reasons for adding one or more nodes to an installation of HP Vertica:

- **Increase system performance.** Add additional nodes due to a high query load or load latency or increase disk space without adding storage locations to existing nodes.

Note: The database response time depends on factors such as type and size of the application query, database design, data size and data types stored, available computational power, and network bandwidth. Adding nodes to a database cluster does not necessarily improve the system response time for every query, especially if the response time is already short, e.g., less than 10 seconds, or the response time is not hardware bound.

- **Make the database K-safe (K-safety=1)** or increase K-safety to 2. See [Failure Recovery](#) for details.
- **Swap a node for maintenance.** Use a spare machine to temporarily take over the activities of an existing node that needs maintenance. The node that requires maintenance is known ahead of time so that when it is temporarily removed from service, the cluster is not vulnerable to additional node failures.
- **Replace a node.** Permanently add a node to remove obsolete or malfunctioning hardware.

Important: : If you installed HP Vertica on a single node without specifying the IP address or hostname (or you used `localhost`), you cannot expand the cluster. You must reinstall HP Vertica and specify an IP address or hostname that is not `localhost/127.0.0.1`.

Adding nodes consists of the following general tasks:

1. [Back up the database.](#)

HP strongly recommends that you back up the database before you perform this significant operation because it entails creating new projections, refreshing them, and then deleting the old projections. See [Backing Up and Restoring the Database](#) for more information.

The process of migrating the projection design to include the additional nodes could take a while; however during this time, all user activity on the database can proceed normally, using the old projections.

2. Configure the hosts you want to add to the cluster.

See [Before you Install HP Vertica](#) in the Installation Guide. You will also need to edit the hosts configuration file on all of the existing nodes in the cluster to ensure they can resolve the new host.

3. [Add one or more hosts to the cluster.](#)
4. [Add the hosts](#) you added to the cluster (in step 3) to the database.

Note: When you add a "host" to the database, it becomes a "node." You can add nodes to your database using either the **Administration Tools** or the **Management Console** (See [Monitoring HP Vertica Using Management Console.](#))

After you add one or more nodes to the database, HP Vertica automatically distributes updated configuration files to the rest of the nodes in the cluster and starts the process of rebalancing data in the cluster. See [Rebalancing Data Across Nodes](#) for details.

Adding Hosts to a Cluster

After you have backed up the database and configured the hosts you want to add to the cluster, you can now add hosts to the cluster using the `update_vertica` script.

You can use MC to add standby nodes to a database, but you cannot add hosts to a cluster using MC.

Prerequisites and Restrictions

- If you installed HP Vertica on a single node without specifying the IP address or hostname (you used localhost), it is not possible to expand the cluster. You must reinstall HP Vertica and specify an IP address or hostname.
- If your database has more than one node already, you can add a node without stopping the server. However, if you are adding a node to a single-node installation, then you must shut down both the database and spread. If you do not, the system returns an error like the following:

```
$ sudo /opt/vertica/sbin/update_vertica --add-hosts node05 --rpm vertica_7.0.x.x86_64.RHEL5.rpm
Vertica 7.0.x Installation Tool
Starting installation tasks...
Getting system information for cluster (this may take a while)...
Spread is running on ['node01']. HP Vertica and spread must be stopped before adding nodes to a 1 node cluster.
Use the admin tools to stop the database, if running, then use the following command to stop spread:
    /etc/init.d/spread stop (as root or with sudo)
Installation completed with errors.
Installation failed.
```

Procedure to Add Hosts

From one of the existing cluster hosts, run the `update_vertica` script with a minimum of the `--add-hosts host(s)` parameter (where *host(s)* is the hostname or IP address of the system(s) that you are adding to the cluster) and the `--rpm` or `--deb` parameter:

```
# /opt/vertica/sbin/update_vertica --add-hosts host(s) --rpm package
```

Note: See [Installing with the Script](#) for the full list of parameters. You must also provide the same options you used when originally installing the cluster.

The `update_vertica` script uses all the same options as `install_vertica` and:

- Installs the HP Vertica RPM on the new host.
- Performs post-installation checks, including RPM version and N-way network connectivity checks.
- Modifies spread to encompass the larger cluster.
- Configures the [Administration Tools](#) to work with the larger cluster.

Important Tips:

- A host can be specified by the hostname or IP address of the system you are adding to the cluster. However, internally HP Vertica stores all host addresses as IP addresses.
- Do not use include spaces in the hostname/IP address list provided with `--add-hosts` if you specified more than one host.
- If a package is specified with `--rpm/--deb`, and that package is newer than the one currently installed on the existing cluster, then, HP Vertica first installs the new package on the existing cluster hosts before the newly-added hosts.
- Use the same command line parameters for the database administrator username, password, and directory path you used when you installed the cluster originally. Alternatively, you can create a properties file to save the parameters during install and then re-using it on subsequent install and update operations. See [Installing HP Vertica Silently](#).
- If you are installing using `sudo`, the database administrator user (`dbadmin`) must already exist on the hosts you are adding and must be configured with passwords and home directory paths identical to the existing hosts. HP Vertica sets up passwordless ssh from existing hosts to the new hosts, if needed.
- If you initially used the `--point-to-point` option to configure spread to use direct, point-to-point communication between nodes on the subnet, then use the `--point-to-point` option whenever you run `install_vertica` or `update_vertica`. Otherwise, your cluster's configuration is reverted to the default (*broadcast*), which may impact future databases.

Examples:

```
--add-hosts host01 --rpm  
--add-hosts 192.168.233.101  
--add-hosts host02,host03
```

Adding Nodes to a Database

Once you have added one or more hosts to the cluster, you can add them as nodes to the database.

You can add nodes to a database using either these methods:

- The Management Console interface
- The Administration Tools interface

To Add Nodes to a Database Using MC

Only nodes in STANDBY state are eligible for addition. STANDBY nodes are nodes included in the cluster but not yet assigned to the database.

You add nodes to a database on MC's **Manage** page. Click the node you want to act upon, and then click Add node in the Node List.

When you add a node, the node icon in the cluster view changes color from gray (empty) to green as the node comes online. Additionally, a task list displays detailed progress of the node addition process.

To Add Nodes to a Database Using the Administration Tools:

1. Open the Administration Tools. (See [Using the Administration Tools.](#))
2. On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If it is not, start it.
3. From the **Main Menu**, select **Advanced Tools Menu** and click **OK**.
4. In the **Advanced Menu**, select **Cluster Management** and click **OK**.
5. In the **Cluster Management** menu, select **Add Host(s)** and click **OK**.
6. Select the database to which you want to add one or more hosts, and then select **OK**.

A list of unused hosts is displayed.

7. Select the hosts you want to add to the database and click **OK**.
8. When prompted, click **Yes** to confirm that you want to add the hosts.
9. When prompted, enter the password for the database, and then select **OK**.
10. When prompted that the hosts were successfully added, select **OK**.

11. HP Vertica now automatically starts the rebalancing process to populate the new node with data. When prompted, enter the path to a temporary directory that the Database Designer can use to rebalance the data in the database and select **OK**.
12. Either press enter to accept the default **K-Safety** value, or enter a new higher value for the database and select **OK**.
13. Select whether HP Vertica should immediately start [rebalancing the database](#), or whether it should create a script to rebalance the database later. You should select the option to automatically start rebalancing unless you want to delay rebalancing until a time when the database has a lower load. If you choose to automatically rebalance the database, the script is still created and saved where you can use it later.
14. Review the summary of the rebalancing process and select **Proceed**.
15. If you chose to automatically rebalance, the rebalance process runs. If you chose to create a script, the script is generated and saved. In either case, you are shown a success screen, and prompted to select **OK** to end the Add Node process.

Removing Nodes

Although less common than adding a node, permanently removing a node is useful if the host system is obsolete or over-provisioned.

Note: You cannot remove nodes if your cluster would not have the minimum number of nodes required to maintain your database's current **K-safety** level (3 nodes for a database with a K-safety level of 1, and 5 nodes for a K-safety level of 2). If you really wish to remove the node or nodes from the database, you first must reduce the K-safety level of your database.

Removing one or more nodes consists of the following general steps:

1. [Back up the database.](#)

HP recommends that you back up the database before performing this significant operation because it entails creating new projections, deleting old projections, and reloading data.

2. [Lower the K-safety of your database](#) if the cluster will not be large enough to support its current level of **K-safety** after you remove nodes.
3. [Remove the hosts from the database.](#)
4. [Remove the nodes from the cluster](#) if they are not used by any other databases.

Lowering the K-Safety Level to Allow for Node Removal

A database with a K-Safety level of 1 requires at least three nodes to operate, and a database with a K-Safety level 2 requires at least 5 nodes to operate. To remove a node from a cluster that is at the minimum number of nodes for its database's K-Safety level, you must first lower the K-Safety level using the [MARK_DESIGN_KSAFE](#) function.

Note: HP does not recommend lowering the K-safety level of a database to 0, since doing so eliminates HP Vertica's fault tolerance features. You should only use this procedure to move from a K-safety level of 2 to 1.

To lower the K-Safety level of the database:

1. Connect to the database, either through the Administration Tools or via **vsqll**.
2. Enter the command: `SELECT MARK_DESIGN_KSAFE(n);` where *n* is the new K-Safety level for the database (0 if you are reducing the cluster to below 3 nodes, 1 if you are reducing the cluster to 3 or 4 nodes).

Removing Nodes From a Database

You can remove nodes from a database using either these methods:

- The Management Console interface
- The Administration Tools interface

Prerequisites

- The node must be empty, in other words there should be no projections referring to the node. Ensure you have followed the steps listed in [Removing Nodes](#) to modify your database design.
- The database must be UP.
- You cannot drop nodes that are critical for **K-safety**. See [Lowering the K-Safety Level to Allow for Node Removal](#).

Remove Unused Hosts From the Database Using MC

You remove nodes from a database cluster on MC's **Manage** page. Click the node you want to act upon, and then click **Remove node** in the Node List.

Using MC, you can remove only nodes that are part of the database cluster and which show a state of DOWN (red). When you remove a node, its color changes from red to clear and MC updates its state to STANDBY. You can add STANDBY nodes back to the database later.

Remove Unused Hosts From the Database Using the Administration Tools

To remove unused hosts from the database using the Administration Tools:

1. Open the Administration Tools. See [Using the Administration Tools](#) for information about accessing the Administration Tools.
2. On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If the database isn't running, start it.
3. From the **Main Menu**, select **Advanced Tools Menu**, and then select **OK**.
4. In the **Advanced** menu, select **Cluster Management**, and then select **OK**.
5. In the **Cluster Management** menu, select **Remove Host(s) from Database**, and then select **OK**.
6. When warned that you must redesign your database and create projections that exclude the hosts you are going to drop, select **Yes**.
7. Select the database from which you want to remove the hosts, and then select **OK**.

A list of all the hosts that are currently being used is displayed.

8. Select the hosts you want to remove from the database, and then select **OK**.
9. When prompted, select **OK** to confirm that you want to remove the hosts. HP Vertica begins the process of rebalancing the database and removing the node or nodes.
10. When informed that the hosts were successfully removed, select **OK**.

Removing Hosts From a Cluster

If a host that you removed from the database is not used by any other database, you can remove it from the cluster using the `update_vertica` script. You can leave the database running (UP) during this operation.

You can [remove hosts from a database](#) on the MC interface, but you cannot remove those hosts from a cluster.

Prerequisites

The host must not be used by any database

Procedure to Remove Hosts

From one of the hosts in the cluster, run `update_vertica` with the `--remove-hosts` switch and provide a comma-separated list of hosts to remove from an existing HP Vertica cluster. A host can be specified by the hostname or IP address of the system.:

```
# /opt/vertica/sbin/update_vertica ---remove-hosts host
```

For example:

```
# /opt/vertica/sbin/update_vertica --remove-hosts host01
```

Note: See [Installing with the Script](#) for the full list of parameters.

The `update_vertica` script uses all the same options as `install_vertica` and:

- Modifies the spread to match the smaller cluster.
- Configures the **Administration Tools** to work with the smaller cluster.

Important Tips:

- Do not include spaces in the hostname list provided with `--remove-hosts` if you specified more than one host.

- If a new RPM is specified with `--rpm`, then HP Vertica will first install it on the existing cluster hosts before proceeding.
- Use the same command line parameters as those used when you installed the original cluster. Specifically if you used non-default values for the database administrator username, password, or directory path, provide the same when you remove hosts; otherwise, the procedure fails. Consider creating a properties file in which you save the parameters during the installation, which you can reuse on subsequent install and update operations. See [Installing HP Vertica Silently](#).

Examples:

```
--remove-hosts host01  
--remove-hosts 192.168.233.101  
-R host01
```

Replacing Nodes

If you have a **K-Safe** database, you can replace nodes, as necessary, without bringing the system down. For example, you might want to replace an existing node if you:

- Need to repair an existing host system that no longer functions and restore it to the cluster
- Want to exchange an existing host system for another more powerful system

Note: HP Vertica does not support replacing a node on a K-safe=0 database. Use the procedures to [add](#) and [remove](#) nodes instead.

The process you use to replace a node depends on whether you are replacing the node with:

- A host that uses the same name and IP address
- A host that uses a different name and IP address

Prerequisites

- Configure the replacement hosts for HP Vertica. See [Before you Install HP Vertica](#) in the Installation Guide.
- Read the Important **Tips** sections under [Adding Hosts to a Cluster](#) and [Removing Hosts From a Cluster](#).
- Ensure that the database administrator user exists on the new host and is configured identically to the existing hosts. HP Vertica will setup passwordless ssh as needed.
- Ensure that directories for Catalog Path, Data Path, and any storage locations are added to the database when you create it and/or are mounted correctly on the new host and have read and write access permissions for the database administrator user. Also ensure that there is sufficient disk space.
- Follow the best practice procedure below for introducing the failed hardware back into the cluster to avoid spurious full-node rebuilds.

Best Practice for Restoring Failed Hardware

Following this procedure will prevent HP Vertica from misdiagnosing missing disk or bad mounts as data corruptions, which would result in a time-consuming, full-node recovery.

If a server fails due to hardware issues, for example a bad disk or a failed controller, upon repairing the hardware:

1. Reboot the machine into runlevel 1, which is a root and console-only mode.

Runlevel 1 prevents network connectivity and keeps HP Vertica from attempting to reconnect to the cluster.

2. In runlevel 1, validate that the hardware has been repaired, the controllers are online, and any RAID recover is able to proceed.

Note: You do not need to initialize RAID recover in runlevel 1; simply validate that it can recover.

3. Once the hardware is confirmed consistent, only then reboot to runlevel 3 or higher.

At this point, the network activates, and HP Vertica rejoins the cluster and automatically recovers any missing data. Note that, on a single-node database, if any files that were associated with a projection have been deleted or corrupted, HP Vertica will delete all files associated with that projection, which could result in data loss.

Replacing a Node Using the Same Name and IP Address

To replace a node with a host system that has the same IP address and host name as the original:

1. [Backing Up and Restoring the Database.](#)
2. From a functioning node in the cluster, run the `install_vertica` script with the `-s` and `-r` parameters. Additionally, use the same additional install parameters that were used when the cluster was originally installed.

```
# /opt/vertica/sbin/install_vertica --hosts host --rpm rpm_package
```

Where `host` is the hostname or IP address of the system you are restoring to the cluster; for example:

```
--hosts host01  
--hosts 192.168.233.101
```

`--rpm` is the name of the rpm package; for example `--rpm vertica_7.0.x.x86_64.RHEL5.rpm`

The installation script verifies system configuration and that HP Vertica, spread, and the Administration Tools metadata are installed on the host.

3. On the new node, create catalog and data directories (unless they both reside in the same top-level directory, then you just need to create the one directory). These are the same top-level directories you specified when creating the database.

Note: You can find the directories used for catalog and data storage by querying the [V_MONITOR.DISK_STORAGE](#) system table. You need to create the directories up to the `v_database_node00xx` portion of the data and catalog path. For example, if the catalog storage location is `/home/dbadmin/vmart/v_vmart_node0001_catalog/Catalog`, you would need to create the `/home/dbadmin/vmart` directory to store the catalog.

4. Use the Administration Tools to restart the host you just replaced.

The node automatically joins the database and recovers its data by querying the other nodes within the database. It then transitions to an UP state.

Note: Do not connect two hosts with the same name and IP address to the same network. If this occurs, traffic is unlikely to be routed properly.

Replacing a Failed Node Using a node with Different IP Address

Replacing a failed node with a host system that has a different IP address from the original consists of the following steps:

1. [Back up the database.](#)

HP recommends that you back up the database before you perform this significant operation because it entails creating new projections, deleting old projections, and reloading data.

2. Add the new host to the cluster. See [Adding Hosts to a Cluster](#).
3. If HP Vertica is still running in the node being replaced, then use the Administration Tools to *Stop Vertica on Host* on the host being replaced.
4. Use the Administration Tools to [replace the original host](#) with the new host. . If you are using more than one database, replace the original host in all the databases in which it is used. See [Replacing Hosts](#).
5. Use the procedure in [Distributing Configuration Files to the New Host](#) to transfer metadata to the new host.
6. [Remove the host from the cluster.](#)
7. Use the Administration Tools to restart HP Vertica on the host. On the **Main Menu**, select **Restart Vertica on Host**, and click **OK**. See [Starting the Database](#) for more information.

Once you have completed this process, the replacement node automatically recovers the data that was stored in the original node by querying other nodes within the database.

Replacing a Functioning Node Using a Different Name and IP Address

Replacing a node with a host system that has a different IP address and host name from the original consists of the following general steps:

1. [Back up the database.](#)

HP recommends that you back up the database before you perform this significant operation because it entails creating new projections, deleting old projections, and reloading data.

2. [Add the replacement hosts to the cluster.](#)

At this point, both the original host that you want to remove and the new replacement host are members of the cluster.

3. Use the Administration Tools to *Stop Vertica on Host* on the host being replaced.

4. Use the Administration Tools to [replace the original host](#) with the new host. If you are using more than one database, replace the original host in all the databases in which it is used. See [Replacing Hosts](#).

5. [Remove the host from the cluster.](#)

6. Restart HP Vertica on the host.

Once you have completed this process, the replacement node automatically recovers the data that was stored in the original node by querying the other nodes within the database. It then transitions to an UP state.

Note: If you do not remove the original host from the cluster and you attempt to restart the database, the host is not invited to join the database because its node address does not match the new address stored in the database catalog. Therefore, it remains in the INITIALIZING state.

Using the Administration Tools to Replace Nodes

If you are replacing a node with a host that uses a different name and IP address, use the Administration Tools to replace the original host with the new host. Alternatively, you can [use the Management Console to replace a node](#).

Replace the Original Host with a New Host Using the Administration Tools

To replace the original host with a new host using the Administration Tools:

1. Back up the database. See [Backing Up and Restoring the Database](#).
2. From a node that is up, and is not going to be replaced, open the **Administration Tools**.
3. On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If it's not running, use the Start Database command on the Main Menu to restart it.
4. On the **Main Menu**, select **Advanced Menu**.
5. In the **Advanced Menu**, select **Stop HP Vertica on Host**.
6. Select the host you want to replace, and then click **OK** to stop the node.
7. When prompted if you want to stop the host, select **Yes**.
8. In the **Advanced Menu**, select **Cluster Management**, and then click **OK**.
9. In the **Cluster Management** menu, select **Replace Host**, and then click **OK**.
10. Select the database that contains the host you want to replace, and then click **OK**.

A list of all the hosts that are currently being used displays.

11. Select the host you want to replace, and then click **OK**.
12. Select the host you want to use as the replacement, and then click **OK**.
13. When prompted, enter the password for the database, and then click **OK**.
14. When prompted, click **Yes** to confirm that you want to replace the host.
15. When prompted that the host was successfully replaced, click **OK**.
16. In the **Main Menu**, select **View Database Cluster State** to verify that all the hosts are running. You might need to start HP Vertica on the host you just replaced. Use **Restart Vertica on Host**.

The node enters a RECOVERING state.

Caution: If you are using a **K-Safe** database, keep in mind that the recovering node counts as one node down even though it might not yet contain a complete copy of the data. This means that if you have a database in which $K \text{ safety}=1$, the current fault tolerance for your database is at a critical level. If you lose one more node, the database shuts down. Be sure that you do not stop any other nodes.

Using the Management Console to Replace Nodes

On the MC **Manage** page, you can quickly replace a DOWN node in the database by selecting one of the STANDBY nodes in the cluster.

A DOWN node shows up as a red node in the cluster. Click the DOWN node and the Replace node button in the Node List becomes activated, as long as there is at least one node in the cluster that is not participating in the database. The STANDBY node will be your replacement node for the node you want to retire; it will appear gray (empty) until it has been added to the database, when it turns green.

Tip: You can resize the Node List by clicking its margins and dragging to the size you want.

When you highlight a node and click **Replace**, MC provides a list of possible STANDBY nodes to use as a replacement. After you select the replacement node, the process begins. A node replacement could be a long-running task.

MC transitions the DOWN node to a STANDBY state, while the node you selected as the replacement will assume the identity of the original node, using the same node name, and will be started.

Assuming a successful startup, the new node will appear orange with a status of RECOVERING until the recovery procedure is complete. When the recovery process completes, the replacement node will turn green and show a state of UP.

Rebalancing Data Across Nodes

HP Vertica automatically rebalances your database when adding or removing nodes. You can also manually trigger a rebalance using the Administration Tools or using SQL functions. Users can rebalance data across nodes through the Management Console interface (see [Rebalancing Data Using Management Console](#) for details).

Whether you start the rebalance process manually or automatically, the process occurs in the following steps:

- For segmented projections, HP Vertica creates new (renamed), segmented projections that are identical in structure to the existing projections, but which have their data distributed across all nodes. The rebalance process then refreshes all new projections, sets the Ancient History Mark (AHM) to the greatest allowable epoch (now), and drops all of the old segmented projections. All new buddy projections have the same base name so they can be identified as a group.

Note: HP Vertica does not maintain custom projection segmentations defined with a specific node list. Node rebalancing distributes data across all nodes, regardless of any custom definitions.

- For unsegmented projections, leaves existing projections unmodified, creates new projections on the new nodes, and refreshes them.
- After the data has been rebalanced, HP Vertica drops:
 - Duplicate buddy projections with the same offset
 - Duplicate replicated projections on the same node

K-safety and Rebalancing

Before data rebalancing completes, HP Vertica operates with the existing **K-safe** value. After rebalancing completes, HP Vertica operates with the K-safe value specified during the rebalance operation.

You can maintain existing K-safety or specify a new value (0 to 2) for the modified database cluster. HP Vertica does not support downgrading K-safety and returns a warning if you attempt to reduce it from its current value: Design k-safety cannot be less than system k-safety level. For more information, see [Lowering the K-Safety Level to Allow for Node Removal](#).

Rebalancing Failure and Projections

If a failure occurs while rebalancing the database, you can rebalance again. If the cause of the failure has been resolved, the rebalance operation continues from where it failed. However, a failed data rebalance can result in projections becoming out of date, so that they cannot be removed automatically.

To locate any such projections, query the V_CATALOG.PROJECTIONS system table as follows:

```
=> SELECT projection_name, anchor_table_name, is_prejoin,  
       is_up_to_date  
       FROM projections  
       WHERE is_up_to_date = false;
```

To remove out-of-date projections, use the [DROP PROJECTION](#) function.

Permissions

Only the **superuser** has permissions to rebalance data.

Rebalancing Data Using the Administration Tools UI

To rebalance the data in your database:

1. Open the Administration Tools. (See [Using the Administration Tools](#).)
2. On the **Main Menu**, select **View Database Cluster State** to verify that the database is running. If it is not, start it.
3. From the **Main Menu**, select **Advanced Tools Menu** and click **OK**.
4. In the **Advanced Menu**, select **Cluster Management** and click **OK**.
5. In the **Cluster Management** menu, select **Re-balance Data** and click **OK**.
6. Select the database you want to rebalance, and then select **OK**.
7. Enter the directory for the Database Designer outputs (for example `/tmp`) and click **OK**.
8. Accept the proposed **K-safety** value or provide a new value. Valid values are 0 to 2.
9. Review the message and click **Proceed** to begin rebalancing data.

The Database Designer modifies existing projections to rebalance data across all database nodes with the K-safety you provided. A script to rebalance data, which you can run manually at a later time, is also generated and resides in the path you specified; for example `/tmp/extend_catalog_rebalance.sql`.

Important: Rebalancing data can take some time, depending on the number of projections and the amount of data they contain. HP recommends that you allow the process to complete. If you must cancel the operation, use `Ctrl+C`.

The terminal window notifies you when the rebalancing operation is complete.

10. Press **Enter** to return to the Administration Tools.

Rebalancing Data Using Management Console

HP Vertica automatically rebalances the database after you add or remove nodes. If, however, you notice data skew where one node shows more activity than another (for example, most queries processing data on a single node), you can manually rebalance the database using MC if that database is imported into the MC interface.

On the **Manage** page, click **Rebalance** in the toolbar to initiate the rebalance operation.

During a rebalance, you cannot perform any other activities on the database cluster, such as start, stop, add, or remove nodes.

Rebalancing Data Using SQL Functions

There are three SQL functions that let you manually control the data rebalancing process. You can use these functions to run a rebalance from a script scheduled to run at an off-peak time, rather than having to manually trigger a rebalance through the Administration Tools.

These functions are:

- [REBALANCE_CLUSTER\(\)](#)
- [START_REBALANCE_CLUSTER\(\)](#)
- [CANCEL_REBALANCE_CLUSTER\(\)](#)

For more information and examples of using these functions, see their entries in the SQL Reference Manual.

Redistributing Configuration Files to Nodes

The add and remove node processes automatically redistribute the HP Vertica configuration files. You may rarely need to redistribute the configuration files to help resolve configuration issues.

To distribute configuration files to a host:

1. Log on to a host that contains these files and start the Administration Tools.

See [Using the Administration Tools](#) for information about accessing the Administration Tools.
2. On the **Main Menu** in the Administration Tools, select **Configuration Menu** and click **OK**.
3. On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.
4. Select **Database Configuration**.
5. Select the database in which you want to distribute the files and click **OK**.

The `vertica.conf` file is distributed to all the other hosts in the database. If it previously existed on a host, it is overwritten.

6. On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.

7. Select **SSL Keys**.

The certifications and keys for the host are distributed to all the other hosts in the database. If they previously existed on a host, they are overwritten.

8. On the **Configuration Menu**, select **Distribute Config Files** and click **OK**.

Select **AdminTools Meta-Data**.

The Administration Tools metadata is distributed to every host in the cluster.

9. [Restart the database](#).

Changing the IP Addresses of an HP Vertica Cluster

This section describes how to change the IP addresses of the nodes in an HP Vertica cluster.

Note: This process requires that you stop the database on all nodes, then subsequently stop the database on individual nodes as you update IP addresses.

These instructions assume you will make the standard OS changes to change the IPs (for example, updating `/etc/hosts`) in Step 4 of this procedure. These instructions detail only the HP Vertica-specific IP changes. Consult the documentation for your particular OS platform for details on changing the IP address of the host.

To change the IP address of one or more nodes in a cluster:

1. Before changing the IP address on the Host, back up the following three files on all nodes:
 - `/opt/vertica/config/admintools.conf`
 - `/opt/vertica/config/vspread.conf`
 - `/etc/sysconfig/spreadd`
2. Stop HP Vertica on all nodes.
3. As root, on each node, stop spread by using the following command:

```
/etc/init.d/spreadd stop
```
4. Change the IP addresses of the hosts as required by your operating system platform.
5. On each node edit `/opt/vertica/config/admintools.conf` and change the IPs as required.

You can use `sed` to change each IP in the file, for example to change `10.10.81.8` to `192.168.150.108` issue the command:

```
sed -i 's/10.10.81.8/192.168.150.108/g' /opt/vertica/config/admintools.conf
```

6. On each node edit `/opt/vertica/config/vspread.conf`:
 - a. Change the old IPs to the new IPs as required.
 - b. Locate the N number for each IP and change it to match the new IP. For example, if the old IP is `10.10.81.8`, then the corresponding N number is `N010010081008`. N numbers consist of 3 digits for each IP number segment, padded with zeros when appropriate (10 becomes 010, 8 become 008, etc.). If the new IP address is `192.168.150.255`, then the new corresponding N number is `N192168250255`.
7. On each node edit `/etc/sysconfig/spreadd` and change the N number to that node's new N number as you specified in `vspread.conf`.

8. As root, on each node start `spread` by using the following command:

```
/etc/init.d/spreadd start
```

9. Start the database.
10. Run `vsq`.
11. In `vsq`, issue the following query to verify the new IP has been updated:

```
select host_name from host_resources;
```

You can also verify IPs with the following shell commands:

- `cat /var/log/spreadd.log`
- `admintools -t list_host`
- `cat /etc/hosts`

12. Update the database to use the new IPs for reporting node status:
 - a. In `vsq`, issue the command `select node_name, node_address from v_catalog.nodes;` to show you the current node names configured.
 - b. For each node in the result, change the hostname to the new IP address. **Note:** the node must be down to change the IP using the `alter node` command. You must bring the node down before altering the `NODE_NAME` property for that node:
 - Bring the node down that you are going to update (don't bring down the node from which you are using `vsq`!). You can bring down the node from the initiator node using `admintools`. For example:

```
admintools -t stop_node -s 192.168.150.255
```


- Update the IP address by issuing the command: `alter node NODE_NAME is hostname 'new.ip.address';` where *NODE_NAME* is the *node_name* and *new.ip.address* is the new IP address of that node.
- Bring the node back up, check it's status in the nodes table (`select node_name, node_state from nodes;`), and wait for that node's status to be *UP*. You can use `admintools` from the initiator node to restart the node. You must provide the database name. For example:

```
admintools -t restart_node -s 192.168.150.255 -d VMart
```
- Repeat the process for the next node in the result. After all nodes have been updated except the node from which you are using `vsql`, log out of `vsql`, then log into `vsql` from another node and update the IP address for the node from which you were previously using `vsql`.

Stopping and Starting Nodes on MC

You can start and stop one or more database nodes through the **Manage** page by clicking a specific node to select it and then clicking the Start or Stop button in the Node List.

Note: The Stop and Start buttons in the toolbar start and stop the database, not individual nodes.

On the **Databases and Clusters** page, you must click a database first to select it. To stop or start a node on that database, click the **View** button. You'll be directed to the Overview page. Click **Manage** in the applet panel at the bottom of the page and you'll be directed to the database node view.

The Start and Stop database buttons are always active, but the node Start and Stop buttons are active only when one or more nodes of the same status are selected; for example, all nodes are UP or DOWN.

After you click a Start or Stop button, Management Console updates the status and message icons for the nodes or databases you are starting or stopping.

Managing Disk Space

HP Vertica detects and reports low disk space conditions in the log file so that the issue can be addressed before serious problems occur. It also detects and reports low disk space conditions via [SNMP traps](#) if enabled.

Critical disk space issues are reported sooner than other issues. For example, running out of catalog space is fatal; therefore, HP Vertica reports the condition earlier than less critical conditions. To avoid database corruption when the disk space falls beyond a certain threshold, HP Vertica begins to reject transactions that update the catalog or data.

Caution: A low disk space report indicates one or more hosts are running low on disk space or have a failing disk. It is imperative to add more disk space (or replace a failing disk) as soon as possible.

When HP Vertica reports a low disk space condition, use the [DISK_RESOURCE_REJECTIONS](#) system table to determine the types of disk space requests that are being rejected and the hosts on which they are being rejected.

These and the other [Using System Tables](#) system tables are described in detail in the [SQL Reference Manual](#).

To add disk space, see [Adding Disk Space to a Node](#). To replace a failed disk, see [Replacing Failed Disks](#).

Monitoring Disk Space Usage

You can use these system tables to monitor disk space usage on your cluster:

| System table | Description |
|------------------------------------|--|
| DISK_STORAGE | Monitors the amount of disk storage used by the database on each node. |
| COLUMN_STORAGE | Monitors the amount of disk storage used by each column of each projection on each node. |
| PROJECTION_STORAGE | Monitors the amount of disk storage used by each projection on each node. |

Adding Disk Space to a Node

This procedure describes how to add disk space to a node in the HP Vertica cluster.

Note: If you are adding disk space to multiple nodes in the cluster, then use the following procedure for each node, one node at a time.

To add disk space to a node:

1. If you must shut down the hardware to which you are adding disk space, then first shut down HP Vertica on the host where disk space is being added.
2. Add the new disk to the system as required by the hardware environment. Boot the hardware if it is was shut down.
3. Partition, format, and mount the new disk, as required by the hardware environment.
4. Create a data directory path on the new volume.

For example:

```
mkdir -p /myNewPath/myDB/host01_data2/
```

5. If you shut down the hardware, then restart HP Vertica on the host.
6. Open a database connection to HP Vertica and add a **storage location** to add the new data directory path. If you are connecting from a different host than the one on which you added storage, then specify the node in [ADD_LOCATION](#), otherwise `ADD_LOCATION` assumes you are referring to the local host.

See [Adding Storage Locations](#) in this guide and the [ADD_LOCATION](#) function in the SQL Reference Manual.

Note: `ADD_LOCATION` is a local command, which must be run on each node to which space is added.

Replacing Failed Disks

If the disk on which the data or catalog directory resides fails, causing full or partial disk loss, perform the following steps:

1. Replace the disk and recreate the data or catalog directory.
2. Distribute the configuration file (`vertica.conf`) to the new host. See [Distributing Configuration Files to the New Host](#) for details.
3. Restart the HP Vertica on the host, as described in [Restart Vertica On Host](#).

See [Catalog and Data Files](#) for information about finding your `DATABASE_HOME_DIR`.

Catalog and Data Files

For the recovery process to complete successfully, it is essential that catalog and data files be in the proper directories.

In HP Vertica, the **catalog** is a set of files that contains information (metadata) about the objects in a database, such as the nodes, tables, constraints, and projections. The catalog files are replicated on all nodes in a cluster, while the data files are unique to each node. These files are installed by default in the following directories:

```
/DATABASE_HOME_DIR/DATABASE_NAME/v_db_nodexxxx_catalog/ /DATABASE_HOME_DIR/DATABASE_NAME/v_db_nodexxxx_catalog/
```

Note: `DATABASE_HOME_DIR` is the path, which you can see from the Administration Tools. See [Using the Administration Tools](#) in the Administrator's Guide for details on using the interface.

To view the path of your database:

1. Run the **Administration Tools**.

```
$ /opt/vertica/bin/admintools
```

2. From the Main Menu, select **Configuration Menu** and click **OK**.
3. Select **View Database** and click **OK**.
4. Select the database you want would like to view and click **OK** to see the database profile.

See [Understanding the Catalog Directory](#) for an explanation of the contents of the catalog directory.

Understanding the Catalog Directory

The catalog directory stores metadata and support files for your database. Some of the files within this directory can help you troubleshoot data load or other database issues. See [Catalog and Data Files](#) for instructions on locating your database's catalog directory. By default, it is located in the database directory. For example, if you created the VMart database in the database administrator's account, the path to the catalog directory is:

```
/home/dbadmin/VMart/v_vmart_nodennnn_catalog
```

where `nodennnn` is the name of the node you are logged into. The name of the catalog directory is unique for each node, although most of the contents of the catalog directory are identical on each node.

The following table explains the files and directories that may appear in the catalog directory.

Note: Do not change or delete any of the files in the catalog directory unless asked to do so by HP Vertica support.

| File or Directory | Description |
|------------------------------------|---|
| <code>bootstrap-catalog.log</code> | A log file generated as the HP Vertica server initially creates the database (in which case, the log file is only created on the node used to create the database) and whenever the database is restored from a backup. |
| <code>Catalog/</code> | Contains catalog information about the database, such as checkpoints. |
| <code>CopyErrorLogs/</code> | The default location for the COPY exceptions and rejections files generated when data in a bulk load cannot be inserted into the database. See Capturing Load Exceptions and Rejections for more information. |
| <code>DataCollector/</code> | Log files generated by the Data Collector . |
| <code>debug_log.conf</code> | Debugging information configuration file. For HP use only. |
| <code>Epoch.log</code> | Used during recovery to indicate the latest epoch that contains a complete set of data. |
| <code>ErrorReport.txt</code> | A stack trace written by HP Vertica if the server process exits unexpectedly. |

| File or Directory | Description |
|-------------------|---|
| Libraries/ | Contains user defined library files that have been loaded into the database. See Developing and Using User Defined Functions in the Programmer's Guide. Do not change or delete these libraries through the file system. Instead, use the CREATE LIBRARY , DROP LIBRARY , and ALTER LIBRARY statements. |
| Snapshots/ | The location where backup snapshots are stored. See Using Database Snapshots for more information. |
| tmp/ | A temporary directory used by HP Vertica's internal processes. |
| UDxLogs/ | Log files written by user defined functions that run in fenced mode. See Fenced Mode in the Programmer's Guide for more information. |
| vertica.conf | The primary configuration file for HP Vertica. If you change any configuration parameter from its default value with the <code>set_config_parameter()</code> function, the changes are contained in this file. |
| vertica.log | The main log file generated by the HP Vertica server process. |
| vertica.pid | The process ID and path to the catalog directory of the HP Vertica server process running on this node. |

Reclaiming Disk Space From Deleted Records

You can reclaim the disk space held by deleted records by [purging the deleted records](#), [rebuilding the table](#) or [dropping a partition](#).

Rebuilding a Table

When it is necessary to do large-scale disk reclamation operations, consider rebuilding the table by following sequence of operations:

1. Create a new table.
2. Create projections for the new table.
3. Populate the new table using `INSERT ... SELECT` to copy the desired table from the old table.
4. Drop the old table and its projections.
5. Use `ALTER TABLE ... RENAME` to give the new table the name of the old table.

Notes

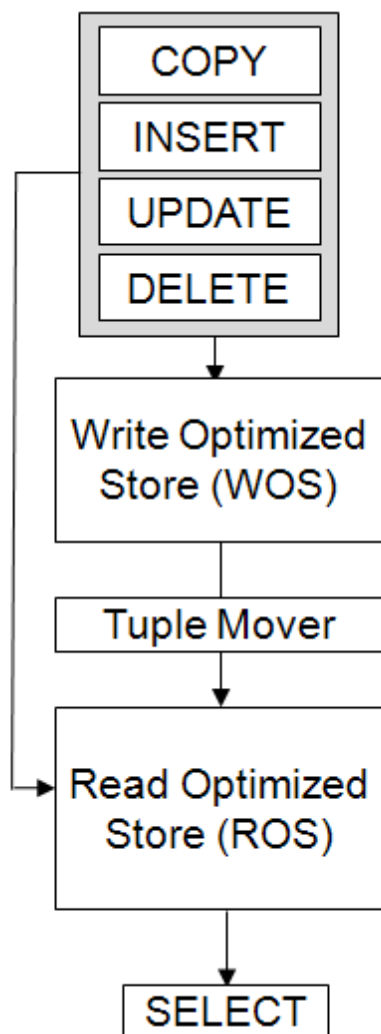
- You must have enough disk space to contain the old and new projections at the same time. If necessary, you can drop some of the old projections before loading the new table. You must, however, retain at least one **superprojection** of the old table (or two **buddy** superprojections to maintain K-safety) until the new table is loaded. (See [Prepare Disk Storage Locations](#) in the Installation Guide for disk space requirements.)
- You can specify different names for the new projections or use the `ALTER PROJECTION ... RENAME` command to change the names of the old projections.
- The relationship between tables and projections does not depend on object names. Instead, it depends on object identifiers that are not affected by rename operations. Thus, if you rename a table, its projections continue to work normally.
- Manually purging a table continues to retain history for rows deleted after the Ancient History Mark. Rebuilding the table results in purging all the history of the table, which means you cannot do historical queries on any older epoch.
- Rather than dropping the old table in Step 4, you might rename it to a different name and use it as a backup copy. Note, however, that you must have sufficient disk space.

Managing Tuple Mover Operations

The Tuple Mover (TM) is the HP Vertica database optimizer component that moves data from memory (WOS) to disk (ROS). The TM also combines small ROS containers into larger ones, and purges deleted data. During moveout operations, the TM is also responsible for adhering to any storage policies that are in effect for the storage location. The Tuple Mover runs in the background, performing some tasks automatically (ATM) at time intervals determined by its configuration parameters. For information about changing the TM configuration parameters, see [Tuple Mover Parameters](#) in the Administrator's Guide for further information.

Under ordinary circumstances, the operations performed by the TM are automatic and transparent, and are therefore of little or no concern to the database administrator. However, when loading data, certain conditions require that you stop the Tuple Mover, perform some operations manually, and restart it. Also, the COPY statement AUTO, DIRECT, and TRICKLE parameters specify how data is loaded (directly into ROS or WOS). See [Choosing a Load Method](#), for more information.

This section discusses [Tuple Mover operations](#) and how to perform TM tasks manually.



Understanding the Tuple Mover

The Tuple Mover performs two operations:

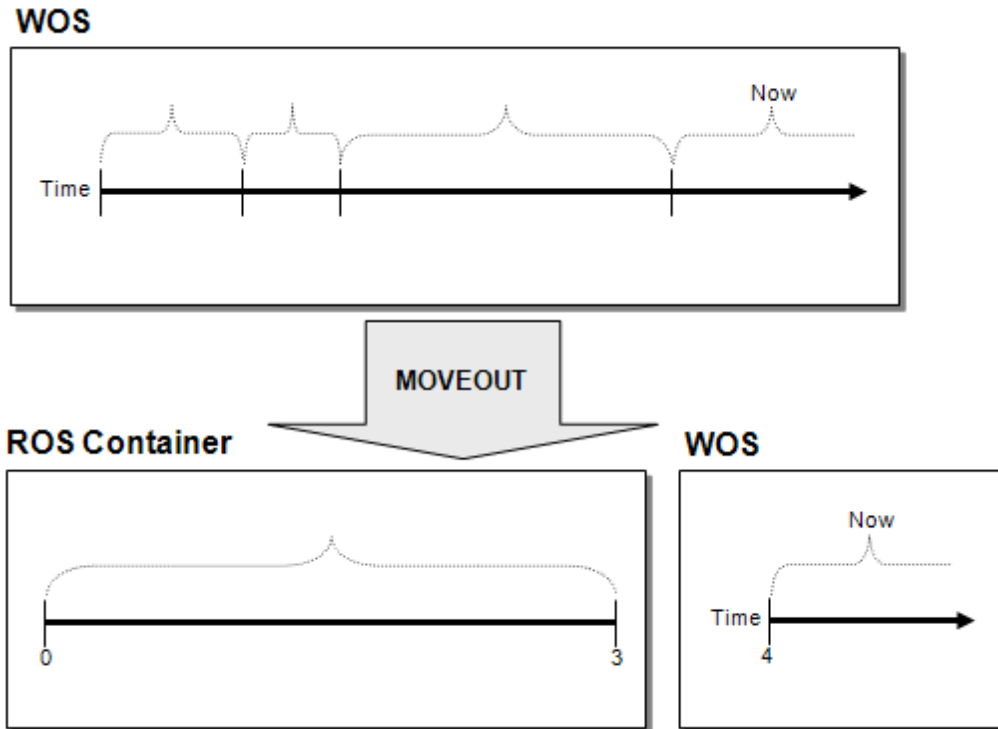
- [Moveout](#)
- [Mergeout](#)

Each of these operations occurs at different intervals across all nodes. The tuple mover runs independently on each node, ensuring that storage is managed appropriately even in the event of data skew.

Moveout

Moveout operations move data from memory (**WOS**) into a new **ROS container**. A moveout "flushes" all historical data from the WOS to the ROS.

The following illustration shows the effect of a projection moveout on a single node:



ROS Containers

A ROS (Read Optimized Store) container is a set of rows stored in a particular group of files. ROS containers are created by operations like Moveout or COPY DIRECT, and can be observed in the STORAGE_CONTAINERS system table. The ROS container layout can differ across nodes due to data variance. Segmentation can deliver more rows to one node than another. Two loads could fit in the WOS on one node and spill on another.

Mergeout

A mergeout is the process of consolidating ROS containers and purging deleted records. Over time, the number of ROS containers increases to a degree that it becomes necessary to merge some of them in order to avoid performance degradation. At that point, the Tuple Mover performs an automatic mergeout, which combines two or more ROS containers into a single container. This process can be thought of as "defragmenting" the ROS.

Vertica keeps data from different partitions separate on disk. When the Tuple Mover consolidates ROS containers, it adheres to this policy by not merging ROS containers from different partitions. When a partition is first created, it is typically the subject of frequent data loads and requires regular attention from the Tuple Mover. As a partition ages, it commonly transitions to a read-only workload that requires much less attention.

The Tuple Mover has two different policies for managing these different partition workloads:

- *Active partitions* are loaded or modified frequently. The Tuple Mover uses a [STRATA](#) mergeout policy that keeps a collection of ROS container sizes to minimize the number of times any individual tuple is subjected to mergeout. The `ActivePartitionCount` parameter identifies how many partitions are being actively loaded.
- *Inactive partitions* are very infrequently loaded or modified. The Tuple Mover consolidates the ROS containers to a minimal set while avoiding merging containers whose size exceeds `MaxMrgOutROSSizeMB`.

Partitions are not explicitly marked by the user as active or inactive; instead, the Tuple Mover uses the following algorithm to order the partitions from oldest to newest:

- If one partition was created before the other partition, it is older.
- If two partitions were created at the same time, but one partition was last updated earlier than the other partition, it is older.
- If two partitions were created and last updated at the same time, the partition with the smaller key is considered older.

If you perform a manual mergeout using the `DO_TM_TASK` function, *all* partitions are consolidated into the smallest possible number of containers, regardless of the value of the `ActivePartitionCount` parameter.

Mergeout of Deletion Markers

When you delete data from the database, HP Vertica does not remove it. Instead, it marks the data as deleted. Using many [DELETE](#) statements to mark a small number of rows relative to the size of the table can result in creation of many small containers to hold these deletion marks. Each of these containers consumes resources, so a large number of these containers can impact performance, especially during recovery.

After it performs a mergeout, the Tuple Mover looks for deletion marker containers that hold few entries. If it finds some, it merges them together into a single larger container. This process helps lower the overhead of tracking deleted data by freeing resources used by the individual containers. It does not purge or otherwise affect the deleted data—it just consolidates the deletion mark containers for greater efficiency.

Note: You can see the number and size of the containers holding the deletion marks by viewing the [V_MONITOR.DELETE_VECTORS](#) system table.

Tuning the Tuple Mover

The **Tuple Mover** comes preconfigured to work for most common workloads. However there are some situations in which tuning the tuple mover behavior is required. You do so by changing its configuration parameters. The following section explains the parameters that tune the Tuple Mover, and the remainder of this section explains how to use them for several situations.

Tuple Mover Configuration Parameters

The following configuration parameters control how the Tuple Mover operates. You can use them to adjust its operation to suit your needs, as described in the following sections.

| Parameters | Description |
|----------------------|---|
| ActivePartitionCount | <p>Sets the number of partitions, called <i>active partitions</i>, that are currently being loaded. For information about how the Tuple Mover treats active (and inactive) partitions during a mergeout operation, see Understanding the Tuple Mover.</p> <p>Default Value: 1</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('ActivePartitionCount', 2);</pre> |
| MergeOutInterval | <p>The number of seconds the Tuple Mover waits between checks for new ROS files to merge out. If ROS containers are added frequently, you may need to decrease this value.</p> <p>Default Value: 600</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('MergeOutInterval', 1200);</pre> |
| MoveOutInterval | <p>The number of seconds the Tuple mover waits between checks for new data in the WOS to move to ROS.</p> <p>Default Value: 300</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('MoveOutInterval', 600);</pre> |
| MoveOutMaxAgeTime | <p>The specified interval (in seconds) after which the tuple mover is forced to write the WOS to disk. The default interval is 30 minutes.</p> <p>Tip: If you had been running the <code>force_moveout.sh</code> script in previous releases, you no longer need to run it.</p> <p>Default Value: 1800</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('MoveOutMaxAgeTime', 1200);</pre> |

| Parameters | Description |
|----------------|--|
| MoveOutSizePct | <p>The percentage of the WOS that can be filled with data before the Tuple Mover performs a moveout operation.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('MoveOutSizePct', 50);</pre> |

Resource Pool Settings

The **Tuple Mover** draws its resources from the TM resource pool. Adding more resources (RAM) to this pool, and changing its concurrency setting, can make the Tuple Mover more effective in dealing with high load rates.

The TM resource pool concurrency setting, `PLANNEDCONCURRENCY`, determines how many merges can occur simultaneously through multiple threads. As a side effect of the concurrency setting, the Tuple Mover dedicates some threads to aggressively address small **ROS containers**, while other threads are reserved to work only on merges of ROS containers in the lower strata.

For the TM pool, `PLANNEDCONCURRENCY` must be proportional to the size of the RAM, the CPU, and the storage subsystem. Depending on the storage type, if you increase `PLANNEDCONCURRENCY` for the Tuple Mover threads, you might create a storage I/O bottleneck. Monitor the storage subsystem; if it becomes saturated with long I/O queues, more than two I/O queues, and long latency in read and write, adjust the `PLANNEDCONCURRENCY` parameter to keep the storage subsystem resources below saturation level. In addition, you might need to:

- Partition storage data files
- Adjust block-size optimization on storage subsystems such as RAID 5 or RAID 10
- Identify the optimal number of disks in the RAID array

The following statement illustrates how to increase the size of the TM resource pool and set the concurrency settings for the pool:

```
=> ALTER RESOURCE POOL tm MEMORYSIZE '4G' PLANNEDCONCURRENCY 4 MAXCONCURRENCY 5;
```

The `WOSDATA` resource pool settings also indirectly affect the Tuple Mover. In automatic mode, `INSERT` and `COPY` commands use the concurrency setting to determine whether data is small enough to store in WOS or if it should be written to ROS. Therefore, set this value to be the number of concurrent loads you expect to perform in your database. The `WOSDATA` resource pool also determines how much RAM the WOS can use.

```
=> ALTER RESOURCE POOL wosdata MAXMEMORYSIZE '4G' PLANNEDCONCURRENCY 3;
```

See [Managing Workloads](#) and [Resource Pool Architecture](#) in this guide and [ALTER RESOURCE POOL](#) and [Built-in Pools](#) in the SQL Reference Manual.

Loading Data

HP Vertica automatically decides whether the data should be placed in **WOS** or stored directly in **ROS** containers based on the amount of data processed by a `COPY` or `INSERT` command. HP Vertica stores large loads directly to disk and stores smaller loads in memory, which it later moves to disk.

For low-latency access to data, use small loads. The automatic Tuple Mover settings are the best option for handling such smaller loads. One exception is for single-node deployments, where a system failure would cause in-memory data to be lost. In this case, you might want to force all data loads to go directly to disk.

For high load rates, you might want the Tuple Mover to check for jobs more frequently by changing the `MergeOutInterval` and `MoveOutInterval` configuration parameters. Reduce the `MoveOutInterval` if you expect the peak load rate to fill the WOS quickly. Reduce `MergeOutInterval` if you anticipate performing many `DIRECT` loads or inserts.

See [COPY](#) and [INSERT](#) in the SQL Reference Manual

Using More Threads

If your database is receiving a large volume of data to load or if it is performing many `DIRECT` loads or inserts, consider allowing the Tuple Mover to perform more operations concurrently by increasing the TM resource pool until it can keep up with the anticipated peak load rate. For example:

```
=> ALTER RESOURCE POOL TM MEMOYRSIZE '4G' PLANNEDCONCURRENCY 4 MAXCONCURRENCY 5;
```

See [ALTER RESOURCE POOL](#) and [Built-in Pools](#) in the SQL Reference Manual.

Active Data Partitions

By default, the Tuple Mover assumes that all loads and updates for partitioned tables are going to the same *active* partition. For example, if a table is partitioned by month, the Tuple Mover expects that after the start of a new month, no data is loaded into the partition for the prior month.

If loads and updates occur to more than one partition, set the `ActivePartitionCount` parameter to reflect the number of partitions that will be loading data. For example, if your database receives data for the current month as well as updates to the prior month, set `ActivePartitionCount` to 2. For tables partitioned by non-temporal attributes, set `ActivePartitionCount` to reflect the number of partitions that will be loaded simultaneously.

See [Table Partitioning](#) in this guide.

See Also

- [Best Practices for Managing Workload Resources](#)

Managing Workloads

HP Vertica provides a sophisticated resource management scheme that allows diverse, concurrent workloads to run efficiently on the database. For basic operations, the built-in [GENERAL pool](#) is pre-configured based on RAM and machine cores, but you can customize this pool to handle specific concurrency requirements.

You can also define new resource pools that you configure to limit memory usage, concurrency, and query priority. You can then optionally restrict each database user to use a specific resource pool, which control memory resources used by their requests.

User-defined pools are useful if you have competing resource requirements across different classes of workloads. Example scenarios include:

- A large batch job takes up all server resources, leaving small jobs that update a web page to starve, which can degrade user experience.

In this scenario, you can create a resource pool to handle web page requests and ensure users get resources they need. Another option is to create a limited resource pool for the batch job, so the job cannot use up all system resources.

- A certain application has lower priority than other applications, and you would like to limit the amount of memory and number of concurrent users for the low-priority application.

In this scenario, you could create a resource pool with an upper limit on the query's memory and associate the pool with users of the low-priority application.

You can also use resource pools to manage resources assigned to running queries. You can assign a run-time priority to a resource pool, as well as a threshold to assign different priorities to queries with different durations. See [Managing Resources At Query Run Time](#) for more information.

For detailed syntax of creating and managing resource pools see the following topics in the SQL Reference Manual:

Statements

- [ALTER RESOURCE POOL](#) alters a resource pool.
- [ALTER USER](#) associates a user with the RESOURCE POOL and MEMORYCAP parameters.
- [CREATE RESOURCE POOL](#) creates a resource pool.
- [CREATE USER](#) adds a name to the list of authorized database users and specifies that user's RESOURCE POOL and MEMORYCAP parameters.
- [DROP RESOURCE POOL](#) drops a user-created resource pool.
- [SET SESSION MEMORYCAP](#) sets the limit on amount of memory that any request issued by

the session can consume.

- [SET SESSION RESOURCE POOL](#) associates a user session with specified resource pool.

System Tables

- [RESOURCE_ACQUISITIONS](#) provides details of resources (memory, open file handles, threads) acquired by each request for each resource pool in the system.
- [RESOURCE_POOL_DEFAULTS \(systab\)](#) lists default values for parameters in each internal and user-defined resource pool.
- [RESOURCE_POOL_STATUS](#) provides configuration settings of the various resource pools in the system, including internal pools.
- [RESOURCE_POOLS](#) displays information about the parameters the resource pool was configured with.
- [RESOURCE_QUEUES](#) provides information about requests pending for various resource pools.
- [RESOURCE_REJECTIONS](#) monitors requests for resources that are rejected by the **Resource Manager**.
- [RESOURCE_REJECTION_DETAILS](#) records an entry for each resource request that HP Vertica denies. This is useful for determining if there are resource space issues, as well as which users/pools encounter problems
- [SYSTEM_RESOURCE_USAGE](#) provides history about system resources, such as memory, CPU, network, disk, I/O.

See Also

- [Managing Resources At Query Run Time](#)
- [Analyzing Workloads](#)

The Resource Manager

On a single-user environment, the system can devote all resources to a single query, getting the most efficient execution for that one query. It's more common, however, that your environment will run several queries at once, which could cause tension between providing each query the maximum amount of resources (fastest run time) and serving multiple queries simultaneously with a reasonable run time.

The HP Vertica Resource Manager (RM) provides lets you resolve this tension, while ensuring that every query eventually gets serviced and that true system limits are respected at all times. For example, when the system experiences resource pressure, the Resource Manager might queue queries until the resources become available or a timeout value is reached. Also, when you

configure various RM settings, you can tune each query's target memory based on the expected number of concurrent queries running against the system.

This section discusses the detailed architecture and operation of the Resource Manager.

Resource Manager Impact on Query Execution

The Resource Manager (RM) impacts individual query execution in various ways. When a query is submitted to the database, the following series of events occur:

1. The query is parsed, optimized to determine an execution plan, and distributed to the participating nodes.
2. The Resource Manager is invoked on each node to estimate resources required to run the query and compare that with the resources currently in use. One of the following will occur:
 - If the memory required by the query alone would exceed the machine's physical memory, the query is rejected - it cannot possibly run. Outside of significantly under-provisioned nodes, this case is very unlikely.
 - If the resource requirements are not currently available, the query is queued. The query will remain on the queue until either sufficient resources are freed up and the query runs or the query times out and is rejected.
 - Otherwise the query is allowed to run.
3. The query starts running when all participating nodes allow it to run.

Note: Once the query is running, the Resource Manager further manages resource allocation using `RUNTIMEPRIORITY` and `RUNTIMEPRIORITYTHRESHOLD` parameters for the resource pool. See [Managing Resources At Query Run Time](#) for more information.

Apportioning resources for a specific query and the maximum number of queries allowed to run depends on the resource pool configuration. See [Resource Pool Architecture](#).

On each node, no resources are reserved or held while the query is in the queue. However, multi-node queries queued on some nodes will hold resources on the other nodes. HP Vertica makes every effort to avoid deadlocks in this situation.

Resource Pool Architecture

The Resource Manager handles resources as one or more resource pools, which are a pre-allocated subset of the system resources with an associated queue.

HP Vertica is preconfigured with a set of [built-in pools](#) that allocate resources to different request types, where the GENERAL pool allows for a certain concurrency level based on the RAM and cores in the machines.

Modifying and Creating Resource Pools

You can configure the build-in GENERAL pool based on actual concurrency and performance requirements, as described in [Guidelines for Setting Pool Parameters](#). You can also create custom pools to handle various classes of workloads and optionally restrict user requests to your custom pools.

You create a pool using the [CREATE RESOURCE POOL](#) command. See the SQL Reference Manual for details.

Monitoring Resource Pools and Resource Usage By Queries

The [Linux top command](#) can be used to determine the overall CPU usage and I/O waits across the system. However, resident memory size indicated by `top` is not a good indicator of actual memory use or reservation because of file system caching and so forth. Instead, HP Vertica provides several monitoring tables that provide detailed information about resource pools, their current memory usage, resources requested and acquired by various requests and the state of the queues.

The [RESOURCE_POOLS](#) table lets you view various resource pools defined in the system (both internal and user-defined), and the [RESOURCE_POOL_STATUS](#) table lets you view the current state of the resource pools.

Examples

The following command returns the various resource pools defined in the system.

```
VMart=> SELECT name, memorysize, maxmemorysize FROM V_CATALOG.RESOURCE_POOLS;
```

| name | memorysize | maxmemorysize |
|----------|------------|---------------|
| general | | Special: 95% |
| sysquery | 64M | |
| sysdata | 100M | 10% |
| wosdata | 0% | 25% |
| tm | 200M | |
| refresh | 0% | |
| recovery | 0% | |
| dbd | 0% | |

```
jvm      | 0%      | 10%
(9 rows)
```

To see only the user-defined resource pools, you can limit your query to return records where IS_INTERNAL is false.

Note: The user-defined pools below are used as examples in subsequent sections related to Workload Management.

The following command returns information on user-defined resource pools:

```
=> SELECT name, memorysize, maxmemorysize, priority, maxconcurrency
      FROM V_CATALOG.RESOURCE_POOLS where is_internal = 'f';
```

| name | memorysize | maxmemorysize | priority | maxconcurrency |
|--------------|------------|---------------|----------|----------------|
| load_pool | 0% | | 10 | |
| ceo_pool | 250M | | 10 | |
| ad hoc_pool | 200M | 200M | 0 | |
| billing_pool | 0% | | 0 | 3 |
| web_pool | 25M | | 10 | 5 |
| batch_pool | 150M | 150M | 0 | 10 |
| dept1_pool | 0% | | 5 | |
| dept2_pool | 0% | | 8 | |

```
(8 rows)
```

The queries borrow memory from the GENERAL pool and show the amount of memory in use from the GENERAL pool.

The following command uses the V_MONITOR.RESOURCE_POOL_STATUS table to return the current state of all resource pools on node0001:

```
=>\x
Expanded display is on
=> SELECT pool_name, memory_size_kb, memory_size_actual_kb, memory_inuse_kb,
      general_memory_borrowed_kb, running_query_count
      FROM V_MONITOR.RESOURCE_POOL_STATUS where node_name ilike '%node0001';
```

| -[RECORD 1]-----+----- | |
|----------------------------|----------|
| pool_name | general |
| memory_size_kb | 2983177 |
| memory_size_actual_kb | 2983177 |
| memory_inuse_kb | 0 |
| general_memory_borrowed_kb | 0 |
| running_query_count | 0 |
| -[RECORD 2]-----+----- | |
| pool_name | sysquery |
| memory_size_kb | 65536 |
| memory_size_actual_kb | 65536 |
| memory_inuse_kb | 0 |
| general_memory_borrowed_kb | 0 |
| running_query_count | 0 |

```

-[ RECORD 3 ]-----+-----
pool_name          | sysdata
memory_size_kb     | 102400
memory_size_actual_kb | 102400
memory_inuse_kb    | 4096
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 4 ]-----+-----
pool_name          | wosdata
memory_size_kb     | 0
memory_size_actual_kb | 0
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 5 ]-----+-----
pool_name          | tm
memory_size_kb     | 204800
memory_size_actual_kb | 204800
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 6 ]-----+-----
pool_name          | refresh
memory_size_kb     | 0
memory_size_actual_kb | 0
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 7 ]-----+-----
pool_name          | recovery
memory_size_kb     | 0
memory_size_actual_kb | 0
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 8 ]-----+-----
pool_name          | dbd
memory_size_kb     | 0
memory_size_actual_kb | 0
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 9 ]-----+-----
pool_name          | jvm
memory_size_kb     | 0
memory_size_actual_kb | 0
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0

```

The following command uses the `V_MONITOR.RESOURCE_ACQUISITIONS` table to show all resources granted to the queries that are currently running:

Note: While running `vmart_query_04.sql` from the [VMart example database](#), notice that the query uses `memory_inuse_kb = 708504` from the GENERAL pool.

```
=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb,
        queue_entry_timestamp, acquisition_timestamp
        FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';
```

| pool_name | thread_count | open_file_handle_count | memory_inuse_kb | queue_entry_timestamp | acquisition_timestamp |
|-----------|--------------|------------------------|-----------------|-------------------------------|-------------------------------|
| sysquery | 4 | 0 | 4103 | 2013-12-05 07:07:08.815362-05 | 2013-12-05 07:07:08.815367-05 |
| ... | ... | ... | ... | ... | ... |
| general | 12 | 18 | 708504 | 2013-12-04 12:55:38.566614-05 | 2013-12-04 12:55:38.566623-05 |
| ... | ... | ... | ... | ... | ... |

To determine how long a query waits in the queue before it is admitted to run, you can get the difference between the `acquisition_timestamp` and the `queue_entry_timestamp` using a query like the following:

```
=> SELECT pool_name, queue_entry_timestamp, acquisition_timestamp,
        (acquisition_timestamp-queue_entry_timestamp) AS 'queue wait'
        FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';
```

| pool_name | queue_entry_timestamp | acquisition_timestamp | queue wait |
|-----------|-------------------------------|-------------------------------|-----------------|
| sysquery | 2013-12-05 07:07:08.815362-05 | 2013-12-05 07:07:08.815367-05 | 00:00:00.000005 |
| sysquery | 2013-12-05 07:07:14.714412-05 | 2013-12-05 07:07:14.714417-05 | 00:00:00.000005 |
| sysquery | 2013-12-05 07:09:57.238521-05 | 2013-12-05 07:09:57.281708-05 | 00:00:00.043187 |
| ... | ... | ... | ... |

See the [SQL Reference Manual](#) for detailed descriptions of the monitoring tables described in this topic.

User Profiles

User profiles are attributes associated with a user that control that user's access to several system resources. These resources include:

- Resource pool to which a user is assigned (RESOURCE POOL)
- Maximum amount of memory a user's session can use (MEMORYCAP)
- Maximum amount of temporary file storage a user's session can use (TEMPSPACECAP)
- Maximum amount of time a user's query can run (RUNTIMECAP)

You can set these attributes with the [CREATE USER](#) statement and modify the attributes later with [ALTER USER](#).

Two strategies limit a user's access to resources: Setting attributes on the user directly to control resource use, or assigning the user to a resource pool. The first method lets you fine tune individual users, while the second makes it easier to group many users together and set their collective resource usage.

The following examples illustrate how to set a user's resource pool attributes. For additional examples, see the scenarios described in [Using User-Defined Pools and User-Profiles for Workload Management](#).

Example

Set the user's RESOURCE POOL attribute to assign the user to a resource pool. To create a user named `user1` who has access to the resource pool `my_pool`, use the command:

```
=> CREATE USER user1 RESOURCE POOL my_pool;
```

To limit the amount of memory for a user without designating a pool, set the user's MEMORYCAP to either a particular unit or a percentage of the total memory available. For example, to create a user named `user2` whose sessions are limited to using 200 MBs memory each, use the command:

```
=> CREATE USER user2 MEMORYCAP '200M';
```

To limit the time a user's queries are allowed to run, set the RUNTIMECAP attribute. To prevent queries for `user2` from running more than five minutes, use this command:

```
=> ALTER USER user2 RUNTIMECAP '5 minutes';
```

To limit the amount of temporary disk space that the user's sessions can use, set the TEMPSPACECAP to either a particular size or a percentage of temporary disk space available. For example, the next statement creates `user3`, and limits her to using 1 GB of temporary space:


```
=> CREATE USER user3 TEMPSPACECAP '1G';
```

You can combine different attributes into a single command. For example, to limit the MEMORYCAP and RUNTIMECAP for user3, include both attributes in an [ALTER USER](#) statement:

```
=> ALTER USER user3 MEMORYCAP '750M' RUNTIMECAP '10 minutes';
ALTER USER
=> \x
Expanded display is on.
=> SELECT * FROM USERS;
-[ RECORD 1 ]-----+-----
user_id      | 45035996273704962
user_name    | release
is_super_user | t
resource_pool | general
memory_cap_kb | unlimited
temp_space_cap_kb | unlimited
run_time_cap | unlimited
-[ RECORD 2 ]-----+-----
user_id      | 45035996273964824
user_name    | user1
is_super_user | f
resource_pool | my_pool
memory_cap_kb | unlimited
temp_space_cap_kb | unlimited
run_time_cap | unlimited
-[ RECORD 3 ]-----+-----
user_id      | 45035996273964832
user_name    | user2
is_super_user | f
resource_pool | general
memory_cap_kb | 204800
temp_space_cap_kb | unlimited
run_time_cap | 00:05
-[ RECORD 4 ]-----+-----
user_id      | 45035996273970230
user_name    | user3
is_super_user | f
resource_pool | general
memory_cap_kb | 768000
temp_space_cap_kb | 1048576
run_time_cap | 00:10
```

See Also

- [ALTER USER](#)
- [CREATE USER](#)

Target Memory Determination for Queries in Concurrent Environments

The resource pool parameters of MEMORYSIZE and PLANNEDCONCURRENCY ([CREATE RESOURCE POOL](#) in the SQL Reference Manual) provide the options that let you tune the target memory allocated to queries. The query_budget_kb column in the V_MONITOR.RESOURCE_POOL_STATUS system table shows the target memory for queries executed on the associated pool. Normally, queries do not require any specific tuning, but if needed, the general formula for computing query_budget_kb is as follows:

- If MEMORYSIZE is set to 0, in which case the pool borrows all memory as needed from the GENERAL pool, the target amount of memory for the query is calculated using the Queueing Threshold of the GENERAL pool / PLANNEDCONCURRENCY.
- If the resource pool for the query has the MEMORYSIZE parameter set, and the pool is standalone (i.e. cannot borrow from General pool) then the target memory is to use the amount of memory in the Queueing Threshold of the pool / PLANNEDCONCURRENCY.
- Otherwise, if MEMORYSIZE is set but the pool is not standalone, the target memory is set to MEMORYSIZE / PLANNEDCONCURRENCY of the pool.

Therefore, by carefully tuning the MEMORYSIZE and PLANNEDCONCURRENCY parameters, it is possible to restrict the amount of memory used by a query to a desired size.

See Also

- [User Profiles](#)
- [RESOURCE_POOL_STATUS](#)

Managing Resources At Query Run Time

The Resource Manager estimates the resources required for queries to run, and then determines when to run queries and when to queue them.

The Resource Manager also lets you manage resources that are assigned to queries that are already running using either of these methods:

- [Setting Run-Time Priority for the Resource Pool](#)—Use resource pool parameters to set the run time priority for queries running within the resource pool.
- [Changing Run-Time Priority of a Running Query](#)—Manually change the run time priority of a running query.

Setting Run-Time Priority for the Resource Pool

For each resource pool, you can manage resources that are assigned to queries that are already running. You assign each resource pool a *run-time priority* of HIGH, MEDIUM, or LOW. These settings determine the amount of run-time resources (such as CPU and I/O bandwidth) assigned to queries in the resource pool when they run. Queries in a resource pool with a HIGH priority are assigned greater runtime resources than those in resource pools with MEDIUM or LOW runtime priorities.

Prioritizing Queries Within a Resource Pool

While run-time priority helps to manage resources for the resource pool, there may be instances where you want some flexibility within a resource pool. For instance, you may want to ensure that very short queries run at a high priority, while also ensuring that all other queries run at a medium or low priority.

The Resource Manager allows you this flexibility by letting you set a *run-time priority threshold* for the resource pool. With this threshold, you specify a time limit (in seconds) by which a query must finish before it is assigned the runtime priority of the resource pool. All queries begin running with a HIGH priority; once a query's duration exceeds the time limit specified in the run-time priority threshold, it is assigned the run-time priority of the resource pool.

How to Set Run-Time Priority and Run-Time Priority Threshold

You specify run-time priority and run-time priority threshold when creating or modifying a resource pool. In the CREATE RESOURCE POOL or ALTER RESOURCE POOL statements, use these parameters:

| Parameter | Description |
|--------------------------|--|
| RUNTIMEPRIORITY | <p>Determines the amount of run-time resources (CPU, I/O bandwidth) the Resource Manager should dedicate to queries already running in the resource pool. Valid values are:</p> <ul style="list-style-type: none"> • HIGH • MEDIUM • LOW <p>Queries with a HIGH run-time priority are given more CPU and I/O resources than those with a MEDIUM or LOW run-time priority.</p> |
| RUNTIMEPRIORITYTHRESHOLD | <p>Specifies a time limit (in seconds) by which a query must finish before the Resource Manager assigns to it the RUNTIMEPRIORITY of the resource pool. All queries begin running at a HIGH priority. When a query's duration exceeds this threshold, it is assigned the RUNTIMEPRIORITY of the resource pool.</p> |

See Also

- [CREATE RESOURCE POOL](#)
- [ALTER RESOURCE POOL](#)

Changing Run-Time Priority of a Running Query

The `CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY` function allows you to change a query's run-time priority. When you run this function, you specify only the transaction ID and the run-time priority value you want to assign. HP Vertica changes the priority of the query that is currently running within the transaction.

Note: You cannot change the run-time priority of a query that has not yet begun executing.

Database administrators can change the run-time priority of any query to any level. Users can change the run-time priority of only their own queries. In addition, users cannot raise the run-time priority of a query to a level higher than that of the resource pools.

How To Change the Run-Time Priority of a Running Query

1. Use the following statement to see the run-time priority of all queries running in the session. Note the transaction ID for the query you want to change; you must specify the transaction ID to change the priority of the current query:

```
SELECT transaction_id, runtime_priority, transaction_description from SESSIONS;
```

2. Run the `CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY` meta-function, specifying the transaction ID for the query whose run-time priority you want to change:

```
SELECT CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY(45035996273705748, 'low')
```

Using `CHANGE_RUNTIME_PRIORITY`

The `CHANGE_RUNTIME_PRIORITY` function allows you to change the priority of a running query. Introduced in a previous release, this function required you to specify both the transaction ID and the statement ID of the query whose priority you wanted to change. With the introduction of `CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY`, you no longer need to specify the statement ID, and should therefore begin using `CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY` instead.

`CHANGE_RUNTIME_PRIORITY` will be removed in a future release. However, if you choose to use this function, you can avoid specifying the query statement ID by setting the statement ID value to `NULL`.

See Also

- [CHANGE_CURRENT_STATEMENT_RUNTIME_PRIORITY](#)
- [CHANGE_RUNTIME_PRIORITY](#)

Restoring Resource Manager Defaults

The system table `V_CATALOG.RESOURCE_POOL_DEFAULTS` stores default values for all parameters for all built-in and user-defined resource pools.

If you have changed the value of any parameter in any of your resource pools and want to restore it to its default, you can simply alter the table and set the parameter to `DEFAULT`. For example, the following statement sets the `RUNTIMEPRIORITY` for the resource pool `sysquery` back to its default value:

```
VMart=> ALTER RESOURCE POOL sysquery RUNTIMEPRIORITY DEFAULT;
```

See Also

- [RESOURCE_POOL_DEFAULTS](#)

Best Practices for Managing Workload Resources

This section provides general guidelines and best practices on how to set up and tune resource pools for various common scenarios.

Note: The exact settings for the pool parameters are heavily dependent on your query mix, data size, hardware configuration, and concurrency requirements. HP recommends performing your own experiments to determine the optimal configuration for your system.

Basic Principles for Scalability and Concurrency Tuning

An HP Vertica database runs on a cluster of commodity hardware. All loads and queries running against the database take up system resources, such as CPU, memory, disk I/O bandwidth, file handles, and so forth. The performance (run time) of a given query depends on how much resource it has been allocated.

When running more than one query concurrently on the system, both queries are sharing the resources; therefore, each query could take longer to run than if it was running by itself. In an efficient and scalable system, if a query takes up all the resources on the machine and runs in X time, then running two such queries would double the run time of each query to $2X$. If the query runs in $> 2X$, the system is not linearly scalable, and if the query runs in $< 2X$ then the single query was wasteful in its use of resources. Note that the above is true as long as the query obtains the minimum resources necessary for it to run and is limited by CPU cycles. Instead, if the system becomes bottlenecked so the query does not get enough of a particular resource to run, then the system has reached a limit. In order to increase concurrency in such cases, the system must be expanded by adding more of that resource.

In practice, HP Vertica should achieve near linear scalability in run times, with increasing concurrency, until a system resource limit is reached. When adequate concurrency is reached without hitting bottlenecks, then the system can be considered as ideally sized for the workload.

Note: Typically HP Vertica queries on segmented tables run on multiple (likely all) nodes of the cluster. Adding more nodes generally improves the run time of the query almost linearly.

Guidelines for Setting Pool Parameters

This section provides guidelines on setting the various parameters of any resource pool. You should tune resource pools only to address specific workload issues. The default configuration is designed for a balanced, high throughput environment. See [Using User-Defined Pools and User-Profiles for Workload Management](#) for examples of situations where you might want to create your own pools.

Note: Consider the following computational resources of your database cluster nodes to ensure that your pool parameter settings optimize the performance of the resource pools:

- CPU
- RAM
- Storage subsystems
- Network bandwidth

| Parameter | Guideline |
|----------------------|---|
| MEMORYSIZE | <p>Ignore if tuning the GENERAL pool.</p> <p>For other pools, you can leave the setting to the default (0%), which allows the pool to borrow memory from the General pool, as needed. Consider setting this in the following situations:</p> <ul style="list-style-type: none"> • To set aside memory for exclusive use of requests issued to that pool. This memory then cannot be used for other purposes, even if there are no requests made to that pool. • In combination with PLANNEDCONCURRENCY, to tune the memory used by a certain query to a certain size, where applicable. This is for expert use only. <p>See Target Memory Determination for Queries in Concurrent Environments.</p> |
| MAXMEMORYSIZE | <p>Ignore if tuning the GENERAL pool.</p> <p>For other pools, use this parameter to set up the resource pool as a standalone pool or to limit how much memory the resource pool can borrow from the GENERAL pool. This provides a mechanism to enforce a hard limit on the memory usage by certain classes of workload; for example, loads should take up no more than 4GB of total available memory.</p> <p>See Scenario: Restricting resource usage and concurrency of ad-hoc application</p> |
| EXECUTIONPARALLELISM | <p>Use this parameter to limit the number of threads that would be used to process any single query issued in this resource pool. Reducing this parameter may increase the throughput of short queries issued in the pool, especially if the queries are executed concurrently. If you choose the default of AUTO, HP Vertica sets this value for you. If you choose to manually set this parameter, set it to a value between 1 and the number of cores.</p> |

| | |
|---------------------------------|--|
| <p>PRIORITY</p> | <p>Use this parameter to prioritize the use of GENERAL pool resources, either by requests made directly to the GENERAL pool, or by requests made to other pools borrowing memory from the GENERAL pool.</p> <p>The PRIORITY for internal pools (SYSQUERY, RECOVERY, and TM) ranges from –110 to 110. Administrator-created pools have a PRIORITY range of –100 to 100. Internal pools should <i>always</i> have a higher priority than created pools.</p> <p>Pool PRIORITY is relative; a pool with a higher PRIORITY value has the same increased access to pool resources as a pool with a lower PRIORITY value, regardless of the difference between their PRIORITY values.</p> <p>When you install HP Vertica, requests made to the RECOVERY pool have highest priority out of the box so that HP Vertica can expeditiously provide resources to recover nodes that are down.</p> <p>Note: The PRIORITY setting has no meaning for a standalone pool, which does not borrow memory from the GENERAL pool. See examples in Scenario: Periodic Batch Loads and Scenario: Setting Priorities on Queries Issued By Different Users.</p> |
| <p>RUNTIMEPRIORITY</p> | <p>Use this parameter to set the priority for running queries in this resource pool. Any query with a duration that exceeds the value in the RUNTIMEPRIORITYTHRESHOLD property will be assigned the run-time priority you specify here.</p> <ul style="list-style-type: none"> • If you want to ensure that short queries will always run at a high priority, set the RUNTIMEPRIORITY parameter to MEDIUM or LOW and set the RUNTIMEPRIORITYTHRESHOLD to a small number that will accommodate your short queries. • If you want all queries in the resource pool to always run a the specified value, set the RUNTIMEPRIORITY parameter to HIGH, MEDIUM, or LOW and also set the RUNTIMEPRIORITYTHRESHOLD to 0. Setting RUNTIMEPRIORITYTHRESHOLD to 0 effectively turns off the RUNTIMEPRIORITYTHRESHOLD feature. All queries will run with the RUNTIMEPRIORITY of the resource pool. |
| <p>RUNTIMEPRIORITYTHRESHOLD</p> | <p>[Default: 2] Use this parameter to specify the duration (in seconds) of queries that should always run with HIGH run-time priority. Because all queries begin running with a RUNTIMEPRIORITY of HIGH, queries that finish within the specified threshold will run at a HIGH priority; all other queries will be assigned the runtime priority assigned to the resource pool.</p> <p>To disable this feature, set the RUNTIMEPRIORITYTHRESHOLD to 0.</p> |

| | |
|---------------|---|
| QUEUE_TIMEOUT | <p>Use this parameter to change the timeout of the pool from the 5-minute default.</p> <p>The timeout can be customized if you need different queuing durations for different classes of workloads. For example, long-running batch workloads could be configured to run on a pool with high timeout values, possibly unlimited, since completion of the task is more critical than response time.</p> <p>For interactive application queries, the timeouts could be set to low or 0 to ensure application gets an immediate error if the query cannot run.</p> <p>Note: Be mindful that increased timeouts will lead to longer queue lengths and will not necessarily improve the overall throughput of the system.</p> |
|---------------|---|

| | |
|---------------------------|--|
| <p>PLANNEDCONCURRENCY</p> | <p>This parameter specifies the typical number of queries running concurrently in the system. Set PLANNEDCONCURRENCY to AUTO to specify that HP Vertica should calculate this number. HP Vertica takes the lower of these two values:</p> <ul style="list-style-type: none"> • Number of cores • Memory/2GB <p>The minimum value for PLANNEDCONCURRENCY is 4.</p> <p>HP Vertica advises changing this value only after evaluating performance over a period of time.</p> <p>The Tuple Mover draws its resources from the TM pool. For the TM pool, the PLANNEDCONCURRENCY parameter must be proportional to the size of the RAM, the CPU, and the storage subsystem. Depending on the storage type, if you increase PLANNEDCONCURRENCY for the Tuple Mover threads, you might create storage I/O bottleneck. Monitor the storage subsystem; if it becomes saturated with long I/O queues, more than two I/O queues, and long latency in read and write, adjust the PLANNEDCONCURRENCY parameter to keep the storage subsystem resources below saturation level. In addition, you might need to:</p> <ul style="list-style-type: none"> • Partition storage data files • Adjust block-size optimization on storage subsystems such as RAID 5 or RAID 10 <p>Identify the optimal number of disks in the RAID array.</p> <p>Notes:</p> <ul style="list-style-type: none"> • Consider the tradeoff between giving each query its maximum amount of resources and allowing many concurrent queries to run in a reasonable amount of time. For more information, see Target Memory Determination for Queries in Concurrent Environments. • This parameter can be used in combination with MEMORYSIZE to tune the memory used by a query down to a specific size. • For clusters where the number of cores differs on different nodes, AUTO can apply differently on each node. Distributed queries run like the minimal effective planned concurrency. Single node queries run with the planned concurrency of the initiator. • If you created or upgraded your database in 4.0 or 4.1, the PLANNEDCONCURRENCY setting on the GENERAL pool defaults to a too-small value for machines with large numbers of cores. To |
|---------------------------|--|

| | |
|-----------------|--|
| | <p>adjust to a more appropriate value:</p> <pre>=> ALTER RESOURCE POOL general PLANNEDCONCURRENCY <#cores>;</pre> <p>This parameter only needs to be set if you created a database before 4.1, patchset 1.</p> |
| MAXCONCURRENCY | <p>Use this parameter if you want to impose a hard limit on the number of concurrent requests that are allowed to run against any pool, including the GENERAL pool.</p> <p>Instead of limiting this at the pool level, it is also possible to limit at the connection level using MaxClientSessions.</p> |
| RUNTIMECAP | <p>Use this parameter to prevent runaway queries. This parameter sets the maximum amount of time any query on the pool can execute. Set RUNTIMECAP using interval, such as '1 minute' or '100 seconds' (see Interval Values for details). This value cannot exceed one year. Setting this value to NONE means there is no time limit on queries running on the pool. If the user or session also has a RUNTIMECAP, the shorter limit applies.</p> |
| SINGLEINITIATOR | <p>This parameter is included for backwards compatibility only. Do not change the default (false) value.</p> |
| CPUAFFINITYSET | <p>Use this parameter if you want to pin the execution of queries in this resource pool to a specified subset of CPUs in the cluster. All nodes in the cluster must have the same number of CPUs. To determine the number of CPUs on a node, issue the command: <code>lscpu grep "^CPU(s)"</code>.</p> <p>This parameter is supported only on user-defined resource pools. However, you limit the CPUs available to the general resource pools by creating a user-defined resource pool that has exclusive use of one or more CPU resources.</p> |
| CPUAFFINITYMODE | <p>Use this parameter to specify if the resource pool has exclusive or shared use of the pinned CPUs defined in CPUAFFINITYSET.</p> |

See Also

- [CREATE RESOURCE POOL](#)
- [ALTER RESOURCE POOL](#)

Setting a Run-Time Limit for Queries

You can set a limit for the amount of time a query is allowed to run using the RUNTIMECAP parameter. You can set this parameter for a:

- User, in the user's profile (See [CREATE USER](#))
- Resource pool (See [CREATE RESOURCE POOL](#))
- Session (See [SET SESSION RUNTIMECAP](#))

In all cases, you set this parameter as an [interval value](#), and the value cannot exceed one year. When `RUNTIMECAP` has been set for two or more of these levels, HP Vertica always uses the shortest value.

Example:

- User1 is assigned to the `ad_hoc_queries` resource pool
- `RUNTIMECAP` for User1 is set to 1 hour
- `RUNTIMECAP` for the `ad_hoc_queries` resource pool is set to 30 minutes

In this example, HP Vertica terminates any of User1's queries if they surpass the 30-minute `RUNTIMECAP` for the resource pool.

See Also

- [RESOURCE_POOLS](#)

Using User-Defined Pools and User-Profiles for Workload Management

The scenarios in this section describe some of the most common workload-management issues and provide solutions with examples.

Scenario: Periodic Batch Loads

Scenario

You do batch loads every night, or occasionally (infrequently) during the day. When loads are running, it is acceptable to reduce resource usage by queries, but at all other times you want all resources to be available to queries.

Solution

Create a separate resource pool for loads with a higher priority than the preconfigured setting on the build-in GENERAL pool.

In this scenario, nightly loads get preference when borrowing memory from the GENERAL pool. When loads are not running, all memory is automatically available for queries.

Note: If you are using the **WOS**, tune the **PLANNEDCONCURRENCY** parameter of the **WOSDATA** pool to the number of concurrent loads. This ensures that **AUTO** spill to **ROS** is configured in an optimal fashion.

Example

Create a resource pool with the **PRIORITY** of the pool set higher than the **GENERAL** pool.

For example, to create a pool designated for loads that has a higher priority than the **GENERAL** pool, set **load_pool** with a priority of 10:

```
=> CREATE RESOURCE POOL load_pool PRIORITY 10;
```

Edit the **WOSDATA** pool **PLANNEDCONCURRENCY**:

```
=> ALTER RESOURCE POOL WOSDATA PLANNEDCONCURRENCY 6;
```

Modify the user's resource pool:

```
=> ALTER USER load_user RESOURCE POOL load_pool;
```

Scenario: The CEO Query

Scenario

The CEO runs a report every Monday at 9AM, and you want to be sure that the report always runs.

Solution

To ensure that a certain query or class of queries always gets resources, you could create a dedicated pool for it as follows:

1. Using the [PROFILE](#) command, run the query that the CEO runs every week to determine how much memory should be allocated:

```
=> PROFILE SELECT DISTINCT s.product_key, p.product_description
-> FROM store.store_sales_fact s, public.product_dimension p
-> WHERE s.product_key = p.product_key AND s.product_version = p.product_version
-> AND s.store_key IN (
->   SELECT store_key FROM store.store_dimension
->   WHERE store_state = 'MA')
-> ORDER BY s.product_key;
```

2. At the end of the query, the system returns a notice with resource usage:

```
NOTICE: Statement is being profiled.HINT: select * from v_monitor.execution_engine_
profiles where
transaction_id=45035996273751349 and statement_id=6;
NOTICE: Initiator memory estimate for query: [on pool general: 1723648 KB,
minimum: 355920 KB]
```

3. Create a resource pool with MEMORYSIZE reported by the above hint to ensure that the CEO query has at least this memory reserved for it:

```
=> CREATE RESOURCE POOL ceo_pool MEMORYSIZE '1800M' PRIORITY 10;
CREATE RESOURCE POOL
=> \x
Expanded display is on.
=> SELECT * FROM resource_pools WHERE name = 'ceo_pool';
-[ RECORD 1 ]-----+-----
name                | ceo_pool
is_internal          | f
memorysize           | 1800M
maxmemorysize       |
priority             | 10
queuetimeout         | 300
plannedconcurrency  | 4
```

```
maxconcurrency      |
singleinitiator     | f
```

4. Assuming the CEO report user already exists, associate this user with the above resource pool using ALTER USER statement.

```
=> ALTER USER ceo_user RESOURCE POOL ceo_pool;
```

5. Issue the following command to confirm that the ceo_user is associated with the ceo_pool:

```
=> SELECT * FROM users WHERE user_name = 'ceo_user';
-[ RECORD 1 ]-+-----
user_id      | 45035996273713548
user_name    | ceo_user
is_super_user | f
resource_pool | ceo_pool
memory_cap_kb | unlimited
```

If the CEO query memory usage is too large, you can ask the Resource Manager to reduce it to fit within a certain budget. See [Target Memory Determination for Queries in Concurrent Environments](#).

Scenario: Preventing Run-Away Queries

Scenario

Joe, a business analyst often runs big reports in the middle of the day that take up the whole machine's resources. You want to prevent Joe from using more than 100MB of memory, and you want to also limit Joe's queries to run for less than 2 hours.

Solution

[User Profiles](#) provides a solution to this scenario. To restrict the amount of memory Joe can use at one time, set a MEMORYCAP for Joe to 100MB using the [ALTER USER](#) command. To limit the amount of time that Joe's query can run, set a RUNTIMECAP to 2 hours using the same command. If any query run by Joe takes up more than its cap, HP Vertica rejects the query.

If you have a whole class of users whose queries you need to limit, you can also create a resource pool for them and set RUNTIMECAP for the resource pool. When you move these users to the resource pool, HP Vertica limits all queries for these users to the RUNTIMECAP you specified for the resource pool.

Example

```
=> ALTER USER analyst_user MEMORYCAP '100M' RUNTIMECAP '2 hours';
```

If Joe attempts to run a query that exceeds 100MB, the system returns an error that the request exceeds the memory session limit, such as the following example:

```
\i vmart_query_04.sqlvsq1:vmart_query_04.sql:12: ERROR: Insufficient resources to initiate plan
on pool general [Request exceeds memory session limit: 137669KB > 102400KB]
```

Only the system database administrator (dbadmin) can increase only the MEMORYCAP setting. Users cannot increase their own MEMORYCAP settings and will see an error like the following if they attempt to edit their MEMORYCAP or RUNTIMECAP settings:

```
ALTER USER analyst_user MEMORYCAP '135M';
ROLLBACK: permission denied
```

Scenario: Restricting Resource Usage of Ad Hoc Query Application

Scenario

You recently made your data warehouse available to a large group of users who are not experienced SQL users. Some of the users run reports that operate on a large number of rows and overwhelm the system. You want to throttle usage of the system by these users.

Solution

The simplest solution is to create a standalone resource pool for the ad hoc applications so that the total MEMORYSIZE is fixed. Recall that in a standalone pool, MAXMEMORYSIZE is set equal to MEMORYSIZE so no memory can be borrowed from the GENERAL pool. Associate this user pool with the database user(s) from which the application uses to connect to the database. In addition, set RUNTIMECAP to limit the maximum duration of an ad hoc query.

Other solutions include limiting the memory usage of individual users such as in the [Scenario: Preventing Run-Away Queries](#).

Tip: Besides adding limits such as the above, it is also a great idea to train the user community on writing good SQL.

Example

To create a standalone resource pool for the ad hoc users, set the MEMORYSIZE equal to the MAXMEMORYSIZE:

```
=> CREATE RESOURCE POOL adhoc_pool MEMORYSIZE '200M' MAXMEMORYSIZE '200M' PRIORITY 0 Q
UEUETIMEOUT 300 PLANNEDCONCURRENCY 4;
=> SELECT pool_name, memory_size_kb, queueing_threshold_kb
FROM V_MONITOR.RESOURCE_POOL_STATUS w
WHERE is_standalone = 'true' AND is_internal = 'false';
```



```

pool_name | memory_size_kb | queueing_threshold_kb
-----+-----+-----
adhoc_pool |          204800 |          153600
(1 row)

```

After the pool has been created, associate the ad hoc users with the adhoc_pool:

```

=> ALTER USER app1_user RESOURCE POOL adhoc_pool;=> ALTER RESOURCE POOL adhoc_pool MEMORY
SIZE '10M' MAXMEMORYSIZE '10M';
\i vmart_query_04.sql
vsq1:vmart_query_04.sql:12: ERROR: Insufficient resources to initiate plan
on pool adhoc_pool [Request Too Large:Memory(KB)
Exceeded: Requested = 84528, Free = 10240 (Limit = 10240, Used = 0)]

```

The query will not borrow memory from the GENERAL pool and gets rejected with a 'Request Too Large' message.

Scenario: Setting a Hard Limit on Concurrency For An Application

Scenario

For billing purposes, analyst Jane would like to impose a hard limit on concurrency for this application. How can she achieve this?

Solution

The simplest solution is to create a separate resource pool for the users of that application and set its MAXCONCURRENCY to the desired concurrency level. Any queries beyond MAXCONCURRENCY are queued.

Tip: HP recommends leaving PLANNEDCONCURRENCY to the default level so the queries get their maximum amount of resources. The system as a whole thus runs with the highest efficiency.

Example

In this example, there are four billing users associated with the billing pool. The objective is to set a hard limit on the resource pool so a maximum of three concurrent queries can be executed at one time. All other queries will queue and complete as resources are freed.

```

=> CREATE RESOURCE POOL billing_pool MAXCONCURRENCY 3 QUEUETIMEOUT 2;
=> CREATE USER bill1_user RESOURCE POOL billing_pool;
=> CREATE USER bill2_user RESOURCE POOL billing_pool;
=> CREATE USER bill3_user RESOURCE POOL billing_pool;
=> CREATE USER bill4_user RESOURCE POOL billing_pool;
=> \x

```

Expanded display is on.

```
=> select maxconcurrency,queuetimeout from resource_pools where name = 'billing_pool';
maxconcurrency | queuetimeout
-----+-----
                3 |                2
(1 row)
> SELECT reason, resource_type, rejection_count FROM RESOURCE_REJECTIONS
WHERE pool_name = 'billing_pool' AND node_name ilike '%node0001';
reason | resource_type | rejection_count
-----+-----+-----
Timeout waiting for resource request | Queries | 16
(1 row)
```

If queries are running and do not complete in the allotted time (default timeout setting is 5 minutes), the next query requested gets an error similar to the following:

```
ERROR: Insufficient resources to initiate plan on pool billing_pool [Timeout waiting fo
r resource request: Request exceeds limits:
Queries Exceeded: Requested = 1, Free = 0 (Limit = 3, Used = 3)]
```

The table below shows that there are three active queries on the billing pool.

```
=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb FROM RESOURCE_
ACQUISITIONS
WHERE pool_name = 'billing_pool';
pool_name | thread_count | open_file_handle_count | memory_inuse_kb
-----+-----+-----+-----
billing_pool | 4 | 5 | 132870
billing_pool | 4 | 5 | 132870
billing_pool | 4 | 5 | 132870
(3 rows)
```

Scenario: Handling Mixed Workloads (Batch vs. Interactive)

Scenario

You have a web application with an interactive portal. Sometimes when IT is running batch reports, the web page takes a long time to refresh and users complain, so you want to provide a better experience to your web site users.

Solution

The principles learned from the previous scenarios can be applied to solve this problem. The basic idea is to segregate the queries into two groups associated with different resource pools. The prerequisite is that there are two distinct database users issuing the different types of queries. If this is not the case, do consider this a best practice for application design.

METHOD 1: Create a dedicated pool for the web page refresh queries where you:

1. Size the pool based on the average resource needs of the queries and expected number of concurrent queries issued from the portal.
2. Associate this pool with the database user that runs the web site queries. (See [Scenario: The CEO Query](#) for detailed procedure on creating a dedicated pool.)
3. This ensures that the web site queries always run and never queue behind the large batch jobs. Leave the batch jobs to run off the GENERAL pool.

For example, the following pool is based on the average resources needed for the queries running from the web and the expected number of concurrent queries. It also has a higher PRIORITY to the web queries over any running batch jobs and assumes the queries are being tuned to take 250M each:

```
CREATE RESOURCE POOL web_pool MEMORYSIZE '250M' MAXMEMORYSIZE NONE PRIORITY 10 MAXCONCURRENCY 5 PLANNEDCONCURRENCY 1
```

METHOD 2: Create a standalone pool to limit the batch reports down to a fixed memory size so memory is always left available for other purposes. (See [Scenario: Restricting Resource Usage of Ad Hoc Query Application](#).)

For example:

```
CREATE RESOURCE POOL batch_pool MEMORYSIZE '4G' MAXMEMORYSIZE '4G' MAXCONCURRENCY 10;
```

The same principle can be applied if you have three or more distinct classes of workloads.

Scenario: Setting Priorities on Queries Issued By Different Users

Scenario

You would like user queries from one department to have a higher priority than queries from another department.

Solution

The solution is similar to the [mixed workload case](#). In this scenario, you do not limit resource usage; you set different priorities. To do so, create two different pools, each with MEMORYSIZE=0% and a different PRIORITY parameter. Both pools borrow from the GENERAL pool, however when competing for resources, the priority determine the order in which each pool's request is granted.

For example:

```
=> CREATE RESOURCE POOL dept1_pool PRIORITY 5;
=> CREATE RESOURCE POOL dept2_pool PRIORITY 8;
```

If you find this solution to be insufficient, or if one department's queries continuously starves another department's users, you could add a reservation for each pool by setting `MEMORYSIZE` so some memory is guaranteed to be available for each department.

For example, since both resources are using the `GENERAL` pool for memory, you could allocate some memory to each resource pool by using the `ALTER RESOURCE POOL` command to change the `MEMORYSIZE` for each pool:

```
=> ALTER RESOURCE POOL dept1_pool MEMORYSIZE '100M';  
=> ALTER RESOURCE POOL dept2_pool MEMORYSIZE '150M';
```

Scenario: Continuous Load and Query

Scenario

You want your application to run continuous load streams, but many have up concurrent query streams. You want to ensure that performance is predictable.

Solution

The solution to this scenario will depend on your query mix; however, below are the general steps to take:

1. Determine the number of continuous load streams required. This may be related to the desired load rate if a single stream does not provide adequate throughput, or may be more directly related to the number of sources of data to load. Also determine if automatic storage is best, or if `DIRECT` is required. Create a dedicated resource pool for the loads, and associate it with the database user that will perform them. See [CREATE RESOURCE POOL](#) for details.

In general, the concurrency settings for the load pool should be less than the number of cores per node. Unless the source processes are slow, it is more efficient to dedicate more memory per load, and have additional loads queue. Adjust the load pool's `QUEUETIMEOUT` setting if queuing is expected.

2. If using automatic targeting of `COPY` and `INSERT`, set the `PLANNEDCONCURRENCY` parameter of the `WOSDATA` pool to the number of concurrent loads expected. Also, set `MEMORYSIZE` of the `WOS` to the expected size of the loaded data to ensure that small loads don't spill to `ROS` immediately. See [Built-In Pools](#) for details.
3. Run the load workload for a while and observe whether the load performance is as expected. If the Tuple Mover is not tuned adequately to cover the load behavior, see [Tuning the Tuple Mover](#) in Administrator's Guide.
4. If there is more than one kind of query in the system (say some queries that must be answered quickly for interactive users, and others that are part of a batch reporting process), follow the advice in [Scenario: Handling Mixed Workloads](#).

- Let the queries run and observe the performance. If some classes of queries are not getting the desired performance, then it may be necessary to tune the GENERAL pool as outlined in [Scenario: Restricting Resource Usage of Ad Hoc Query Application](#), or to create further dedicated resource pools for those queries. See [Scenario: The CEO Query](#) and [Scenario: Handling Mixed Workloads](#).

See the sections on [Managing Workloads](#) and [CREATE RESOURCE POOL](#) for additional details and tips for obtaining predictable results in mixed workload environments.

Scenario: Prioritizing Short Queries At Run Time

Scenario

You recently created a resource pool for users who are not experienced with SQL and who frequently run ad hoc reports. You have managed resource allocation by creating a standalone resource pool that will prevent these queries from borrowing resources from the GENERAL pool, but now you want to manage resources at run time and ensure that short queries always run with a high priority and are never queued as a result of limited run-time resources.

Solution

Set the `RUNTIMEPRIORITY` for the resource pool to `MEDIUM` or `LOW`. Set the `RUNTIMEPRIORITYTHRESHOLD` for the resource pool to the duration of queries you want to ensure always run at a high priority. For instance, if you set this value to 5, all queries that complete within 5 seconds will run at high priority. Any other query that exceeds 5 seconds will drop down to the `RUNTIMEPRIORITY` assigned to the resource pool (`MEDIUM` or `LOW`).

Example

To ensure that all queries with a duration of less than 5 seconds always run at a high priority, modify `adhoc_pool` as follows:

- Set the `RUNTIMEPRIORITY` to `MEDIUM`
- Set the `RUNTIMETHRESHOLD` to 5

```
=> ALTER RESOURCE POOL ad_hoc_pool RUNTIMEPRIORITY medium RUNTIMEPRIORITYTHRESHOLD 5;
```

Scenario: Dropping the Runtime Priority of Long Queries

Scenario

You want most queries in a resource pool to run at a `HIGH` runtime priority; however, you'd like to be able to drop jobs longer than 1 hour to a lower priority.

Solution

Set the `RUNTIMEPRIORITY` for the resource pool to `LOW` and set the `RUNTIMEPRIORITYTHRESHOLD` to a number that cuts off only the longest jobs.

Example

To ensure that all queries with a duration of more than 3600 seconds (1 hour) are assigned a low runtime priority, modify the resource pool as follows:

- Set the `RUNTIMEPRIORITY` to `LOW`.
- Set the `RUNTIMEPRIORITYTHRESHOLD` to 3600

```
=> ALTER RESOURCE POOL ad_hoc_pool RUNTIMEPRIORITY low RUNTIMEPRIORITYTHRESHOLD 3600;
```

Tuning the Built-In Pools

The scenarios in this section describe how to tune the built-in pools.

Scenario: Restricting HP Vertica to Take Only 60% of Memory

Scenario

You have a single node application that embeds HP Vertica, and some portion of the RAM needs to be devoted to the application process. In this scenario, you want to limit HP Vertica to use only 60% of the available RAM.

Solution

Set the MAXMEMORYSIZE parameter of the GENERAL pool to the desired memory size. See [Resource Pool Architecture](#) for a discussion on resource limits.

Scenario: Tuning for Recovery

Scenario

You have a large database that contains a single large table with two projections, and with default settings, recovery is taking too long. You want to give recovery more memory to improve speed.

Solution

Set the PLANNEDCONCURRENCY and MAXCONCURRENCY setting of the recovery pool to 1 so that recovery can take as much memory as possible from the GENERAL pool and run only one thread at once.

Note: This setting could slow down other queries in your system.

Scenario: Tuning for Refresh

Scenario

When a **refresh** operation is running, system performance is affected and user queries get rejected. You want to reduce the memory usage of the refresh job.

Solution

Set the MEMORYSIZE parameter of the refresh pool to a fixed value. The Resource Manager then tunes the refresh query to only use this amount of memory.

Tip: Remember to reset the refresh pool MEMORYSIZE back to 0% after the refresh operation completes so memory can be used for other operations.

Scenario: Tuning Tuple Mover Pool Settings

Scenario

During loads, you occasionally notice spikes in the number of **ROS** containers, and you would like to make the **Tuple Mover** more aggressive.

Solution

Increase the MAXCONCURRENCY parameter of the TM pool to 3 or higher. This setting ensures that the Tuple Mover can run more than one **mergeout** thread, so if a large mergeout is in progress, smaller ROS containers can also be merged, thus preventing a buildup of ROS containers.

Reducing Query Run-Time

The run time of queries depends on the complexity of the query, the number of operators in the plan, data volumes, and projection design. If the system is bottlenecked on either I/O or CPU, queries could run more slowly than expected. In most cases, high CPU usage can be alleviated by better projection design, and high I/O is usually due to contention because of operations like joins and sorts that spill to disk. However, there is no single solution to fix high CPU or high I/O usage, so queries must be examined and tuned individually.

Two primary ways to determine why a query is slow are:

- Examine the query plan using the EXPLAIN command
- Examine the execution profile by querying the EXECUTION_ENGINE_PROFILES system table

Examining the query plan usually reveals one or more more of the following:

- Suboptimal sort order of a projection
- Cases where predicate evaluation occurs on an unsorted or unencoded column

Note: Although you cannot see that a partitioned hash join occurred in the plan, you can see that when the optimizer chose a hash join.

- Presence of group by hash rather than pipeline

See [Creating Custom Designs](#) to understand projection design techniques. The Database Designer automatically applies these techniques to suggest optimal designs for queries.

Real-Time Profiling

HP Vertica provides profiling mechanisms that let you determine how well the database is performing. For example, HP Vertica can collect profiling data for a single statement, a single session, or for all sessions on all nodes.

Real-time profiling is always "on", without profiling being explicitly enabled.

For details, see in the [Profiling Database Performance](#) Administrator's Guide and, in particular:

- [Profiling a Single Statement](#)
- [About Real-Time Profiling](#)
- [Viewing Profiling Data](#)
- [Viewing real-time profile data](#)

See Also

- [EXECUTION_ENGINE_PROFILES](#)

Managing System Resource Usage

You can use the [Using System Tables](#) to track overall resource usage on your cluster. These and the other system tables are described in the [SQL Reference Manual](#).

If your queries are experiencing errors due to resource unavailability, you can use the following system tables to obtain more details:

| System Table | Description |
|--|--|
| RESOURCE_REJECTIONS | Monitors requests for resources that are rejected by the Resource Manager . |
| DISK_RESOURCE_REJECTIONS | Monitors requests for resources that are rejected due to disk space shortages. See Managing Disk Space for more information. |

When requests for resources of a certain type are being rejected, do one of the following:

- Increase the resources available on the node by adding more memory, more disk space, and so on. See [Managing Disk Space](#).
- Reduce the demand for the resource by reducing the number of users on the system (see [Managing Sessions](#)), rescheduling operations, and so on.

The `LAST_REJECTED_VALUE` field in `RESOURCE_REJECTIONS` indicates the cause of the problem. For example:

- The message `Usage of a single requests exceeds high limit` means that the system does not have enough of the resource available for the single request. A common example occurs when the file handle limit is set too low and you are loading a table with a large number of columns.
- The message `Timed out or Canceled waiting for resource reservation` usually means that there is too much contention for the resource because the hardware platform cannot support the number of concurrent users using it.

See Also

- [Guidelines for Setting Pool Parameters](#)

Managing Sessions

HP Vertica provides powerful methods for database administrators to view and control sessions. The methods vary according to the type of session:

- External (user) sessions are initiated by vsql or programmatic (ODBC or JDBC) connections and have associated client state.

- Internal (system) sessions are initiated by the HP Vertica database process and have no client state.

You can view a list of currently active sessions (including internal sessions) and can interrupt or close external sessions when necessary, particularly when [shutting down the database](#).

By default HP Vertica allows 50 client sessions and an additional 5 administrator sessions. You can modify connection settings with the `MaxClientSessions` parameter. For example, to increase the number of `MaxClientSessions` to 100, issue the following command at a vsql prompt:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 100);
```

To prevent new non-dbadmin sessions from connecting, set `MaxClientSessions` to 0:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

Viewing Sessions

HP Vertica provides the `SESSIONS` table to view the session status of your database. `SESSIONS` contains information about external sessions and returns one row per session. This table is described in the SQL Reference Manual.

Note: Superuser has unrestricted access to all database metadata. Users have significantly reduced access to metadata based on their privileges. See [Metadata Privileges](#).

Interrupting and Closing Sessions

- Interrupting a running statement returns an enclosing session to an idle state, meaning no statements or transactions are running, no locks are held, and the database is doing no work on behalf of the session. If no statement is running, you get an error.
- Closing a session interrupts the session and disposes of all state related to the session, including client socket connections for external sessions.

These actions are provided in the form of SQL functions, described in the SQL Reference Manual:

- [INTERRUPT_STATEMENT](#)
- [CLOSE_SESSION](#)
- [CLOSE_ALL_SESSIONS](#)
- [SHUTDOWN](#)

`SELECT` statements that call these functions return when the interrupt or close message has been delivered to all nodes, not after the interrupt or close has completed. This means there might be a

delay after the statement returns and the interrupt or close taking effect throughout the cluster. To determine if the session or transaction has ended, you can monitor the SESSIONS system table.

Controlling Sessions

The database administrator must be able to disallow new incoming connections in order to shut down the database. On a busy system, database shutdown is prevented if new sessions connect after the CLOSE_SESSION or CLOSE_ALL_SESSIONS() command is invoked—and before the database actually shuts down.

One option is for the administrator to issue the SHUTDOWN('true') command, which forces the database to shut down and disallow new connections. See [SHUTDOWN](#) in the SQL Reference Manual.

Another option is to modify the MaxClientSessions parameter from its original value to 0, in order to prevent new non-dbadmin users from connecting to the database.

1. Determine the original value for the MaxClientSessions parameter by querying the V_MONITOR.CONFIGURATIONS_PARAMETERS system table:

```
=> SELECT CURRENT_VALUE FROM CONFIGURATION_PARAMETERS WHERE parameter_name='MaxClient
Sessions';
CURRENT_VALUE
-----
50
(1 row)
```

2. Set the MaxClientSessions parameter to 0 to prevent new non-dbadmin connections:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 0);
```

Note: The previous command allows up to five administrators to log in.

3. Issue the CLOSE_ALL_SESSIONS() command to remove existing sessions:

```
=> SELECT CLOSE_ALL_SESSIONS();
```

4. Query the SESSIONS table:

```
=> SELECT * FROM SESSIONS;
```

When the session no longer appears in the SESSIONS table, disconnect and run the [Stop Database](#) command.

5. Restart the database.

6. Restore the MaxClientSessions parameter to its original value:

```
=> SELECT SET_CONFIG_PARAMETER('MaxClientSessions', 50);
```

Administrator's Guide

SQL Reference Manual

See Also

- [Configuration Parameters](#)
- [Stopping a Database](#)
- [Shutdown Problems](#)
- [CONFIGURATION_PARAMETERS](#)
- [CLOSE_ALL_SESSIONS](#)
- [CLOSE_SESSION](#)
- [INTERRUPT_STATEMENT](#)
- [SESSIONS](#)
- [SHUTDOWN](#)

Managing Load Streams

You can use the [Using System Tables](#) to keep track of data being loaded on your cluster.

| System Table | Description |
|------------------------------|--|
| LOAD_STREAMS | Monitors load metrics for each load stream on each node. |

These and the other SQL Monitoring API system tables are described in detail in the [SQL Reference Manual](#).

When a COPY statement using the DIRECT option is in progress, the ACCEPTED_ROW_COUNT field can increase to the maximum number of rows in the input file as the rows are being parsed.

If COPY reads input data from multiple named pipes, the PARSE_COMPLETE_PERCENT field will remain at zero (0) until *all* named pipes return an **EOF**. While COPY awaits an EOF from multiple pipes, it may seem to be hung. Before canceling the COPY statement, however, check your [system CPU and disk accesses](#) to see if any activity is in progress.

In a typical load, PARSE_COMPLETE_PERCENT can either increase slowly to 100%, or jump to 100% quickly if you are loading from named pipes or STDIN, while SORT_COMPLETE_PERCENT is at 0.

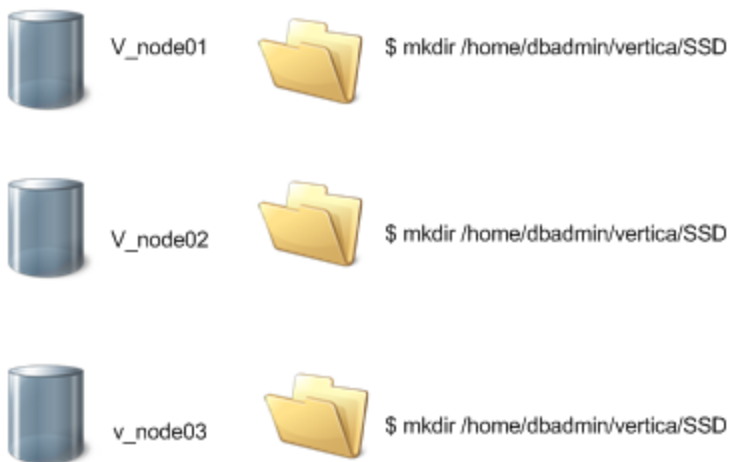
Once `PARSE_COMPLETE_PERCENT` reaches 100%, `SORT_COMPLETE_PERCENT` increases to 100%. Depending on the data sizes, a significant lag can occur between the time `PARSE_COMPLETE_PERCENT` reaches 100% and the time `SORT_COMPLETE_PERCENT` begins to increase.

Working With Storage Locations

HP Vertica *storage locations* are the specific paths you designate as places to store data and temp files. Every node in the cluster requires at least one area in which to store data, and another separate area in which to store database catalog files. These two storage locations are the required defaults that must exist on each cluster node. You set up these locations as part of installation and setup. (See [Prepare Disk Storage Locations](#) in the Installation Guide for disk space requirements.)

A storage location consists of an existing path on one or more nodes. HP Vertica recommends creating the same storage locations on each cluster node, rather than on a single node. Once the directories exist, you create a storage location using [ADD_LOCATION\(\)](#).

This example shows a three-node cluster, each with a `vertica/SSD` directory for storage. Calling the `add_location()` function with that path, and an empty string as the `nodes` value (`' '`), creates the storage location on each node:



```
Vmart=>select add_location('/home/dbadmin/vertica/ssd','',  
'DATA');
```

Other storage locations can be on the HP Vertica cluster nodes, or available on local SANs or other storage systems as you determine your site's requirements and needs for more storage. You add storage locations as shown, creating a directory path on every node, and adding the path as a designated storage location, preferably on every cluster node.

How HP Vertica Uses Storage Locations

Every time you add data to the database, or perform a DML operation, the new data is held in memory (WOS) and moved to storage locations on disk (ROS) at regular intervals. If the object to store has no associated storage policy, HP Vertica uses available storage locations and stores data using its default storage algorithms.

If the object to store has a storage policy, HP Vertica stores the object's data at the default labeled location. See [Creating Storage Policies](#).

If storage locations are no longer required at your site, you can retire or drop them, as described in [Retiring Storage Locations](#) and [Dropping Storage Locations](#).

Viewing Storage Locations and Policies

You can monitor information about available storage, location labels, and your site's current storage policies.

Viewing Disk Storage Information

Query the [V_MONITOR.DISK_STORAGE](#) system table for disk storage information on each database node. For more information, see [Monitoring Using System Tables](#) and [Altering Storage Location Use](#).

Note: The [V_MONITOR.DISK_STORAGE](#) system table includes a CATALOG annotation, indicating that the location is used to store catalog files. You cannot add or remove a catalog storage location. HP Vertica creates and manages this storage location internally, and the area exists in the same location on each cluster node.

Viewing Location Labels

Three system tables have information about storage location labels in their `location_labels` columns:

- `storage_containers`
- `storage_locations`
- `partitions`

Use a query such as the following for relevant columns of the `storage_containers` system table:

```
VMART=> select node_name,projection_name, location_label from v_monitor.storage_containers;
 node_name      | projection_name | location_label
-----+-----+-----
 v_vmart_node0001 | states_p_node0001 |
 v_vmart_node0001 | states_p_node0001 |
 v_vmart_node0001 | t1_b1           |
 v_vmart_node0001 | newstates_b0    | FAST3
 v_vmart_node0001 | newstates_b0    | FAST3
 v_vmart_node0001 | newstates_b1    | FAST3
 v_vmart_node0001 | newstates_b1    | FAST3
 v_vmart_node0001 | newstates_b1    | FAST3
 .
 .
 .
```


Use a query such as the following for columns of the `v_catalog.storage_locations` system table:

```
VMart=> select node_name, location_path, location_usage, location_label from storage_locations;
node_name | location_path | location_usage | location_label
-----+-----+-----+-----
v_vmart_node0001 | /home/dbadmin/VMart/v_vmart_node0001_data | DATA,TEMP | 
v_vmart_node0001 | /home/dbadmin/SSD/schemas | DATA | 
v_vmart_node0001 | /home/dbadmin/SSD/tables | DATA | SSD
v_vmart_node0001 | /home/dbadmin/SSD/schemas | DATA | Schema
v_vmart_node0002 | /home/dbadmin/VMart/v_vmart_node0002_data | DATA,TEMP | 
v_vmart_node0002 | /home/dbadmin/SSD/tables | DATA | 
v_vmart_node0002 | /home/dbadmin/SSD/schemas | DATA | 
v_vmart_node0003 | /home/dbadmin/VMart/v_vmart_node0003_data | DATA,TEMP | 
v_vmart_node0003 | /home/dbadmin/SSD/tables | DATA | 
v_vmart_node0003 | /home/dbadmin/SSD/schemas | DATA | 
(10 rows)
```

Use a query such as the following for columns of the `v_monitor.partitions` system table:

```
VMART=> select partition_key, projection_name, location_label from v_monitor.partitions;
partition_key | projection_name | location_label
-----+-----+-----
NH | states_b0 | FAST3
MA | states_b0 | FAST3
VT | states_b1 | FAST3
ME | states_b1 | FAST3
CT | states_b1 | FAST3
.
.
.
```

Viewing Storage Tiers

Query the `storage_tiers` system table to see the labeled and unlabeled storage containers and information about both:

```
VMart=> select * from v_monitor.storage_tiers;
location_label | node_count | location_count | ros_container_count | total_occupied_size
-----+-----+-----+-----+-----
- | | | | 
SSD | 1 | 2 | 17 | 297039391
Schema | 1 | 1 | 9 | 1506
(3 rows)
```

Viewing Storage Policies

Query the `storage_policies` system table to view the current storage policy in place.

```
VMART=> select * from v_monitor.storage_policies;
 schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
 public      | public      | Schema         | F4
 public      | lineorder   | Partition [4, 4] | M3
(2 rows)
```

Adding Storage Locations

Configuring new storage locations provides additional space, and lets you control what type of data to store at a location. You can add a new storage location from one node to another node, or from a single node to all cluster nodes. Do not use a shared directory on one node for other cluster nodes to access.

You can add and configure storage locations (other than the required defaults) to provide additional storage for these purposes:

- Isolating execution engine temporary files from data files.
- Creating **labeled locations** to use in storage policies.
- Creating a storage locations based on predicted or measured access patterns.
- Creating USER storage locations for specific users or user groups.

Planning Storage Locations

Adding a storage location requires minimal planning:

- Verify that the directory you plan to use for a storage location destination is an empty directory with write permissions for the HP Vertica process.
- Plan the labels to use if you want to label the location as you create it.
- Determine the type of information to store in the storage location:
 - DATA — Persistent data and temp table data
 - TEMP — Temporary files that are generated and dumped to disk such as those generated by sort, group by, join, and so on
 - DATA,TEMP — Both data and temp files (the default)
 - USER — Gives access to non-dbadmin users so they can use the storage location after being granted read or write privileges. You cannot assign this location type for use in a storage policy.

Tip: Storing temp and data files in different storage locations is advantageous because the

two types of data have different disk I/O access patterns. Temp data is distributed across locations based on available storage space, while data can be stored on different storage locations based on predicted or measured access patterns.

Adding the Location

Make a directory at the path to use for storage. For example:

```
$ mkdir /home/dbadmin/storage/SSD
```

HP Vertica recommends that you create the same directory path on each cluster node. This is the path to use when creating a storage location.

Use the `ADD_LOCATION()` function add a storage location.

1. Specify the new data directory path to the host, the node where the location is available (optional), and the type of information to be stored. If you specify the node as an empty string (''), the function creates the storage locations on all cluster nodes in a single transaction.

Note: For user access (non-dbadmin users), you must create the storage location with the USER usage type. You cannot change an existing storage location to have USER access. Once a USER storage location exists, you can grant one or more users access to the area. User areas can store only data files, not temp files. You cannot assign a USER storage location to a storage policy.

The following example adds a location available on all nodes to store only data:

```
SELECT ADD_LOCATION ('/secondVerticaStorageLocation/' , '' , 'DATA');
```

The following example adds a location that is available on only the **initiator node** to store data and temporary files:

```
SELECT ADD_LOCATION ('/secondVerticaStorageLocation/');
```

2. If you are using a storage location for data files and want to create ranked storage locations, where columns are stored on different disks based on their measured performance, you should first:
 - a. [Measure the performance of the storage location.](#)
 - b. [Set the performance of the storage location.](#)

Note: Once a storage location exists, you can alter the type of information it stores, with some

restrictions. See [Altering Storage Location Use](#).

Storage Location Subdirectories

You cannot create a storage location in a subdirectory of an existing location. For example, if you create a storage location at one location, you cannot add a second storage location in a subdirectory of the first:

```
dbt=> select add_location ('/myvertica/Test/KMM','','DATA','SSD');
          add_location
-----
/myvertica/Test/KMM added.
(1 row)
dbt=> select add_location ('/myvertica/Test/KMM/SSD','','DATA','SSD');
ERROR 5615: Location [/myvertica/Test/KMM/SSD] conflicts with existing location [/myvertica/Test/KMM] on node v_node0001
ERROR 5615: Location [/myvertica/Test/KMM/SSD] conflicts with existing location [/myvertica/Test/KMM] on node v_node0002
ERROR 5615: Location [/myvertica/Test/KMM/SSD] conflicts with existing location [/myvertica/Test/KMM] on node v_node0003
```

Adding Labeled Storage Locations

You can add a storage location with a descriptive label. You use labeled locations to set up storage policies for your site. See [Creating Storage Policies](#).

This example creates a storage location on `v_vmart_node0002` with the label `SSD`:

```
VMART=> select add_location ('/home/dbadmin/SSD/schemas','v_vmart_node0002','data','SSD');
          add_location
-----
/home/dbadmin/SSD/schemas added.
(1 row)
```

This example adds a new `DATA` storage location with a label, `SSD`. The label identifies the location when you create storage policies. Specifying the `node` parameter as an empty string adds the storage location to all cluster nodes in a single transaction:

```
VMART=> select add_location ('home/dbadmin/SSD/schemas', '', 'DATA', 'SSD');
          add_location
-----
home/dbadmin/SSD/schemas added.
(1 row)
```

The new storage location is listed in the `v_monitor.disk_storage` system table:

```
VMART=> select * from v_monitor.disk_storage;
.
```

```
.
-[ RECORD 7 ]-----+-----
node_name          | v_vmart_node0002
storage_path       | /home/dbadmin/SSD/schemas
storage_usage      | DATA
rank               | 0
throughput         | 0
latency            | 0
storage_status     | Active
disk_block_size_bytes | 4096
disk_space_used_blocks | 1549437
disk_space_used_mb  | 6053
disk_space_free_blocks | 13380004
disk_space_free_mb  | 52265
disk_space_free_percent | 89%
.
.
.
```

Adding a Storage Location for USER Access

You can create USER storage locations for a non-dbadmin user to access the storage location once granted appropriate privileges.

The following examples create a storage location, BillyBobStore, with a USER usage parameter, on node v_mcdb_node0007:

```
dbadmin=> SELECT ADD_LOCATION('/home/dbadmin/UserStorage/BillyBobStore',
                             'v_mcdb_node0007', 'USER');
          ADD_LOCATION
-----
/home/dbadmin/UserStorage/BillyBobStore' added.
(1 row)
```

Note: A data location can already exist and be in use before you identify it as a storage location with the ADD_LOCATION function. For instance, as a superuser, you can set up an area in which to store and test external tables. When you are ready for user BOB to access the location, you add a storage location at the path you used for testing and grant BOB the required privileges.

The following example grants user BillyBob READ/WRITE permissions to the /BillyBobStore location:

```
dbadmin=> GRANT ALL ON LOCATION '/home/dbadmin/UserStorage/BillyBobStore' TO BillyBob;
GRANT PRIVILEGE
```

For more information about configuring user privileges, see [Managing Users and Privileges](#) in the Administrator's Guide and the [GRANT \(Storage Location\)](#) and [REVOKE \(Storage Location\)](#) functions in the SQL Reference Manual.

Altering Storage Location Use

You can make changes to the type of files that HP Vertica stores at a storage location, with these restrictions:

- Labeled locations can be used only to store DATA files.
- You cannot change labeled storage to TEMP or DATA, or TEMP.
- Storage locations created with the USER option can store only DATA files.

To modify a storage location, use the [ALTER_LOCATION_USE](#) function.

This example alters the storage location on `v_vmartdb_node0004` to store only data files:

```
=> SELECT ALTER_LOCATION_USE ('/thirdVerticaStorageLocation/' , 'v_vmartdb_node0004' , 'DATA');
```

USER Storage Location Restrictions

You cannot change a storage location from a USER usage type if you created the location that way, or to a USER type if you did not. You can change a USER storage location to specify DATA (storing TEMP files is not supported). However, doing so does not affect the primary objective of a USER storage location, to be accessible by non-dbadmin users with assigned privileges.

Effects of Altering Storage Location Use

Before altering a storage location use type, at least one location must remain for storing data and temp files on a node. Data and temp files can be stored in the same, or separate, storage locations.

Altering an existing storage location has the following effects:

| Alter use from: | To store only: | Has this effect: |
|------------------------------------|----------------|---|
| Temp and data files (or Data only) | Temp files | Data content eventually merged out through ATM , per its policies. You can also merge out data from the storage location manually using DO_TM_TASK . The location stores only temp files from that point forward. |
| Temp and data files (or Temp only) | Data files | Continues to run all statements that use temp files (such as queries and loads). Subsequent statements no longer use the changed storage location for temp files, and the location stores only data files from that point forward. |

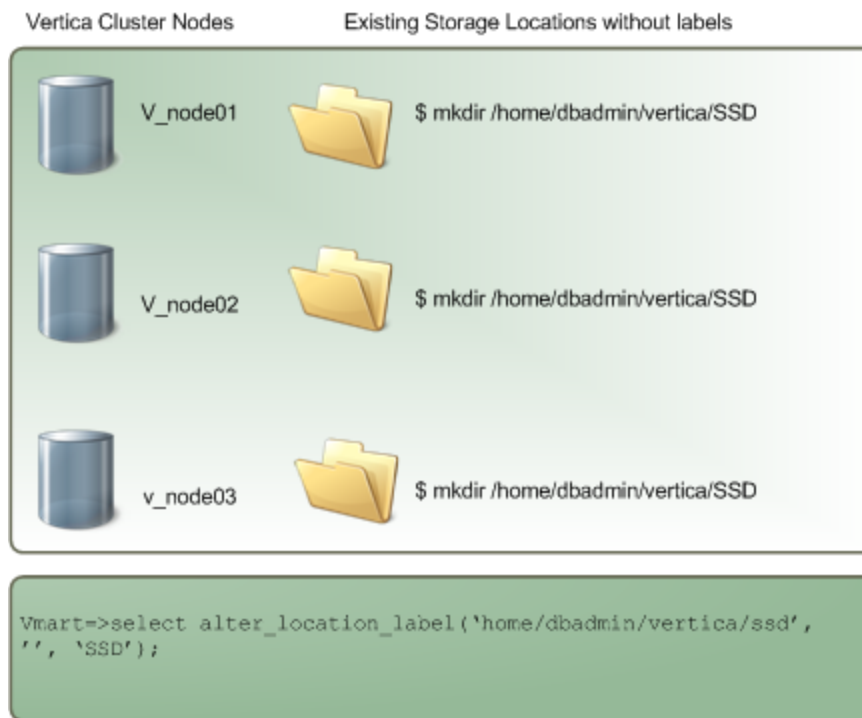
Altering Location Labels

You can add a label to an unlabeled storage location, change an existing label, or remove a label by specifying an empty string for the *location-label* parameter. You can also use this function to perform cluster-wide operations, by specifying an empty string for the function's *node* parameter ('').

Note: If you label an existing storage location that already contains data, and then include the labeled location in one or more storage policies, existing data could be moved. If the **ATM** determines data stored on a labeled location does not comply with a storage policy, the ATM moves the data elsewhere.

Adding a Location Label

To alter a location label, use the [ALTER_LOCATION_LABEL](#) function. The illustration shows how to add a location label, ('SSD'), to the existing storage locations on all cluster nodes (''):



Removing a Location Label

The next example removes the SSD label for the storage location on all nodes by specifying empty strings ('') for both *node* and *location_label* parameters:

```
VMART=> select alter_location_label('/home/dbadmin/SSD/tables','', '');  
alter_location_label
```

```
-----  
/home/dbadmin/SSD/tables label changed.  
(1 row)
```

Note: You cannot remove a location label if the name being removed is used in a storage policy, *and* the location from which you are removing the label is the last available storage for its associated objects.

Effects of Altering a Location Label

Altering a location label has the following effects:

| Alter label: | To: | Has this effect: |
|---------------|-----------|---|
| No name | New label | Lets you use the labeled storage within a storage policy. See note above regarding data storage being moved to other locations if you add a label to a storage location with existing data. |
| Existing name | New name | You can use the new label in a storage policy. If the existing name is used in a storage policy, you cannot change the label. |
| Existing name | No name | You cannot use an unlabeled storage in a storage policy. If the existing name is used in a storage policy, you cannot remove the label. |

Creating Storage Policies

You create a **storage policy** to associate a database object with a labeled storage location using the `SET_OBJECT_STORAGE_POLICY` function. Once a storage policy exists, HP Vertica uses the labeled location as the default storage location for the object data. Storage policies let you determine where to store your critical data. For example, you can create a storage location with the label `SSD` representing the fastest available storage on the cluster nodes, and then create storage policies to associate tables with that labeled location. One storage policy can exist per database object.

Note: You cannot include temporary files in storage policies. Storage policies are for use only with data files, and only on storage locations for `DATA`, not `USER` locations.

You can create a storage policy for any database object (database, schemas, tables, and partition ranges). Each time data is loaded and updated, HP Vertica checks to see whether the object has a storage policy. If it does, HP Vertica automatically uses the labeled storage location. If no storage policy exists for an object, or its parent entities, data storage processing continues using standard storage algorithms on available storage locations. If all storage locations are labeled, HP Vertica uses one of them.

Creating one or more storage policies does not require that policies exist for all database objects. A site can support objects with or without storage policies. You can add storage policies for a discrete set of priority objects, and let other objects exist without a policy, so they use available storage.

Creating Policies Based on Storage Performance

You can measure the performance of any disk storage location (see [Measuring Storage Performance](#)). Then, using the performance measurements, set the storage location performance. HP Vertica uses the performance measurements you set to rank its storage locations and, through ranking, to determine which key projection columns to store on higher performing locations, as described in [Setting Storage Performance](#).

If you have already set the performance of your site's storage locations, and decide to use storage policies, any storage location with an associated policy has a higher priority than the storage ranking setting.

Storage Levels and Priorities

HP Vertica assigns storage levels to database objects. The database is the highest storage level (since nothing exists above the database level), and partition `min_` and `max_` key ranges are considered the lowest level objects. In addition to storage levels, storage priorities exist. The lower the storage level of an object, the higher its storage priority.

Consider this example of database objects, listed in storage level order, with the highest level, `Sales` database, first:

| Object | Storage Level | Storage Policy | Storage Priority | Labeled Location |
|---------------------------------|---------------|----------------|------------------|------------------|
| <code>Sales</code> (database) | Highest | YES | Lower | STANDARD |
| <code>Region</code> (schema) | Medium | NO | Medium | N/A |
| <code>Income</code> (table) | Lower | YES | Higher/highest | FAST |
| <code>Month</code> (partitions) | Lowest | NO | Highest | N/A |

Storage policies exist for the database and table objects, with default storage on the locations `STANDARD` and `FAST`, respectively.

When TM operations occur, such as moveout and mergeout, table data has the highest priority (in this case). The TM moves data from ROS to WOS to the `FAST` labeled location.

Any schema data changes are prioritized after table data. Since the `Region` schema has no storage policy, HP Vertica searches up the storage levels for a policy. In this case, that is the `Sales` database itself. If a database storage policy is in effect, `Region` schema data is moved from ROS to WOS to the `STANDARD` storage location, using its parent object's default storage location.

If the `Sales` database object had no storage policy, the TM operations would use existing storage locations and mechanisms.

Using the SET_OBJECT_STORAGE_POLICY Function

To set a storage policy, use the [SET_OBJECT_STORAGE_POLICY](#) function.

This example sets a storage policy for the table states to use the storage labeled SSD as its default location:

```
VMART=> select set_object_storage_policy ('states', 'SSD');
         set_object_storage_policy
-----
Default storage policy set.
(1 row)
```

You can query existing storage policies, listed in the `location_label` column of the `v_monitor.storage_containers` system table:

```
VMART=> select node_name, projection_name, storage_type, location_label from v_monitor.storage_containers;
 node_name | projection_name | storage_type | location_label
-----+-----+-----+-----
v_vmart_node0001 | states_p_node0001 | ROS | 
v_vmart_node0001 | states_p_node0001 | ROS | 
v_vmart_node0001 | t1_b1 | ROS | 
v_vmart_node0001 | newstates_b0 | ROS | LEVEL3
v_vmart_node0001 | newstates_b0 | ROS | LEVEL3
v_vmart_node0001 | newstates_b1 | ROS | LEVEL3
v_vmart_node0001 | newstates_b1 | ROS | LEVEL3
v_vmart_node0001 | newstates_b1 | ROS | LEVEL3
v_vmart_node0001 | states_p_v1_node0001 | ROS | LEVEL3
v_vmart_node0001 | states_p_v1_node0001 | ROS | LEVEL3
v_vmart_node0001 | states_p_v1_node0001 | ROS | LEVEL3
v_vmart_node0001 | states_p_v1_node0001 | ROS | LEVEL3
v_vmart_node0001 | states_p_v1_node0001 | ROS | LEVEL3
v_vmart_node0001 | states_p_v1_node0001 | ROS | LEVEL3
v_vmart_node0001 | states_b0 | ROS | SSD
v_vmart_node0001 | states_b0 | ROS | SSD
v_vmart_node0001 | states_b1 | ROS | SSD
v_vmart_node0001 | states_b1 | ROS | SSD
v_vmart_node0001 | states_b1 | ROS | SSD
.
```

Effects of Creating Storage Policies

Creating storage policies has the following effects:

| Create policy for... | Storage effect: |
|--------------------------------------|--|
| Database | The highest storage level and the lowest storage priority. This is the default policy when no lower-level or higher priority policies exist. At storage time, HP Vertica uses the database policy for all objects without storage policies. |
| Schema | The mid-level storage, also with a medium priority, compared to lower storage level objects. If a table's schema has no policy, the TM searches the next higher level, the database, using that policy, if it exists. If it does not, the TM uses existing storage mechanisms. |
| Table | A lower storage level than a schema, with the highest storage priority, if no policy exists for the table's partition key ranges. If a table has no storage policy, HP Vertica checks the next higher storage level (the schema) for a policy and uses that. If the schema has no policy, it checks the next higher level, the database, and uses that. If no database policy exists, the TM uses existing storage mechanisms. |
| Partition min_key and max_key ranges | The lowest level policy that can be in effect. During storage processing, partition key ranges with a policy have the highest priority. If no policy exists, the parent table is checked, and so on as described for the other database objects. |

Moving Data Storage Locations

You can use the `SET_OBJECT_STORAGE_POLICY` function to move data storage from an existing location (labeled or not) to another labeled location. Using this function accomplishes two tasks:

1. Creates a new storage policy for the object.

-or-

Updates an existing policy to the target labeled location.

2. Moves all existing data for the specified object(s) to the target storage location.

Moving existing data occurs as part of the next TM moveout activity. Alternatively, you can enforce the data move to occur in the current transaction using the function's `force_storage_move` parameter.

Before actually moving the object to the target storage location, HP Vertica calculates the required storage and checks available space at the target. If there is insufficient free space, the function generates an error and stops execution. The function does not attempt to find sufficient storage at another location.

Note: Checking available space on the new target location before starting to move data cannot guarantee space will continue to exist during a move execution. However, checking target

space does prevent attempts to move any data if insufficient space is available.

Moving Data Storage While Setting a Storage Policy

You can use the `SET_OBJECT_STORAGE_POLICY` function to update an existing storage policy, or create a new policy, and move object data to a new or different labeled storage location. The following example uses the function to set a storage policy for the table object `states`, and to move the table's existing stored data to the labeled location, `SSD`. You force the move to occur during the function transaction by specifying the last parameter as `true`:

```
VMart=> select set_object_storage_policy('states', 'SSD', 'true');
           set_object_storage_policy
```

```
-----
-----
Object storage policy set.
Task: moving storages
(Table: public.states) (Projection: public.states_p1)
(Table: public.states) (Projection: public.states_p2)
(Table: public.states) (Projection: public.states_p3)
(1 row)
```

Note: Moving an object's current storage to a new target is a cluster-wide operation, so a failure on any node results in a warning message. The function then attempts to continue executing on other cluster nodes.

You can view the storage policies that are in effect:

```
VMart=> select * from storage_policies;
 schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
 public      | states      | Table          | SSD
(1 row)
```

Effects of Moving a Storage Location

Moving an object from one labeled storage location to another has the following effects:

| Object type: | Effect: |
|--|--|
| Schema or table | <p>If data storage exists, moves data from source to target destination. Source data can reside on a labeled or unlabeled storage location, but will be moved to specified labeled location.</p> <p>Cluster nodes unavailable when existing data is copied are updated by the TM when they rejoin the cluster (unless you enforce data moving as part of the function transaction, specifying last parameter as <code>true</code>).</p> <p>If a storage policy was in effect, the default storage location changes from the source to target location for all future TM operations, such as moveout and mergeout activities.</p> |
| Table with specified partition min-keys and max_keys | Sets a policy or moves existing data only for the <code>key_min</code> and <code>key_max</code> ranges. Separate partition key ranges can have different storage policies from other ranges, or from the parent table. |

Clearing Storage Policies

You can clear a storage policy by object name after you have defined storage policies. To see existing policies, query the `storage_policies` system table, described in [Viewing Storage Locations and Policies](#).

To clear a storage policy, use the `CLEAR_OBJECT_STORAGE_POLICY` function, specifying the object name associated with the labeled location:

```
release=> select clear_object_storage_policy('lineorder');      clear_object_storage_poli
cy
-----
Default storage policy cleared.
(1 row)
```

Effects on Same-Name Storage Policies

The effects of clearing a storage policy depend on which policy you clear.

For example, consider the following storage. The table `lineorder` has a storage policy for default storage to the location label `F2`, and the table's partition ranges, also `lineorder` objects, have storage policies for other default storage locations:

```
release=> select * from v_monitor.storage_policies;
schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
public      | public      | Schema         | F4
public      | lineorder   | Table          | F2
public      | lineorder   | Partition [0, 0] | F1
public      | lineorder   | Partition [1, 1] | F2
public      | lineorder   | Partition [2, 2] | F4
```

```
public          | lineorder      | Partition [3, 3] | M1
public          | lineorder      | Partition [4, 4] | M3
(7 rows)
```

For this example, clearing the storage policy for an objected named `lineorder`, removes the policy for the table, while retaining storage policies for its partitions, which have their own policies.

The function determines which `lineorder` object policy to clear because no partition range values are specified in the function call:

```
release=> select clear_object_storage_policy('lineorder');
           clear_object_storage_policy
-----
Default storage policy cleared.
(1 row)
release=> select * from v_monitor.storage_policies;
 schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
public       | public      | Schema         | F4
public       | lineorder   | Partition [0, 0] | F1
public       | lineorder   | Partition [1, 1] | F2
public       | lineorder   | Partition [2, 2] | F4
public       | lineorder   | Partition [3, 3] | M1
public       | lineorder   | Partition [4, 4] | M3
(6 rows)
```

Further, using a partition key range with the `lineorder` object name clears the storage policy for only the specified partition range(s). The storage policy for the parent table objects, and other partition ranges persist:

```
release=> select clear_object_storage_policy ('lineorder','0','3');
           clear_object_storage_policy
-----
Default storage policy cleared.
(1 row)
release=> select * from storage_policies;
 schema_name | object_name | policy_details | location_label
-----+-----+-----+-----
public       | public      | Schema         | F4
public       | lineorder   | Table          | F2
public       | lineorder   | Partition [4, 4] | M3
(2 rows)
```

Measuring Storage Performance

HP Vertica lets you measure disk I/O performance on any storage location at your site. You can use the returned measurements to set performance so that it has a rank. Depending on your storage needs, you can also use performance to determine the storage locations to use for critical data as part of your site's storage policies. Storage performance measurements are applicable only to DATA storage locations, not temporary storage locations.

Measuring storage location performance calculates the time it takes to read and write 1MB of data from the disk, which equates to:

```
IO time = time to read/write 1MB + time to seek = 1/throughput + 1/Latency
```

- Throughput is the average throughput of sequential reads/writes (units in MB per second)
- Latency is for random reads only in seeks (units in seeks per second)

Thus, the I/O time of a faster storage location is less than a slower storage location.

Note: Measuring storage location performance requires extensive disk I/O, which is a resource-intensive operation. Consider starting this operation when fewer other operations are running.

HP Vertica has two ways to measure storage location performance, depending on whether the database is running. Both methods return the throughput and latency for the storage location. Record or capture the throughput and latency information so you can use it to set the location performance (see [Setting Storage Performance](#)).

Measuring Performance on a Running HP Vertica Database

Use the [MEASURE_LOCATION_PERFORMANCE\(\)](#) function to measure performance for a storage location when the database is running. This function has the following requirements:

- The storage path must already exist in the database.
- You need RAM*2 free space available in a storage location to measure its performance. For example, if you have 16GB RAM, you need 32GB of available disk space. If you do not have enough disk space, the function errors out.

Use the system table [DISK_STORAGE](#) to obtain information about disk storage on each database node.

The following example measures the performance of a storage location on v_vmartdb_node0004:

```
=> SELECT MEASURE_LOCATION_PERFORMANCE('/secondVerticaStorageLocation/', 'v_vmartdb_node0004');
WARNING:  measure_location_performance can take a long time. Please check logs for progress
          measure_location_performance
-----
Throughput : 122 MB/sec. Latency : 140 seeks/sec
```

Measuring Performance Before a Cluster Is Set Up

You can measure disk performance before setting up a cluster. This is useful for verifying that the disk is functioning within normal parameters. This method requires only that HP Vertica be installed.

To measure disk performance, use the following `vsql` command:

```
opt/vertica/bin/vertica -m <path to disk mount>
```

For example:

```
opt/vertica/bin/vertica -m /secondVerticaStorageLocation/node0004_data
```

Setting Storage Performance

You can use the measurements returned from the [MEASURE_LOCATION_PERFORMANCE](#) function as input values to the [SET_LOCATION_PERFORMANCE\(\)](#) function.

Note: You must set the throughput and latency parameters of this function to 1 or more.

The following example sets the performance of a storage location on `v_vmartdb_node0004` to a throughput of 122 MB/second and a latency of 140 seeks/second. These were the values returned for this location from the [MEASURE_LOCATION_PERFORMANCE](#) function.

```
=> SELECT SET_LOCATION_PERFORMANCE(v_vmartdb_node0004', '/secondVerticaStorageLocation/', '122', '140');
```

How HP Vertica Uses Location Performance Settings

Once set, HP Vertica automatically uses performance data to rank storage locations whenever it stores projection columns.

HP Vertica stores columns included in the projection sort order on the fastest storage locations. Columns not included in the projection sort order are stored on slower disks. Columns for each projection are ranked as follows:

- Columns in the sort order are given the highest priority (numbers > 1000).
- The last column in the sort order is given the rank number 1001.
- The next-to-last column in the sort order is given the rank number 1002, and so on until the first column in the sort order is given 1000 + # of sort columns.
- The remaining columns are given numbers from 1000–1, starting with 1000 and decrementing by one per column.

HP Vertica then stores columns on disk from the highest ranking to the lowest ranking, with the highest ranking columns placed on the fastest disks, and the lowest ranking columns placed on the slowest disks.

Using Location Performance Settings With Storage Policies

After measuring location performance, and setting it in the HP Vertica database, you can also use the performance results to determine the fastest storage to use in your storage policies. The locations with the highest performance can be set as the default locations for critical data. Slower locations can become default locations for older, or less-important data, or may not require policies at all if you do not want to specify default locations.

Data is stored as follows, depending on whether a storage policy exists:

| Storage Policy | Label | # Locations | HP Vertica Action |
|----------------|-------|-------------|---|
| No | No | Multiple | Uses ranking (as described), choosing a location from all locations that exist. |
| Yes | Yes | Single | Uses that storage location exclusively. |
| Yes | Yes | Multiple | Ranks storage (as described) among all same-name labeled locations. |

Dropping Storage Locations

To drop a storage location, use the [DROP_LOCATION\(\)](#) function. The following example drops a storage location on `v_vmartdb_node0002` that was used to store temp files:

```
=> SELECT DROP_LOCATION('/secondVerticaStorageLocation/' , 'v_vmartdb_node0002');
```

Dropping a storage location is a permanent operation and cannot be undone. When you drop a storage location, the operation cascades to associated objects including any granted privileges to the storage. If the storage was being used for external table access, subsequent queries on the external table will fail with a COPY COMMAND FAILED message.

Since dropping a storage location cannot be undone, HP recommends that you consider first retiring a storage location (see [Retiring Storage Locations](#)). Retiring a storage location before dropping it lets you verify that there will be no adverse effects on any data access. Additionally, you can restore a retired storage location (see [Restoring Retired Storage Locations](#)).

Altering Storage Locations Before Dropping Them

You can drop only storage locations containing temp files. If you alter a storage location to the TEMP usage type so that you can drop it, and data files still exist on the storage, HP Vertica

prevents you from dropping the storage location. Deleting data files does not clear the storage location, and can result in database corruption. To handle a storage area containing data files so that you can drop it, use one of these options:

- Manually merge out the data files
- Wait for the **ATM** to mergeout the data files automatically
- Manually [drop partitions](#)

Dropping USER Storage Locations

Storage locations that you create with the USER usage type can contain only data files, not temp files. However, unlike a storage location not designated for USER access, you can drop a USER location, regardless of any remaining data files.

Retiring Storage Locations

To retire a storage location, use the [RETIRE_LOCATION\(\)](#) function.

The following example retires a storage location on v_vmartdb_node0004:

```
=> SELECT RETIRE_LOCATION('/secondHP VerticaStorageLocation/' , 'v_vmartdb_node0004');
```

Retiring a location prevents HP Vertica from storing data or temp files to it, but does not remove the actual location. Any data previously stored on the retired location is eventually merged out by the **Automatic Tuple Mover (ATM)** per its policies.

Note: You cannot retire a location if it is used in a storage policy, *and* is the last available storage for its associated objects.

Data and temp files can be stored in one, or separate, storage locations. If the location you are retiring was used to store temp files only, you can remove it. See [Dropping Storage Locations](#).

Restoring Retired Storage Locations

You can restore a previously retired storage location that continues to be used in queries. Once restored, HP Vertica re-ranks the storage location and uses the restored location to process queries as determined by its rank.

Use the [RESTORE_LOCATION\(\)](#) function to restore a retired storage location.

The following example restores a retired storage location on v_vmartdb_node0004:

```
=> SELECT RESTORE_LOCATION('/secondHP VerticaStorageLocation/' , 'v_vmartdb_node0004');
```

Backing Up and Restoring the Database

Creating regular database backups is an important part of basic maintenance tasks at your site. HP Vertica supplies a comprehensive utility, the `vbr.py` Python script. The utility lets you back up, restore, list backups, and copy your database to another cluster. You can create full and incremental database backups, as well as backups of schemas or tables (object-level backups) for use with a multitenant database. Once a full backup exists, you can restore one or more objects from the backup, or the entire database as required.

Using `vbr.py`, you can save your data to a variety of locations:

- A local directory on the nodes in the cluster
- One or more hosts outside of the cluster
- A different HP Vertica cluster (effectively cloning your database)

Note: Saving a database backup on a different cluster does not provide Disaster Recovery. The cloned database you create with `vbr.py` is entirely separate from the original, and is not kept in sync with the database from which it is cloned.

Compatibility Requirements for Using `vbr.py`

Creating backups with `vbr.py` requires restoring backups with the same utility. The `vbr.py` script supports creating and restoring backups between 6.x versions. Object-level backups were not supported before HP Vertica 6.x.

HP Vertica does not support restoring a backup created in 5.x after you have upgraded to 6.x.

The backup and restore scripts used in 5.x and earlier, `backup.sh` and `restore.sh`, are obsolete. The `vbr.py` utility is incompatible with these scripts. If you created backups using the obsolete `backup.sh` script, you must restore them with `restore.sh`.

Automating Regular Backups

The `vbr.py` utility helps to automate backing up your database, because you can configure `vbr` with the required runtime parameters in a script. The ability to configure runtime parameters facilitates adding the utility to a cron or other task scheduler to fully automate regular database backups.

Types of Backups

The `vbr.py` utility supports several kinds of backups:

- Full backups
- Object-level backups
- Hard link local backups

You can refer to full or object-level backups by a user-defined descriptive name, such as `FullDBSnap`, `Schema1Bak`, `Table1Bak`, and so on.

Note: This section uses *backup* as a noun (create a full backup), and *back up* as a multi-word verb (be sure to back up the database).

Full Backups

A full backup is a complete copy of the database catalog, its schemas, tables, and other objects. It is a consistent image, or snapshot, of the database at the time the backup occurred. You can use a full backup for disaster recovery to restore a damaged or incomplete database. You can also restore objects from a full backup .

Once a full backup exists, `vbr.py` creates incremental backups as successive snapshots following the full-backup, consisting of new or changed data since the last full, or incremental, backup occurred.

Archives consist of a collection of same-name backups. Each archive can have a different retention policy. For example, if `TBak` is the name of an object-level backup of table `T`, and you create a daily backup each week, the seven backups are part of the `TBak` archive. Keeping a backup archive lets you revert back to any one of the saved backups.

Full and object-level backups reside on *backup hosts*, the computer system(s) on which backups and archives are stored. For more information about backup hosts, see [Configuring Backup Hosts](#).

Backups are saved in a specific *backup location*, the directory on a backup host. This location can comprise multiple backups, including associated archives. All backups in the same backup location share data files (through hard links). The backups are also compatible, meaning that you can restore any object-level backup from the same location, after restoring a full database backup.

Object-Level Backups

An object-level backup consists of one or more schemas or tables, or a group of such objects. The conglomerate parts of the object-level backup do not contain the entire database. Once an object-level backup exists, you can restore all of its contents, but not individually. You can restore one or more objects from a full backup.

Object level backups consist of three object types:

| | |
|-------------------|--|
| Selective objects | Objects you choose to be part of an object-level backup. For example, if you specify tables <code>T1</code> and <code>T2</code> to include in an object-level backup, they are the selected objects. |
|-------------------|--|

| | |
|-------------------|--|
| Dependent objects | Objects that must be included as part of an object-level backup due to dependencies. For example, to create an object-level backup that includes a table with a foreign key, <code>vbr .py</code> automatically includes the primary key table due to table constraints. Projections anchored on a table in the selected objects are also <i>dependent objects</i> . |
| Principal objects | The objects on which both selected and dependent objects depend are called <i>principal objects</i> . For example, each table and projection has an owner, and each is a <i>principal object</i> . |

In earlier Vertica versions, object-level backups could not exist because a backup always contained the entire database.

Hard Link Local Backups

A hard link local backup is a full or object-level backup consisting of a complete copy of the database catalog, and a set of hard file links to corresponding data files. A hard link local backup must be saved on the file system used by the catalog and database files.

When to Back up the Database

In addition to any guidelines established by your corporation, HP Vertica recommends that you back up your database:

- Before you upgrade HP Vertica to another release.
- Before you drop a partition.
- After you load a large volume of data.
- If the epoch in the latest backup is earlier than the current ancient history mark (**AHM**).
- Before and after you add, remove, or replace nodes in your database cluster.
- After recovering a cluster from a crash.

Note: When you restore a full database backup, you must restore to a cluster that is identical to the one on which you created the backup. For this reason, always create a new full backup after adding, removing, or replacing nodes.

Ideally, create regular backups to back up your data. You can run the HP Vertica `vbr .py` from a cron job or other task scheduler.

Configuring Backup Hosts

The `vbr .py` utility lets you back up your database to one or more hosts (called backup hosts), that can be outside of your database cluster. Using backup hosts external to your database cluster facilitates offsite data backups.

Note: You can use one or more backup hosts to back up your database. Use the `vbr.py` configuration file to specify which backup host each node in your cluster should use. See [Backup Configuration Options](#) for details.

Before you back up to hosts outside of the local cluster, configure the target backup locations to work with the `vbr.py` utility. The backup hosts you use must:

- Have sufficient backup disk space.
- Be accessible from your database cluster.
- Have passwordless SSH access for the **database administrator** account.
- Test SSH between cluster nodes and backup node(s).
- Have a copy of the same versions of Python and `rsync` that were installed by the HP Vertica installer.

Note: The version of `rsync` included in the HP Vertica installer supports a combined maximum 500 full- and object-level backups at one backup location. All backup hosts must use the `rsync` version supplied with HP Vertica 6.0.

Configuring Single-Node Database Hosts for Backup

Installing HP Vertica on a single-node database host automatically sets up the node with passwordless SSH access, as the installation does for cluster nodes. Instructions for performing this step manually are available in the Installation Guide section, [Enable Secure Shell \(SSH\) Logins](#).

The `vbr.py` utility requires that all database hosts (including single-node hosts) and backup location hosts have passwordless SSH access, and fulfill other necessary requirements, described in this section.

Creating Configuration Files for Backup Hosts

Create separate configuration files for full- or object-level backups, using distinct names for each configuration file. Also, use the same node, backup host, and directory location pairs. The backup directory location should be used for the backups from only one database.

Note: If possible, for optimal network performance when creating a backup, HP Vertica recommends having each node in the cluster use a dedicated backup host.

Estimating Backup Host Disk Requirements

Wherever you plan to save data backups, consider the disk requirements for incremental backups at your site. Also, if you use more than one archive, it potentially requires more disk space.

Regardless of the specifics of your site's backup schedule and retention requirements, HP Vertica recommends that each backup host has space for at least twice the database footprint size.

To estimate the database size from the `used_bytes` column of the `storage_containers` system table:

```
VMart=> select sum(used_bytes) as total_size from v_monitor.storage_containers;
total_size
-----
      302135743
(1 row)
```

If your site uses multiple backup host locations, you can estimate the database size requirements per node with a query such as the following, substituting a backup host name for `node_name`:

```
select node_name,sum(used_bytes) as size_in_bytes from v_monitor.storage_containers group
by node_name;
```

Estimating Log File Disk Requirements

When you run the `vbr.py --setupconfig` command to create the configuration file and configure advanced parameters (see [Configuring Advanced VBR Parameters](#)), one of the parameters is `tempDir`. This parameter specifies the backup host location where `vbr.py` writes its log files, and some other temp files (of negligible size). The default location is the `/tmp/vbr` directory on each backup host. You can change the default location by specifying a different path in the configuration file.

The temporary storage directory also contains local log files describing the progress, throughput, and any errors encountered for each node. Each time you run `vbr.py`, the script creates a separate log file, so the directory size increases over time, depending on how frequently you run the utility. The utility stores older log files in separate subdirectories, each named with a timestamp. When the `vbr.py` utility is executing, it updates the current log file each second with one line of information, so the log file size is proportional to the actual backup transfer size. As a rough estimate, a one-hour backup will populate the log file with 3600 lines, or approximately 100KB. HP Vertica recommends allotting 1GB disk space on each backup host for the `vbr.py` log files.

The `vbr.py` log files are not removed automatically, so delete older log files manually as necessary.

Making Backup Hosts Accessible

To make the backup hosts accessible to your source database cluster, add each of the backup hosts to the hosts file of all your database nodes, unless you plan on using IP addresses.

Any firewalls between the source database nodes and the target backup hosts must allow connections for SSH and rsync.

The backup hosts must be running the same Linux distribution and have the same processor architecture as your database nodes, because each must have identical versions of rsync and Python as those supplied in the HP Vertica installation package.

Setting Up Passwordless SSH Access

To access a backup host, the database superuser must meet two requirements to run the `vbr.py` utility:

- Have an account on each backup host, with write permissions to the backup directory
- Have passwordless SSH access from each cluster host to the corresponding backup host

How you fulfill these requirements depends on your platform and infrastructure.

If your site does not use a centralized login system (such as LDAP), you can usually add a user with the `useradd` command, or through a GUI administration tool. See the documentation for your Linux distribution for details.

If your platform supports it, you can enable passwordless SSH logins using the `ssh-copy-id` command to copy a database superuser's SSH identity file to the backup location from one of your database nodes. For example, to copy the SSH identity file from a node to a backup host named `backup01`:

```
> ssh-copy-id -i dbadmin@backup01|  
Password:
```

Try logging into the machine with "`ssh dbadmin@backup01`", and check the contents of the `~/.ssh/authorized_keysfile` to make sure you have not added extra keys that you were not expecting.

```
> ssh backup01  
Last login: Mon May 23 11:44:23 2011 from host01
```

Repeat the steps to copy a superuser's SSH identity to all backup hosts you will use to back up your database.

After copying a superuser's SSH identity, you should be able to log in to the backup host from any of the nodes in the cluster without being prompted for a password.

Testing SSH Access

A best practice after setting up passwordless SSH is to test the SSH connection on each cluster node, and the backup host.

To test that passwordless SSH is working correctly:

1. Log into each cluster node and SSH to and from all other cluster nodes. Add the key to the RSA store as noted above.
2. If you cannot SSH to and from all nodes without entering a password, SSH is not properly set up.
3. To set up correctly, see [Enable Secure Shell \(SSH\) Logins](#), in the Installation Guide.

Changing the Default SSH Port on Backup Hosts

Internally, the `vbr.py` utility uses the default SSH port 22. If your backup hosts are using SSH on a different port, you can override the default by manually adding the `ssh_port_backup` parameter as follows:

1. Open the backup configuration file (*backup_file.ini*) in your favorite editor.
2. Find the Transmission section.
3. Add the `ssh_port_backup` parameter to the section, specifying with the backup hosts' SSH port number:

```
[Transmission]encrypt = False
checksum = False
port_rsync = 50000
ssh_port_backup = 25
```

4. Save the configuration file.

Note: The `vbr.py` utility supports a non-default SSH port with the backup task, but not with `copycluster`.

Increasing the SSH Maximum Connection Settings for a Backup Host

If your configuration requires backing up multiple nodes to one backup host (n:1), increase the number of concurrent SSH connections to the SSH daemon (`sshd`). By default, the number of concurrent SSH connections on each host is 10, as set in the `sshd_config` file with the `MaxStartups` keyword. The `MaxStartups` value for each backup host should be greater than the total number of hosts being backed up.

To increase the `MaxStartups` value:

1. Log on as root to access the config file.
2. Open the SSH configuration file (`/etc/ssh/sshd_config`) in your favorite editor.
3. Locate the `#MaxStartups` parameter and increase the value.
4. Save the file.
5. Exit from root.

Copying Rsync and Python to the Backup Hosts

All backup hosts must use the version of rsync and Python installed by the HP Vertica installation package.

Copy rsync to the same directory on each of the backup hosts. The rsync file is located here:

```
/opt/vertica/bin/rsync
```

1. Copy the python directory contents recursively to the `/opt/vertica/oss/python` directory on each backup host using a command such as this:

```
> cp -r /opt/vertica/oss/python <backup_host1>/oss/python
```

2. To preserve the file characteristics (including permissions) on the target backup host, add the `-p` option:

```
> cp -r -p /opt/vertica/oss/python <backup_host1>/oss/python
```

Note: If you do not use preserve file characteristics (using `cp -r -p`) when copying the rsync and Python files, make sure that the permissions are set after you copy the files. The permissions must enable the database superuser to run rsync and Python on the backup hosts.

Configuring Hard Link Local Backup Hosts

When specifying the `backupHost` parameter for your hard link local configuration files, use the database host names (or IP addresses) as known to Admintools, rather than the node names. Host names (or IP addresses) are what you used when setting up the cluster. Do not use `localhost` for the `backupHost` parameter.

Listing Host Names

To query node names and host names:

```
VMart=> select node_name, host_name from node_resources;
  node_name | host_name
-----+-----
v_vmart_node0001 | 192.168.223.11
v_vmart_node0002 | 192.168.223.22
v_vmart_node0003 | 192.168.223.33
(3 rows)
```

Since you are creating a local backup, the `host_name` listed under the [Mapping] section in the configuration file, after each node name is the name of the host on which you are creating the local backup. For example, the following Mapping section shows mappings for a 3-node cluster. The

backupHost value you supplied when creating the configuration file is the name of the cluster host (192.168.223.33) you are using as the hard link local backup host for the cluster:

```
[Mapping]
v_vmart_node0001 = 192.168.223.33:/home/dbadmin/data/backups
v_vmart_node0002 = 192.168.223.33:/home/dbadmin/data/backups
v_vmart_node0003 = 192.168.223.33:/home/dbadmin/data/backups
```

Creating vbr.py Configuration Files

The `vbr.py` utility uses a configuration file for the information required to back up and restore a full- or object-level backup, or to copy a cluster. You cannot run `vbr.py` without a configuration file, since no default file exists. You can create a configuration file using the instructions in this section, or copy and edit an existing configuration file, and save the changes to a different file name.

Note: You must be logged on as `dbadmin`, not `root`, to create the `vbr` configuration file.

To create a configuration file, enter this command:

```
> /opt/vertica/bin/vbr.py --setupconfig
```

The script prompts you to answer several questions, as shown in the following summarized example:

```
[dbadmin@localhost ~]$ /opt/vertica/bin/vbr.py --setupconfig
Snapshot name (backup_snapshot): fullbak
Backup vertica configurations? (n) [y/n]: y
Number of restore points (1): 3
Specify objects (no default):
Vertica user name (dbadmin):
Save password to avoid runtime prompt? (n) [y/n]: y
Password to save in vbr config file (no default):
Node v_vmart_node0001
Backup host name (no default): 127.0.0.1
Backup directory (no default): /home/dbadmin/backups
Config file name (fullbak1.ini):
Change advanced settings? (n) [y/n]: n
Saved vbr configuration to fullbak1.ini.
```

The following sections describe the information you supply.

Specifying a Backup Name

Specify the name of the backup `vbr.py` will create:

```
Snapshot name (backup_snapshot):
```

You cannot leave this blank. Choose a name that clearly identifies the backup. The name is used in the directory tree structure that `vbr.py` creates for each node. You do not need to add dates to the backup name, since date and time indicators are added automatically in backup subdirectories.

Create different configuration files with specific backup names for full- and object-level backups, but use the same backup directory for both types of backups. For example, the configuration file for

a full database backup, called `fullbak.ini` would have these `snapshotName` and `backupDir` parameter values:

```
snapshotName=fullbak  
backupDir=/home/dbadmin/data/backups
```

The configuration file for the object-level backup, called `objectbak.ini`, would have these parameter values:

```
snapshotName=objectbak  
backupDir=/home/dbadmin/data/backups
```

Backing Up the Vertica Configuration File

Enter `y` to include a copy of the `vertica.conf` file in the backup. The file is not saved by default:

```
Backup vertica configurations? (n) [y/n]:
```

The `vertica.conf` file contains configuration parameters you have changed. The file is stored in the catalog directory. Press `Enter` to accept the default, or `n` to omit the file explicitly.

Saving Multiple Restore Points

Specify how many backups to save as a number of restore points:

```
Number of restore points (1):
```

The default value is 1, indicating that `vbr.py` always retains one additional restore point. Saving multiple restore points gives you the option to recover from one of several backups. For example, specifying 3 results in eventually having 1 current backup, and 3 backup archives. The value you enter here is stored as the `restorePointLimit` parameter in the `vbr.py` configuration file.

For more information about how `vbr.py` organizes the backup and archives in the backup directory, see [Backup Directory Structure and Contents](#).

Specifying Full or Object-Level Backups

Indicate whether you want to create a full- or object-level backup:

```
Specify objects (no default):
```

Press `Enter` to continue creating a configuration file for a full database backup.

Enter the names of one or more schemas or tables to create an object-level backup configuration file. Separate each name with a comma (`,`). The objects you enter are listed in the `Objects` parameter of the configuration file.

Entering the User Name

Enter the user name of the person who will invoke `vbr.py`:

```
Vertica user name (dbadmin):
```

Press `Enter` to leave this blank and accept the default user name `dbadmin`. Enter another user name if you have assigned `dbadmin` privileges to a different user.

Saving the Account Password

Specify whether `vbr.py` will prompt for an account password at runtime:

```
Save password to avoid runtime prompt? (n) [y/n]:
```

Press `Enter` to accept the default `(n)` and continue. Since you are not saving the password in the configuration file, you must enter it when you run `vbr.py`.

To save your password in the configuration file, enter `y`. The utility prompts for you to enter the password, but does not display the text as you type:

```
Password to save in vbr config file (no default):
```

The parameter name in the configuration file indicating whether to prompt for a password is `dbPromptForPassword`.

Specifying the Backup Host and Directory

The utility lists each node name for your cluster. Enter the backup host name and directory for each cluster node at the prompts:

```
Node v_vmart_node0001  
Backup host name (no default):  
Backup directory (no default):
```

No defaults exist for either the host name or the backup directory.

In the configuration file, all cluster nodes are listed by name under the `[Mapping]` section, each containing a line that includes the cluster node, backup host, and backup directory location:

```
[Mapping]  
v_vmart_node0001 = 127.0.0.1:/home/dbadmin/backups
```

Saving the Configuration File

Enter a configuration file name of your choice, or press (Enter) to save the configuration file with its default name:

```
Config file name (fullbak.ini):
```

The default name consists of the backup name you supplied (fullbak in this example), with an .ini file extension. The vbr.py utility confirms that it saved the file:

```
Saved vbr configuration to fullbak.ini.
```

NOTE: Since the backup configuration file is typically stored on the cluster you are backing up, VERTICA recommends that you also save a copy of the configuration file on the backup host. In this way, the configuration file will be available if the cluster node is lost.

Continuing to Advanced Settings

To continue with advanced configuration settings, enter y to continue:

```
Change advanced settings? (n) [y/n]:
```

See [Configuring Advanced VBR Parameters](#) for more information.

To continue without advanced settings and to save the configuration file with the information you just entered, press Enter.

Sample Configuration File

Following are the contents of the fullbak.ini configuration file created from the entries used in this example. Notice that any account password (mydbpassword) you supplied when running the script is stored as plain text, and is not concealed in any way:

```
[Misc]
snapshotName = fullbak
verticaConfig = True
restorePointLimit = 3

[Database]
dbName = VMart
dbUser = dbadmin
dbPassword = mydbpassword

[Transmission]

[Mapping]
v_vmart_node0001 = 127.0.0.1:/home/dbadmin/backups
```

Changing the Overwrite Parameter Value

You can also specify an `overwrite` parameter in the configuration file once the file exists. The `overwrite` parameter is associated only with restoring object-level backups. For more information, see [Creating Object-Level Backups](#).

Configuring Required VBR Parameters

To invoke the `vbr.py` utility to create a configuration file, enter this command:

```
> /opt/vertica/bin/vbr.py --setupconfig
```

The utility prompts you to answer the questions listed in the following table. Each is followed by its default value, if any exists. Type `Enter` to accept a default value. For more information about the questions and their values, see the [Creating vbr.py Configuration Files](#).

The second table column lists the name of the parameter associated with the question, as it will be stored in the configuration file. See also the [VBR Configuration File Reference](#):

| | |
|---|---------------------|
| Snapshot name: [snapshotName] | snapshotName |
| Backup Vertica configurations? (n)[y/n] | verticaConfig |
| Number of restore points? (1): | restorePointLimit |
| Specify objects (no default): | objects |
| Vertica user name (current user): | dbUser |
| Save password to avoid runtime prompt (n) [y/n] | dbPromptForPassword |
| Password to save in vbr config file (no default): | dbPassword |
| Backup host name (no default): | backupHost |
| Backup directory (no default): | backupDir |
| Config file name (no default): | N/A |
| Change advanced settings? (n) [y/n]: | N/A |

To change any advanced parameters, enter `y` at the last question, `Change advanced settings?`.

After successfully completing all of the required questions, `vbr.py` generates a configuration file with the information you supplied. Create separate configuration files for a full backup, and for each object-level backup. Use distinct backup names in each configuration file.

When the setup completes processing, enter a configuration file name. Use this file name when you run the `--task backup` or other commands. The utility uses the configuration file contents for both backup and restore tasks, as well as for the `--copycluster` task, described in [Copying the Database to Another Cluster](#).

If you do not successfully complete all of the required questions, `vbr.py` lists error messages and hints, but does not create a configuration file. You can then run the `--setupconfig` command again to respecify any missing or incorrect information.

Sample Session Configuring Required Parameters

Following is a test session showing the required configuration file parameters. The utility detects the HP Vertica node names in the cluster, so you do not have to supply them (`v_example_node0001`, `v_example_node0002`, and `v_example_node0003`):

```
> /opt/vertica/bin/vbr.py --setupconfig
Snapshot name (snapshotName): ExampleBackup
Backup Vertica configurations? (n) [y/n] y
Number of restore points? (1): 5
Specify objects (no default): dim, dim2
Vertica user name (current_user): dbadmin
Save password to avoid runtime prompt? (n)[y/n]: y
Password to save in vbr config file (no default): mypw234
Node v_example_node0001
Backup host name (no default): backup01
Backup directory (no default): /home/dbadmin/backups
Node v_example_node0002
Backup host name: backup02
Backup directory: /home/dbadmin/backups
Node v_example_node0003
Backup host name: backup03
Backup directory: /home/dbadmin/backups
Config file name: exampleBackup.ini
Change advanced settings? (n)[y/n]: n
Saved vbr configuration to ExampleBackup.ini.
```

Configuring Advanced VBR Parameters

When you use `vbr.py` to create your configuration file, you can configure advanced parameters after configuring the required ones. To invoke utility initially, enter this command:

```
> /opt/vertica/bin/vbr.py --setupconfig
```

To continue from advanced settings after completing the Required parameter options, enter **y** to the last question:

```
Change advanced settings? (n)[y/n]: y
```

Following is a list of the advanced parameter questions, their default values, and the name of the associated parameter, which will be in the completed configuration file.

Type Enter to accept the default value for any parameter. For more details, see the [VBR Configuration File Reference](#):

| Questions and default value | Configuration parameter |
|--|-------------------------|
| Temp directory (/tmp/vbr): | tempDir |
| Number of times to retry? (2): | retryCount |
| Seconds between retry attempts (1): | retryDelay |
| Encrypt data during transmission? (n) [y/n]: | encrypt |
| Use checksum for data integrity (not file date and size)? (n) [y/n]: | checksum |
| Port number for rsync daemon (50000): | port_rsync |
| Transfer bandwidth limit in KBps or 0 for unlimited (0): | bwlimit |

Note: For additional information about the tempDir configuration parameter, see [Configuring Backup Hosts](#).

Example of Configuring Advanced Parameters

Following is a test session providing some of the advanced configuration file parameters, starting after the required parameters:

```
Change advanced settings? (n)[y/n]: y
Temp directory (/tmp/vbr):
Number of times to retry backup? (2): 5
Seconds between retry attempts? (1): 3
Encrypt data during transmission? (n) [y/n] n
Use checksum for data integrity (not file date and size)? (n)[y/n]: n
Port number for Rsync daemon (50000):
Transfer bandwidth limit in KBPS or 0 for unlimited (0): 0
Saved vbr configuration to exampleBackup.ini.
```

Configuring the Hard Link Local Parameter

Creating hard link local backups requires manually adding the hardLinkLocal=True parameter to the [Transmission] section of the vbr.py configuration file.

Note: To create a local backup without hard file links, omit the hardLinkLocal=True parameter from the configuration file. Specify the backupDir parameter as a location on the same file system as the database catalog and data files. Then, the vbr.py utility creates a backup by copying the files, even when they are located on the same file system.

Generating a configuration file with the vbr.py --setupconfig always includes a [Transmission] section. If you have an existing vbr.py configuration file (backup_name.ini), add the hardLinkLocal parameter to the [Transmission] section.

1. For a configuration file *without* advanced options, add the parameter as the sole entry in the [Transmission] section:

```
[Transmission]
hardLinkLocal = True
```

2. Save the configuration file.

-or-

1. For a configuration file *with* advanced options, add the hardLinkLocal parameter as the last entry in the [Transmission] section:

```
[Transmission]
encrypt = False
checksum = False
port_rsync = 50000
bwlimit = 0
hardLinkLocal = True
```

2. Save the configuration file.

Restrictions for Backup Encryption Option

The encrypt parameter is one of the advanced vbr.py configuration file options. However, you cannot use encryption when creating a hard link local backup. If you hardlinkLocal=true to a configuration file that includes encrypt=true, vbr.py issues a warning, but ignores the encryption parameter.

Example Backup Configuration File

The following example configuration file shows most of the options described in [Configuring Required VBR Parameters](#) and [Configuring Advanced VBR Parameters](#). Each node is backed up to a corresponding non-cluster host (except for hard-link local backups). See [Creating Hard Link Local Backups](#).

```
[Misc]
; Section headings are enclosed by square brackets.
; Comments have leading semicolons.
; Option and values are separated by an equal sign.
snapshotName = exampleBackup
; For simplicity, use the same temp directory location on
; all backup hosts. The utility must be able to write to this
; directory.
tempDir = /tmp/vbr
; Vertica binary directory should be the location of
; vsql & bootstrap. By default it's /opt/vertica/bin
;verticaBinDir =
```

```
; include vertica configuration in the backup
verticaConfig = True
; how many times to retry operations if some error occurs.
retryCount = 5
retryDelay = 1
restorePointLimit = 5

[Database]
; db parameters
dbName = exampleDB
dbUser = dbadmin
dbPassword = password
; if this parameter is True, vbr will prompt user for db password every time
dbPromptForPassword = False

[Transmission]
encrypt = False
checksum = False
port_rsync = 50000
; bandwidth limit in KBPS, 0 for unlimited
bwlimit = 0

[Mapping]
; backupDir ignored for copy cluster task
v_exampledb_node0001 = backup01:/home/dbadmin/backups
v_exampledb_node0002 = backup02:/home/dbadmin/backups
v_exampledb_node0003 = backup03:/home/dbadmin/backups
```

Using Hard File Link Local Backups

You can use the `vbr.py` utility `hardLinkLocal` option to create a full-, or object-level, backup with hard file links on a local database host.

Creating hard link local backups can provide the following advantages over a remote host backup:

- **Speed** — Hard link local backups are significantly faster than a remote host backup. In a hard link local backup, `vbr.py` does not copy files (as long as the backup directory exists on the same file system as the database catalog and data directories).
- **Reduced network activities** — The hard link local backup minimizes network load because it does not require `rsync` to copy files to a remote backup host.
- **Less disk space** — Since the backup includes a copy of the catalog and hard file links, the local backup uses significantly less disk space than a backup with copies of database data files. However, since a hard link local backup saves a full copy of the catalog each time you run `vbr.py`, the disk size will increase with the catalog size over time.

Hard link local backups are useful during experimental designs and development cycles. Database designers and developers can create hard link local object backups of schemas and tables on a regular schedule during design and development phases. If any new developments are unsuccessful, developers can restore one or more objects easily and efficiently.

Note: Running `vbr.py` does not affect active database applications. The `vbr.py` utility supports creating backups while concurrently running applications executing DML statements, including `COPY`, `INSERT`, `UPDATE`, `DELETE`, and `SELECT`.

Planning Hard Link Local Backups

If you plan to use hard link local backups as a standard site procedure, consider storing all of the data files on one file system per node when designing the overall database setup and hardware configuration. Such a configuration has the advantage of being set up for hard link local backups automatically. However, using one file system per node to support hard link local backups does preclude the use of external storage locations on separate file systems.

Specifying Backup Directory Locations

The `backupDir` parameter of the configuration file specifies the location of the top-level backup directory. Hard link local backups require that the backup directory be located on the same Linux file system as the database data and catalog files. The Linux operating system cannot create hard file links to another file system.

Do not create the hard link local backup directory in a database data storage location. For example, as a best practice, the database data directory should not be at the top level of the file system, such as this:

```
/home/dbadmin/data/VMart/v_vmart_node0001
```

Instead, HP Vertica recommends adding another subdirectory for data above the database level, such as in this example:

```
/home/dbadmin/data/dbdata/VMart/v_vmart_node0001
```

You can then create the hard link local backups subdirectory as a peer of the data directory you just created, such as in this example:

```
/home/dbadmin/data/backups  
/home/dbadmin/data/dbdata
```

For more information about how `vbr.py` creates directories for backups and archives, see [Backup Directory Structure and Contents](#).

Understanding Hard Link Local Backups and Disaster Recovery

Hard link local backups are only as reliable as the disk on which they are stored. If the local disk becomes corrupt, so does the hard link local backup. In this case, you will be unable to restore the database from the hard link local backup, since it is also corrupt.

All sites should maintain full backups externally for disaster recovery. Since hard link local backups do not actually copy any database files, HP Vertica strongly recommends that you not use hard link local backups as the sole means of recovery, unless the backups are copied to tape or other external media, as described in [Creating Hard Link Local Backups](#).

Creating Full and Incremental Backups

Before you create a database backup, ensure the following:

- Your database is running. It is unnecessary for all nodes to be up in a **K-safe** database. However, any nodes that are down are not backed up.
- All of the backup hosts are up and available (see [Configuring Backup Hosts](#)).
- The backup host (either on the database cluster or not) has sufficient disk space to store the backups
- The user account of whoever starts the utility (`dbadmin`, or other) has write access to the target directories on the host backup location.

Run the `vbr.py` script from a terminal using the **database administrator** account from an initiator node in your database cluster. You cannot run the utility as root.

Note: If you have upgraded from HP Vertica 5.x to 6.x, backups created with 5.x are incompatible with 6.x. Once you have created new backups with the 6.x `vbr.py` utility, consider removing older backups created with 5.x.

Running Vbr Without Optional Commands

You can run the `vbr.py` with only its required commands:

- `--task backup`
- `--config-file config_file`

If your configuration file does not contain the database administrator password, `vbr.py` prompts you to enter the password, but does not display what you type:

```
[dbadmin@node02 ~]$ vbr.py --task backup --config-file nuvmartbak.ini
Enter vertica password for user dbadmin:
Preparing...
Found Database port: 5433
.
.
.
```

The utility requires no further interaction after you invoke it.

To run the `vbr.py` utility:

1. Use the `--task backup` and `--config-file filename` directives as shown in this example:

```
> vbr.py --task backup --config-file myconfig.ini Copying...
[=====] 100%
All child processes terminated successfully.
Committing changes on all backup sites...
backup done!
```

By default, there is no screen output, other than the progress bar. To include additional progress information, use the `--debug` option, with a value between 1 – 3.

2. If the utility does not locate the configuration you specify, it fails with an error and exits.

Best Practices for Creating Backups

When creating backup configuration files:

- Create separate configuration files to create full- and object-level backups
- Use the same backup host directory location for both kinds of backups
- For best network performance, use one backup host per cluster node

- Use one directory on each backup-node to store successive backups
- For future reference, append the major Vertica version number to the configuration file name (mybackup76x)

Using the same backup host directory location for full- and object-level backups results in the backups sharing disk space and being compatible when performing a restore. Each cluster node must also use the same directory location on its designated backup host.

The selected objects of a backup can include one or more schemas or tables, or a combination of both. For example, you can include schema S1 and tables T1 and T2 in an object-level backup. Multiple backups can be combined into a single backup. A schema-level backup can be integrated with a database backup (and a table backup integrated with a schema-level backup, and so on).

Object-Level Backups

After you create a full backup, you can create object-level backups, as described in [Creating Object-Level Backups](#).

To see existing full and incremental backups, see [Viewing Backups](#).

Backup Locations and Storage

Backups are stored in the directory you specify in the configuration file. The directory containing the backup has a subdirectory for each node backed up to that location. In turn, that area contains a subdirectory with the name of the backup, or multiple backup names, appended with the date and time if you are keeping more than one backup. The backup name reflects the `snapshotName` parameter value in the configuration file. See the [Backup Directory Structure and Contents](#) for further information.

Saving Incremental Backups

Each time you back up your database with the same configuration file, `vbr.py` creates an incremental backup, copying new storage containers, which can include data that existed the last time you backed up the database, along with new and changed data since then. By default, `vbr.py` saves one archive backup, unless you set the `restorePointLimit` parameter value in the configuration file to a value greater than 1.

For creating backup files, `vbr.py` uses the following directory naming conventions:

snapshotname_archivedate_timestamp

snapshotname

The value of the `snapshotName` parameter in the configuration file. This is the name used as a prefix to the container directory on the backup host.

| | |
|-------------------------------------|---|
| <code>_archivedate_timestamp</code> | <p>The prefix (<code>_archive</code>) indicates the directory contains an archive. The <code>vbr.py</code> utility appends the date, in the form <code>date_timestamp</code> to the directory name when the backup is created. For instance, if you specify the backup name as <code>mysnap</code>, and create backups over three consecutive days, multiple subdirectories with archive names such as these will exist:</p> <ul style="list-style-type: none"> • <code>mysnap_archive20111111_205841</code> • <code>mysnap_archive20111112_205844</code> • <code>mysnap_archive20111113_206841</code> |
|-------------------------------------|---|

You can restore from any archive, as described in [Restoring from a Full Backup](#).

When vbr.py Deletes Older Backups

Running the `vbr.py` utility with the `--task backup` command deletes the oldest backup whenever the total number that exist exceeds the `restorePointLimit` value in the configuration file. For instance, if the `restorePointLimit = 5`, and five archives exist, running the `vbr.py --task backup` utility again deletes the backup with the oldest `date_timestamp`, before the utility completes the current backup command.

When you invoke `vbr.py` to create a backup, this is what occurs:

1. The utility obtains the value of the `restorePointLimit` parameter value to determine how many backups should exist in the archive.
2. If creating the next backup will exceed the restore point limit, `vbr.py` deletes the oldest archived backup.
3. `vbr.py` continues processing and initially creates the backup on the database cluster.
4. When the new backup is complete, `vbr.py` copies it from the database cluster to the designated backup location.
5. After the new backup is successfully copied to the backup location, `vbr.py` removes the backup from the database cluster.

Backup Directory Structure and Contents

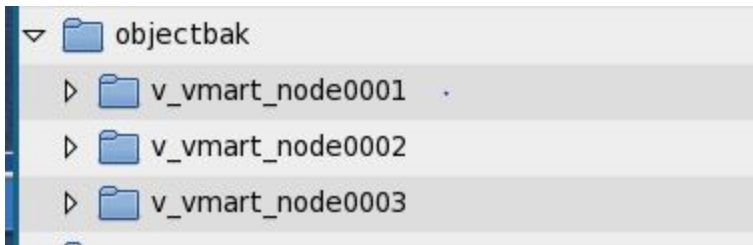
The first time you execute the `vbr.py` utility with a new configuration file, it creates a set of backup directories. Subsequently running the utility also creates subdirectories, but within the structure defined during the first invocation, which this section describes.

For information about where to create hard link local backups, see [Using Hard File Link Local Backups](#).

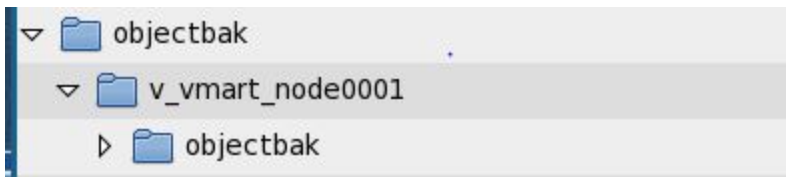
Directory Tree

The directory structure starts at the location you specify as the `backupDir` parameter value. In that directory, `vbr.py` creates the top-level backup subdirectory using the `snapshotName` value in the configuration file. The default configuration file name consists of the `snapshotName` value, with an `.ini` suffix. You can use any valid file name for the configuration file, or change the name once the file exists. Throughout this example, the backup name is `objectbak`, and the configuration file name is `objectbak.ini`.

- The top level directory starts with the backup name you specified in the `snapshotName` parameter value, and creates a structure based on the cluster nodes listed in the configuration file. In the following example, the configuration file parameter `snapshotName=objectbak`, and the three `dbNode` parameters specified are `v_vmart_node0001`, `v_vmart_node0002`, and `v_vmart_node0003`:



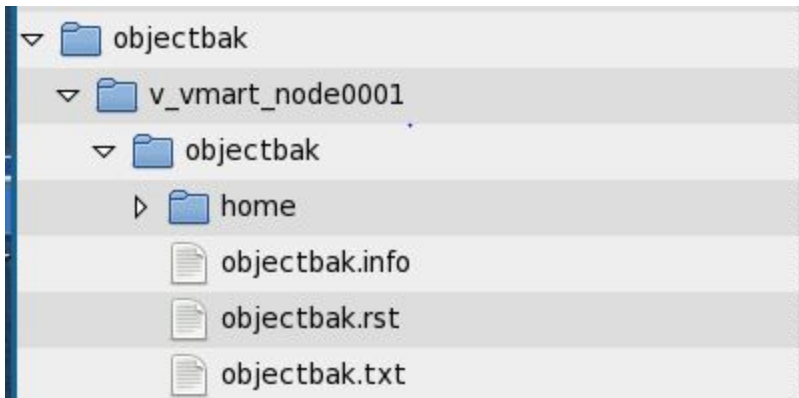
- When you create a backup, for each designated node, `vbr.py` creates a subdirectory below the node directory, also named by the `snapshotName` parameter value for the backup name:



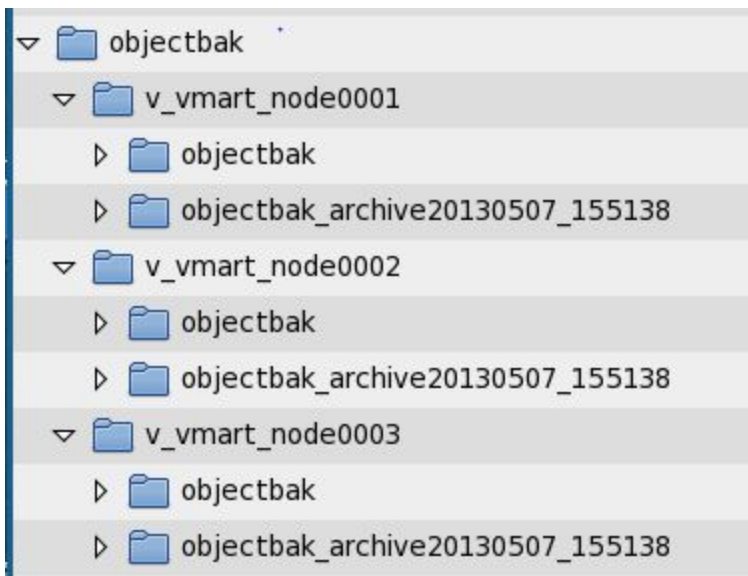
Multiple Restore Points

Each time you invoke `vbr.py` to create a new backup, the utility verifies the `restorePointLimit` value.

- If the value is greater than 1, the utility checks to see how many backups and archives exist in each node's backup name subdirectory. If no archive files exist, the utility creates the first backup in a new subdirectory with the backup name, one per node.
- The backup name subdirectory for each node also contains a `home` subdirectory that includes two important files, `backupname.info` and `backupname.txt`. Among other information, these files contain the archive OID of each backup (unique across all nodes), and the backup epoch:



- The next time you create a backup using the same configuration file, the `restorePointLimit` value is verified again. If `vbr.py` is to save another backup, it gets the OID from the info file of the existing backup, and saves that previous backup to a new directory. The new directory for the last backup is then an archive, named with the backup name (`objectbak` in this example), and the OID suffix `_archivedatestamp_timestamp`.
- The following example lists the backup directory, `objectbak`, with subdirectories for each node. Each node then has one subdirectory for the backup (`objectbak`) and a peer subdirectory for the first archive:



- Thereafter, each time you create a backup, the utility continues this process until the number of backups and archives together equal the `restorePointLimit` value. Then, `vbr.py` deletes the oldest archive backup, and saves a new backup.

Once you save multiple backups, you can restore directly from the archive of your choice, as described in [Restoring Full Database Backups](#).

Creating Object-Level Backups

You can create object-level backups consisting of one or more schemas and tables. Object-level backups are especially useful for multi-tenanted database sites. For example, an international airport could use a multi-tenanted database to represent different airlines in its schemas. Then, tables could maintain various types of information for the airline, including ARRIVALS, DEPARTURES, and PASSENGER information. With such an organization, creating object-level backups of the specific schemas would let you restore by airline tenant, or any other important data segment.

Note: You cannot restore part of an object-level backup; you must restore the entire backup. For example, if you create an object-level backup containing two schemas, `schema1` and `schema2`, and later want to restore `schema1`, you cannot do so without also restoring `schema2`. To restore a single object, you can create a single object backup.

There are two configuration file parameters used explicitly with object-level backups, and described in the section [VBR Configuration File Reference](#):

- Object
- Overwrite

For more information about creating configuration files for full- or object-level backups, see [Configuring Required VBR Parameters](#).

Invoking `vbr.py` Backup

After creating the configuration file specifying which objects to backup, you can create an object backup. The following command uses the `objectbak.ini` configuration file:

```
[dbadmin@node01 ~]$ vbr.py --task backup --config-file objectbak.ini
Preparing...
Found Database port: 5433
Copying...
[=====] 100%
All child processes terminated successfully.
Committing changes on all backup sites...
backup done!
[dbadmin@node01 ~]$
```

Backup Locations and Naming

You can use one top-level backup directory to store both full- and object-level backups. For more information about directory usage for backups, see [Backup Directory Structure and Contents](#).

Note: You must use unique names for full- and object-level backups stored at one location. Otherwise, creating successive backups will overwrite the previous version.

To see existing full- and object-level backups, see [Viewing Backups](#).

Best Practices for Object-Level Backups

To create one or more object-level backups, create a configuration file specifying the backup location, the object-level backup name, and a list of objects to include (one or more schemas and tables). When creating configuration backup files:

- Create one configuration file for each object-level backup
- Create a different configuration file to create a full database backup
- For best network performance, use one backup host per cluster node
- Use one directory on each backup-node to store successive backups
- For future reference, append the major Vertica version number to the configuration file name (mybackup7x)

Using the same backup host directory location for full- and object-level backups results in the backups sharing disk space and being compatible when performing a restore. Each cluster node must also use the same directory location on its designated backup host.

The selected objects of a backup can include one or more schemas or tables, or a combination of both. For example, you can include schema S1 and tables T1 and T2 in an object-level backup. Multiple backups can be combined into a single backup. A schema-level backup can be integrated with a database backup (and a table backup integrated with a schema-level backup, and so on).

Naming Conventions

Give each object-level backup configuration file a distinct and descriptive name. For instance, at the airport terminal, schema-based backup configuration files use a naming convention with an airline prefix, followed by further description, such as:

AIR1_daily_arrivals_snapshot

AIR2_hourly_arrivals_snapshot

AIR2_hourly_departures_snapshot

AIR3_daily_departures_snapshot

Once database and object-based backups exist, you can recover the backup of your choice. For more information, see [Restoring Object-Level Backups](#).

Note: Do not change object names in an object-level configuration file once a backup already exists. When the backup name and backup locations are the same, even if you change the objects listed in the configuration file, the later backup overwrites the first so you cannot restore from the earlier backup. Instead, create a different configuration file.

Creating Backups Concurrently

In a multi-tenanted database, you need to create object-level snapshots concurrently. For example, at the airport terminal, AIR1 and AIR2 both need schema snapshots each hour. To accomplish this goal without concurrently backing up the same snapshot, the `vbr.py` utility currently permits only one instance of the backup script per initiator node.

Due to this present behavior, HP Vertica suggests the following:

- Assign one initiator node to create backup for a given tenant.
- Give each object backup initiated on a different node a unique backup name.
- Start the backup script on different initiator nodes to create their specific tenant backups concurrently.

Determining Backup Frequency

HP Vertica recommends a best practice of taking frequent backups to avoid backup bloating if database contents diverge in significant ways.

Always take backups after any event that significantly modifies the database, such as performing a rebalance. Mixing many backups with significant differences can weaken data k-safety, such as taking backups before a rebalance, and again after the rebalance when the backups are all part of one archive.

Understanding Object-Level Backup Contents

While a full database backup contains a complete collection of the database data files, with the catalog and all of its objects, an object-level backup comprises only the elements necessary to restore the schema or table, including the selected, dependent, and principal objects (see [Types of Backups](#)). In summary, an object-level backup includes the following contents:

- Storage: Data files belonging to the specified object (s)
- Metadata: Including the cluster topology, timestamp, epoch, AHM, and so on
- Catalog snippet: persistent catalog objects serialized into the principal and dependent objects

Some of the elements that comprise AIR2, for instance, include its parent schema, tables, named sequences, primary key and foreign key constraints, and so on. To create such a backup, `vbr.py` script saves the objects directly associated with the table, along with any dependencies, such as foreign key (FK) tables, and creates an object map from which to restore the backup.

Note: Because the data in local temp tables persists only within a session, local temporary tables are excluded when you create an object-level schema. For global temporary tables, `vbr.py` stores the table's definition.

Making Changes After an Object-Level Backup

After creating an object-level backup, dropping schemas and tables from the database means the objects will also be dropped from subsequent backups. If you do not save an archive of the object backup, such objects could be lost permanently.

Changing a table name after creating a table backup will not persist after restoring the backup. If you drop a user after a backup, and the user is the owner of any selected or dependent objects, restoring the backup also restores the user.

If you drop a table (t1), and then create a new table, also called t1, restoring an object backup causes an error. While tables in the backup and the current database have identical names, their object identification numbers (OIDs) differ.

To restore a dropped table from a backup:

1. Rename the newly created table from t1 to t2.
2. Restore the backup containing t1.

K-safety may have increased after an object backup. If this occurs, a backup restore will fail if any table in the backup has insufficient projections.

Understanding the Overwrite Parameter

The configuration file `Overwrite` parameter determines what happens during an object-level restore when two objects have identical OIDs. The `overwrite` parameter has a default value set internally to `vbr.py`.

To change the default setting for the `overwrite` parameter, you must add it to the configuration file explicitly. It is not included as a question or prompt when running `vbr.py` to create a configuration file.

Overwrite applies to any objects associated with the schema and table saved in the backup (including users and named sequences), whose OIDs remain the same. For instance, consider this use of overwriting in a backup:

1. Create an object backup of `mytable`.
2. After the backup, rename `mytable` to `mytable2`.
3. Restoring the backup causes `mytable` to overwrite `mytable2` (`Overwrite=true`).

The overwrite occurs because although the table names differ, their OIDs do not.

Conversely, creating a new table of the same name (with a different OID) is handled differently.

1. Create a table backup of `mytable`.
2. Drop `mytable`.

3. Create a new table, called mytable.
4. Restoring the backup does NOT overwrite the table, and causes an error, since there is an OID conflict.

Changing Principal and Dependent Objects

If you create a backup and then drop a principal object, restoring the backup restores the principal object. For example, if you drop the owner of a table included in a backup, restoring the snapshot recreates the user, along with any permissions that existed when the backup was taken.

Adding dependent objects after backing up a parent object will drop the dependent objects when the parent is restored. For example, restore will drop a constraint or a projection added to a table after the table backup, as will columns added to a table, or tables added to a schema.

Identity and auto-increment sequences are dependent objects, since they cannot exist without their tables. Such objects are backed up along with the tables on which they depend.

Named sequences are not dependent objects, since they exist autonomously. A named sequence remains after dropping the table in which the sequence is used. In this case, the named sequence is a principal object that must be backed up with the table and regenerated if it does not already exist when you restore the table. If the sequence does exist, it is used, unmodified. Sequence values could repeat, if you restore the full database and then restore a table backup to a newer epoch.

Considering Constraint References

You must backup all database objects that are related through constraints. For example, a schema with tables whose constraints reference only tables in the same schema can be backed up, but a schema containing a table with an FK/PK constraint on a table in another schema cannot, unless you include the other schema in the list of selected objects.

Configuration Files for Object-Level Backups

The `vbr.py` utility automatically associates configurations with different backup names, but the same backup location.

Always create a cluster-wide configuration file and one or more object-level configuration files pointing to the same backup location. Storage between backups is shared, so you will not have multiple copies of the same data. For object-level backups, using the same backup location causes `vbr.py` to encounter fewer OID conflict prevention techniques and have less problems when restoring the backup.

By using cluster and object configuration files with the same backup location, the utility includes additional provisions to ensure that the object-level backups can be used following a full cluster restore. One scenario to complete a full cluster restore is to use a full database backup to bootstrap the cluster. Once the cluster is operational again, you can restore the most recent object-level backups for schemas and tables.

Note: Attempting to restore a full database using an object-level configuration file fails,

resulting in this error:

```
VMart=> /tmp/vbr.py --config-file=Table2.ini -t restore
Preparing...
Invalid metadata file. Cannot restore.
restore failed!
```

Backup Epochs

Each backup includes the epoch to which its contents can be restored. This epoch will be close to the latest epoch, though the epoch could change during the time the backup is being written. If an epoch change occurs while a backup is being written, the storage is split to indicate the different epochs.

The `vbr.py` utility attempts to create an object-level backup five times before an error occurs and the backup fails.

Maximum Number of Backups

There is a limit of 500 full- and object-level backups at each backup location. This maximum is set by `rsync`, and does not include archives, so the total number of saved backups at the same location can exceed 500.

For example, if a database has 500 schemas, S1 – S499, the full database, including archives of earlier snapshots, can be backed up along with backups for each schema.

Creating Hard Link Local Backups

Before creating a full hard link local database backup, ensure the following:

- Your database is running. All nodes need not be up in a **K-safe** database for `vbr.py` to run. However, keep in mind that any nodes that are down are not backed up.
- The user account of whoever starts the utility (`dbadmin`, or other) has write access to the target backup directories.

When you create a full- or object-level hard link local backup, these are the backup contents:

| Backup | Catalog | Database files |
|---------------------|-----------|--|
| Full backup | Full copy | Hard file links to all database files |
| Object-level backup | Full copy | Hard file links for all objects listed in the configuration file, and any of their dependent objects |

Run the `vbr.py` script from a terminal using the **database administrator** account from a node in your database cluster. You cannot run the utility as `root`.

To create a full or object-level backup, enter the following command:

```
> /opt/vertica/bin/vbr.py --task backup --config fullbak.ini
```

Note: While not required, HP Vertica recommends that you first create a full backup before creating any object-level backups.

Specifying the Hard Link Local Backup Location

If you add the `hardLinkLocal=True` parameter to the configuration file, but specify a backup directory on a different node, `vbr.py` issues a warning message and stops processing the backup. Change the configuration file to include a backup directory on the same host and file system as the database and catalog files, and run the backup utility again.

If you add the `hardLinkLocal=True` parameter but specify a backup destination directory on a different file system from the database and catalog files, but on the same node, `vbr.py` issues a warning message and proceeds with the backup by copying the files on the node from one file system to the other.

Creating Hard Link Local Backups for Tape Storage

You can use hard link local backups as a staging mechanism to backup to tape or other forms of storage media. You can also use the hard link local backup to restore the hard file links to the database files.

The following steps present a simplified approach to saving, and then restoring, hard link local backups from tape storage:

1. Create a configuration file using a command such as this:

```
/opt/vertica/bin/vbr.py --setupconfig
```

2. Edit the configuration file (`localbak.ini` in this example) to include the `hardLinkLocal=True` parameter in the `[Transmission]` section.
3. Run the backup utility with the configuration file:

```
/opt/vertica/bin/vbr.py --task backup --config-file localbak.ini
```

4. Copy the hard link local backup directory with a separate process (not `vbr.py`) to tape or other external media.
5. If the database becomes corrupted, create the directory structure that existed when you created the hard link local backup.
6. Transfer the backup files from tape to their original backup directory.
7. Using the configuration file you used to create the hard link local backup (Step 3), restore the

database using the following command:

```
/opt/vertica/bin/vbr.py --task restore --config-file localbak.ini
```

When you restore from a hard link local backup (copied from tape), `vbr.py` creates hard links from the backup files to the database directory, if possible, again saving significant disk space, and time.

Interrupting the Backup Utility

To cancel a backup, use Ctrl+C or send a SIGINT to the Python process running the backup utility. The utility stops the backup procedure after it has completed copying the data.

The files generated by an interrupted backup process remain in the target backup location directory. The next backup process picks up where the interrupted process left off.

Backup operations are atomic, so that interrupting a backup operation does not affect the previous backup. The previous backup is replaced only as the very last step of backing up your database.

The `restore` or `copy-cluster` operations overwrite the database catalog directory, so interrupting either of these processes leaves the database unusable until you restart the process and allow it to finish.

Viewing Backups

These are the ways to view backups:

1. Using `vbr.py`, list the backups that reside on the local or remote backup host (requires a configuration file).
2. Monitor backup information while `vbr.py` is executing, as described in the [DATABASE_SNAPSHOTS](#) system table description in the SQL Reference Manual.
3. View historical information about backups with the [DATABASE_BACKUPS](#) system table.

Note: Since the `database_backups` system table contains historical information, it is not updated when you delete the backups, as described in [Removing Backups](#).

List Backups With `vbr.py`

To list backups on the backup hosts, use the `vbr.py --listbackup` task, with a specific configuration file. The following example uses an object-level configuration file, `table2bak.ini`, which results in listing the table object backup, `test2`:

```
dbadmin@node01 temp]$ /opt/vertica/bin/vbr.py --task listbackup --config-file /home/dbadm
in/table2bak.ini
```

```

Found 1 backup(s) with given config file /home/dbadmin/table2bak.ini.
backup          epoch  objects  hosts(nodes)
                                     table2bak_20130516_072002  180  tes
t2   v_vmart_node0001(192.168.223.33), v_vmart_node0002(192.168.223.33), v_vmart_node000
3(192.168.223.33)

```

The `vbr.py` output information includes the backup OID, the epoch, which object(s) (if applicable) were backed up, and the backup host (192.168.223.33) used for each node. The identifier appended to the backup name (20130516_072002 in this example), is the value you need when specifying an archive to restore.

Monitor database_snapshots

You can monitor the backups while `vbr.py` is in process. The following examples show the system table while a single host node is being backed up:

```

VMart=> select * from database_snapshots;

node_name      | snapshot_name | is_durable_snapshot | total_size_bytes | storage_cost_
bytes | acquisition_timestamp
-----+-----+-----+-----+-----
v_vmart_node0001 | fullbak      | t                    | 299089541 | 8
63439 | 2013-09-04 14:15:23-07
(1 row)

```

Query database_backups

Use the following query to list historical information about backups. The `objects` column lists which objects were backed up in object-level backups. However, do not use the `backup_timestamp` value when restoring an archive. Use the values in the `vbr.py --task listbackup`, described in the previous section when restoring an archive.

```

VMart=> select * from v_monitor.database_backups;
-[ RECORD 1 ]-----+-----
backup_timestamp | 2013-05-10 14:41:12.673381-04
node_name       | v_vmart_node0003
snapshot_name   | schemabak
backup_epoch    | 174
node_count      | 3
objects         | public, store, online_sales
-[ RECORD 2 ]-----+-----
backup_timestamp | 2013-05-13 11:17:30.913176-04
node_name       | v_vmart_node0003
snapshot_name   | kantibak
backup_epoch    | 175
node_count      | 3
objects         |
-[ RECORD 13 ]-----+-----
backup_timestamp | 2013-05-16 07:02:23.721657-04
node_name       | v_vmart_node0003

```

```

snapshot_name | objectbak
backup_epoch  | 180
node_count    | 3
objects       | test, test2
-[ RECORD 14 ]-----
backup_timestamp | 2013-05-16 07:19:44.952884-04
node_name        | v_vmart_node0003
snapshot_name    | table1bak
backup_epoch     | 180
node_count       | 3
objects          | test
-[ RECORD 15 ]-----
backup_timestamp | 2013-05-16 07:20:18.585076-04
node_name        | v_vmart_node0003
snapshot_name    | table2bak
backup_epoch     | 180
node_count       | 3
objects          | test2

```

Restoring Full Database Backups

To restore a full database backup, you must ensure that:

- The database is down. You cannot restore a full backup when the database is running.
- All of the backup hosts are up and available.
- The backup directory exists and contains the backups from which to restore.
- The cluster to which you are restoring the backup has the same number of hosts as the one used to create the backup. The node names and the IP addresses must also be identical.
- The database you are restoring must already exist on the cluster to which you are restoring data. The database can be completely empty without any data or schema. As long as the database name matches the name in the backup, and all of the node names match the names of the nodes in the configuration file, you can restore to it.

You can use only a full database backup to restore a database from scratch. If you have saved multiple backup archives, you can restore from either the last backup, or a specific archive.

When you restore a full database backup, OIDs from each backup (including all archives) in the same backup are injected into the restored catalog of the full database backup. Additionally, the OID generator and current epoch are set to the respective maximum OID and epoch from the full database backup.

For information about monitoring database restoration, see [Monitoring Recovery](#).

Restoring the Most Recent Backup

To perform a full database restore, the cluster must be DOWN. When a node or cluster is DOWN the expected use case is to return the cluster to its most recent state, by restoring a full database

backup. You can restore any full database backup from the archive by identifying the name in the configuration file.

To begin a full database restore, log in using the **database administrator's** account. You cannot run the utility as root.

To restore from the most recent backup, use the configuration file used to create the backup, specifying `vbr.py` with the `--task restore`. If your configuration file does not contain the database superuser password, the utility prompts you to enter it at run time.

The following example uses the `db.ini` configuration file, which includes the superuser's password:

```
> vbr.py --task restore --config-file db.ini
Copying...
1871652633 out of 1871652633, 100%
All child processes terminated successfully.
restore done!
```

You can restore a snapshot only to the database from which it was taken. You cannot restore a snapshot into an empty database.

Restoring an Archive

If you saved multiple backups, you can specify a specific archive to restore. To list the archives that exist to choose one to restore, use the `vbr.py --listbackup` task, with a specific configuration file. For more information, see [Viewing Backups](#).

To restore from one of several archives:

1. Log in using the **database administrator's** account. You cannot run the utility as root.

Invoke the utility with the `--task restore` command, the configuration file with which you created the backup, followed by the `--archive` parameter with the `date_timestamp` suffix of the directory name to identify which archive to restore. For example:

```
> vbr.py --task restore --config-file fullbak.ini --archive=20121111_205841
```

2. The `vbr.py` utility restores the snapshot.

The `--archive` parameter identifies the archive subdirectory, in this example, created on 11-11-2012 (`_archive20121111`), at time 205841 (20:58:41). You need specify only the `_archive` suffix, because the configuration file identifies the snapshot name of the subdirectory, and the OID identifier indicates the backup is an archive.

Attempting to Restore a Node That Is UP

During a full database restore, the node must be DOWN. If you start the restore process and the node is UP, `vbr.py` displays the following message:

```

doc:tests/doc/tools $ vbr.py --config-file=doc.ini -t restore --nodes=v_doc_node0001
Warning: trying to restore to an UP cluster
Warning: Node state of v_doc_node0001 is UP; node must be DOWN for restore; ignoring restore on this node.
Nothing to do
restore done!

```

To restore the node, first bring it down, and then run the utility again.

Attempting to Restore to an Alternate Cluster

You can restore a full backup ONLY to a cluster with the node names and IP addresses used to create the backup. The `vbr.py` utility does NOT support restoring a full database backup to an alternate cluster with different host names and IP addresses.

Attempting to restore a full backup to an alternate cluster when the source database is accessible can render the source database unusable. The backup you are restoring contains critical references to the source database and uses that information during the restore process.

Restoring Object-Level Backups

To restore an object-level backup to the database from which it was taken, the database must be UP. The `vbr.py` configuration file you use for the restore task specifies the backup to restore, as described in [Creating Object-Level Backups](#). To restore a full database backup, see [Restoring Full Database Backups](#). You cannot restore any object-level backup into an empty database.

Note: You cannot restore part of an object-level backup; you must restore the entire backup. For example, if you create an object-level backup containing two schemas, `schema1` and `schema2`, and later want to restore `schema1`, you cannot do so without also restoring `schema2`. To restore a single object, you can create a single object backup.

Backup Locations

Backups created in the same backup location are compatible. After restoring a full database backup, you can restore object-level backups saved in the same backup location as the full backup.

Note: Using different backup locations in which to create full- or object-level backups results in incompatible object-level backups. Attempting to restore an object-level backup after restoring a full database will fail.

Restoring an object-level backup does not affect any schemas or tables that were not included in the original backup you are restoring. For instance, in a multi-tenanted database, if you create schema backups for each of four different tenants, restoring one tenant schema does not overwrite the other three.

Cluster Requirements for Object-Level Restore

To restore from a full database backup, the cluster must be `DOWN`. To restore from an object-level backup, the database cluster must be `UP`.

After successfully restoring from a full database backup, after the cluster is `UP`, you can restore from an object-level backup, as long as it was created in the same backup location as the full backup you just restored.

Regardless of the node states when a backup was taken, you do not have to manage node states before restoring an object-level backup. Any node that is `DOWN` when you restore an object-level backup is updated when the node rejoins the cluster and comes `UP` after an object-level restore.

You restore a backup by name. An archive can comprise multiple snapshots, including both full- and object-level backups.

Restoring Objects to a Changed Cluster Topology

Unlike restoring from a full database backup, `vbr.py` supports restoring object-level backups after adding nodes to the cluster. Any nodes that were not in the cluster when you created the object-level backup will be updated as necessary when `vbr.py` restores the backup.

You cannot restore an object-level backup after removing nodes, altering node names, or changing IP addresses. Trying to restore an object-level backup after such changes causes `vbr.py` to fail and display this message:

```
Preparing...
Topology changed after backup; cannot restore.
restore failed!
```

Projection Epoch After Restore

All object-level backup and restore events are treated as DDL events. If a table does not participate in an object-level backup, possibly due to a node being down, restoring the backup has this effect on the projection:

- Its epoch is reset to 0
- It must recover any data that it does not have (by comparing epochs and other recovery procedures)

For more information, see [Failure Recovery](#).

Catalog Locks During Backup Restore

As with other databases, HP Vertica transactions follow strict locking mechanisms to maintain data integrity. For more information about session-scoped transaction locking, see ["Transactions" on page 1](#) in the Concepts Guide and the system table column descriptions in ["LOCK_USAGE" on page 1](#) of the SQL Reference Manual.

When restoring an object-level backup into a cluster that is UP, `vbr.py` begins by copying data and managing storage containers, potentially splitting the containers if necessary. While copying and managing data is the largest part of the restore process, it does not require any database locks.

After completing the first part of the process, `vbr.py` requires a table object lock (O-lock), and then a global catalog lock (GCLX). If other database operations already have locks on any associated tables, the restore process is blocked until the locks are released.

Once the restore process obtains a table O-lock, `vbr.py` blocks other operations that require a lock on the same table. To guarantee catalog consistency, processes can hold a GCLX for a minimal duration. When the restore locks are in effect, any concurrent table modifications are blocked until the locks are released. Database system operations, such as the tuple mover (TM) transferring data from memory to disk, are canceled to allow the object-level restore to proceed.

Catalog Restore Events

Each object-level backup includes a section of the database catalog, called a *snippet*, which contains the selected objects, their dependent objects, and principal objects. The catalog snippet is similar in format to the database catalog, but consists of a subset representing the object information. Objects being restored can be read from the catalog snippet and used to update the global and local catalogs.

Each object from a restored backup is updated in the catalog. If the object no longer exists, `vbr.py` adds the object to the catalog. Any dependent objects that are not in the backup are dropped from the catalog.

The `vbr.py` utility uses existing dependency verification methods to check the catalog, and adds a restore event to the catalog for each restored table. That event also includes the epoch at which the event occurred. Any node that misses the restore event will recover projections anchored on the given table.

A DML statement that references an object being restored must wait for the restore to complete before being able to lock the table. If a DML statement is in progress when the restore attempts to place an O-lock on the table, the DML statement must finish before the restore can complete.

Restoring Hard Link Local Backups

This section describes issues around restoring from hard link local full- and object-level backups.

Restoring Full- and Object-Level Hard Link Local Backups

If you have created both full- and object-level backups and the database fails, first restore the full database backup. You can then restore from the object-level backups.

Avoiding OID and Epoch Conflicts

If you create full- and object-level backups in the same backup directory (recommended), when you restore a full backup, `vbr.py` determines the latest OID and epoch of the object-level backups as well.

Consider the following scenario:

1. Create a full hard link local backup in backup directory `/home/dbadmin/backups`, with configuration file `mybak.ini`:

```
[dbadmin@node01 ~]$ /opt/vertica/bin/vbr.py --task backup --config-file mybak.ini
```

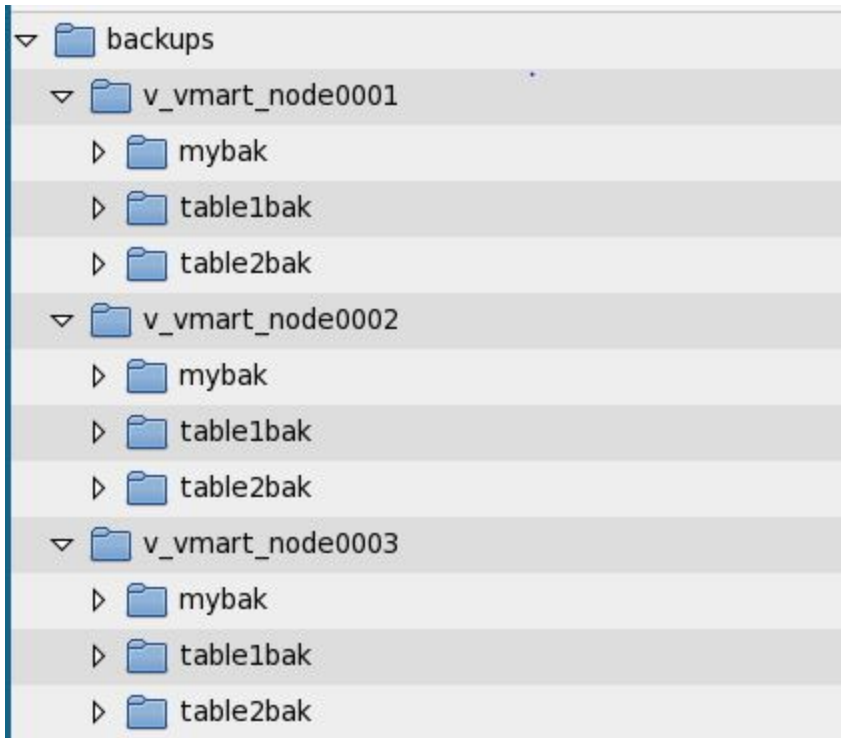
2. Create an object-level hard link local backup for `Table1` using the same backup directory, with configuration file `table1bak.ini`:

```
[dbadmin@node01 ~]$ /opt/vertica/bin/vbr.py --task backup --config-file table1bak.ini
```

3. Create an object-level hard link local backup for `Table2`, using the same backup directory, with configuration file `table2bak.ini`.

```
[dbadmin@node01 ~]$ /opt/vertica/bin/vbr.py --task backup --config-file table2bak.ini
```

After creating these hard link local backups, using the `/backups` directory, the following directory structure exists:



Given this situation, the following occurs when you need to restore from a hard link local backup:

- When restoring a full DB (when `table1` and `table2` backups exist in the same backup directory), `vbr.py` detects the maximum object ID (OID) and epochs from the object-level table

backups and sets them in the restored database. This prevents OID and epoch conflicts from occurring when object-level backups are restored after the newly restored database.

- If the database and object-level table backups are not in the same backup directory, `vbr.py` reverts the maximum OID and epoch for a full database restore back to the table OID and epoch. Further attempts to restore either `table1` or `table2` backups will then fail, preventing any potential conflicts.

Transferring Backups to and From Remote Storage

Once a full hard link local backup exists, you can use a utility (other than `vbr.py`) to transfer the backup to another storage media, such as tape. Transferring hard link local backups to another storage media may copy the data files associated with the hard file links. The external media will then contain copies of the files.

Complete the following steps to restore hard link local backups from external media:

1. If the original backup directory no longer exists on one or more local backup host nodes, recreate the directory. The directory structure into which you restore hard link backup files must be identical to what existed when the backup was created. For example, if you created hard link local backups at the following backup directory, then recreate that directory structure:

```
/home/dbadmin/backups/localbak
```

2. Copy the backup files to their original backup directory, as specified for each node in the configuration file with the `backupHost` and `backupDir` parameters. For example, this configuration file shows the `backupDir` parameter for `v_vmart_node0001`:

```
[Mapping0]dbNode = v_vmart_node0001
backupHost = node03
backupDir = /home/dbadmin/backups/localbak
```

3. To restore the latest version of the backup, move the backup files to this directory:

```
/home/dbadmin/backups/localbak/node_name/snapshotname
```

4. To restore a different backup version, move the backup files to this directory:

```
/home/dbadmin/backups/localbak/node_name/snapshotname_archivedate_timestamp
```

5. When the backup files are returned to their original backup directory, use the original configuration file to invoke `vbr.py` as follows:

```
>/opt/vertica/bin/vbr.py --task restore --config-file localbak.ini
```

If the physical files are restored from tape into the correct directories, and you use the configuration file that specifies `hardLinkLocal = true`, restoring the backup succeeds.

Note: You can use a different directory when you return the backup files to the hard link local backup host. However, you must also change the `backupDir` parameter value in the configuration file before restoring the backup.

Restoring to the Same Cluster

Restoring a database on the same cluster from a full-database backup consists of following these steps:

1. [Stopping the Database](#) you intend to restore.

Note: If you restore data to a single node, the node has already stopped. You do not need to stop the database.

2. [Restoring Full Database Backups](#).
3. Starting the database, see [Starting the Database \(admintools\)](#).

Note: If you restored all the nodes using the backup, you are using a manual recovery method, as described in [Failure Recovery](#). Administration Tools returns the message, "Database startup failed," after you attempt to restart the database and then offers to restart the database from an earlier epoch. Click **Yes**.

4. After the database starts, connect to it through the Administration Tools or MC and verify that it was successfully restored by running some queries.

Removing Backups

To remove existing backups:

1. Use `vbr.py --task listbackup` to identify the backup and archive names that reside on the local or remote backup host (requires a configuration file).
2. Delete the backup directories.

Deleting Backup Directories

Removing existing local or remote backup host backups requires deleting the backup directory. For example, to remove the sample `table2bak` backup in the following example:

```

dbadmin@node01 temp]$ /opt/vertica/bin/vbr.py --task listbackup --config-file /home/dbadm
in/table2bak.ini
Found 1 backup(s) with given config file /home/dbadmin/table2bak.ini.
backup                epoch  objects  hosts(nodes)

table2bak_20130516_072002  180    test2    v_vmart_node0001(192.168.223.33), v_vmart_node
0002(192.168.223.33), v_vmart_node0003(192.168.223.33)

```

1. Get these parameter values from the configuration file:
 - snapshotName
 - backupHost
 - backupDir
2. Also note the dbNode parameter values, indicating which nodes were backed up.
3. Connect to the backup host.
4. Navigate to the backup directory (the backupDir parameter value in the configuration file).
5. To delete all backups, delete the top-level snapshot directory

-or-
6. To delete an archive, navigate to the subdirectory for each database node, located below the top-level snapshot directory, and delete the archive directory.

Copying the Database to Another Cluster

You can use the `vbr.py` utility to copy the entire database to another HP Vertica cluster. This feature makes it easy to perform tasks such as copying a database between a development and a production environment. Copying your database to another cluster is essentially a simultaneous backup and restore — the data is backed up from the source database cluster and restored to the destination cluster in a single operation.

The directory locations for the HP Vertica catalog, data, and temp directories must be identical on the source and target database. Use the following `vsqL` query to see the source database directory locations. This example sets expanded display for illustrative purposes, and lists the columns of most interest, `node_name`, `storage_path`, and `storage_usage`.

```

VMart=> \xExpanded display is on.
VMart=> select node_name,storage_path, storage_usage from disk_storage;
-[ RECORD 1 ]+-----
node_name      | v_vmart_node0001
storage_path   | /home/dbadmin/VMart/v_vmart_node0001_catalog/Catalog
storage_usage  | CATALOG
-[ RECORD 2 ]+-----
node_name      | v_vmart_node0001

```

```

storage_path | /home/dbadmin/VMart/v_vmart_node0001_data
storage_usage | DATA,TEMP
-[ RECORD 3 ]+-----
node_name    | v_vmart_node0001
storage_path | home/dbadmin/SSD/schemas
storage_usage | DATA
-[ RECORD 4 ]+-----
node_name    | v_vmart_node0001
storage_path | /home/dbadmin/SSD/tables
storage_usage | DATA
-[ RECORD 5 ]+-----
node_name    | v_vmart_node0001
storage_path | /home/dbadmin/SSD/schemas
storage_usage | DATA
-[ RECORD 6 ]+-----
node_name    | v_vmart_node0002
storage_path | /home/dbadmin/VMart/v_vmart_node0002_catalog/Catalog
storage_usage | CATALOG
-[ RECORD 7 ]+-----
node_name    | v_vmart_node0002
storage_path | /home/dbadmin/VMart/v_vmart_node0002_data
storage_usage | DATA,TEMP
-[ RECORD 8 ]+-----
node_name    | v_vmart_node0002
storage_path | /home/dbadmin/SSD/tables
storage_usage | DATA
.
.
.

```

Notice the directory paths for the Catalog, Data, and Tempstorage. These paths are the same on all nodes in the source database, and must be the same in the target database.

Note: Copying a database to another cluster overwrites any existing data on the target cluster. If the target data is identical to the source database, data is not transferred again. However, if the target cluster contains data you want to retain, create a full database backup before invoking the `copycluster vbr.py` task.

Identifying Node Names for Target Cluster

You need to know the exact names that `Admintools` supplied to all nodes in the source database before configuring the target cluster.

To see the node names, run a query such as this:

```

VMART=> select node_name from nodes;
  node_name
-----
v_vmart_node0001
v_vmart_node0002
v_vmart_node0003
(3 rows)

```

-or-

To run `Admintools` from the command line, enter a command such as this for the VMart database:

```
$ /opt/vertica/bin/admintools -t node_map -d VMART
DATABASE | NODENAME          | HOSTNAME
-----|-----|-----
VMART    | v_vmart_node0001    | 192.168.223.xx
VMART    | v_vmart_node0002    | 192.168.223.yy
VMART    | v_vmart_node0003    | 192.168.223.zz
```

Configuring the Target Cluster

Configure the target to allow the source database to connect to it and restore the database. The target cluster must:

- Have the same number of nodes the source cluster.
- Have a database with the same name as the source database. The target database can be completely empty.
- Have the same node names as the source cluster. The nodes names listed in the [NODES](#) system tables on both clusters must match.
- Be accessible from the source cluster.
- Have the same **database administrator** account, and all nodes must allow a database administrator of the source cluster to login through SSH without a password.

Note: Having passwordless access *within* the cluster is not the same as having passwordless access *between* clusters. The SSH ID of the administrator account on the source cluster is likely not the same as the SSH ID of the administrator account on the target cluster. You need to configure each host in the target cluster to accept the SSH authentication of the source cluster. See [Configuring Backup Hosts](#) for more information.

- Have adequate disk space for the `vbr.py --task copycluster` command to complete.

Creating a Configuration File for CopyCluster

You must create a configuration file specifically for copying your database to another cluster. In the configuration file, specify the host names of nodes in the target cluster as the backup hosts. When using the `copycluster` command, the `vbr.py` requires that you define the `backupHost`, but ignores the `backupDir` option, and always stores the data in the catalog and data directories of the target database.

You cannot use an object-level backup with the `copycluster` command. You must use a full database backup.

The following example configuration file is set up to copy a database on a three node cluster (`v_vmart_node0001`, `v_vmart_node0002`, and `v_vmart_node0003`) to another cluster consisting of nodes named `test-host01`, `test-host02`, and `test-host03`.

```

[Misc]
snapshotName = CopyVmart
restorePointLimit = 5
verticaConfig = False
tempDir = /tmp/vbrretryCount = 5
retryDelay = 1
[Database]
dbName = vmart
dbUser = dbadmin
dbPassword = password
dbPromptForPassword = False
[Transmission]
encrypt = False
checksum = False
port_rsync = 50000
bwlimit = 0
[Mapping]
; backupDir is not used for cluster copy
v_vmart_node0001= test-host01:/home/dbadmin/backups
v_vmart_node0002= test-host02:/home/dbadmin/backups
v_vmart_node0003= test-host03:/home/dbadmin/backups

```

Copying the Database

The target cluster must be stopped before you invoke copycluster.

To copy the cluster, run `vbr.py` from a node in the source database using the database administrator account, passing the `--task copycluster --config-file CopyVmart.ini` command.

The following example demonstrates copying a cluster using a configuration file located in the current directory.

```

> vbr.py --config-file CopyVmart.ini --task copyclusterCopying...
1871652633 out of 1871652633, 100%
All child processes terminated successfully.
copycluster done!

```


Backup and Restore Utility Reference

This section provides reference information about both the `vbr.py` utility commands, and its associated configuration file parameters.

VBR Utility Reference

The HP Verticavbr.py utility lets you back up and restore either the full database, or one or more schema and table objects of interest. You can also copy a cluster and list backups you created previously. The utility is located in the HP Vertica binary directory (`/opt/vertica/bin/vbr.py` on most installations).

Syntax

```
/opt/vertica/bin/vbr.py { command }
... [ --archive file ]
... [ --config-file file ]
... [ --debug <em>level</em> ]
... [ --nodes node1 [, node2, ...] ]
... [ --showconfig ]
```

Where *command* is one of the following:

| Full Command | Short Command | Description |
|--|-----------------|--|
| <code>--help</code> | <code>-h</code> | Shows a brief usage guide for the command. |
| <code>--setupconfig</code> | | Asks a series of questions and generates a configuration file. See Configuring the Backup Script for details. |
| <code>--task {backup copycluster listbackup restore }</code> | <code>-t</code> | Performs the specified task: <ul style="list-style-type: none"> <code>backup</code> creates a full-database, or object-level backup, depending on what you have specified in the configuration file <code>copycluster</code> copies the database to another HP Vertica cluster. <code>listbackup</code> displays the existing backups associated with the configuration file you supply. Use the information in this display to get the name of a snapshot you want to restore. See Restoring Full Database Backups, and Restoring Object-Level Backups. <code>restore</code> Restores a full- or object-level database backup. Requires a configuration file. |

Parameters

| Parameter | Description |
|----------------------------------|---|
| <code>--archive file</code> | <p>Used with <code>--task backup</code> or <code>--task restore</code> commands, specifies the name of the archive backup to create (backup) or restore. Use this option with the <code>restore</code> command when you have saved more than one restore point, using a command such as:</p> <pre>> vbr.py --task restore --config-file myconfig.ini --archive=snapDB20111114_205841</pre> <p>See the <code>restorePointLimit</code> parameter in Backup Configuration Options for details of saving multiple backups.</p> |
| <code>--config-file file</code> | <p>Indicates the configuration file to use. If this file does not exist, an error occurs and the utility cannot continue. The file parameter can be absolute or relative path to the location from which you start the backup utility.</p> |
| <code>--nodes node1[,...]</code> | <p>Specifies the node or nodes (in a comma-separated list if more than one node exists), on which to perform a <code>vbr.py</code> task. The nodes in the list are the same names in the Mapping section of the configuration file. See VBR Configuration File Reference for details.</p> <p>Caution: If you create a backup for only some nodes in the cluster, you are creating a partial database backup. Partial backups can result in lost data if not used correctly. Do not try to restore the entire database cluster from a partial database backup created from a subset of the nodes.</p> |
| <code>--debug level</code> | <p>Indicates the level of debugging messages (from 0 to 3) that the <code>vbr.py</code> utility provides. Level 3 has the most verbose debugging messages, while level 0 supplies no messages. The default value when running the utility is level 0 (no output).</p> |
| <code>--showconfig</code> | <p>The configuration values being used to perform the task. The parameters are shown in a raw JSON format before <code>vbr.py</code> starts processing.</p> |

VBR Configuration File Reference

The configuration file options are grouped into sections within the configuration file. The following tables describe each parameter section.

[Misc] Miscellaneous Settings

This section collects basic settings, including the name and location of the backup. The section also indicates whether you are keeping more than a single backup file, as specified by the (`restorePointLimit` parameter).

| Parameter | Default | Configuration question and Description |
|-------------------|------------------|--|
| snapshotName | snapshotName | <p>Specifies the name of the top-level directory <code>vbr.py</code> creates for the full or object-level backup. A <code>snapshotName</code> can include only alphanumeric characters, including:</p> <ul style="list-style-type: none"> • a — z • A — Z • 0 — 9 • period (.) • hyphen (-) • underscore (_) |
| tempDir | /tmp | <p>Specifies an absolute path to a temporary storage area on the cluster nodes. The <code>vbr.py</code> utility uses this directory as a temporary location while it is copying files from the source cluster node to the destination backup location. See Configuring Backup Hosts for further information.</p> <p>Note: The <code>tmp</code> path must be the same on all nodes in the cluster.</p> |
| verticaBinDir | /opt/vertica/bin | Specifies a full path to the HP Vertica binary directory, if the path is something other than the default. |
| verticaConfig | False | Indicates whether the HP Vertica configuration file is included in the backup, in addition to the database data. See Creating vbr.py Configuration Files . |
| restorePointLimit | 1 | <p>Specifies the number of backups to retain. For example, if you set <code>restorePointLimit=3</code>, there will be a maximum of three backups, in addition to the most recent backup. By default, HP Vertica maintains a single archive. Saving multiple backups lets you back up incrementally. Enter a value from 1 - 99.</p> <p>If you set <code>restorePointLimit</code> to more than 1, you can save multiple archive snapshots to the same location, with the benefit that storage common to multiple snapshots is shared (through hard links). In this case, each snapshot begins with the same prefix but has a unique time and date suffix, such as follows:</p> <pre>mynsnapshot_archive20111111_205841</pre> |

| Parameter | Default | Configuration question and Description |
|------------|---------|---|
| objects | None | <p>Specifies whether <code>vbr.py</code> creates a full or object-level backup. If you do not specify any objects, <code>vbr.py</code> creates a full backup. Otherwise, specify the object names (schemas or tables) to include in a backup. To enter more than one object, enter multiple names in a comma-separated list.</p> <p>Object names can include UTF-8 alphanumeric characters (as for the <code>snapshotName</code> parameter, above). Object names cannot include escape characters, single quote (') or double quote (") characters.</p> <p>To use non-alphanumeric characters, use a backslash (\) followed by a hex value. For instance, if the table name is a <code>my table</code> (<code>my</code> followed by a space character, then <code>table</code>), enter the object name as follows:</p> <pre>objects=my\20table</pre> |
| overwrite | True | <p>Specifies whether to overwrite an object of the same name when restoring a schema- or table-level snapshot. This parameter is related only to schema or table snapshots. By default, conflicting objects are overwritten when you restore a snapshot and an conflict between object IDs (OIDs) occurs. To prevent the snapshot from overwriting the schema or table while restoring the snapshot, set this parameter to <code>false</code> in the related object-specific snapshot configuration file.</p> <p>NOTE: This parameter is not included in the configuration file you create, even if you specify one or more values for the preceding <code>objects</code> parameter, and the default value is in use when restoring a snapshot. To change the <code>overwrite</code> default value, edit your configuration file before restoring the snapshot, and enter <code>overwrite = false</code> in the file's [MISC] section.</p> |
| retryCount | 2 | <p>Indicates the number of times the backup operation attempts to complete execution after an error occurs. If the failure continues to occur after the number of retry attempts, the utility reports an error and stops processing.</p> |
| retryDelay | 1 | <p>Defines the number of seconds to wait between backup retry attempts in the event of a failure.</p> |

[Database] Database Access Settings

Sets options for accessing the database.

| Parameter | Default | Description |
|---------------------|-------------------|--|
| dbName | N/A | Specifies the name of the database to back up. If you do not supply a database name, the <code>vbr.py</code> utility selects the current database to back up. HP Vertica recommends that you provide a database name. |
| dbUser | Current user name | Identifies the login name of the person who can run <code>vbr.py</code> to create a snapshot, or perform other tasks. The <code>vbr.py</code> utility obtains this information automatically as the current user of the person who invoked the <code>--setupconfig</code> command. You must be logged on as the database administrator to back up the database. |
| dbPromptForPassword | True | Controls whether the utility prompts for a password. If you set this parameter to <code>False</code> (indicating no prompt at run time), then you must also enter the database administrator password in the <code>dbPassword</code> parameter. If you do not supply a password in the configuration file, the utility prompts for one at run time. |
| dbPassword | None | Identifies the database administrator's password. Enter a password if you set <code>dbPromptForPassword</code> to <code>False</code> , so that you will not be prompted at runtime, and the utility needs no further intervention. The <code>vbr.py</code> utility saves your password as plain text. Do not include a password unless you are confident that no unauthorized personnel have access to the <code>vbr.py</code> configuration file. Note: You cannot enter an empty string for the <code>dbPassword</code> in the configuration file. If a superuser's password consists of an empty string (not recommended), you must set the <code>dbPromptForPassword</code> parameter to <code>True</code> , leave the <code>dbPassword</code> option blank, and enter the empty string at the prompt each time you run the backup utility. |

[Transmission] Data Transmission During Backup Process

Sets options for transmitting the data when using backup hosts.

| Parameter | Default | Description |
|-----------------|---------|---|
| encrypt | False | <p>Controls whether the transmitted data is encrypted while it is being copied to the target backup location. Choose this option if you are performing a backup over an untrusted network (for example, backing up to a remote host across the Internet).</p> <p>Note: Encrypting data transmission causes significant processing overhead and slows transfer. One of the processor cores of each database node is consumed during the encryption process. Use this option only if you are concerned about the security of the network used when transmitting backup data.</p> |
| checksum | False | <p>Controls whether the <code>vbr.py</code> utility has <code>rsync</code> use the <code>md5</code> checksum to determine whether files are identical before and after network transmission. By default, <code>rsync</code> does not perform checksum. Instead, it performs minimal file checking, confirming that the file size and time of last modification are identical before and after transmission.</p> <p>Note: Calculating checksum values increases processor usage during the backup process. For more details, see Wikipedia rsync.</p> |
| port_rsync | 50000 | Changes the default port number for the <code>rsync</code> protocol. Change this value if the default <code>rsync</code> port is in use on your cluster, or you need <code>rsync</code> to use another port to avoid a firewall restriction. |
| bwlimit | 0 | Indicates the transfer bandwidth limit (in KBs per second) for data transmission on each node in the database. If you do not specify a value, then no limit is imposed. |
| hardLinkLocal | False | Creates a full- or object-level backup using hard file links on the local file system, rather than copying database files to a remote backup host. Add this configuration parameter manually to the Transaction section of the configuration file, as described in Configuring the Hard Link Local Parameter . |
| ssh_port_backup | 22 | <p>Overrides the default SSH port setting (22) for the backup hosts. Enter the required SSH port for your site.</p> <p>Changing the default SSH port is supported only when using the backup and restore tasks. Using a non-default SSH port with the <code>copycluster</code> task is not supported.</p> <p>NOTE: This parameter is not included in the configuration file automatically. See Configuring Backup Hosts to enter the parameter manually.</p> |

[Mapping]

There is one [Mapping] section for all of the nodes in your database cluster. The section must exist in your configuration file, since it specifies all database nodes being included in the backup, along

with the backup host and directory for each node.

NOTE: Previous configuration files used separate mapping sections for each database node. While `vbr.py` continues to support that format, you cannot combine formats. If you edit an existing configuration file to add a Mapping in the current style, you must combine information from all existing Mappings into the new section. Alternatively, you can use `vbr.py` with the `--task=create-config` option to generate a new configuration file, as described in [Creating vbr.py Configuration Files](#).

In the following example, the Mapping section indicates a single node to be backed up (`v_vmart_node0001`). The node is assigned to the backup host (`127.0.0.1`), and the backup directory (`/home/dbadmin/backups`). Notice that, even though this example is for a single node cluster, and the backup host and the database node are the same system, you specify them differently. The backup host and directory specifications use a colon (`:`) as a separator:

```
[Mapping]
v_vmart_node0001 = 127.0.0.1:/home/dbadmin/backups
```

While the configuration file `[Mapping]` section no longer uses named parameters, the elements of the simplified format continue to represent the following parameters:

```
dbNode = backupHost:backupDir
```

| Parameter | Default | Description |
|------------|---------|---|
| backupHost | None | Indicates the target host name or IP address on which to store this node's backup. The backupHost name is different from dbNode, also described in this table. When you create a configuration file to copy the database to another HP Vertica cluster (with the <code>copycluster</code> task), supply the host name of a node in the target cluster. |
| backupDir | None | Identifies the full path to the directory on the backup host or node where the backup will be stored. This directory must already exist when you run the utility with the <code>--task backup</code> option, and must be writable by the user account used to run the backup utility. This setting is not used for the <code>copycluster</code> command. |
| dbNode | None | The name of the database node, as recognized by HP Vertica. This is not the node's host name, but rather the name HP Vertica uses internally to identify the node, usually in the form of: <code>v_databasename_node00xx</code> To find database node names in your cluster, query the <code>node_name</code> column of the NODES system table. |

Recovering the Database

Recovering a database can consist of any of the following:

- [Restarting HP Vertica on a host](#)
- [Restarting the Database](#)
- [Recovering the Cluster From a Backup](#)
- [Replacing Failed Disks](#)
- [Copying the Database to Another Cluster](#)
- [Exporting a Catalog](#) for support purposes.

You can [monitor a recovery](#) in progress by viewing log messages posted to the `vertica.log` file on each host.

See Also

- [Failure Recovery](#)

Failure Recovery

Recovery is the process of restoring the database to a fully functional state after one or more nodes in the system has experienced a software- or hardware-related failure. HP Vertica recovers nodes by querying replicas of the data stored on other nodes. For example, a hardware failure can cause a node to lose database objects or to miss changes made to the database (INSERTs, UPDATEs, and so on) while offline. When the node comes back online, it recovers lost objects and catches up with changes by querying the other nodes.

K-safety is a measure of fault tolerance in the database cluster. The value K represents the number of replicas of the data in the database that exist in the database cluster. These replicas allow other nodes to take over for failed nodes, allowing the database to continue running while still ensuring data integrity. If more than K nodes in the database fail, some of the data in the database may become unavailable. In that case, the database is considered unsafe and automatically shuts down.

It is possible for an HP Vertica database to have more than K nodes fail and still continue running safely, because the database continues to run as long as every data segment is available on at least one functioning cluster node. Potentially, up to half the nodes in a database with a K-safety level of 1 could fail without causing the database to shut down. As long as the data on each failed node is available from another active node, the database continues to run.

Note: If half or more of the nodes in the database cluster fail, the database will automatically

shut down even if all of the data in the database is technically available from replicas. This behavior prevents issues due to network partitioning.

In HP Vertica, the value of K can be zero (0), one (1), or two (2). The physical schema design must meet certain requirements. To create designs that are K-safe, HP recommends using the **Database Designer**.

Note : You can monitor the cluster state through the **View Database Cluster** state menu option.

Recovery Scenarios

Recovery comes into play when a node or the database is started. Depending upon how the node or database was shut down, and how it is restored, there are three possibilities for a K-Safe database:

- **Recovery of failed nodes:** One or more nodes have failed, but the database continues to run since the remaining nodes in the database are able to fill in for the failed nodes. The failed nodes can be restarted through the [Administration Tools](#) using the [Restart HP Vertica on host](#) option. The nodes being restarted have a RECOVERING status while they rebuild some of the data from the remaining nodes. Once rebuilding is finished, the nodes transition to an UP status. The database can continue to commit transactions during the recovery process, except for a short period at the end of the recovery process.
- **Recovery after a Clean Shutdown:** The database had been shut down cleanly via the Administration Tools Stop Database option. In this case, the database should be restarted using the [Start Database](#) option. Upon restart all nodes that were 'UP' at the time of shutdown immediately transition to 'UP'. It is possible that at the time of shutdown, the database had one or more failed nodes. If these nodes are now available, they go through the 'RECOVERING' state as described in 'Recovery of failed nodes' case above.
- **Recovery after an Unclean Shutdown (Manual Recovery):** The database was not shut down cleanly, which means that the database became unsafe due to a failure. In this case, the database possibly did not write all the data from the WOS to disk. There are several reasons for unclean shutdowns, such as:
 - A critical node failed, leaving part of the database's data unavailable.
 - A site-wide event, such as a power failure that causes all nodes to reboot.
 - HP Vertica processes on the nodes exited due to a software or hardware failure.

When the database is started through the Administration Tools Start Database option, recovery determines that a normal startup is not possible. It goes on to determine a point in time in which the data was consistent on all nodes. This is called the **Last Good Epoch**. As part of Start Database processing, the administrator is prompted to accept recovery with the suggested epoch. If accepted, the database recovers and any data changes made after the Last Good Epoch are lost. If not accepted, startup is aborted and the database is not started on any of the nodes.

Instead of accepting the given epoch, the administrator can instead choose to [recover from a backup](#) or select an epoch for an even earlier point using the Roll Back Database to Last Good Epoch option in the Administration Tools Advanced Menu. This is useful in special situations, for example if the failure occurs during a batch of loads, for which it is easier to go back to the beginning of the batch, rather than starting in the middle, even though some of the work must be repeated. In most scenarios, it is sufficient and recommended to accept the given epoch.

Notes

- In HP Vertica 5.0, manual recovery is possible as long as the nodes that are being started can supply all of the partition segments in the database. This means that more than K nodes can remain down at startup, and the database can still successfully start as long as all of the data is available from the remaining nodes in the cluster.
- In HP Vertica 4.1, the default for the `HistoryRetentionTime` configuration parameter changed to 0, which means that HP Vertica only keeps historical data when nodes are down. This default setting effectively prevents the use of the **Administration Tools** 'Roll Back Database to Last Good Epoch' option because the **AHM** remains close to the current epoch and a rollback is not permitted to an epoch prior to the AHM. If you rely on the Roll Back option to remove recently loaded data, consider setting a day-wide window for removing loaded data; for example:

```
=> SELECT SET_CONFIG_PARAMETER ('HistoryRetentionTime', '86400');
```

See [Epoch Management Parameters](#) in the Administrator's Guide.

- Starting in 4.0, manual recovery is possible even if up to K nodes are out of commission; for example, physically removed for repair or not reachable at the time of recovery. Once the nodes are back in commission, they recover and rejoin the cluster, as described in the "Recovery after failure of up to K nodes" section above.
- **IMPORTANT:** When a node is down, it can take a full minute or more for the HP Vertica processes to time out during its attempt to form a cluster when manual recovery is needed. Wait approximately one minute until the system returns the manual recovery prompt. Do not press CTRL-C during database startup.

See Also

- [High Availability and Recovery](#)

Restarting HP Vertica on a Host

When one node in a running database cluster fails, or if any files from the catalog or data directories are lost from any one of the nodes, you can check the status of failed nodes using either the Administration Tools or the Management Console.

Restarting HP Vertica on a Host Using the Administration Tools

1. Run **Administration Tools**.
2. From the Main Menu, select **Restart HP Vertica on Host** and click **OK**.
3. Select the database host you want to recover and click **OK**.

Note: You might see additional nodes in the list, which are used internally by the Administration Tools. You can safely ignore these nodes.

4. Verify recovery state by selecting **View Database Cluster State** from the **Main Menu**.

After the database is fully recovered, you can check the status at any time by selecting **View Database Cluster State** from the Administration Tools **Main Menu**.

Restarting HP Vertica on a Host Using the Management Console

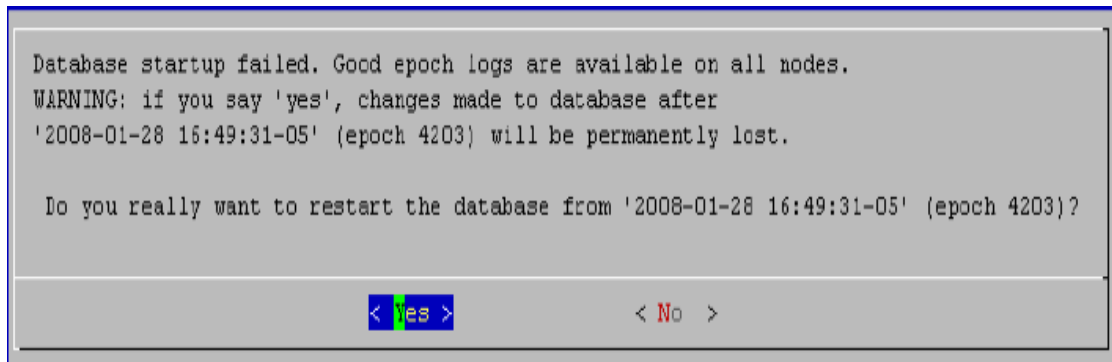
1. Connect to a cluster node (or the host on which MC is installed).
2. Open a browser and [connect to MC](#) as an MC administrator.
3. On the MC **Home** page, double-click the running database under the **Recent Databases** section.
4. Within the **Overview** page, look at the node status under the Database sub-section and see if all nodes are up. The status will indicate how many nodes are up, critical, down, recovering, or other.
5. If a node is down, click **Manage** at the bottom of the page and inspect the graph. A failed node will appear in red.
6. Click the failed node to select it and in the Node List, click the **Start node** button.

Restarting the Database

If you lose the HP Vertica process on more than one node (for example, due to power loss), or if the servers are shut down without properly shutting down the HP Vertica database first, the database cluster indicates that it did not shut down gracefully the next time you start it.

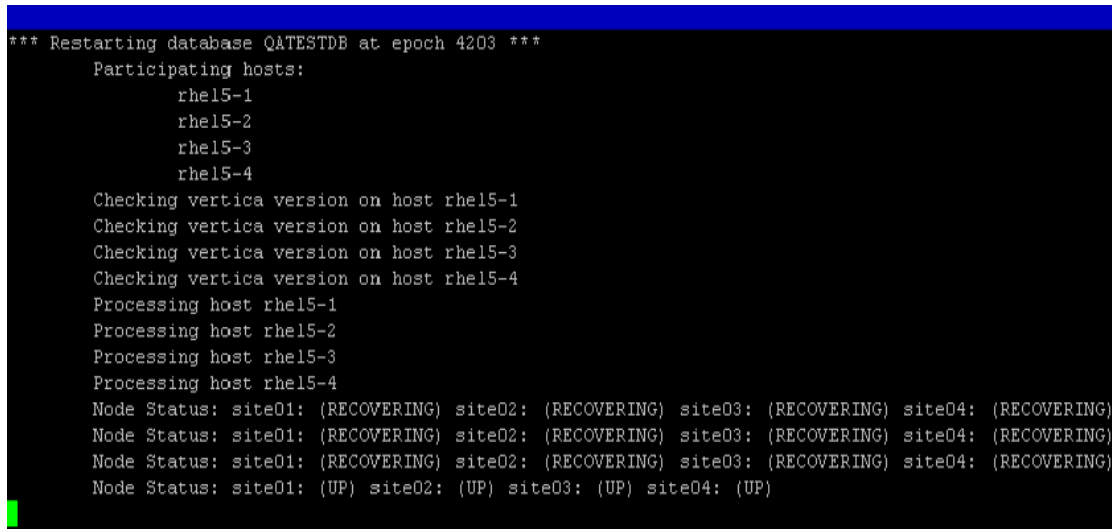
The database automatically detects when the cluster was last in a consistent state and then shuts down, at which point an administrator can restart it.

From the Main Menu in the **Administration Tools**:



Caution: If you do not want to start from the last good epoch, you may instead restore the data from a backup and attempt to restart the database. For this to be useful, the backup must be more current than the last good epoch.

HP Vertica continues to initialize and recover all data prior to the last good epoch.



If recovery takes more than a minute, you are prompted to answer <Yes> or <No> to "Do you want to continue waiting?"

When all the nodes' status have changed to RECOVERING or UP, selecting <No> lets you exit this screen and monitor progress via the Administration Tools Main Menu. Selecting <Yes> continues to display the database recovery window.

Note: Be sure to reload any data that was added after the last good epoch date to which you have recovered.

Recovering the Cluster From a Backup

To recover a cluster from a backup, refer to the following topics in this guide:

- [Backing Up the Database \(vbr.py\)](#)
- [Restoring Full Database Backups](#)

Monitoring Recovery

There are several ways to monitor database recovery:

- Log files on each host
- Admintools (View Database Cluster State)
- System tables

This section describes the different ways to monitor recovery.

Viewing Log Files on Each Node

During database recovery, HP Vertica adds logging information to the `vertica.log` on each host. Each message is identified with a `[Recover]` string.

Use the `tail` command to monitor recovery progress by viewing the relevant status messages, as follows.

```
$ tail -f catalog-path/database-name/node-name_catalog/vertica.log
01/23/08 10:35:31 thr:Recover:0x2a98700970 [Recover] <INFO> Changing host node01
startup state from INITIALIZING to RECOVERING
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Recovering to specified
epoch 0x120b6
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running 1 split queries
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running query: ALTER
PROJECTION proj_tradesquotes_0 SPLIT node01 FROM 73911;
```

Viewing the Cluster State and Recover Status

Use the `admintools view_cluster` tool from the command line to see the cluster state:

```
$ /opt/vertica/bin/admintools -t view_cluster
DB | Host | State
-----+-----+-----
<data_base> | 112.17.31.10 | RECOVERING
```

```
<data_base> | 112.17.31.11 | UP  
<data_base> | 112.17.31.12 | UP  
<data_base> | 112.17.31.17 | UP
```

Using System Tables to Monitor Recovery

Use the following system tables to monitor recover:

- [RECOVERY_STATUS](#)
- [PROJECTION_RECOVERIES](#)

Specifically, the `recovery_status` system table includes information about the node that is recovering, the epoch being recovered, the current recovery phase, and running status:

```
=>select node_name, recover_epoch, recovery_phase, current_completed, is_running from reco  
very_status;  
node_name | recover_epoch | recovery_phase | current_completed | is_running  
-----+-----+-----+-----+-----  
node01 | | | 0 | f  
node02 | 0 | historical pass 1 | 0 | t  
node03 | 1 | current | 0 | f
```

The `projection_recoveries` system table maintains history of projection recoveries. To check the recovery status, you can summarize the data for the recovering node, and run the same query several times to see if the counts change. Differing counts indicate that the recovery is working and in the process of recovering all missing data.

```
=> select node_name, status , progress from projection_recoveries;  
node_name | status | progress  
-----+-----+-----  
v_<data_base>_node0001 | running | 61
```

To see a single record from the `projection_recoveries` system table, add `limit 1` to the query.

Monitoring Cluster Status After Recovery

When recovery has completed:

1. Launch Administration Tools.
2. From the Main Menu, select **View Database Cluster State** and click **OK**.

The utility reports your node's status as UP.

Note: You can also monitor the state of your database nodes on the Management Console Overview page under the Database section, which tells you the number of nodes that are up, critical, recovering, or down. To get node-specific information, click Manage at the bottom of

the page.

See Also

- [Monitoring HP Vertica](#)

Exporting a Catalog

When you export a catalog you can quickly move a catalog to another cluster. Exporting a catalog transfers schemas, tables, constraints, projections, and views. System tables are not exported.

Exporting catalogs can also be useful for support purposes.

See the [EXPORT_CATALOG](#) function in the SQL Reference Manual for details.

Best Practices for Disaster Recovery

To protect your database from site failures caused by catastrophic disasters, maintain an off-site replica of your database to provide a standby. In case of disaster, you can switch database users over to the standby database. The amount of data loss between a disaster and fail over to the offsite replica depends on how frequently you save a full database backup.

The solution to employ for disaster recover depends upon two factors that you must determine for your application:

- **Recovery point objective (RPO):** How much data loss can your organization tolerate upon a disaster recovery?
- **Recovery time objective (RTO):** How quickly do you need to recover the database following a disaster?

Depending on your RPO and RTO, HP Vertica recommends choosing from the following solutions:

1. **Dual-load:** During each load process for the database, simultaneously load a second database. You can achieve this easily with off-the-shelf ETL software.
2. **Periodic Incremental Backups:** Use the procedure described in [Copying the Database to Another Cluster](#) to periodically copy the data to the target database. Remember that the script copies only files that have changed.
3. **Replication solutions provided by Storage Vendors:** If you are using a SAN, evaluate your storage vendor's replication (SRDF) solutions.

The following table summarizes the RPO, RTO, and the pros and cons of each approach:

| | Dual Load | Periodic Incremental | Storage Replication |
|-----|-----------------------|-----------------------|-----------------------|
| RPO | Up to the minute data | Up to the last backup | Recover to the minute |

| | Dual Load | Periodic Incremental | Storage Replication |
|------|---|---|---|
| RTO | Available at all times | Available except when backup in progress | Available at all times |
| Pros | <ul style="list-style-type: none"> Standby database can have different configuration Can use the standby database for queries | <ul style="list-style-type: none"> Built-in scripts High performance due to compressed file transfers | Transparent to the database |
| Cons | <ul style="list-style-type: none"> Possibly incur additional ETL licenses Requires application logic to handle errors | Need identical standby system | <ul style="list-style-type: none"> More expensive Media corruptions are also replicated |

Monitoring HP Vertica

This section describes some of the ways in which you can monitor the health of your HP Vertica database.

Monitoring Log Files

When a Database Is Running

When an HP Vertica database is running, each **node** in the **cluster** writes messages into a file named `vertica.log`. For example, the **Tuple Mover** and the transaction manager write INFO messages into `vertica.log` at specific intervals even when there is no **WOS** activity.

To monitor a running database in real time:

1. Log in to the database administrator account on any or all nodes in the cluster.
2. In a terminal window (such as `vsq`) enter:

```
$ tail -f catalog-path/database-name/node-name_catalog/vertica.log
```

| | |
|----------------------|---|
| <i>catalog-path</i> | The catalog pathname specified when you created the database. See Creating a Database in the Administrator's Guide. |
| <i>database-name</i> | The database name (case sensitive) |
| <i>node-name</i> | The node name, as specified in the database definition. See Viewing a Database in the Administrator's Guide. |

When the Database / Node Is Starting up

During startup before the `vertica.log` has been initialized to write messages, each node in the cluster writes messages into a file named `dbLog`. This log is useful to diagnose situations where database fails to start before it can write messages into `vertica.log`. The `dblog` can be found at the following path, using `catalog-path` and `database-name` as described above:

```
catalog-path/database-name/dbLog
```

See Also

- [Rotating Log Files](#)

Rotating Log Files

The `logrotate` utility, which is included with most Linux distributions, helps simplify log file administration on systems that generate many log files. Logrotate allows for automatic rotation, compression, removal, and mailing of log files and can be configured to perform these tasks at specific intervals or when the log file reaches a particular size.

If `logrotate` is present when HP Vertica is installed (which is typical for most Linux distributions), then HP Vertica automatically sets `logrotate` to look for configuration files in the `/opt/vertica/config/logrotate` directory. The utility also creates the file `vertica` in the `/etc/logrotate.d/` directory, which includes the line:

```
include /opt/vertica/config/logrotate
```

If `logrotate` is not present but installed at a later time, either reinstall the HP Vertica RPM on every node in the cluster or add a file in the `/etc/logrotate.d/` directory that instructs `logrotate` to include the `logrotate` directory contents. For example:

1. Create the file `/etc/logrotate.d/vertica`.
2. Add the following line:

```
include /opt/vertica/config/logrotate
```

When a database is created, HP Vertica creates database-specific `logrotate` configurations which are used by the `logrotate` utility. For example, a file `/opt/vertica/config/logrotate/<dbname>` is created for each individual database.

Using Administration Tools Logrotate Utility

The administration tools provide a `logrotate` option to help configure `logrotate` scripts for a database and to distribute it across the cluster. Only a few basic options are supported - how often to rotate logs, how large the log can get before rotation and how long to keep the logs. For other options, you can manually create `logrotate` scripts as described later in this topic.

Example:

The following example sets up log rotation on a weekly schedule and keeps for 3 months (12 logs).

```
$ admintools -t logrotate -d <dbname> -r weekly -k 12
```

See [Writing Administration Tools Scripts](#) for full usage description.

Manually Rotating Logs

To perform manual log rotation, use the following procedure to implement a custom log rotation process. No log messages are lost during the procedure.

1. Rename or archive the *vertica.log* file that is produced. For example:

```
$ mv vertica.log vertica.log.1
```

2. Send the HP Vertica process the USR1 signal. For example:

```
$ killall -USR1 vertica
```

or

```
$ ps -ef | grep -i vertica  
$ kill -USR1 process-id
```

Manually Creating Logrotate Scripts

If your needs are not met by the administration tools logrotate utility, you may create your own scripts. The following script is an example:

```
/mydb/site01_catalog/vertica.log {  
    # rotate weekly  
    weekly  
    # and keep for 52 weeks  
    rotate 52  
    # no complaining if vertica did not start yet  
    missingok  
    # compress log after rotation  
    compress  
    # no creating a new empty log, vertica will do that  
    nocreate  
    # if set, only rotates when log size is greater than X  
    size 10M  
    # delete files after 90 days (not all logrotate pkgs support this keyword)  
    # maxage 90  
    # signal vertica to reopen and create the log  
    postrotate  
        kill -USR1 `head -1 /mydb/site01_catalog/vertica.pid 2> /dev/null` 2> /dev/null ||  
true  
    endscript  
}
```

The following script is an example of the typical default setting for the dbLog file:

```
/mydb/dbLog {  
    # rotate weekly  
    weekly  
    # and keep for 52 weeks  
    rotate 52  
    # no complaining if vertica did not start yet  
    missingok
```

```
# compress log after rotation
compress
# this log is stdout, so rotate by copying it aside and truncating
copytruncate
}
```

For details about additional settings, issue the `man logrotate` command.

See Also

- [Monitoring Log Files](#)

Monitoring Process Status (ps)

You can use `ps` to monitor the database and Spread processes running on each node in the cluster. For example:

```
$ ps aux | grep /opt/vertica/bin/vertica  
$ ps aux | grep /opt/vertica/sbin/spread
```

You should see one HP Vertica process and one Spread process on each node for common configurations. To monitor Administration Tools and connector processes:

```
$ ps aux | grep vertica
```

There can be many connection processes but only one Administration Tools process.

Monitoring Linux Resource Usage

You should monitor system resource usage on any or all nodes in the cluster. You can use System Activity Reporting (SAR) to monitor resource usage.

HP recommends that you install `pstack` and `sysstat` to help monitor Linux resources. The `SYSSTAT` package contains utilities for monitoring system performance and usage activity, such as `sar`, as well as tools you can schedule via `cron` to collect performance and activity data. See the `SYSSTAT` Web page for details.

The `pstack` utility lets you print a stack trace of a running process. See the `PSTACK` man page for details.

1. Log in to the database administrator account on any node.
2. Run the `top` utility

```
$ top
```

A high CPU percentage in `top` indicates that HP Vertica is CPU-bound. For example:

```
top - 11:44:28 up 53 days, 23:47, 9 users, load average: 0.91, 0.97, 0.81
Tasks: 123 total, 1 running, 122 sleeping, 0 stopped, 0 zombie
Cpu(s): 26.9%us, 1.3%sy, 0.0%ni, 71.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 4053136 total, 3882020k used, 171116 free, 407688 buffers
Swap: 4192956 total, 176k used, 4192780 free, 1526436 cached
  PID USER      PR  NI  VIRT  RES  SHR  S %CPU %MEM    TIME+  COMMAND
13703 dbadmin    1   0 1374m 678m 55m  S 99.9 17.1   6:21.70 vertica
 2606 root       16   0 32152  11m 2508  S  1.0  0.3   0:16.97 X
     1 root       16   0  4748   52 456  S  0.0  0.0   0:01.51 init
     2 root       RT  -5     0     0   0  S  0.0  0.0   0:04.92 migration/0
     3 root       34  19     0     0   0  S  0.0  0.0   0:11.75 ksoftirqd/0
...
```

Some possible reasons for high CPU usage are:

- The **Tuple Mover** runs automatically and thus consumes CPU time even if there are no connections to the database.
- The `pdflush` process (a set of worker threads for writing back dirty filesystem data) is consuming a great deal of CPU time, possibly driving up the load. Adding RAM appears to make the problem worse. Log in to root and change the Linux parameter `swappiness` to 0.

```
# echo 0 > /proc/sys/vm/swappiness
```

- Some information sources:

Red Hat

Indiana University Unix Systems Support Group

3. Run the **iostat** utility. A high idle time in **top** at the same time as a high rate of blocks read in **iostat** indicates that HP Vertica is disk-bound. For example:

```
$ /usr/bin/iostat
Linux 2.6.18-164.el5 (qa01)      02/05/2011
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.77    2.32   0.76   0.68   0.00   95.47

Device:            tps    Blk_read/s    Blk_wrtn/s    Blk_read    Blk_wrtn
hda                  0.37         3.40         10.37       2117723     6464640
sda                  0.46         1.94         18.96       1208130     11816472
sdb                  0.26         1.79         15.69       1114792     9781840
sdc                  0.24         1.80         16.06       1119304     10010328
sdd                  0.22         1.79         15.52       1117472     9676200
md0                  8.37         7.31         66.23       4554834     41284840
```

Monitoring Disk Space Usage

You can use these system tables to monitor disk space usage on your cluster:

| System table | Description |
|------------------------------------|--|
| DISK_STORAGE | Monitors the amount of disk storage used by the database on each node. |
| COLUMN_STORAGE | Monitors the amount of disk storage used by each column of each projection on each node. |
| PROJECTION_STORAGE | Monitors the amount of disk storage used by each projection on each node. |

See [Managing Disk Space](#) for more information.

Monitoring Database Size for License Compliance

If your HP Vertica license includes a raw data storage allowance, you should regularly monitor the size of your database. This monitoring allows you to plan to either schedule deleting old data to keep your database in compliance with your license agreement, or budget for a license upgrade to allow for the continued growth of your database.

Viewing the Current License State

HP Vertica periodically runs an audit of the database size to verify that your database remains compliant with your license. You can view the results of the most recent audit by calling the [GET_COMPLIANCE_STATUS](#) function.

```
GET_COMPLIANCE_STATUS
-----
Raw Data Size: 2.00GB +/- 0.003GB
License Size : 4.000GB
Utilization  : 50%
Audit Time   : 2011-03-09 09:54:09.538704+00
Compliance Status : The database is in compliance with respect to raw data size.
License End Date: 04/06/2011
Days Remaining: 28.59
(1 row)
```

Periodically running GET_COMPLIANCE_STATUS to monitor your database's license status is usually enough to ensure that your database remains compliant with your license. If your database begins to near its data allowance, you may want to use the other auditing functions described below to determine where your database is growing and how recent deletes have affected the size of your database.

Manually Running an Audit of the Entire Database

You can trigger HP Vertica's automatic audit of your database at any time using the [AUDIT_LICENSE_SIZE](#) SQL function. This function triggers the same audit that HP Vertica performs periodically. The audit runs in the background, so you need to wait for the audit to complete. You can then view the audit results using [GET_COMPLIANCE_STATUS](#).

An alternative to [AUDIT_LICENSE_SIZE](#) is to use the [AUDIT](#) SQL function to audit the size of your entire database by passing it an empty string. Unlike [AUDIT_LICENSE_SIZE](#), this function operates synchronously, returning when it has estimated the size of the database.

```
=> SELECT AUDIT('');
      AUDIT
-----
      76376696
(1 row)
```

The size of the database is reported in bytes. The [AUDIT](#) function also allows you to control the accuracy of the estimated database size using additional parameters. See the entry for the [AUDIT](#) function in the SQL Reference Manual for full details

Note: HP Vertica does not count the results of the [AUDIT](#) function as an official audit. It takes no license compliance actions based on the results.

Targeted Auditing

If your audits find your database to be unexpectedly large, you may want to find which schemas, tables, or partitions are using the most storage. You can use the [AUDIT](#) function to perform targeted audits of schemas, tables, or partitions by supplying the name of the entity whose size you want to find. For example, to find the size of the `online_sales` schema in the [VMart](#) example database, run the following command:

```
VMart=> SELECT AUDIT('online_sales');
        AUDIT
        -----
        35716504
        (1 row)
```

You can also change the granularity of an audit to report the size of each entity in a larger entity (for example, each table in a schema) by using the granularity argument of the AUDIT function. See the [AUDIT](#) function's entry in the SQL Reference Manual.

Using Management Console to Monitor License Compliance

You can also get information about raw data storage through the Management Console. This information is available in the database **Overview** page, which displays a grid view of the database's overall health.

- The needle in the license meter adjusts to reflect the amount used in megabytes.
- The grace period represents the term portion of the license.
- The Audit button returns the same information as the AUDIT() function in a graphical representation.
- The Details link within the License grid (next to the Audit button) provides historical information about license usage. This page also shows a progress meter of percent used toward your license limit.

Monitoring Shared Node Connections

If you want to monitor which nodes are sharing connections, view the `check.txt` file by issuing the following command at a shell prompt:

```
# watch cat /etc/keepalived/check.txtEvery 2.0s: cat /etc/keepalived/check.txt   Wed Nov
 3 10:02:20 2010
N192168051057
N192168051056
N192168051055
```

The `check.txt` is a file located in the `/etc/keepalived/` directory, and it gets updated when you submit changes to the kernel using `sysctl -p`, described in [Disable the Address Resolution Protocol \(ARP\)](#). For example, the `spread.pl` script (see [Configuring the Directors](#)), writes to the `check.txt` file, which is then modified to include only the remaining nodes in the event of a node failure. Thus, the virtual server knows to stop sending vsql requests to the failed node.

You can also look for messages by issuing the following command at a shell prompt:

```
# tail -f /var/log/messages
```

```
Nov 3 09:21:00 p6 Keepalived: Starting Keepalived v1.1.17 (05/17,2010)Nov 3 09:21:00 p6
Keepalived: Starting Healthcheck child process, pid=32468
Nov 3 09:21:00 p6 Keepalived: Starting VRRP child process, pid=32469
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Using LinkWatch kernel netlink reflector...
Nov 3 09:21:00 p6 Keepalived_vrrp: Using LinkWatch kernel netlink reflector...
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Netlink reflector reports IP 10.10.51.55 ad
ded
Nov 3 09:21:00 p6 Keepalived_vrrp: Netlink reflector reports IP 10.10.51.55 added
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Netlink reflector reports IP 192.168.51.55
added
Nov 3 09:21:00 p6 Keepalived_vrrp: Netlink reflector reports IP 192.168.51.55 added
Nov 3 09:21:00 p6 Keepalived_vrrp: Registering Kernel netlink reflector
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Registering Kernel netlink reflector
Nov 3 09:21:00 p6 Keepalived_vrrp: Registering Kernel netlink command channel
Nov 3 09:21:00 p6 Keepalived_vrrp: Registering gratuitous ARP shared channel
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Registering Kernel netlink command channel
Nov 3 09:21:00 p6 Keepalived_vrrp: Opening file '/etc/keepalived/keepalived.conf'.
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Opening file '/etc/keepalived/keepalived.co
nf'.
Nov 3 09:21:00 p6 Keepalived_vrrp: Configuration is using : 63730 Bytes
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Configuration is using : 16211 Bytes
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Activating healthcheckers for service [10.1
0.51.55:5433]
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Activating healthcheckers for service [10.1
0.51.56:5433]
Nov 3 09:21:00 p6 Keepalived_healthcheckers: Activating healthcheckers for service [10.1
0.51.57:5433]
Nov 3 09:21:00 p6 Keepalived_vrrp: VRRP sockpool: [ifindex(2), proto(112), fd(10,11)]
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Misc check to [10.10.51.56] for [/etc/keepa
lived/check.pl 192.168.51.56] failed.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Removing service [10.10.51.56:5433] from VS
[10.10.51.180:5433]
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connecte
d.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Misc check to [10.10.51.55] for [/etc/keepa
lived/check.pl 192.168.51.55] failed.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Removing service [10.10.51.55:5433] from VS
[10.10.51.180:5433]
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connecte
d.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Misc check to [10.10.51.57] for [/etc/keepa
lived/check.pl 192.168.51.57] failed.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Removing service [10.10.51.57:5433] from VS
[10.10.51.180:5433]
Nov 3 09:21:01 p6 Keepalived_healthcheckers: Remote SMTP server [127.0.0.1:25] connecte
d.
Nov 3 09:21:01 p6 Keepalived_healthcheckers: SMTP alert successfully sent.
Nov 3 09:21:10 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Transition to MASTER STATE
Nov 3 09:21:20 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Entering MASTER STATE
Nov 3 09:21:20 p6 Keepalived_vrrp: VRRP_Instance(VI_1) setting protocol VIPs.
Nov 3 09:21:20 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0 f
or 10.10.51.180
Nov 3 09:21:20 p6 Keepalived_healthcheckers: Netlink reflector reports IP 10.10.51.180 a
dded
Nov 3 09:21:20 p6 Keepalived_vrrp: Remote SMTP server [127.0.0.1:25] connected.
Nov 3 09:21:20 p6 Keepalived_vrrp: Netlink reflector reports IP 10.10.51.180 added
Nov 3 09:21:20 p6 Keepalived_vrrp: SMTP alert successfully sent.
Nov 3 09:21:25 p6 Keepalived_vrrp: VRRP_Instance(VI_1) Sending gratuitous ARPs on eth0
```

for 10.10.51.1

Monitoring Elastic Cluster Rebalancing

HP Vertica includes system tables that can be used to monitor the rebalance status of an elastic cluster and gain general insight to the status of elastic cluster on your nodes.

- The [REBALANCE_TABLE_STATUS](#) table provides general information about a rebalance. It shows, for each table, the amount of data that has been separated, the amount that is currently being separated, and the amount to be separated. It also shows the amount of data transferred, the amount that is currently being transferred, and the remaining amount to be transferred (or an estimate if storage is not separated).

Note: If multiple rebalance methods were used for a single table (for example, the table has unsegmented and segmented projections), the table may appear multiple times - once for each rebalance method.

- [REBALANCE_PROJECTION_STATUS](#) can be used to gain more insight into the details for a particular projection that is being rebalanced. It provides the same type of information as above, but in terms of a projection instead of a table.

In each table, *separated_percent* and *transferred_percent* can be used to determine overall progress.

Historical Rebalance Information

Historical information about work completed is retained, so use the predicate "*where is_latest*" to restrict the output to only the most recent or current rebalance activity. The historical data may include information about dropped projections or tables. If a table or projection has been dropped and information about the anchor table is not available, then NULL is displayed for the *table_id* and "<unknown>" is displayed for the *table_name*. Information on dropped tables is still useful, for example, in providing justification for the duration of a task.

Monitoring Parameters

The following table describes the monitoring parameters for configuring HP Vertica.

| Parameters | Description |
|---------------------------|--|
| SnmptTrapDestinationsList | <p>Defines where HP Vertica send traps for SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: none</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SnmptTrapDestinationsList', 'localhost 162 public');</pre> |
| SnmptTrapsEnabled | <p>Enables event trapping for SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SnmptTrapsEnabled', 1);</pre> |
| SnmptTrapEvents | <p>Define which events HP Vertica traps through SNMP. See Configuring Reporting for SNMP.</p> <p>Default Value:Low Disk Space, Read Only File System, Loss of K Safety, Current Fault Tolerance at Critical Level, Too Many ROS Containers, WOS Over Flow, Node State Change, Recovery Failure, and Stale Checkpoint</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SnmptTrapEvents', 'Low Disk Space, Recovery Failure');</pre> |
| SyslogEnabled | <p>Enables event trapping for syslog. See Configuring Reporting for Syslog.</p> <p>Default Value: 0</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SyslogEnabled', 1);</pre> |

| Parameters | Description |
|----------------|---|
| SyslogEvents | <p>Defines events that generate a syslog entry. See Configuring Reporting for Syslog.</p> <p>Default Value: none</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SyslogEvents', 'Low Disk Space, Recovery Failure');</pre> |
| SyslogFacility | <p>Defines which SyslogFacility HP Vertica uses. See Configuring Reporting for Syslog.</p> <p>Default Value: user</p> <p>Example:</p> <pre>SELECT SET_CONFIG_PARAMETER ('SyslogFacility' , 'ftp');</pre> |

Monitoring Events

To help you monitor your database system, HP Vertica traps and logs significant events that affect database performance and functionality if you do not address their root causes. This section describes where events are logged, the types of events that HP Vertica logs, how to respond to these events, the information that HP Vertica provides for these events, and how to configure event monitoring.

Event Logging Mechanisms

HP Vertica posts events to the following mechanisms:

| Mechanism | Description |
|---------------|---|
| vertica.log | All events are automatically posted to vertica.log. See Monitoring the Log Files . |
| ACTIVE_EVENTS | This SQL system table provides information about all open events. See Using System Tables and ACTIVE_EVENTS . |
| SNMP | To post traps to SNMP, enable global reporting in addition to each individual event you want trapped. See Configuring Event Reporting . |
| Syslog | To log events to syslog, enable event reporting for each individual event you want logged. See Configuring Event Reporting . |

Event Severity Types

Event names are sensitive to case and spaces. HP Vertica logs the following events:

| Event Name | Event Type | Description | Action |
|---|------------|--|---|
| Low Disk Space | 0 | The database is running out of disk space or a disk is failing or there is a I/O hardware failure. | <p>It is imperative that you add more disk space or replace the failing disk or hardware as soon as possible.</p> <p>Check <code>dmesg</code> to see what caused the problem.</p> <p>Also, use the DISK_RESOURCE_REJECTIONS system table to determine the types of disk space requests that are being rejected and the hosts on which they are being rejected. See Managing Disk Space within the Database Administrator's Guide for more information about low disk space.</p> |
| Read Only File System | 1 | The database does not have write access to the file system for the data or catalog paths. This can sometimes occur if Linux remounts a drive due to a kernel issue. | Modify the privileges on the file system to give the database write access. |
| Loss Of K Safety | 2 | <p>The database is no longer K-Safe because there are insufficient nodes functioning within the cluster. Loss of K-safety causes the database to shut down.</p> <p>In a four-node cluster, for example, K-safety=1. If one node fails, the fault tolerance is at a critical level. If two nodes fail, the system loses K-safety.</p> | If a system shuts down due to loss of K-safety, you need to recover the system. See Failure Recovery in the Administrator's Guide. |
| Current Fault Tolerance at Critical Level | 3 | One or more nodes in the cluster have failed. If the database loses one more node, it is no longer K-Safe and it shuts down. (For example, a four-node cluster is no longer K-safe if two nodes fail.) | Restore any nodes that have failed or been shut down. |

| Event Name | Event Type | Description | Action |
|-------------------------|------------|--|---|
| Too Many ROS Containers | 4 | <p>Due to heavy data load conditions, there are too many ROS containers. This occurs when the Tuple Mover falls behind in performing mergeout operations. The resulting excess number of ROS containers can exhaust all available system resources. To prevent this, HP Vertica automatically rolls back all transactions that would load data until the Tuple Mover has time to catch up.</p> | <p>You might need to adjust the Tuple Mover's configuration parameters to compensate for the load pattern or rate. See Tuning the Tuple Mover in the Administrator's Guide for details.</p> <p>You can query the TUPLE_MOVER_OPERATIONS table to monitor mergeout activity. However, the Tuple Mover does not immediately start a mergeout when a projection reaches the limit of ROS containers, so you may not see a mergeout in progress when receiving this error.</p> <p>If waiting for a mergeout does not resolve the error, the problem probably is related to insufficient RAM.. A good rule of thumb is that system RAM in MB divided by 6 times the number of columns in the largest table should be greater than 10. For example, for a 100 column table you would want at least 6GB of RAM ($6144\text{MB} / (6 * 100) = 10.24$) to handle continuous loads.</p> |
| WOS Over Flow | 5 | <p>The WOS cannot hold all the data that you are attempting to load. This means that the copy fails and the transaction rolls back.</p> <p>Note: This event does not occur in HP Vertica 4.0 or later.</p> | <p>Consider loading the data to disk (ROS) instead of memory (WOS) or splitting the fact table load file into multiple pieces and then performing multiple loads in sequence.</p> <p>You might also consider making the Tuple Mover's moveout operation more aggressive. See Tuning the Tuple Mover in Administrator's Guide.</p> |
| Node State Change | 6 | The node state has changed. | Check the status of the node. |

| Event Name | Event Type | Description | Action |
|-------------------------------------|-------------------|--|--|
| Recovery Failure | 7 | The database was not restored to a functional state after a hardware or software related failure. | The reason for recovery failure can vary. See the event description for more information about your specific situation. |
| Recovery Error | 8 | The database encountered an error while attempting to recover. If the number of recovery errors exceeds Max Tries, the Recovery Failure event is triggered. See Recovery Failure within this table. | The reason for a recovery error can vary. See the event description for more information about your specific situation. |
| Recovery Lock Error | 9 | A recovering node could not obtain an S lock on the table. If you have a continuous stream of COPY commands in progress, recovery might not be able to obtain this lock even after multiple re-tries. | Either momentarily stop the loads or pick a time when the cluster is not busy to restart the node and let recovery proceed. |
| Recovery Projection Retrieval Error | 10 | HP Vertica was unable to retrieve information about a projection. | The reason for a recovery projection retrieval error can vary. See the event description for more information about your specific situation. |
| Refresh Error | 11 | The database encountered an error while attempting to refresh. | The reason for a refresh error can vary. See the event description for more information about your specific situation. |
| Refresh Lock Error | 12 | The database encountered a locking error during refresh. | The reason for a refresh error can vary. See the event description for more information about your specific situation. |
| Tuple Mover Error | 13 | The database encountered an error while attempting to move the contents of the Write Optimized Store (WOS) into the Read Optimized Store (ROS). | The reason for a Tuple Mover error can vary. See the event description for more information about your specific situation. |
| Timer Service Task Error | 14 | An error occurred in an internal scheduled task. | Internal use only |

| Event Name | Event Type | Description | Action |
|------------------|------------|--|---|
| Stale Checkpoint | 15 | Data in the WOS has not been completely moved out in a timely manner. An UNSAFE shutdown could require reloading a significant amount of data. | Be sure that Moveout operations are executing successfully. Check the <code>vertica.log</code> files for errors related to Moveout. |

Event Data

To help you interpret and solve the issue that triggered an event, each event provides a variety of data, depending upon the event logging mechanism used.

The following table describes the event data and indicates where it is used.

| <code>vertica.log</code> | ACTIVE_EVENTS (column names) | SNMP | Syslog | Description |
|--------------------------|------------------------------|------------|------------|--|
| N/A | NODE_NAME | N/A | N/A | The node where the event occurred. |
| Event Code | EVENT_CODE | Event Type | Event Code | A numeric ID that indicates the type of event. See Event Types in the previous table for a list of event type codes. |
| Event Id | EVENT_ID | Event OID | Event Id | A unique numeric ID that identifies the specific event. |

| vertica.log | ACTIVE_ EVENTS (column names) | SNMP | Syslog | Description |
|--------------------|--------------------------------------|----------------|-----------------|--|
| Event Severity | EVENT_ SEVERITY | Event Severity | Event Severity | <p>The severity of the event from highest to lowest. These events are based on standard syslog severity types:</p> <ul style="list-style-type: none"> 0 - Emergency 1 - Alert 2 - Critical 3 - Error 4 - Warning 5 - Notice 6 - Info 7 - Debug |
| PostedTimestamp | EVENT_ POSTED_ TIMESTAMP | N/A | PostedTimestamp | <p>The year, month, day, and time the event was reported. Time is provided as military time.</p> |

| vertica.log | ACTIVE_ EVENTS (column names) | SNMP | Syslog | Description |
|----------------------|--------------------------------------|-------------------------|----------------------|---|
| ExpirationTimestamp | EVENT_ EXPIRATION | N/A | ExpirationTimestamp | The time at which this event expires. If the same event is posted again prior to its expiration time, this field gets updated to a new expiration time. |
| EventCodeDescription | EVENT_ CODE_ DESCRIPTION | Description | EventCodeDescription | A brief description of the event and details pertinent to the specific situation. |
| ProblemDescription | EVENT_ PROBLEM_ DESCRIPTION | Event Short Description | ProblemDescription | A generic description of the event. |
| N/A | REPORTING_ NODE | Node Name | N/A | The name of the node within the cluster that reported the event. |
| DatabaseName | N/A | Database Name | DatabaseName | The name of the database that is impacted by the event. |

| vertica.log | ACTIVE_EVENTS (column names) | SNMP | Syslog | Description |
|--------------------|---|--------------|---------------|---|
| N/A | N/A | Host Name | Hostname | The name of the host within the cluster that reported the event. |
| N/A | N/A | Event Status | N/A | The status of the event. It can be either: 1 - Open 2 - Clear |

Configuring Event Reporting

Event reporting is automatically configured for `vertica.log`, and current events are automatically posted to the `ACTIVE_EVENTS` system table. You can also configure HP Vertica to post events to `syslog` and `SNMP`.

Configuring Reporting for Syslog

Syslog is a network-logging utility that issues, stores, and processes meaningful log messages. It is designed so DBAs can keep machines up and running, and it is a useful way to get heterogeneous data into a single data repository.

To log events to syslog, enable event reporting for each individual event you want logged. Messages are logged, by default, in `/var/log/messages`.

Configuring event reporting to syslog consists of:

1. Enabling HP Vertica to trap events for syslog.
2. Defining which events HP Vertica traps for syslog.

HP strongly suggests that you trap the Stale Checkpoint event.

3. Defining which syslog facility to use.

Enabling HP Vertica to Trap Events for Syslog

To enable event trapping for syslog, issue the following SQL command:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogEnabled', 1 );
       SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

To disable event trapping for syslog, issue the following SQL command:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogEnabled', 0 );
       SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

Defining Events to Trap for Syslog

To define events that generate a syslog entry, issue the following SQL command, where `Event_Name` is one of the events described in the list below the command:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogEvents', 'Event_Name' , 'Event_Name');
       SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

- Low Disk Space
- Read Only File System
- Loss Of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Recovery Error
- Recovery Lock Error
- Recovery Projection Retrieval Error
- Refresh Error

- Refresh Lock Error
- Tuple Mover Error
- Timer Service Task Error
- Stale Checkpoint

The following example generates a syslog entry for low disk space and recovery failure:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogEvents', 'Low Disk Space, Recovery Failure');  
       SET_CONFIG_PARAMETER  
-----  
Parameter set successfully  
(1 row)
```

Defining the SyslogFacility to Use for Reporting

The syslog mechanism allows for several different general classifications of logging messages, called facilities. Typically, all authentication-related messages are logged with the `auth` (or `authpriv`) facility. These messages are intended to be secure and hidden from unauthorized eyes. Normal operational messages are logged with the `daemon` facility, which is the collector that receives and optionally stores messages.

The `SyslogFacility` directive allows all logging messages to be directed to a different facility than the default. When the directive is used, *all* logging is done using the specified facility, both authentication (secure) and otherwise.

To define which `SyslogFacility` HP Vertica uses, issue the following SQL command:

```
=> SELECT SET_CONFIG_PARAMETER('SyslogFacility' , 'Facility_Name');
```

Where the facility-level argument `<Facility_Name>` is one of the following:

| | |
|---|--------------------------------------|
| • <code>auth</code> | • <code>uucp</code> (UUCP subsystem) |
| • <code>authpriv</code> (Linux only) | • <code>local0</code> (local use 0) |
| • <code>cron</code> | • <code>local1</code> (local use 1) |
| • <code>daemon</code> | • <code>local2</code> (local use 2) |
| • <code>ftp</code> (Linux only) | • <code>local3</code> (local use 3) |
| • <code>lpr</code> (line printer subsystem) | • <code>local4</code> (local use 4) |
| • <code>mail</code> (mail system) | • <code>local5</code> (local use 5) |

| | |
|---|--|
| <ul style="list-style-type: none">• news (network news subsystem) | <ul style="list-style-type: none">• local6 (local use 6) |
| <ul style="list-style-type: none">• user (default system) | <ul style="list-style-type: none">• local7 (local use 7) |

See Also

- [Event Reporting Examples](#)
- [Configuration Parameters](#)

Configuring Reporting for SNMP

Configuring event reporting for SNMP consists of:

1. Configuring HP Vertica to enable event trapping for SNMP as described below.
2. Importing the HP Vertica Management Information Base (MIB) file into the SNMP monitoring device.

The HP Vertica MIB file allows the SNMP trap receiver to understand the traps it receives from HP Vertica. This, in turn, allows you to configure the actions it takes when it receives traps.

HP Vertica supports the SNMP V1 trap protocol, and it is located in `/opt/vertica/sbin/VERTICA-MIB`. See the documentation for your SNMP monitoring device for more information about importing MIB files.

3. Configuring the SNMP trap receiver to handle traps from HP Vertica.

SNMP trap receiver configuration differs greatly from vendor to vendor. As such, the directions presented here for configuring the SNMP trap receiver to handle traps from HP Vertica are generic.

HP Vertica traps are single, generic traps that contain several fields of identifying information. These fields equate to the event data described in [Monitoring Events](#). However, the format used for the field names differs slightly. Under SNMP, the field names contain no spaces. Also, field names are pre-pended with "vert". For example, Event Severity becomes `vertEventSeverity`.

When configuring your trap receiver, be sure to use the same hostname, port, and community string you used to configure event trapping in HP Vertica.

Examples of network management providers:

- [HP Software Network Node Manager](#)
- IBM Tivoli

- AdventNet
- Net-SNMP (Open Source)
- Nagios (Open Source)
- Open NMS (Open Source)

See Also

- [Configuration Parameters](#)

Configuring Event Trapping for SNMP

The following events are trapped by default when you configure HP Vertica to trap events for SNMP:

- Low Disk Space
- Read Only File
- System
- Loss of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Stale Checkpoint

To Configure HP Vertica to Trap Events for SNMP

1. Enable HP Vertica to trap events for SNMP.
2. Define where HP Vertica sends the traps.
3. Optionally redefine which SNMP events HP Vertica traps.

Note: After you complete steps 1 and 2 above, HP Vertica automatically traps the default SNMP events. Only perform step 3 if you want to redefine which SNMP events are trapped.

HP strongly suggests that you trap the Stale Checkpoint event even if you decide to reduce the number events HP Vertica traps for SNMP. The setting has no effect on traps sent to the log. All events are trapped to the log.

To Enable Event Trapping for SNMP

Use the following SQL command:

```
=> SELECT SET_CONFIG_PARAMETER('SnmpTrapsEnabled', 1 );
```

To Define Where HP Vertica Send Traps

Use the following SQL command, where Host_name and port identify the computer where SNMP resides, and CommunityString acts like a password to control HP Vertica's access to the server:

```
=> SELECT SET_CONFIG_PARAMETER('SnmpTrapDestinationsList',  
    'host_name port CommunityString' );
```

For example:

```
=> SELECT SET_CONFIG_PARAMETER('SnmpTrapDestinationsList',  
    'localhost 162 public' );
```

You can also specify multiple destinations by specifying a list of destinations, separated by commas:

```
=> SELECT SET_CONFIG_PARAMETER('SnmpTrapDestinationsList',  
    'host_name1 port1 CommunityString1,hostname2 port2 CommunityString2' );
```

Note: : Setting multiple destinations sends any SNMP trap notification to all destinations listed.

To Define Which Events HP Vertica Traps

Use the following SQL command, where Event_Name is one of the events in the list below the command:

```
=> SELECT SET_CONFIG_PARAMETER('SnmpTrapEvents', 'Event_Name, Event_Name');
```

- Low Disk Space
- Read Only File System

- Loss Of K Safety
- Current Fault Tolerance at Critical Level
- Too Many ROS Containers
- WOS Over Flow
- Node State Change
- Recovery Failure
- Recovery Error
- Recovery Lock Error
- Recovery Projection Retrieval Error
- Refresh Error
- Tuple Mover Error
- Stale Checkpoint

Note: The above values are case sensitive.

The following is an example that uses two different event names:

```
=> SELECT SET_CONFIG_PARAMETER('SnmpTrapEvents', 'Low Disk Space, Recovery  
Failure');
```

Verifying SNMP Configuration

To create a set of test events that checks SNMP configuration:

1. Set up SNMP trap handlers to catch HP Vertica events.
2. Test your setup with the following command:

```
SELECT SNMP_TRAP_TEST();  
      SNMP_TRAP_TEST  
-----  
Completed SNMP Trap Test  
(1 row)
```

Event Reporting Examples

Vertica.log

The following example illustrates a Too Many ROS Containers event posted and cleared within vertica.log:

```
08/14/08 15:07:59 thr:nameless:0x45a08940 [INFO] Event Posted: Event Code:4 Event Id:0 Event Severity: Warning [4] PostedTimestamp: 2008-08-14 15:07:59.253729 ExpirationTimestamp: 2008-08-14 15:08:29.253729 EventCodeDescription: Too Many ROS Containers ProblemDescription: Too many ROS containers exist on this node. DatabaseName: QATESTDB Hostname: fc6-1.verticacorp.com
08/14/08 15:08:54 thr:Ageout Events:0x2aaab0015e70 [INFO] Event Cleared: Event Code:4 Event Id:0 Event Severity: Warning [4] PostedTimestamp: 2008-08-14 15:07:59.253729 ExpirationTimestamp: 2008-08-14 15:08:53.012669 EventCodeDescription: Too Many ROS Containers ProblemDescription: Too many ROS containers exist on this node. DatabaseName: QATESTDB Hostname: fc6-1.verticacorp.com
```

SNMP

The following example illustrates a Too Many ROS Containers event posted to SNMP:

```
Version: 1, type: TRAPREQUESTEnterprise OID: .1.3.6.1.4.1.31207.2.0.1
Trap agent: 72.0.0.0
Generic trap: ENTERPRISESPECIFIC (6)
Specific trap: 0
.1.3.6.1.4.1.31207.1.1 ---> 4
.1.3.6.1.4.1.31207.1.2 ---> 0
.1.3.6.1.4.1.31207.1.3 ---> 2008-08-14 11:30:26.121292
.1.3.6.1.4.1.31207.1.4 ---> 4
.1.3.6.1.4.1.31207.1.5 ---> 1
.1.3.6.1.4.1.31207.1.6 ---> site01
.1.3.6.1.4.1.31207.1.7 ---> suse10-1
.1.3.6.1.4.1.31207.1.8 ---> Too many ROS containers exist on this node.
.1.3.6.1.4.1.31207.1.9 ---> QATESTDB
.1.3.6.1.4.1.31207.1.10 ---> Too Many ROS Containers
```

Syslog

The following example illustrates a Too Many ROS Containers event posted and cleared within syslog:

```
Aug 14 15:07:59 fc6-1 vertica: Event Posted: Event Code:4 Event Id:0 Event Severity: Warning [4] PostedTimestamp: 2008-08-14 15:07:59.253729 ExpirationTimestamp: 2008-08-14 15:08:29.253729 EventCodeDescription: Too Many ROS Containers ProblemDescription: Too many ROS containers exist on this node. DatabaseName: QATESTDB Hostname: fc6-
```

```
1.verticacorp.com
Aug 14 15:08:54 fc6-1 vertica: Event Cleared: Event Code:4 Event Id:0 Event Severity:
Warning [4] PostedTimestamp: 2008-08-14 15:07:59.253729 ExpirationTimestamp:
2008-08-14 15:08:53.012669 EventCodeDescription: Too Many ROS Containers ProblemDescripti
on:
Too many ROS containers exist on this node. DatabaseName: QATESTDB Hostname: fc6-1.vertic
acorp.com
```

Using System Tables

HP Vertica provides an API (application programming interface) for monitoring various features and functions within a database in the form of system tables. These tables provide a robust, stable set of views that let you monitor information about your system's resources, background processes, workload, and performance, allowing you to more efficiently profile, diagnose, and view historical data equivalent to load streams, query profiles, tuple mover operations, and more. Because HP Vertica collects and retains this information automatically, you don't have to manually set anything.

You can write queries against system tables with full SELECT support the same way you perform query operations on base and temporary tables. You can query system tables using expressions, predicates, aggregates, analytics, subqueries, and joins. You can also save system table query results into a user table for future analysis. For example, the following query creates a table, `mynode`, selecting three node-related columns from the `V_CATALOG.NODES` system table:

```
VMart=> CREATE TABLE mynode AS SELECT node_name, node_state, node_address
FROM nodes;
CREATE TABLE
VMart=> SELECT * FROM mynode;
  node_name      | node_state | node_address
-----+-----+-----
v_vmart_node0001 | UP         | 192.168.223.11
(1 row)
```

Note: You cannot query system tables if the database cluster is in a recovering state. The database refuses connection requests and cannot be monitored. HP Vertica also does not support **DDL** and **DML** operations on system tables.

Where System Tables Reside

System tables are grouped into the following schemas:

- `V_CATALOG` – information about persistent objects in the catalog
- `V_MONITOR` – information about transient system state

These schemas reside in the default search path so there is no need to specify `schema.table` in your queries unless you [change the search path](#) to exclude `V_MONITOR` or `V_CATALOG` or both.

The system tables that make up the monitoring API are described fully in the [SQL Reference Manual](#). You can also use the following command to view all the system tables and their schema:

```
SELECT * FROM system_tables ORDER BY table_schema, table_name;
```

How System Tables Are Organized

Most of the tables are grouped into the following areas:

- System information
- System resources
- Background processes
- Workload and performance

HP Vertica reserves some memory to help monitor busy systems. Using simple system table queries makes it easier to troubleshoot issues. See also `SYSQUERY` and `SYSDATA` pools under [Built-in pools](#) topic in SQL Reference Manual.

Note: You can use external monitoring tools or scripts to query the system tables and act upon the information, as necessary. For example, when a host failure causes the **K-safety** level to fall below the desired level, the tool or script can notify the database administrator and/or appropriate IT personnel of the change, typically in the form of an e-mail.

Querying Case-Sensitive Data in System Tables

Some system table data might be stored in mixed case. For example, HP Vertica stores mixed-case `identifier` names the way you specify them in the `CREATE` statement, even though case is ignored when you reference them in queries. When these object names appear as data in the system tables, you'll encounter errors if you use the equality (`=`) predicate because the case must match the stored identifier. In particular, `V_CATALOG.TABLES.TABLE_SCHEMA` and `V_CATALOG.TABLES.TABLE_NAME` columns are case sensitive with equality predicates.

If you don't know how the identifiers are stored, use the case-insensitive operator `ILIKE` instead of equality predicates.

For example, given the following schema:

```
=> CREATE SCHEMA SS;=> CREATE TABLE SS.TT (c1 int);
=> CREATE PROJECTION SS.TTP1 AS SELECT * FROM ss.tt UNSEGMENTED ALL NODES;
=> INSERT INTO ss.tt VALUES (1);
```

If you run a query using the `=` predicate, HP Vertica returns 0 rows:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema = 'ss';
table_schema | table_name
-----+-----
(0 rows)
```

Using the case-insensitive `ILIKE` predicate returns the expected results:

```
=> SELECT table_schema, table_name FROM v_catalog.tables WHERE table_schema ILIKE 'ss';
table_schema | table_name
-----+-----
SS           | TT
```

(1 row)

Examples

The following query uses the VMart example database (see [Introducing the VMart Example Database](#)) to obtain the number of rows and size occupied by each table in the database.

```
=> SELECT t.table_name AS table_name,  
        SUM(ps.wos_row_count + ps.ros_row_count) AS row_count,  
        SUM(ps.wos_used_bytes + ps.ros_used_bytes) AS byte_count  
FROM tables t  
JOIN projections p ON t.table_id = p.anchor_table_id  
JOIN projection_storage ps on p.projection_name = ps.projection_name  
WHERE (ps.wos_used_bytes + ps.ros_used_bytes) > 500000  
GROUP BY t.table_name  
ORDER BY byte_count DESC;  
table_name | row_count | byte_count  
-----+-----+-----  
online_sales_fact | 5000000 | 171987438  
store_sales_fact | 5000000 | 108844666  
store_orders_fact | 300000 | 9240800  
product_dimension | 60000 | 2327964  
customer_dimension | 50000 | 2165897  
inventory_fact | 300000 | 2045900  
(6 rows)
```

The rest of the examples illustrate simple ways to use system tables in queries.

```
=> SELECT table_name FROM columns WHERE data_type ILIKE 'Numeric' GROUP BY  
table_name;  
table_name  
-----  
n1  
(1 row)  
=> SELECT current_epoch, designed_fault_tolerance, current_fault_tolerance  
FROM SYSTEM;  
current_epoch | designed_fault_tolerance | current_fault_tolerance  
-----+-----+-----  
492 | 1 | 1  
(1 row)  
=> SELECT node_name, total_user_session_count, executed_query_count FROM  
query_metrics;  
node_name | total_user_session_count | executed_query_count  
-----+-----+-----  
node01 | 53 | 42  
node02 | 53 | 0  
node03 | 42 | 120  
node04 | 53 | 0  
(4 rows)  
=> SELECT table_schema FROM primary_keys;  
table_schema  
-----  
public
```

```
public  
public  
public  
public  
public  
public  
public  
public  
public  
store  
online_sales  
online_sales  
(12 rows)
```

Retaining Monitoring Information

When you query an HP Vertica system table (described in [Using System Tables](#)), you can get information about currently running queries, the state of various components, and other run-time information. During query execution, HP Vertica examines the current state of the system and returns information in the result set.

Data Collector

HP Vertica also provides a utility called the Data Collector (DC), which collects and retains history of important system activities and records essential performance and resource utilization counters.

Data Collector extends system table functionality by:

- Providing a framework for recording events
- Making the information available in system tables
- Requiring few configuration parameter tweaks
- Having negligible impact on performance

You can use the information the Data Collector retains to query the past state of system tables and extract aggregate information, as well as do the following:

- See what actions users have taken
- Locate performance bottlenecks
- Identify potential improvements to HP Vertica configuration

DC does not collect data for nodes that are down, so no historical data would be available for that node.

Data Collector works in conjunction with the Workload Analyzer (WLA), an advisor tool that intelligently monitors the performance of SQL queries and workloads and recommends tuning actions based on observations of the actual workload history. See [Analyzing Workloads](#) for more information about WLA.

Where Is DC Information retained?

Collected data is stored on disk in the `DataCollector` directory under the HP Vertica `/catalog` path. This directory also contains instructions on how to load the monitoring data into another HP Vertica database. See [Working with Data Collection Logs](#) for details.

DC retains the data it gathers based on retention policies, which a superuser can configure. See [Configuring Data Retention Policies](#).

Data Collector is on by default, but a superuser can disable it if performance issues arise. See [Data Collector Parameters](#) and [Enabling and Disabling Data Collector](#).

DC Tables

Caution: Data Collector tables (prefixed by `dc_`) reside in the `V_INTERNAL` schema and are provided for informational purposes only. They are provided as-is and are subject to removal or change without notice. If you use Data Collector tables in scripts or monitoring tools, you might need to change your scripts and tools after an HP Vertica upgrade. HP recommends that you use the [Workload Analyzer](#) instead of accessing the Data Collector tables directly.

See Also

-
-
-
-

Enabling and Disabling Data Collector

Data Collector is on by default and retains information for all sessions. If performance issues arise, a superuser can disable Data Collector at any time, such as if performance issues arise.

To disable the Data Collector:

```
=> SELECT SET_CONFIG_PARAMETER('EnableDataCollector', '0');
```

To re-enable the Data Collector:

```
=> SELECT SET_CONFIG_PARAMETER('EnableDataCollector', '1');
```

See Also

- [Data Collector Parameters](#)

Viewing Current Data Retention Policy

To view the current retention policy for a **Data Collector** component, use the [GET_DATA_COLLECTOR_POLICY\(\)](#) function and supply the component name as the function's argument.

To retrieve a list of all current component names, query the `V_MONITOR.DATA_COLLECTOR` system table, which returns Data Collector components, their current retention policies, and statistics about how much data is retained. For example:

```
mcdb=> \xExpanded display is on.
```

```
mddb=> SELECT * from data_collector;
-[ RECORD 1 ]-----+-----
node_name          | v_mddb_node0001
component          | AllocationPoolStatistics
table_name         | dc_allocation_pool_statistics
description        | Information about global memory pools, which generally cannot be
recovered without restart
access_restricted  | t
in_db_log          | f
in_vertica_log     | f
memory_buffer_size_kb | 64
disk_size_kb       | 256
record_too_big_errors | 0
lost_buffers       | 0
lost_records       | 0
retired_files      | 1429
retired_records    | 647358
current_memory_records | 0
current_disk_records | 1493
current_memory_bytes | 0
current_disk_bytes  | 215737
first_time         | 2012-11-30 07:04:30.000726-05
last_time          | 2012-11-30 07:16:56.000631-05
kb_per_day         | 24377.3198211312
-[ RECORD 2 ]-----+-----
```

The following command returns the retention policy for a specific component, NodeState.

```
=> SELECT get_data_collector_policy('NodeState');
```

The results let you know that 10KB is retained in memory and 100KB on disk:

```
get_data_collector_policy-----
10KB kept in memory, 100KB kept on disk.
(1 row)
```

Configuring Data Retention Policies

Data Collector retention policies hold the following information:

- Which component to monitor
- How much memory to retain
- How much disk space to retain

A **superuser** can modify policies, such as change the amount of data to retain, by invoking the [SET_DATA_COLLECTOR_POLICY\(\)](#) function, as follows:

```
SET_DATA_COLLECTOR_POLICY('component', 'memoryKB', 'diskKB' )
```

The `SET_DATA_COLLECTOR_POLICY()` function sets the retention policy for the specified component on all nodes, and lets you specify memory and disk size to retain in kilobytes. Failed nodes receive the new setting when they rejoin the cluster.

For example, the following statement specifies that the `NodeState` component be allocated 50KB of memory and 250KB of disk space:

```
=> SELECT SET_DATA_COLLECTOR_POLICY('NodeState', '50', '250');
      SET_DATA_COLLECTOR_POLICY
-----
      SET
(1 row)
```

Before you change a retention policy, you can view the current setting by calling the `GET_DATA_COLLECTOR_POLICY()` function.

You can also use the `GET_DATA_COLLECTOR_POLICY()` function to verify changed settings. For example, the following query retrieves a brief statement about the retention policy for the `NodeState` component:

```
=> SELECT GET_DATA_COLLECTOR_POLICY('NodeState');
      GET_DATA_COLLECTOR_POLICY
-----
50KB kept in memory, 250KB kept on disk.
(1 row)
```

Tip: If you do not know the name of a component, you can query the `V_MONITOR.DATA_COLLECTOR` system table to get a full list. For example, the following query returns all current Data Collector components and a description of each: `=> SELECT DISTINCT component, description FROM data_collector ORDER BY 1 ASC;`

See Also

- [GET_DATA_COLLECTOR_POLICY](#)
- [SET_DATA_COLLECTOR_POLICY](#)

Working with Data Collection Logs

Upon startup, an HP Vertica database creates a `DataCollector` directory within the `/catalog` directory.

The `DataCollector` directory holds the disk-based data collection logs, where retained data is kept in files named `<component>.<timestamp>.log`. HP Vertica might maintain several log files, per component, at any given time. See [Querying Data Collector Tables](#) for an example of how to view this information.

Also upon startup, HP Vertica creates two additional files, per component, in the `DataCollector` directory. These are SQL files that contain examples on how to load Data Collector data into another HP Vertica instance. These files are:

- `CREATE_<component>_TABLE.sql` — contains the SQL DDL needed to create a table into which Data Collector logs for the component can be loaded.
- `COPY_<component>_TABLE.sql` — contains example SQL to load (using `COPY` commands) the data log files into the table that the `CREATE` script creates.

Two functions let you manipulate these log files.

| If you want to ... | See these topics in the SQL Reference Manual |
|---|--|
| Clear memory and disk records from Data Collector retention and reset collection statistics | CLEAR_DATA_COLLECTOR() |
| Flush the Data Collector logs | FLUSH_DATA_COLLECTOR() |
| Retrieve a list of all current Data Collector components | V_MONITOR.DATA_COLLECTOR |

Clearing the Data Collector

If you want to clear the Data Collector of all memory and disk records and reset the collection statistics in the `V_MONITOR.DATA_COLLECTOR` system table, call the `CLEAR_DATA_COLLECTOR()` function. You can clear records for all components on all nodes or you can specify individual components, one at a time.

To clear records on a single component, pass the function the component argument. For example, the following command clears records for the `ResourceAcquisitions` component only, returning a result of `CLEAR` (success):

```
=> SELECT clear_data_collector('ResourceAcquisitions');
clear_data_collector
-----
CLEAR
(1 row)
```

The following command clears the Data Collector records for all components on all nodes:

```
=> SELECT clear_data_collector();
clear_data_collector
-----
CLEAR
(1 row)
```

Note: After you clear the DataCollector log, the information is no longer available for querying.

Flushing Data Collector Logs

If you want to flush Data Collector information for all components or for an individual component, use the `FLUSH_DATA_COLLECTOR()` function. This function waits until memory logs are moved to disk and flushes the Data Collector, synchronizing the log with the disk storage:

To flush data collection for all components on all nodes:

```
=> SELECT flush_data_collector();
flush_data_collector
-----
FLUSH
(1 row)
```

To flush records on a single component, pass a component argument to the function. For example, the following command flushes the ResourceAcquisitions component:

```
=> SELECT flush_data_collector('ResourceAcquisitions');
flush_data_collector
-----
FLUSH
(1 row)
```

See Also

- [Data Collector Functions](#)
- [DATA_COLLECTOR](#)

Monitoring Data Collection Components

Query the `V_MONITOR.DATA_COLLECTOR` system table to get a list of **Data Collector** components, their current retention policies, and statistics about how much data is retained and how much has been discarded for various reasons. `DATA_COLLECTOR` also calculates the approximate collection rate, to aid in sizing calculations.

The following is a simple query that returns all the columns in this system table. See [V_MONITOR.DATA_COLLECTOR](#) in the SQL Reference Manual for additional details.

```
=> \xExpanded display is on.
=> SELECT * FROM data_collector;
-[ RECORD 1 ]-----+-----
node_name          | v_vmartdb_node0001
component          | AllocationPoolStatistics
table_name         | dc_allocation_pool_statistics
description        | Information about global memory pools, ...
in_db_log          | f
in_vertica_log     | f
memory_buffer_size_kb | 64
```

```

disk_size_kb          | 256
record_too_big_errors | 0
lost_buffers         | 0
lost_records         | 0
retired_files        | 120
retired_records      | 53196
current_memory_records | 0
current_disk_records | 1454
current_memory_bytes | 0
current_disk_bytes   | 214468
first_time           | 2011-05-26 12:25:52.001109-04
last_time            | 2011-05-26 12:31:55.002762-04
kb_per_day           | 49640.4151810525
-[ RECORD 2 ]-----+-----
...

```

Related Topics

[V_MONITOR.DATA_COLLECTOR](#) and [Data Collector Functions](#) in the SQL Reference Manual
[Retaining Monitoring Information](#) and [How HP Vertica Calculates Database Size](#) in this guide

Querying Data Collector Tables

Caution: Data Collector tables (prefixed by `dc_`) reside in the `V_INTERNAL` schema and are provided for informational purposes only. They are provided as-is and are subject to removal or change without notice. If you use Data Collector tables in scripts or monitoring tools, you might need to change your scripts and tools after an HP Vertica upgrade. HP recommends that you use the [Workload Analyzer](#) instead of accessing the Data Collector tables directly.

Here's one useful example you can use to check on the state of your database. Upon startup, the HP Vertica database creates, under its catalog directory, a `DataCollector` directory. This directory holds the disk-based data collection logs. The main data is kept in files named `<component>.<timestamp>.log`.

When you start your database an entry is created in the `dc_startups` table. The following is the result of querying this table.

```

=> SELECT * FROM dc_startups;
-[ RECORD 1 ]-----+-----
time          | 2011-05-26 17:35:40.588589-04
node_name     | v_vmartdb_node0001
version       | Vertica Analytic Database v5.0.4-20110526
command_line  | /opt/vertica/bin/vertica -C vmartdb -D /home/vmartdb/
              | catalog/vmartdb/v_vmartdb_node0001_catalog -h
              | 10.10.50.123 -p 5608
codename      | 5.0
build_tag     | vertica(v5.0.4-20110526) built by root@build2 from trunk@69652 on 'Thu
              | May 26 3:37:18 2011' $BuildId$
build_type    | 64-bit Optimized Build

```

```

compiler_version | 4.1.1 20070105 (Red Hat 5.1.1-52)
server_locale    | UTF-8
database_path   | /home/vmartdb/catalog/vmartdb/v_vmartdb_node0001_catalog
alt_host_name   | 10.10.50.123
alt_node_name   |
start_epoch     |
-[ RECORD 2 ]-----+-----
time            | 2011-05-26 17:35:40.218999-04
node_name       | v_vmartdb_node0004
version         | Vertica Analytic Database v5.0.4-20110526
command_line    | /opt/vertica/bin/vertica -C vmartdb -D /home/vmartdb/
                | catalog/vmartdb/v_vmartdb_node0004_catalog -h
                | 10.10.50.126 -p 5608
codename        | 5.0
build_tag       | vertica(v5.0.4-20110526) built by root@build2 from trunk@69652 on 'Thu
                | May 26 3:37:18 2011' $BuildId$
build_type      | 64-bit Optimized Build
compiler_version | 4.1.1 20070105 (Red Hat 5.1.1-52)
server_locale   | UTF-8
database_path   | /home/vmartdb/catalog/vmartdb/v_vmartdb_node0004_catalog
alt_host_name   | 10.10.50.126
alt_node_name   |
start_epoch     |
-[ RECORD 3 ]-----+-----
time            | 2011-05-26 17:35:40.931353-04
node_name       | v_vmartdb_node0003
version         | Vertica Analytic Database v5.0.4-20110526
command_line    | /opt/vertica/bin/vertica -C vmartdb -D /home/vmartdb/
                | catalog/vmartdb /v_vmartdb_node0003_catalog -h
                | 10.10.50.125 -p 5608
codename        | 5.0
build_tag       | vertica(v5.0.4-20110526) built by root@build2 from trunk@69652 on 'Thu
                | May 26 3:37:18 2011' $BuildId$
build_type      | 64-bit Optimized Build
compiler_version | 4.1.1 20070105 (Red Hat 5.1.1-52)
server_locale   | UTF-8
database_path   | /home/vmartdb/catalog/vmartdb/v_vmartdb_node0003_catalog
alt_host_name   | 10.10.50.125
alt_node_name   |
start_epoch     |
-[ RECORD 4 ]-----+-----

```

Clearing PROJECTION_REFRESHES History

Information about a refresh operation—whether successful or unsuccessful—is maintained in the [PROJECTION_REFRESHES](#) system table until either the [CLEAR_PROJECTION_REFRESHES\(\)](#) function is executed or the storage quota for the table is exceeded. The `PROJECTION_REFRESHES.IS_EXECUTING` column returns a boolean value that indicates whether the refresh is currently running (t) or occurred in the past (f).

To immediately purge this information, use the `CLEAR_PROJECTION_REFRESHES()` function:

```

=> SELECT clear_projection_refreshes();
clear_projection_refreshes

```

```
-----  
CLEAR  
(1 row)
```

Note: Only the rows where the `PROJECTION_REFRESHES.IS_EXECUTING` column equals `false` are cleared.

See Also

- [CLEAR_PROJECTION_REFRESHES](#)
- [PROJECTION_REFRESHES](#)

Monitoring Query Plan Profiles

To monitor real-time flow of data through a query plan, query the [V_MONITOR.QUERY_PLAN_PROFILES](#) system table. Information returned by this table is useful for letting you know *what* a query did *when*, which occurs throughout a plan in a series of steps, called **paths**.

See [Profiling query plans](#) for more information.

Monitoring Partition Reorganization

When you use the `ALTER TABLE ... REORGANIZE`, the operation reorganizes the data in the background.

You can monitor details of the reorganization process by polling the following system tables:

- [V_MONITOR.PARTITION_STATUS](#) displays the fraction of each table that is partitioned correctly.
- [V_MONITOR.PARTITION_REORGANIZE_ERRORS](#) logs any errors issued by the background REORGANIZE process.
- [V_MONITOR.PARTITIONS](#) displays NULLS in the `partition_key` column for any ROS's that have not been reorganized.

Note: The corresponding foreground process to `ALTER TABLE ... REORGANIZE` is [PARTITION_TABLE\(\)](#).

[Partitioning, repartitioning, and Reorganizing Tables](#) for more information.

Monitoring Resource Pools and Resource Usage By Queries

The [Linux top command](#) can be used to determine the overall CPU usage and I/O waits across the system. However, resident memory size indicated by `top` is not a good indicator of actual memory use or reservation because of file system caching and so forth. Instead, HP Vertica provides several monitoring tables that provide detailed information about resource pools, their current memory usage, resources requested and acquired by various requests and the state of the queues.

The [RESOURCE_POOLS](#) table lets you view various resource pools defined in the system (both internal and user-defined), and the [RESOURCE_POOL_STATUS](#) table lets you view the current state of the resource pools.

Examples

The following command returns the various resource pools defined in the system.

```
VMart=> SELECT name, memorysize, maxmemorysize FROM V_CATALOG.RESOURCE_POOLS;
```

| name | memorysize | maxmemorysize |
|----------|------------|---------------|
| general | | Special: 95% |
| sysquery | 64M | |
| sysdata | 100M | 10% |
| wosdata | 0% | 25% |
| tm | 200M | |
| refresh | 0% | |
| recovery | 0% | |
| dbd | 0% | |
| jvm | 0% | 10% |

(9 rows)

To see only the user-defined resource pools, you can limit your query to return records where `IS_INTERNAL` is false.

Note: The user-defined pools below are used as examples in subsequent sections related to Workload Management.

The following command returns information on user-defined resource pools:

```
=> SELECT name, memorysize, maxmemorysize, priority, maxconcurrency  
FROM V_CATALOG.RESOURCE_POOLS where is_internal = 'f';
```

| name | memorysize | maxmemorysize | priority | maxconcurrency |
|--------------|------------|---------------|----------|----------------|
| load_pool | 0% | | 10 | |
| ceo_pool | 250M | | 10 | |
| ad hoc_pool | 200M | 200M | 0 | |
| billing_pool | 0% | | 0 | 3 |

```

web_pool      | 25M      |          |      10 |          5
batch_pool    | 150M     | 150M     |      0 |          10
dept1_pool    | 0%       |          |      5 |
dept2_pool    | 0%       |          |      8 |
(8 rows)

```

The queries borrow memory from the GENERAL pool and show the amount of memory in use from the GENERAL pool.

The following command uses the V_MONITOR.RESOURCE_POOL_STATUS table to return the current state of all resource pools on node0001:

```

=>\x
Expanded display is on
=> SELECT pool_name, memory_size_kb, memory_size_actual_kb, memory_inuse_kb,
general_memory_borrowed_kb,running_query_count
FROM V_MONITOR.RESOURCE_POOL_STATUS where node_name ilike '%node0001';

-[ RECORD 1 ]-----+-----
pool_name          | general
memory_size_kb     | 2983177
memory_size_actual_kb | 2983177
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 2 ]-----+-----
pool_name          | sysquery
memory_size_kb     | 65536
memory_size_actual_kb | 65536
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 3 ]-----+-----
pool_name          | sysdata
memory_size_kb     | 102400
memory_size_actual_kb | 102400
memory_inuse_kb    | 4096
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 4 ]-----+-----
pool_name          | wosdata
memory_size_kb     | 0
memory_size_actual_kb | 0
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 5 ]-----+-----
pool_name          | tm
memory_size_kb     | 204800
memory_size_actual_kb | 204800
memory_inuse_kb    | 0
general_memory_borrowed_kb | 0
running_query_count | 0
-[ RECORD 6 ]-----+-----
pool_name          | refresh

```

```

memory_size_kb          | 0
memory_size_actual_kb   | 0
memory_inuse_kb         | 0
general_memory_borrowed_kb | 0
running_query_count     | 0
-[ RECORD 7 ]-----+-----
pool_name               | recovery
memory_size_kb         | 0
memory_size_actual_kb   | 0
memory_inuse_kb         | 0
general_memory_borrowed_kb | 0
running_query_count     | 0
-[ RECORD 8 ]-----+-----
pool_name               | dbd
memory_size_kb         | 0
memory_size_actual_kb   | 0
memory_inuse_kb         | 0
general_memory_borrowed_kb | 0
running_query_count     | 0
-[ RECORD 9 ]-----+-----
pool_name               | jvm
memory_size_kb         | 0
memory_size_actual_kb   | 0
memory_inuse_kb         | 0
general_memory_borrowed_kb | 0
running_query_count     | 0

```

The following command uses the `V_MONITOR.RESOURCE_ACQUISITIONS` table to show all resources granted to the queries that are currently running:

Note: While running `vmart_query_04.sql` from the [VMart example database](#), notice that the query uses `memory_inuse_kb = 708504` from the GENERAL pool.

```

=> SELECT pool_name, thread_count, open_file_handle_count, memory_inuse_kb,
        queue_entry_timestamp, acquisition_timestamp
        FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';

-[ RECORD 1 ]-----+-----
pool_name          | sysquery
thread_count       | 4
open_file_handle_count | 0
memory_inuse_kb    | 4103
queue_entry_timestamp | 2013-12-05 07:07:08.815362-05
acquisition_timestamp | 2013-12-05 07:07:08.815367-05
-[ RECORD 2 ]-----+-----
...
-[ RECORD 8 ]-----+-----
pool_name          | general
thread_count       | 12
open_file_handle_count | 18
memory_inuse_kb    | 708504
queue_entry_timestamp | 2013-12-04 12:55:38.566614-05
acquisition_timestamp | 2013-12-04 12:55:38.566623-05
-[ RECORD 9 ]-----+-----

```

...

To determine how long a query waits in the queue before it is admitted to run, you can get the difference between the `acquisition_timestamp` and the `queue_entry_timestamp` using a query like the following:

```
=> SELECT pool_name, queue_entry_timestamp, acquisition_timestamp,
       (acquisition_timestamp-queue_entry_timestamp) AS 'queue wait'
       FROM V_MONITOR.RESOURCE_ACQUISITIONS WHERE node_name ILIKE '%node0001';

-[ RECORD 1 ]-----+-----
pool_name      | sysquery
queue_entry_timestamp | 2013-12-05 07:07:08.815362-05
acquisition_timestamp | 2013-12-05 07:07:08.815367-05
queue wait     | 00:00:00.000005
-[ RECORD 2 ]-----+-----
pool_name      | sysquery
queue_entry_timestamp | 2013-12-05 07:07:14.714412-05
acquisition_timestamp | 2013-12-05 07:07:14.714417-05
queue wait     | 00:00:00.000005
-[ RECORD 3 ]-----+-----
pool_name      | sysquery
queue_entry_timestamp | 2013-12-05 07:09:57.238521-05
acquisition_timestamp | 2013-12-05 07:09:57.281708-05
queue wait     | 00:00:00.043187
-[ RECORD 4 ]-----+-----
...
```

See the [SQL Reference Manual](#) for detailed descriptions of the monitoring tables described in this topic.

Monitoring Recovery

There are several ways to monitor database recovery:

- Log files on each host
- Admintools (View Database Cluster State)
- System tables

This section describes the different ways to monitor recovery.

Viewing Log Files on Each Node

During database recovery, HP Vertica adds logging information to the `vertica.log` on each host. Each message is identified with a `[Recover]` string.

Use the `tail` command to monitor recovery progress by viewing the relevant status messages, as follows.

```
$ tail -f catalog-path/database-name/node-name_catalog/vertica.log
```



```
01/23/08 10:35:31 thr:Recover:0x2a98700970 [Recover] <INFO> Changing host node01
startup state from INITIALIZING to RECOVERING
```

```
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Recovering to specified
epoch 0x120b6
```

```
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running 1 split queries
```

```
01/23/08 10:35:31 thr:CatchUp:0x1724b80 [Recover] <INFO> Running query: ALTER
PROJECTION proj_tradesquotes_0 SPLIT node01 FROM 73911;
```

Viewing the Cluster State and Recover Status

Use the `admintools view_cluster` tool from the command line to see the cluster state:

```
$ /opt/vertica/bin/admintools -t view_cluster
DB | Host | State
-----+-----+-----
<data_base> | 112.17.31.10 | RECOVERING
<data_base> | 112.17.31.11 | UP
<data_base> | 112.17.31.12 | UP
<data_base> | 112.17.31.17 | UP
```

Using System Tables to Monitor Recovery

Use the following system tables to monitor recover:

- [RECOVERY_STATUS](#)
- [PROJECTION_RECOVERIES](#)

Specifically, the `recovery_status` system table includes information about the node that is recovering, the epoch being recovered, the current recovery phase, and running status:

```
=>select node_name, recover_epoch, recovery_phase, current_completed, is_running from reco
very_status;
node_name | recover_epoch | recovery_phase      | current_completed | is_running
-----+-----+-----+-----+-----
node01    |               |                    | 0                 | f
node02    | 0             | historical pass 1  | 0                 | t
node03    | 1             | current            | 0                 | f
```

The `projection_recoveries` system table maintains history of projection recoveries. To check the recovery status, you can summarize the data for the recovering node, and run the same query several times to see if the counts change. Differing counts indicate that the recovery is working and in the process of recovering all missing data.

```
=> select node_name, status , progress from projection_recoveries;
node_name          | status      | progress
-----+-----+-----
v_<data_base>_node0001 | running    | 61
```

To see a single record from the `projection_recoveries` system table, add `limit 1` to the query.

Monitoring Cluster Status After Recovery

When recovery has completed:

1. Launch Administration Tools.
2. From the Main Menu, select **View Database Cluster** State and click **OK**.

The utility reports your node's status as UP.

Note: You can also monitor the state of your database nodes on the Management Console Overview page under the Database section, which tells you the number of nodes that are up, critical, recovering, or down. To get node-specific information, click Manage at the bottom of the page.

See Also

- [Monitoring HP Vertica](#)

Monitoring HP Vertica Using MC

Management Console gathers and retains history of important system activities about your MC-managed database cluster, such as performance and resource utilization. You can use MC charts to locate performance bottlenecks on a particular node, to identify potential improvements to HP Vertica configuration, and as a reference for what actions users have taken on the MC interface.

Note: MC directly queries Data Collector tables on the MC-monitored databases themselves. See [Management Console Architecture](#) in the Concepts Guide.

The following list describes some of the areas you can monitor and troubleshoot through the MC interface:

- Multiple database cluster states and key performance indicators that report on the cluster's overall health
- Information on individual cluster nodes specific to resources
- Database activity in relation to CPU/memory, networking, and disk I/O usage
- Query concurrency and internal/user sessions that report on important events in time
- Cluster-wide messages
- Database and agent log entries
- MC user activity (what users are doing while logged in to MC)
- Issues related to the MC process
- Error handling and feedback

About Chart Updates

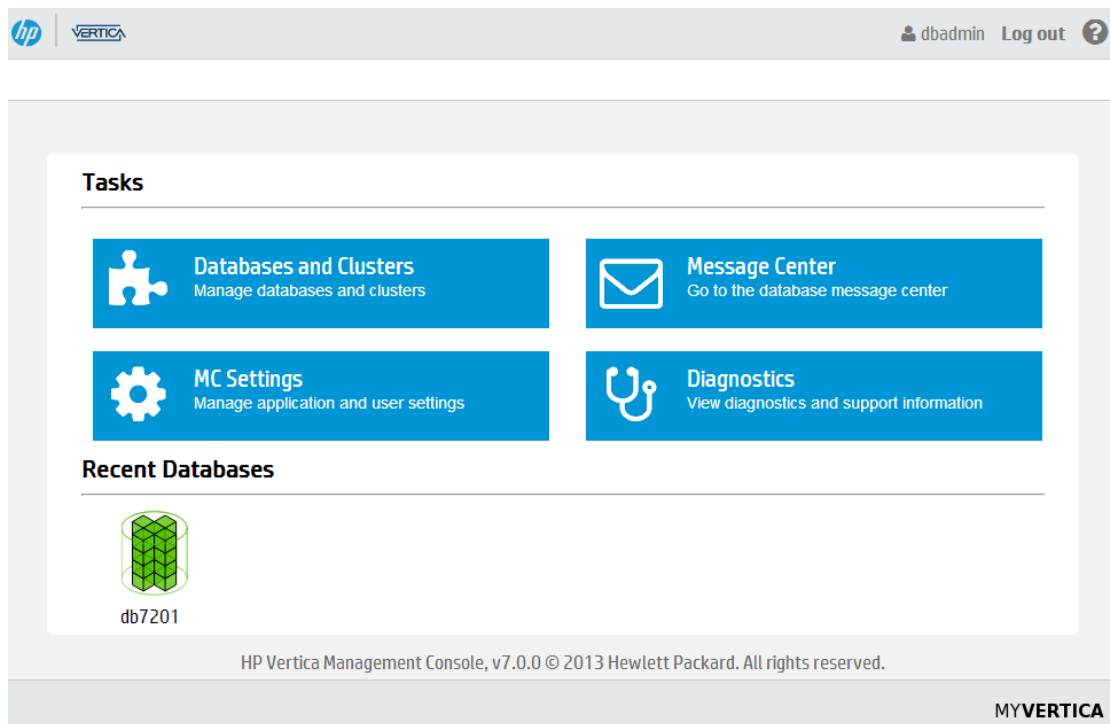
MC charts update dynamically with text, color, and messages Management Console receives from the **agents** on the database cluster. This information can help you quickly resolve problems.

Each client session on MC uses a connection from `MaxClientSessions`, a database configuration parameter that determines the maximum number of sessions that can run on a single database cluster node. If multiple MC users are mapped to the same database account and are concurrently monitoring the Overview and Activity pages, graphs could be slow to update while MC waits for a connection from the pool.

Tip: You can increase the value for `MaxClientSessions` on an MC-monitored database to take extra sessions into account. See [Managing Sessions](#) for details.

Viewing MC Home Page

After you [connect to MC](#) and sign in, the **Home** page displays. This page is the entry point to all MC-managed HP Vertica database clusters and users. Information on this page, as well as throughout the MC interface, will appear or be hidden, based on the permissions ([access levels](#)) of the user who is logged in. The following image is what an MC super administrator sees.



Tasks

Operations you can perform on MC are grouped into the following task-based areas:

- **Databases and Clusters.** Create, import, manage and monitor one or more databases on one or more clusters—this includes creating new empty databases and importing existing database clusters into the MC interface. See [Managing Database Clusters on MC](#).
- **MC Settings.** Configure MC and user settings, as well as use the MC interface to install HP Vertica on a cluster of hosts. See [Managing MC Settings](#).
- **Message Center.** View, sort, and search database messages and optionally export messages to a file. See [Monitoring MC-managed Database Messages](#).
- **Diagnostics.** View and resolve MC-related issues, as well as browse HP Vertica agent and audit logs. See [Troubleshooting Management Console](#).

Recent Databases

The area directly below **Tasks** displays all databases that you created on or imported into the MC interface. You can install one or more databases, on the same cluster or on different clusters, but you can have only one database running on a single cluster at a time. UP databases appear in green and DOWN databases are red. An empty space under Recent Databases means that you have not yet created or imported a database into the MC interface.

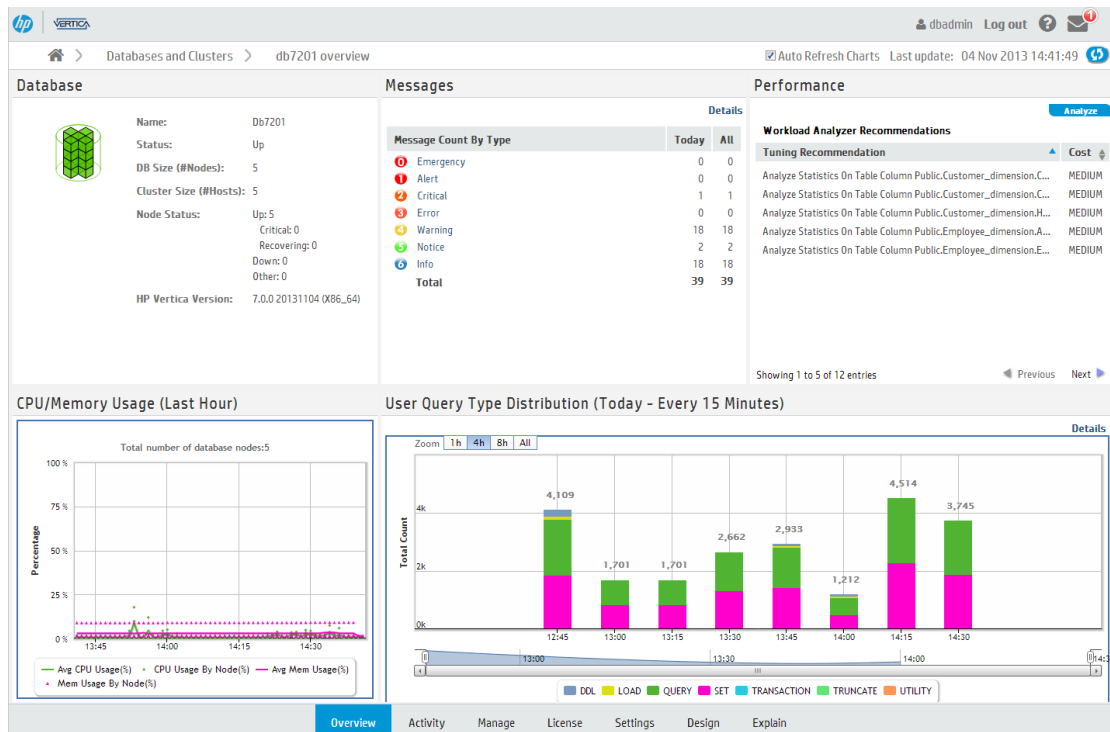
Monitoring Same-Name Databases on MC

If you are monitoring two databases with identical names on different clusters, you can determine which database is associated with which cluster by clicking the database icon on MC's Databases and Clusters page to view its dialog box. Information in the dialog displays the cluster on which the selected database is associated.

Monitoring Cluster Resources

For a dynamic, "dashboard" view of your MC-managed database cluster—including resources (CPU and memory), distribution of query type, query performance, and a summary of database messages by message type—monitor the MC **Overview** page. Information on this page updates every minute, but you can postpone updates by clearing Auto Refresh Charts in the toolbar.

The following image shows all activity on an example cluster and represents what a user with MC database administrator (**ADMIN (db)**) privileges sees. See [About MC Users](#) for more information about the MC privilege model.



The MC Overview dashboard displays information about the following cluster functionality.

Database

The Database subsection summarizes general information about the selected database cluster. Content in this area changes only if the database or node state changes; for example if the state changes from Up to Initializing or Recovering, states that indicate a recovery operation is in progress. See [Failure Recovery](#) for more information.

Messages

The Messages subsection displays counts for database-related messages, by severity type, and lets you quickly identify anything that requires immediate attention. To open the Message Center, click the Details link. See [Monitoring MC-managed Database Messages](#).

Performance

If your queries are performing suboptimally, the **Performance** subsection on the Overview page could help you identify issues by providing results from the last Workload Analyzer (WLA) run. WLA analyzes system information retained in [SQL system tables](#) and provides tuning recommendations through a SQL command you can run, along with the cost (low, medium, or high) of running the command. Workload Analyzer begins 60 seconds after MC starts and then runs once per day. If you want WLA to analyze your system workload/queries immediately, click **Analyze**. See [Analyzing Workloads](#) for more information about this utility.

CPU/Memory Usage

The CPU/Memory subsection provides a graph-based overview of cluster resources during the last hour. The chart displays the number of nodes in the database cluster and plots average and per-node percentages for both CPU and memory. Click a data point (that represents a node), and the Node Details page opens. See [Monitoring Cluster CPU/Memory](#) for more information.

User Query Type Distribution

The Query Distribution chart provides an overview of user and system query activity and reports the type of operation that ran, which MC gathers from the production database. Hover your cursor over chart points for more details. To zoom, adjust the slider bar at the bottom of the chart around an area of interest. Double clicking any point in the graph opens the Activity page. See [Monitoring Database Activity](#) for further details.

See Also

- [Monitoring Cluster Performance](#)

Monitoring Cluster Nodes

For a visual overview of all cluster nodes, click the running database on the Databases and Clusters page and then click the **Manage** tab at the bottom of the page to open the cluster status page.

The cluster status page displays the nodes in your cluster. Healthy nodes appear green. A small arrow to the right of the points upward to indicate the node is up. A node in a critical state appears yellow, and a warning symbol appears over the arrow. Nodes that are down appear red, with a red arrow to the right pointing downwards. You can get information about a particular node by clicking it, an action that opens the [node details](#) page.

Filtering What You See

If you have a large cluster, where it might be difficult to view dozens to hundreds of nodes on the MC interface, you can filter what you see. The Zoom filter shows more or less detail on the cluster overall, and the Health Filter lets you view specific node activity; for example, you can slide the bar all the way to the right to show only nodes that are down. A message next to the health filter indicates how many nodes in the cluster are hidden from view.

On this page, you can perform the following actions on your database cluster:

- Add, remove and replace nodes
- Rebalance data across all nodes
- Stop or start (or restart) the database
- Refresh the view from information MC gathers from the production database
- View key performance indicators (KPI) on node state, CPU, memory, and storage utilization (see [Monitoring Cluster Performance](#) for details)

Note: Starting, stopping, adding, and dropping nodes and rebalancing data across nodes works with the same functionality and restrictions as those same tasks performed through the **Administration Tools**.

If You don't See What You Expect

If the cluster grid does not accurately reflect the current state of the database (for example if the MC interface shows a node in INITIALIZING state, but when you use the Administration Tools to View Database Cluster State, you see that all nodes are UP), click the Refresh button in the toolbar. Doing so forces MC to immediately synchronize with the **agents** and update MC with new data.

Don't press the F5 key, which redisplay the page using data from MC and ignores data from the agent. It can take several seconds for MC to enable all database action buttons.

See Also

- [Monitoring Node Activity](#)

Monitoring Cluster CPU/Memory

On the MC Overview page, the **CPU/Memory** subsection provides a graph-based overview of cluster resources during the last hour, which lets you quickly monitor resource distribution across nodes.

This chart plots average and per-node percentages for both CPU and memory with updates every minute—unless you clear Auto Refresh Charts in the toolbar. You can also filter what the chart displays by clicking components in the legend at the bottom of the subsection to show/hide those components. Yellow data points represent individual nodes in the cluster at that point in time.

Investigating Areas of Concern

While viewing cluster resources, you might wonder why resources among nodes become skewed. To zoom in, use your mouse to drag around the problem area surrounding the time block of interest.

After you release the mouse, the chart refreshes to display a more detailed view of the selected area. If you hover your cursor over the node that looks like it's consuming the most resources, a dialog box summarizes that node's percent usage.

For more information, click a data point (node) on the graph to open MC's node details page. To return to the previous view, click **Reset zoom**.

See Also

- [Monitoring Node Activity](#)

Monitoring Cluster Performance

Key Performance Indicators (KPIs) are a type of performance measurement that let you quickly view the health of your database cluster through MC's **Manage** page. These metrics, which determine a node's color, make it easy for you to quickly identify problem nodes.

Metrics on the database are computed and averaged over the latest 30 seconds of activity and dynamically updated on the cluster grid.

How to Get Metrics on Your Cluster

To view metrics for a particular state, click the menu next to the **KPI View** label at the bottom of the Manage page, and select a state.

MC reports KPI scores for:

- **Node state**—(default view) shows node status (up, down, k-safety critical) by color; you can filter which nodes appear on the page by sliding the health filter from left to right
- **CPU Utilization**—average CPU utilization
- **Memory Utilization**—average percent RAM used
- **Storage Utilization**—average percent storage used

After you make a selection, there is a brief delay while MC transmits information back to the requesting client. You can also click **Sync** in the toolbar to force synchronization between MC and the client.

Node Colors and What They Mean

Nodes in the database cluster appear in color. Green is the most healthy and red is the least healthy, with varying color values in between.

Each node has an attached information dialog box that summarizes its score. It is the score's position within a range of 0 (healthiest) to 100 (least healthy) that determines the node's color *bias*. Color bias means that, depending on the value of the health score, the final color could be slightly biased; for example, a node with score 0 will be more green than a node with a score of 32, which is still within the green range but influenced by the next base color, which is yellow. Similarly, a node with a score of 80 appears as a dull shade of red, because it is influenced by orange.

MC computes scores for each node's color bias as follows:

- 0-33: green and shades of green
- 34-66: yellow and shades of yellow
- 67-100: red and shades of red shades

If the unhealthy node were to consume additional resources, its color would change from a dull orange-red to a brighter red.

Filtering Nodes From the View

The health filter is the slider in the lower left area of page. You can slide it left to right to show or hide nodes; for example, you might want to hide nodes with a score smaller than a certain value so the UI displays only the unhealthy nodes that require immediate attention. Wherever you land on the health filter, an informational message appears to the right of the filter, indicating how many nodes are hidden from view.



Filtering is useful if you have many nodes and want to see only the ones that need attention, so you can quickly resolve issues on them.

Monitoring System Resources

MC's **Activity** page provides immediate visual insight into potential problem areas by giving you graph-based views of query performance by runtime in milliseconds, current load rates, real-time hardware and memory impacts, the type of query or operation that the user or system ran, concurrently-running processes, and system bottlenecks on nodes.

Select one of the following charts in the toolbar menu:

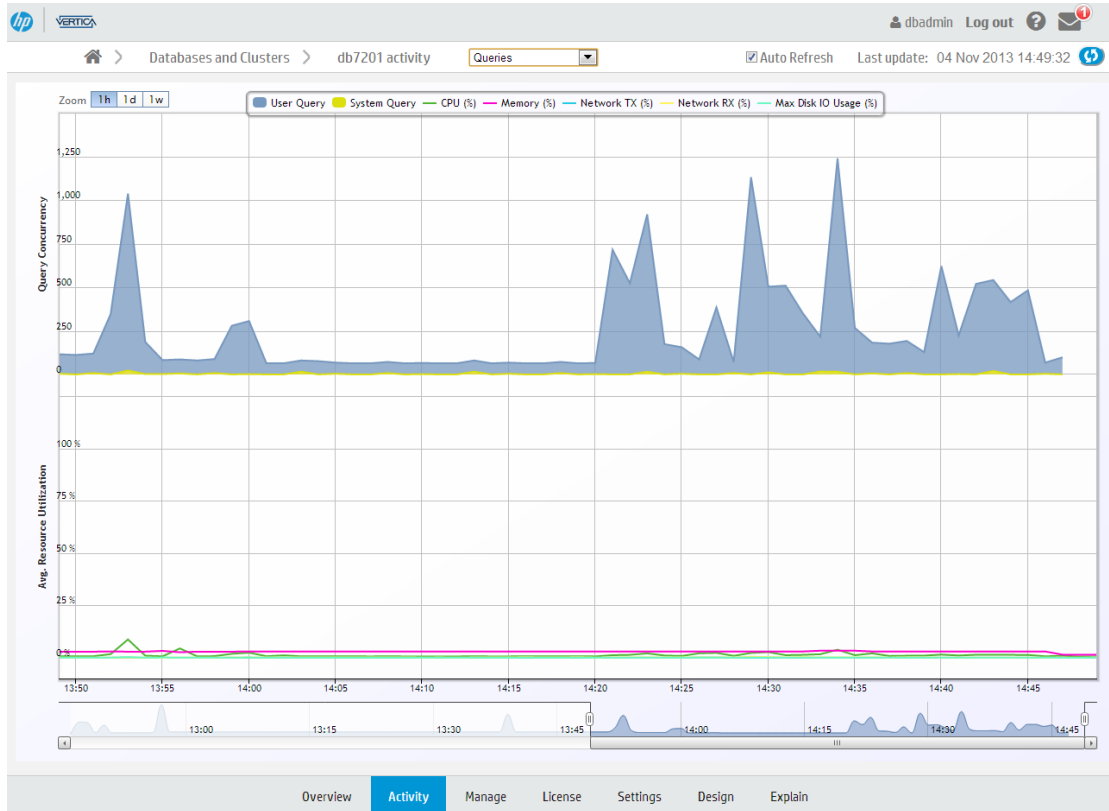
- [Queries](#)
- [Internal Sessions](#)
- [User Sessions](#)
- [Memory Usage](#)
- [Bottleneck Usage](#)

How up to Date Is the information?

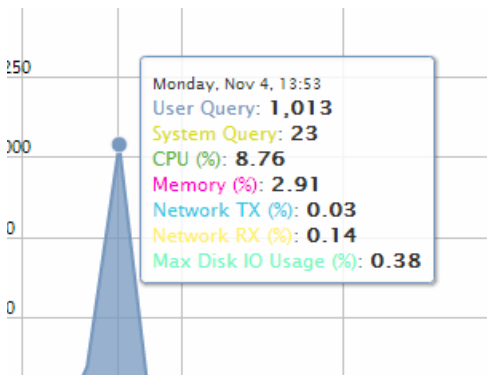
System-level activity charts automatically update every five minutes, unless you clear Auto Refresh in the toolbar. Depending on your system, it could take several moments for the charts to display when you first access the page or change the kind of resource you want to view.

Monitoring Query Activity

The Queries chart displays information about query concurrency and average resource usage for CPU/memory, network activity, and disk I/O percent based on maximum rated bandwidth.



Hover over a data point for more information about percent usage for each of the resource types.



If you click a data point, MC opens a details page for that point in time, summarizing the number of user queries and system queries. This page can help you identify long-running queries, along with the query type. You can sort table columns and export the report to a file.

Monitoring Key Events

On the main Queries page, MC reports when a key event occurred, such as a Workload Analyzer or rebalance operation, by posting a **WLA** (Workload Analyzer) and/or **RBL** (rebalance) label on the resource section of the chart.

Filtering Chart Results

The default query concurrency is over the last hour. The chart automatically refreshes every five minutes, unless you clear the Auto Refresh option in the toolbar. You can filter results for 1 hour, 1 day, or up to 1 week, along with corresponding average resource usage. You can also click different resources in the legend to show or hide those resources.

To return to the main Queries page, use the slider bar or click the 1h button.

Viewing More Detail

To zoom in for detail, click-drag the mouse around a section or use the sliding selector bar at the bottom of the chart. After the detailed area displays, hover your cursor over a data point to view the resources anchored to that point in time.

For more detail about user or system queries, click a data point on one of the peaks. A **Detail** page opens to provide information about the queries in tabular format, including the query type, session ID, node name, query type, date, time, and the actual query that ran.

The bottom of the page indicates the number of queries it is showing on the current page, with Previous and Next buttons to navigate through the pages. You can sort the columns and export contents of the table to a file.

To return to the main Queries page, click **<database name> Activity** in the navigation bar.

Monitoring Internal Sessions

The Internal Sessions chart provides information about HP Vertica system activities, such as Tuple Mover and rebalance cluster operations, along with their corresponding resources, such as CPU/memory, networking, and disk I/O percent used.

Hover your cursor over a bar for more information. A dialog box appears and provides details.

Filtering Chart Results

You can filter what the chart displays by selecting options for the following components. As you filter, the **Records Requested** number changes:

- **Category**—Filter which internal session types (moveout, mergeout, rebalance cluster) appear in the graph. The number in parentheses indicates how many sessions are running on that operation.
- **Session duration**—Lists time, in milliseconds, for all sessions that appear on the graph. The minimum/maximum values on which you can filter (0 ms to *n* ms) represent the minimum/maximum elapsed times within all sessions currently displayed on the graph. After you choose a value, the chart refreshes to show only the internal sessions that were greater than or equal to the value you select.
- **Records requested**—Represents the total combined sessions for the Category and Session Duration filters.

Monitoring User Sessions

The User Sessions chart provides information about HP Vertica user activities for all user connections open to MC.

What Chart Colors Mean

Sessions are divided into two colors, yellow and blue.

- Yellow bars represent user (system) sessions. If you click a yellow bar, MC opens a Detail page that shows all queries that ran or are still running within that session.
- Blue bars represent user requests (transactions within the session). If you click a blue section in the graph, MC opens a Detail page that includes information for that query request only.

When you hover your mouse over a transaction bar, a dialog box provides summary information about that request, such as which user ran the query, how long the transaction took to complete, or whether the transaction is still running.

Chart Results

Extremely busy systems will show a lot of activity on the interface, perhaps more than you can interpret at a glance. You can filter session results by selecting a smaller number of records, a specific user, or by changing the session duration (how long a session took to run).

As you apply filters, the Records requested number changes to represent the total combined sessions based on your filters:

- Show records—decide how many records to show in increments of 10.
- User—select the user for which you want to show sessions.
- Session duration—list time, in milliseconds, for all the sessions that appear on the graph. The minimum/maximum values on which you can filter (0 ms to n ms) represent the minimum/maximum elapsed times within all sessions currently displayed on the graph. After you choose a value, the chart refreshes to show only the sessions that were greater than or equal to the value you select.
- Records requested—represents the total combined user sessions for the User and Session duration filters.

Monitoring System Memory Usage

The Memory Usage chart shows how system memory is used on individual nodes over time. Information the chart displays is stored based on **Data Collector** retention policies, which a superuser can configure. See [Configuring Data Retention Policies](#).

The first time you access the Memory Usage chart, MC displays the first node in the cluster. MC remembers the node you last viewed and displays that node when you access the Activity page again. To choose a different node, select one from the Nodes drop-down list at the bottom of the chart. The chart automatically refreshes every five minutes unless you disable the Auto Refresh option.

Tip: On busy systems, the Node list might cover part of the graph you want to see. You can move the list out of the way by dragging it to another area on the page.

Types of System Memory

The Memory Usage chart displays a stacking area for the following memory types:

- swap
- free
- fcache (file cache)

- buffer
- other (memory in use by all other processes running on the system besides the main Vertica process, such as the MC process or **agents**)
- HP Vertica
- rcache (HP Vertica ROS cache)
- catalog

When you hover over a data point, a dialog box displays percentage of memory used during that time period for the selected node.

Monitoring System Bottlenecks

The System Bottlenecks chart helps you quickly locate performance bottlenecks on a particular node. The first time you access the Activity page, MC displays the first node in the cluster. To choose a different node, select one from the Nodes drop-down list at the bottom of the chart.

How MC Gathers System Bottleneck Data

Every 15 minutes, MC takes the maximum percent values from various system resources and plots a single line with a data point for the component that used the highest resources at that point in time. When a different component uses the highest resources, MC displays a new data point and changes the line color to make the change in resources obvious. Very busy databases can cause frequent changes in the top resources consumed, so you might notice heavy chart activity.

The Components MC Reports on

MC reports maximum percent values for the following system components:

- Average percent CPU usage
- Average percent memory usage
- Maximum percent disk I/O usage
- Percent data sent over the network (TX)
- Percent data received over the network (RX)

How MC Handles Conflicts in Resources

If MC encounters two metrics with the same maximum percent value, it displays one at random. If two metrics are very close in value, MC displays the higher of the two.

Note: The System Bottlenecks chart reports what MC identifies as the *most* problematic resource during a given time interval. This chart is meant to be a starting point for further investigation and cannot represent everything going on in the system.

Example

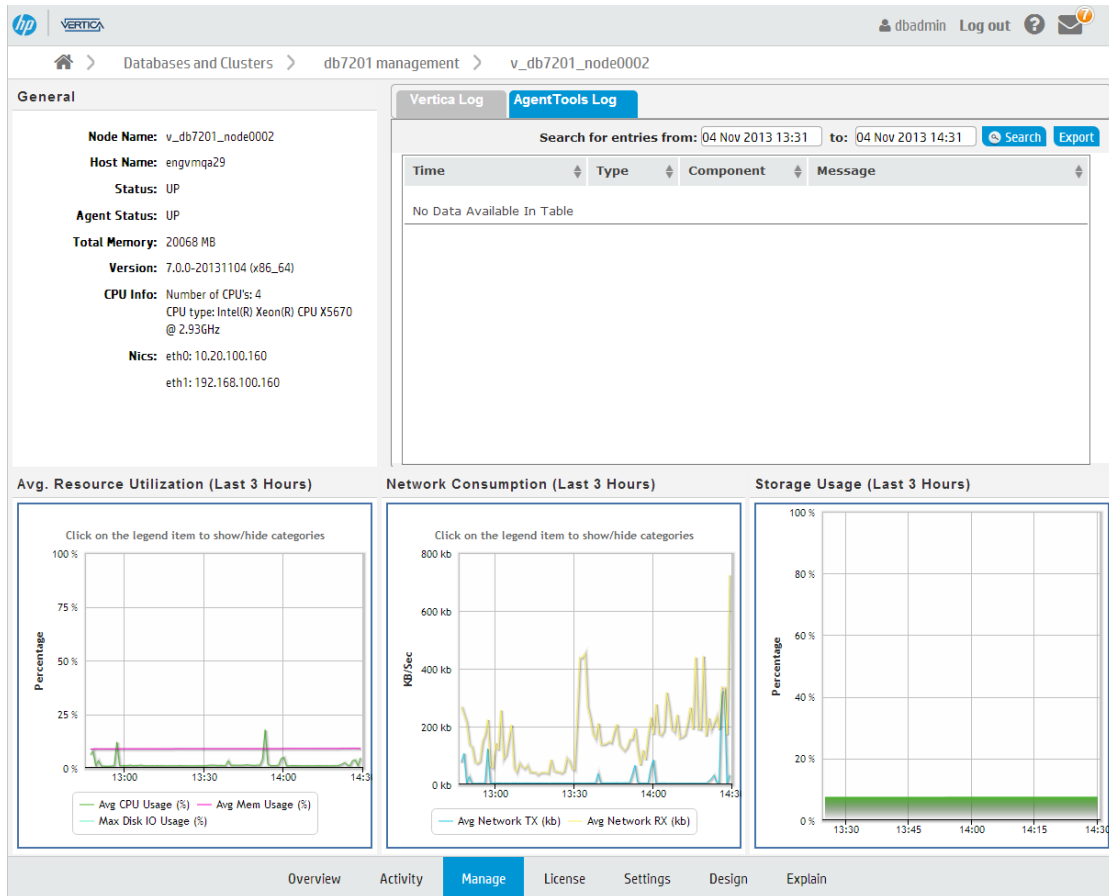
The following image shows that, over the last three hours, the most-used resources on the selected node were disk I/O, CPU, and memory. Line colors map to the components in the legend.

According to the above chart, just before 07:40 the maximum resources used changed from CPU to memory, denoted by a change from green to pink and a new data point. The following image provides a closeup of the change.

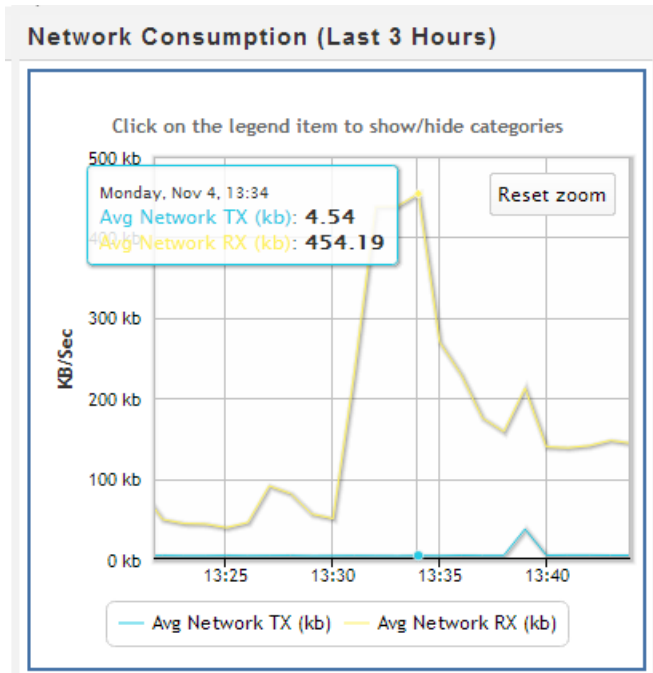
Monitoring Node Activity

If a node fails on an MC-managed cluster or you notice one node is using higher resources than other cluster nodes—which you might observe when monitoring the [Overview page](#)—open the **Manage** page and click the node you want to investigate.

The Node Details page opens and provides summary information for the node (state, name, total memory, and so on), as well as resources the selected node has been consuming for the last three hours, such as average CPU, memory, disk I/O percent usage, network consumption in kilobytes, and the percentage of disk storage the running queries have been using. You can also browse and export log-level data from AgentTools and Vertica log files. MC retains a maximum of 2000 log records.



For a more detailed view of node activity, use the mouse to drag-select around a problem area in one of the graphs, such as the large spike in network traffic in the above image. Then hover over the high data point for a summary.



See Also

- [Monitoring Cluster Resources](#)
- [Monitoring Cluster Performance](#)

Monitoring MC-managed Database Messages

You can view an overview of your database-related messages on the Overview page.

| Messages | | |
|-----------------------|-----------|-----------|
| | Details | |
| Message Count By Type | Today | All |
| ① Emergency | 0 | 0 |
| ① Alert | 0 | 0 |
| ② Critical | 1 | 1 |
| ③ Error | 0 | 0 |
| ④ Warning | 17 | 17 |
| ⑤ Notice | 2 | 2 |
| ⑥ Info | 17 | 17 |
| Total | 37 | 37 |

If you click the Details link MC opens the **Message Center**, which lists up to 500 of the most current messages and reports on the following database-related conditions:

- Low disk space
- Read-only file system
- Loss of **K-safety**
- Current fault tolerance at critical level
- Too many **ROS containers**
- **WOS** overflow
- Change in node state
- Recovery error
- Recovery failure
- Recovery lock error
- Recovery projection retrieval error
- Refresh error
- Refresh lock error

- **Workload Analyzer** operations
- **Tuple Mover** error
- Timer service task error
- **Last Good Epoch (LGE)** lag
- License size compliance
- License term compliance

Message Severity

Messages are color coded to represent the level of severity. For example:

- Red - Emergency, Alert, and Critical
- Orange - Error
- Yellow - Warning
- Green - Notice
- Blue - Informational

The icons that correspond with a message on the interface use the same color conventions.

Viewing Message Details

Within the Message Center, if you click the arrow next to a message, you can get more information about the issue, which can help you determine what action to take, if any.

Tip: You can also query the [V_MONITOR.EVENT_CONFIGURATIONS](#) table to get information about events. See the SQL Reference Manual for details.

Search and Export Messages

You can also [search](#) and sort database messages, mark messages read/unread and delete them, filter messages by message type, and [export](#) messages.

Searching MC-managed Database Messages

The Management Console Message Center displays the 500 most recent database messages, starting with the most recent record. If fewer than 500 records exist, MC displays all of them. If more than 500 exist, MC returns a message letting you know that only the most recent 500 entries are being shown, so you have the option to filter.

By default, MC reports on all message types (Emergency, Alert, Critical, Error, Warning, Notice) except Info. If you want to view Info messages, you must select that type, but because Info messages could quickly reach the 500-message limit, the Message Center might not have room to show other message types. You can filter what message types the Message Center returns.

Changing Message Search Criteria

To specify which messages to retrieve, click **Change Search** and choose a date range, one or more message types, or one or more message types within a specific date range.



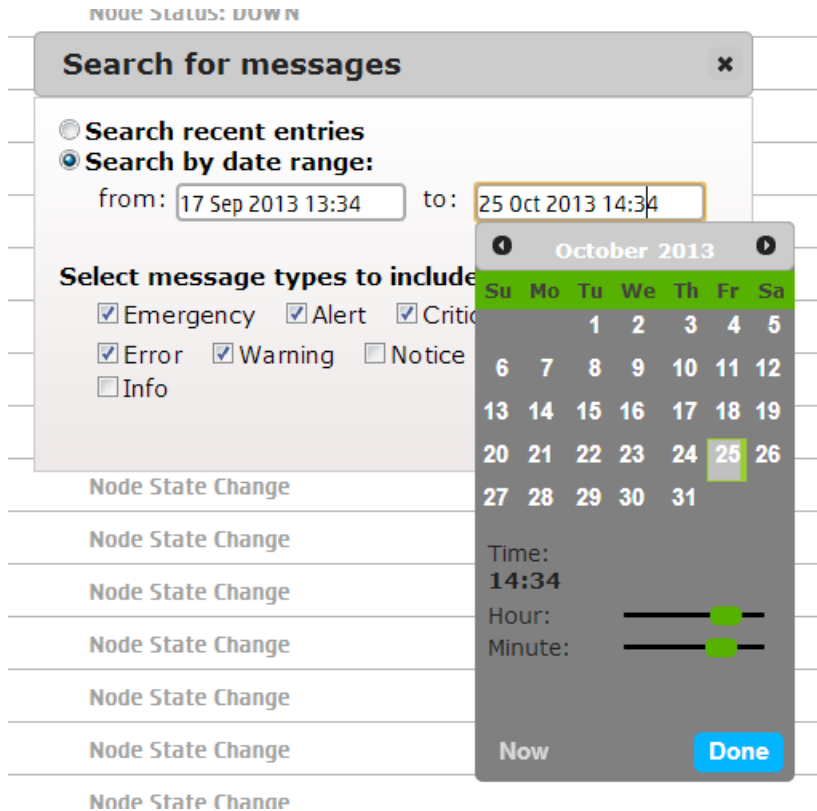
After you click **OK**, the Message Center returns up to 500 messages and updates the toolbar with your search criteria. By default, the Message Center displays, "showing recent entries, all alert types," but if you requested specific message types the toolbar displays "alert types filtered." The toolbar will also show the date range if you supplied one.

Specifying Date Range Searches

For date range searches, MC starts at the beginning of the time range and either returns all messages up to the specified end time or 500 messages, whichever comes first. You can filter message searches on the following date ranges:

- Any date-to-date period, including hour and minute
- Any time period up to now (forward searches)
- Any time period before now (backward searches)

Based on the likelihood that MC will find more than 500 messages, a date search from *<date>* to *<now>* will return a different set of messages than it would from a "search recent entries" request.



After you specify a date range, click **Done** to close the calendar, and then click **OK** to see the results in the Message Center.

Filtering Messages Client Side

In a selection pane at the bottom of the Message Center page, you can further refine search results through the client interface.



Client-side filtering works only on messages that MC has already returned, whether you can see them or not. If you select or clear message types in this filter, the toolbar status does not change. Client-side selections also do not change based on the choices you make in your server-side search. The goal of client-side filtering is to let you further refine server-side results.

Exporting MC-managed Database Messages and Logs

You can export the contents of database messages, log details, query details, and MC user activity to a file. Information comes directly from the MC interface. This means that if the last five minutes of `vertica.log` information displays on the interface, you can save that five minutes of data to a file, not the entire `vertica.log`, for which MC retains a maximum of 2000 records.

Before you can export messages/logs, you first need to search the logs. Exported log files have a `.log` extension and contain all records found within the time range you specify for the search. See [Searching MC-managed Database Messages](#) for additional details.

Depending on how you set your browser preferences, when you export messages you can view the output immediately or specify a location to save the file. System-generated filenames include a timestamp for uniqueness. If you notice a slight discrepancy between results on the MC interface and the contents of an exported file, this is because MC handles log searching (both viewing and exporting) by the minute, and a discrepancy can occur if new information comes in for the search range's end minute.

The following table shows, by record type, the MC pages that contain content you can export, the name of the system-generated file, and what that file's output contains:

| Message type | Where you can export on MC | System-generated filename | Contents of exported file |
|------------------------------|----------------------------|---|---|
| All db-related message types | Message Center page | vertica-alerts- <i><timestamp></i> .csv | Exports messages in the Message Center to a .csv file. Message contents are saved under the following headings: <ul style="list-style-type: none"> • Create time • Severity • Database • Summary (of message) • Description (more details) |
| MC log files | Diagnostics page | mconsole- <i><timestamp></i> .log | Exports MC log search results from MC to a .log file under the following headings: <ul style="list-style-type: none"> • Time • Type (message severity) • Component (such as TM, Txn, Recover, and so on) • Message |

| Message type | Where you can export on MC | System-generated filename | Contents of exported file |
|--------------|---|--|--|
| Vertica logs | <p>Manage page</p> <p>Double-click any node to get to the details and then click the VerticaLog tab.</p> | vertica-vertica- <i><db></i> - <i><timestamp></i> .log | <p>Exports vertica log search results from MC to a .log file under the following headings:</p> <ul style="list-style-type: none"> • Time • Type (message severity) • Component (such as TM, Txn, Recover, and so on) • Message |
| Agent logs | <p>Manage page</p> <p>Click any node to get to the details and then click the AgentTools Log tab.</p> | vertica-agent- <i><db></i> - <i><timestamp></i> .log | <p>Exports agent log search results from MC to a .log file under the following headings:</p> <ul style="list-style-type: none"> • Time • Type (message severity) • Component (such as TM, Txn, Recover, and so on) • Message |

| Message type | Where you can export on MC | System-generated filename | Contents of exported file |
|------------------|--|---|--|
| Query details | <p>Activity page</p> <p>Click any query spike in the graph to get to the Detail page.</p> | vertica-querydetails-<db>-<timestamp>.dat | <p>Exports query details for the database between <timestamp> and <timestamp> as a tab-delimited .dat file. Content is saved under the following headings:</p> <ul style="list-style-type: none"> • Query type • Session ID • Node name • Started • Elapsed • User name • Request/Query |
| MC user activity | <p>Diagnostics page</p> <p>Click the Audit Log task</p> | vertica_audit<timestamp>.csv | <p>Exports MC user-activity results to a .csv file. Content is saved under the following headings:</p> <ul style="list-style-type: none"> • Time • MC User • Resource • Target User • Client IP • Activity |

Monitoring MC User Activity

When an MC user makes changes on the MC interface, whether to an MC-managed database or to the MC itself, their action generates a log entry that records a timestamp, the MC user name, the

database and client host (if applicable), and the operation the user performed. You monitor user activity on the **Diagnostics > Audit Log** page.

MC records the following types of user operations:

- User log-on/log-off activities
- Database creation
- Database connection through the console interface
- Start/stop a database
- Remove a database from the console view
- Drop a database
- Database rebalance across the cluster
- License activity views on a database, as well as new license uploads
- **Workload Analyzer** views on a database
- Database password changes
- Database settings changes (individual settings are tracked in the audit record)
- Syncing the database with the cluster (who clicked Sync on grid view)
- Query detail viewings of a database
- Node changes (add, start, stop, replace)
- User management (add, edit, enable, disable, delete)
- LDAP authentication (enable/disable)
- Management Console setting changes (individual settings are tracked in the audit record)
- SSL certificate uploads
- Message deletion and number deleted
- Console restart from the browser interface
- Factory reset from the browser interface

Background Cleanup of Audit Records

An internal MC job starts every day and, if required, clears audit records that exceed a specified timeframe and size. The default is 90 days and 2K in log size. MC clears whichever limit is first reached.

You can adjust the time and size limits by editing the following lines in the `/opt/vconsole/config/console.properties` file:

```
vertica.audit.maxDays=90vertica.audit.maxRecords=2000
```

Filter and Export Results

You can manipulate the output of the audit log by sorting column headings, scrolling through the log, refining your search to a specific date/time and you can export audit contents to a file.

If you want to export the log, see [Exporting the User Audit Log](#).

If You Perform a Factory Reset

If you perform a factory reset on MC's Diagnostics page (restore it to its pre-configured state), MC prompts you to export audit records before the reset occurs.

Analyzing Workloads

If your queries are performing suboptimally, you can get tuning recommendations for them, as well as for hints about optimizing database objects, by using the Workload Analyzer (WLA).

About the Workload Analyzer

The Workload Analyzer is an HP Vertica utility that analyzes system information held in [SQL system tables](#). WLA intelligently monitors the performance of your SQL queries, workload history, resources, and configurations to identify the root causes of poor query performance.

WLA returns a set of tuning recommendations, which is based on a combination of statistics, system and **data collector** events, and database/table/projection design. These recommendations let you quickly and easily tune query performance without needing sophisticated skills.

You run the Workload Analyzer:

- By calling the [ANALYZE_WORKLOAD\(\)](#) function
- Through the [Management Console interface](#)

See [Understanding WLA Triggering Conditions](#) for the most common triggering conditions and recommendations.

Getting Tuning Recommendations Through an API

You can get Workload Analyzer to return tuning recommendations for queries and database objects by calling the [ANALYZE_WORKLOAD\(\)](#) function and passing it arguments that tell WLA exactly what and when to analyze.

What and When

The 'scope' argument tells WLA what to analyze:

- An empty string ("") returns recommendations for all database objects
- 'table_name' returns recommendations related to the specified table
- 'schema_name' returns recommendations on all database objects in the specified schema

The 'since_time' argument tells WLA when by limiting recommendations from all events that you specified in 'scope' since the time you specify in this argument, up to the current system status. If you omit the since_time parameter, ANALYZE_WORKLOAD returns recommendations on events since the last recorded time that you called ANALYZE_WORKLOAD().

You must explicitly cast strings that you use for the since_time parameter to `TIMESTAMP` or `TIMESTAMPTZ`. The following statements show four different ways of expressing the same query

using different formats for the `since_time` parameter. All four queries return the same result for workloads on table `t1` since October 4, 2010.

```
=> SELECT ANALYZE_WORKLOAD('t1', TIMESTAMP '2010-10-04 11:18:15');  
=> SELECT ANALYZE_WORKLOAD('t1', '2010-10-04 11:18:15'::TIMESTAMP);  
=> SELECT ANALYZE_WORKLOAD('t1', 'October 4, 2010'::TIMESTAMP);  
=> SELECT ANALYZE_WORKLOAD('t1', '10-04-10'::TIMESTAMP);
```

Record the Events

Instead of supplying a specific time, you can instruct WLA to record this particular call of `WORKLOAD_ANALYZER()` in the system by supplying an optional `'true'` parameter along with the scope. The default value is `false` (do not record). If recorded, subsequent calls to `ANALYZE_WORKLOAD` analyze only the events that have occurred since this recorded time, ignoring all prior events.

For example, the following statement runs WLA and returns recommendations for all database objects in all schemas and records this analysis invocation.

```
=> SELECT ANALYZE_WORKLOAD('', true);
```

The next invocation of `ANALYZE_WORKLOAD()` will analyze events from this point forward.

```
-[ RECORD 1 ]-----+-----  
observation_count   | 1  
first_observation_time |  
last_observation_time |  
tuning_parameter   | public.locations  
tuning_description  | run database designer on table public.locations  
tuning_command      |  
tuning_cost         | HIGH  
-[ RECORD 2 ]-----+-----  
observation_count   | 1  
first_observation_time |  
last_observation_time |  
tuning_parameter   | public.new_locations  
tuning_description  | run database designer on table public.new_locations  
tuning_command      |  
tuning_cost         | HIGH  
-[ RECORD 3 ]-----+-----  
observation_count   | 1  
first_observation_time |  
last_observation_time |  
tuning_parameter   | release  
tuning_description  | set password for user 'release'  
tuning_command      | alter user release identified by 'new_password'  
tuning_cost         | LOW
```

Observation Count and Time

The `observation_count` column returns an integer that represents the total number of events WLA observed for this tuning recommendation, so in each case above, WLA is making its first recommendation. Null results in the `observation_time` columns mean only that the recommendations are from the current system status instead of from a prior event.

Knowing What to Tune

The `tuning_parameter` column returns the object on which WLA recommends that you apply the tuning action. The parameter of `release` in the example above notifies the DBA to set a password for user release.

The Tuning Description (recommended action) and Command

Workload Analyzer's output returns a brief description of tasks you should consider in the `tuning_description` column, along with a SQL command you can run, where appropriate, in the `tuning_command` column. In the above fragment for RECORDS 1 and 2, WLA recommends that you run the **Database Designer** on two tables and consider setting a user's password in record 3. Note that RECORD 3 also provides the `ALTER USER` command to run because the tuning action is a SQL command.

What a Tuning Operation Costs

The output in the `tuning_cost` column is based on the type of tuning recommendation WLA has made. It can be low, medium or high:

- A **LOW** cost has minimal impact on resources from running the tuning command. You can perform the tuning operation at any time, like changing the user's password in Record 3 above.
- A **MEDIUM** cost has moderate impact on resources from running the tuning command
- A **HIGH** cost has maximum impact on resources from running the tuning command. Depending on the size of your database or table, consider running high-cost operations after hours instead of during peak load times.

Examples

The following statement tells WLA to analyze all events for a table named `locations`:

```
=> SELECT ANALYZE_WORKLOAD('locations');
```

The recommendation `s` that you run the Database Designer on the table, an operation that, depending on the size of the `locations` table, could potentially have a high cost, so you might want to run Database Designer after hours.

```

-[ RECORD 1 ]-----+-----
observation_count   | 1
first_observation_time |
last_observation_time |
tuning_parameter    | public.locations
tuning_description  | run database designer on table public.locations
tuning_command      |
tuning_cost         | HIGH
    
```

The following statement analyzes workloads on all tables in the VMart example database (see [Introducing the VMart Example Database](#)) since one week before today:

```
=> SELECT ANALYZE_WORKLOAD('', NOW() - INTERVAL '1 week');
```

Workload Analyzer found two issues which are described below the sample output.

```

-[ RECORD 1 ]-----+-----
observation_count   | 4
first_observation_time | 2012-02-17 13:57:17.799003-04
last_observation_time | 2011-04-22 12:05:26.856456-04
tuning_parameter    | store.store_orders_fact.date_ordered
tuning_description  | analyze statistics on table column store.store_orders_fact.date_
ordered
tuning_command      | select analyze_statistics('store.store_orders_fact.date_order
d');
tuning_cost         | MEDIUM
-[ RECORD 2 ]-----+-----
...
-[ RECORD 14 ]-----+-----
observation_count   | 2
first_observation_time | 2012-02-19 17:52:03.022644-04
last_observation_time | 2012-02-19 17:52:03.02301-04
tuning_parameter    | SELECT x FROM t WHERE x > (SELECT SUM(DISTINCT x) FROM
t GROUP BY y) OR x < 9;
tuning_description  | consider incremental design on query
tuning_command      |
tuning_cost         | HIGH
    
```

In RECORD 1, the `date_ordered` column in the `store.store_orders_fact` table likely has [stale statistics](#), so WLA suggests running `ANALYZE_STATISTICS()` on that column. The function output also returns the query to run. For example:

```
=> SELECT ANALYZE_STATISTICS('store.store_orders_fact.date_ordered');
```

In RECORD 14, WLA identifies an underperforming query and recommends that you use the Database Designer to run an incremental design. This information is shown in the `tuning_parameter` column. Note that the potential cost of running this operation is HIGH.

This statement analyzes workloads on all tables two days before today (now):

```
=> SELECT ANALYZE_WORKLOAD('', NOW() - '2 DAYS'::INTERVAL);
```


Getting Recommendations From System Tables

You can also get tuning recommendations by querying system table [V_MONITOR.TUNING_RECOMMENDATIONS](#), which returns tuning recommendation results from the last `ANALYZE_WORKLOAD()` call.

```
=> SELECT * FROM tuning_recommendations;
```

System information that WLA uses for its recommendations is held in [SQL system tables](#), so querying the `TUNING_RECOMMENDATIONS` system table does not run the Workload Analyzer.

Understanding WLA's Triggering Events

For information about what operations trigger WLA to make a tuning recommendation, see [Understanding WLA Triggering Conditions](#).

See Also

- [Collecting Database Statistics](#)

Getting Tuning Recommendations Through MC

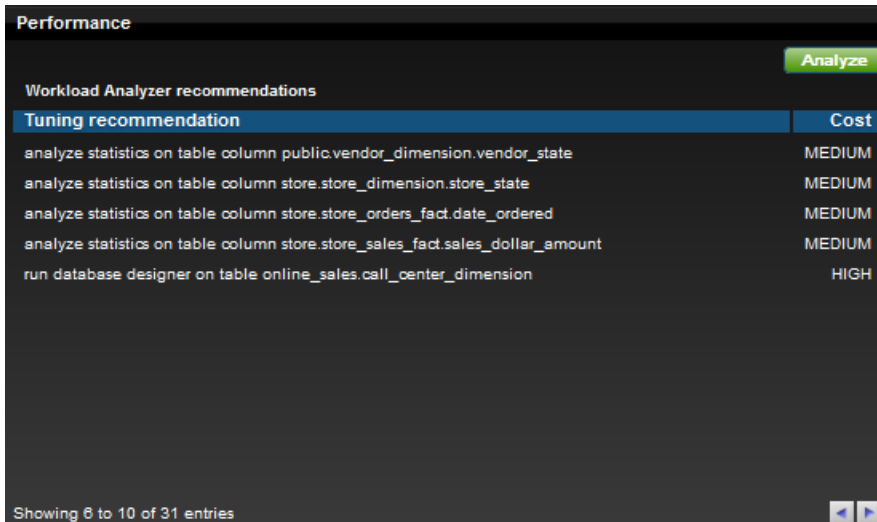
On MC, the Workload Analyzer automatically begins monitoring data one minute after the MC process starts. WLA then runs once per day, or immediately after you add a database to the MC interface, where it continually gathers data in the background, as long as the database is running. If you haven't created a database yet, or if the database is down, WLA does nothing until the database is back up.

WLA recommendations are available on the MC **Overview** page, which is the 3x2 grid that provides a snapshot of the database, cluster, performance, alerts, jobs, and license activity.

The tuning recommendations are in the Performance subsection, along with their cost

The following image shows a short list of WLA tuning recommendations. In the output, WLA suggests that you run the [ANALYZE_STATISTICS\(\)](#) function on specific columns in four different tables and provides the cost estimate of running these operations. WLA also advises that you run the Database Designer on one of the tables.

Note that the cost of running Database Designer is HIGH. When you see a high cost, you might want to run the recommended tuning action after hours. For additional information about tuning recommendations and cost see [ANALYZE_WORKLOAD\(\)](#) in the SQL Reference Manual.



The screenshot shows a table titled "Performance" with a sub-header "Workload Analyzer recommendations". A green "Analyze" button is in the top right. The table has two columns: "Tuning recommendation" and "Cost". It lists five recommendations with their respective costs. At the bottom left, it says "Showing 6 to 10 of 31 entries" and at the bottom right, there are navigation arrows.

| Tuning recommendation | Cost |
|---|--------|
| analyze statistics on table column public.vendor_dimension.vendor_state | MEDIUM |
| analyze statistics on table column store.store_dimension.store_state | MEDIUM |
| analyze statistics on table column store.store_orders_fact.date_ordered | MEDIUM |
| analyze statistics on table column store.store_sales_fact.sales_dollar_amount | MEDIUM |
| run database designer on table online_sales.call_center_dimension | HIGH |

Management Console displays five recommendations per page and includes a message at bottom left, which lets you know where you are within the total number of recommendations; for example, "Showing *n-nn* of *nn* entries." Depending on the total number of recommendations, you can move forward and backward through them by clicking the arrows at bottom right.

Tip: You can force the WLA task to run immediately by clicking **Analyze** over the Cost column.

For additional information about tuning recommendations and their triggering event, see [Understanding WLA Triggering Conditions](#).

See Also

- [Getting Tuning Recommendations Through an API](#)
- [ANALYZE_WORKLOAD](#)

Understanding WLA Triggering Conditions

Workload Analyzer (WLA) monitors suspicious system activity and makes recommendations based on its observations. When you run Workload Analyzer, the utility returns the following information:

- The tuning description
- The objects on which WLA applies tuning action
- The suggested SQL command for you to run (where appropriate)

In rare circumstances, tuning recommendation WLA proposes might not resolve the underlying problem. Because you might occasionally need help understanding WLA's recommendations, the

following table lists some of the most common triggering conditions, along with the recommendations to resolve the issue and a pointer to more information, when available.

| The triggering condition | What to do about it | Where to get more information |
|--|---|--|
| Internal configuration parameter is not the same across nodes. | Reset configuration parameter. <pre>SELECT set_config_parameter ('parameter ', 'new_value');</pre> | See SET_CONFIG_PARAMETER in the SQL Reference Manual |

| The triggering condition | What to do about it | Where to get more information |
|---|---|---|
| <p>An unused projection meets the following conditions:</p> <ul style="list-style-type: none"> • No queries on projection for more than 30 days but, projection's anchor table has been queried more than 10 times • Projection's anchor table is not a temp or system table • Projection's table is not small, where the number of bytes of disk storage in use by | <p>Drop the projection (<i>projection-name</i>).</p> <pre>DROP PROJECTION public.T1_fact_super_P1_B1;</pre> | <p>See the following topics in the SQL Reference Manual:</p> <ul style="list-style-type: none"> • DROP PROJECTION • PROJECTION_STORAGE • PROJECTIONS • TABLES |

| The triggering condition | What to do about it | Where to get more information |
|---|---|--|
| the projection (used_bytes) is more than 10M | | |
| User with dbadmin/pseudosuperuser role has empty password. | Set the password for user. <pre>ALTER USER (user) IDENTIFIED BY ('new_password');</pre> | See ALTER USER in the SQL Reference Manual |
| Table with too many partition count. | Alter the table's partition expression. <pre>ALTER TABLE (schema.table) PARTITION BY (new_partition_expression) REORGANIZE;</pre> | See ALTER TABLE in the SQL Reference Manual |
| LGE threshold setting is lower than the default setting. | Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support. | |
| Tuple Mover's MoveOutInterval parameter is set greater than the default setting. | Decrease the MoveOutInterval configuration parameter setting. <pre>SELECT set_config_parameter('MoveOutInterval',default_value);</pre> | See the following topics: <ul style="list-style-type: none"> • Monitoring Events • Tuple Mover Parameters • ACTIVE_EVENTS in the SQL Reference Manual |

| The triggering condition | What to do about it | Where to get more information |
|---|---|-------------------------------|
| Tuple Mover has been disabled. | Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support. | |
| Too Many ROS containers since the last mergeout operation; configuration parameters are set lower than the default. | Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support. | |
| Too Many ROS containers since the last mergeout operation; the TM Mergeout service is disabled. | Workload Analyzer does not trigger a tuning recommendation for this scenario unless you altered settings and/or services under the guidance of technical support. | |

| The triggering condition | What to do about it | Where to get more information |
|---|---|---|
| Average CPU usage exceeds 95% for 20 minutes. | Check system processes or change the settings for PLANNEDCONCURRENCY and/or MAXCONCURRENCY for the RESOURCE POOL. | See the following topics: <ul style="list-style-type: none"> • Guidelines for Setting Pool Parameters • ALTER RESOURCE POOL and Built-In Pool Configuration in the SQL Reference Manual |
| Partitioned table data is not fully reorganized after repartitioning. | Reorganize data in partitioned table public.T1. <pre>ALTER TABLE public.T1 REORGANIZE;</pre> | See ALTER TABLE in the SQL Reference Manual |
| Table has multiple partition keys within the same ROS container. | Reorganize data in partitioned table public.T1. <pre>ALTER TABLE public.T1 REORGANIZE;</pre> | See ALTER TABLE in the SQL Reference Manual |

| The triggering condition | What to do about it | Where to get more information |
|---|--|---|
| Excessive swap activity; average memory usage exceeds 99% for 10 minutes. | Check system processes | |
| A table does not have any Database Designer-designed projections. | Run database designer on table public.T1. | See Creating an Incremental Design Using the |
| Statistics are stale (no histogram or predicate falls outside histogram). | Run ANALYZE_STATISTICS() on table column public.<table>.<column> <pre>SELECT analyze_statistics ('public.t.a');</pre> | See the following topics: <ul style="list-style-type: none"> • ANALYZE_STATISTICS() in the SQL Reference Manual • Collecting Statistics in this guide |
| Data distribution in segmented projection is skewed. | Re-segment projection public.t_super on high-cardinality column (s). | See Designing for Segmentation |

| The triggering condition | What to do about it | Where to get more information |
|--------------------------|--|--|
| GROUP BY spill event. | Consider running an incremental design on query. | See Creating an Incremental Design Using the |

See Also

- [ANALYZE_WORKLOAD](#)

Collecting Database Statistics

The HP Vertica cost-based query optimizer relies on representative statistics on the data, which it uses to determine the final plan to execute a query.

Various optimizer decisions rely on having up-to-date statistics, where it chooses between:

- Multiple eligible projections to answer the query
- The best order in which to perform joins
- Plans involving different algorithms, such as hash join/merge join or group by hash/group by pipelined operations
- Data distribution algorithms, such as broadcast and re-segmentation

Without reasonably accurate statistics, the optimizer could choose a suboptimal projection or a suboptimal join order for a query.

To understand how HP Vertica collects statistics, consider the following common scenario:

Scenario: You have a large table into which you load timestamp on an ongoing basis (hourly, daily, etc.). Then you run queries that select the recently-loaded rows from that table.

How the optimizer decides on a plan: You load days 1 through 15 into the table and run the [ANALYZE_STATISTICS\(\)](#) function. When you next run a query that requests yesterday's data by filtering on the timestamp column, the optimizer chooses an optimized plan. If on the next day, you load day 16 data and run the same query—but you do not run [ANALYZE_STATISTICS\(\)](#) again—the optimizer might conclude that the predicate results in only one row being returned because the date range falls outside the **histogram** range and the data becomes stale. When the optimizer detects that statistics are not current for a particular predicate (such as when a timestamp predicate is outside a histogram's boundary), HP Vertica plans those queries using other considerations, such as FK-PK constraints, when available.

Resolution: Run [ANALYZE_STATISTICS\(\)](#) after you load new data, or day 16 in this example. You can also look for statistics in the [EXPLAIN](#) plan; for example, when statistics are off outside a histogram's boundaries, the [EXPLAIN](#) plan is annotated with a status. See [Reacting to Stale Statistics](#) for details.

See the SQL Reference Manual for additional details about functions and system tables described in the topics in this section.

| For information about how to... | See ... |
|--|--------------------------------------|
| Collect a fixed-size statistical data sampling | ANALYZE_HISTOGRAM() |
| Collect a specified percentage of disk data sampling | ANALYZE_STATISTICS() |
| Run a utility that analyzes information held in system tables and returns query tuning recommendations | ANALYZE_WORKLOAD() |
| Generate an XML file that contains statistics for the database | EXPORT_STATISTICS() |

| For information about how to... | See ... |
|---|--|
| Import statistics from the XML file generated by the EXPORT_STATISTICS command | IMPORT_STATISTICS() |
| Remove statistics for a table and optionally specify the category of statistics to drop | DROP_STATISTICS() |
| Monitor information about projection columns, such as encoding type, sort order, type of statistics, and the time at which columns statistics were last updated | V_CATALOG.PROJECTION_COLUMNS |

Statistics Used By the Query Optimizer

HP Vertica uses the estimated values of the following statistics in its cost model:

- Number of rows in the table
- Number of distinct values of each column (**cardinality**)
- Minimum/maximum values of each column
- An equi-height **histogram** of the distribution of values each column
- Space occupied by the column on disk

Note: The HP Vertica query optimizer and the **Database Designer** both use the same set of statistics. When there are ties, the optimizer chooses the projection that was created first.

How Statistics Are Collected

Statistics collection is a cluster-wide operation that accesses data using a **historical query** (at epoch latest) without any locks. Once computed, statistics are stored in the catalog and replicated on all nodes. The storage operation requires a brief, exclusive lock on the catalog, similar to when a DDL operation occurs. In fact, these operations require a COMMIT for the current transaction.

HP Vertica provides three ways to manually collect statistics:

- ANALYZE ROW COUNT
- ANALYZE_STATISTICS
- ANALYZE_HISTOGRAM

Using the ANALYZE ROW COUNT Operation

The ANALYZE ROW COUNT is a lightweight operation that automatically collects the number of rows in a projection every 60 seconds to collect a minimal set of statistics and aggregate row counts

calculated during loads. You can use the `AnalyzeRowCountInterval` configuration parameter to change the default collection interval (60 seconds). See [Configuration Parameters](#) for additional information.

To change the 60-second interval to 1 hour (3600 seconds), use the following command:

```
=> SELECT SET_CONFIG_PARAMETER('AnalyzeRowCountInterval', 3600);
```

To reset the interval to the default of 1 minute (60 seconds):

```
=> SELECT SET_CONFIG_PARAMETER('AnalyzeRowCountInterval', 60);
```

You can also invoke this function manually using the `DO_TM_TASK('analyze_row_count')` function.

Using ANALYZE_STATISTICS

The `ANALYZE_STATISTICS` function computes full statistics on all objects, or on a per-table or per-column basis. You must invoke this function explicitly.

The `ANALYZE_STATISTICS()` function:

- Lets you analyze tables on a per-column basis for improved performance.
- Performs faster data sampling, which expedites the analysis of relatively small tables with a large number of columns.
- Includes data from **WOS**.
- Recognizes deleted data, instead of ignoring delete markers.
- Lets you cancel the function mid analysis by issuing CTRL-C on vsql or invoking the `INTERRUPT_STATEMENT()` function.
- Records the last time statistics were run for a table so that subsequent calls to the function can be optimized. See [V_CATALOG.PROJECTION_COLUMNS](#) for details.

Using ANALYZE_HISTOGRAM

`ANALYZE_STATISTICS()` is an alias for `ANALYZE_HISTOGRAM()`. The difference between the two functions is that `ANALYZE_HISTOGRAM` lets you specify what percentage of data to read from disk, so you have more control over deciding between sample accuracy and speed.

The `ANALYZE_HISTOGRAM percent` parameter specifies the amount of column data (from 1 - 100%) that the function reads from disk. The default value is 10%, but you can use this parameter to specify a smaller or larger percentage. Changing the percent value affects both the data collection time and the histogram accuracy:

- A smaller percent value reads less data off disk. Data collection is faster than with a larger value, but histogram accuracy *decreases*, since the function samples less data.
- A larger percent value reads more data off disk. Data collection takes longer than for a smaller value, but the histogram accuracy *increases*, since the function samples a larger percentage of collected data.

Regardless of the `percent` parameter value, `ANALYZE_HISTOGRAM` uses at most 128K (128,000) rows of column data. This sample size (128K) consistently creates an accurate representation of the sample data, even for columns with more than 1,000,000 rows. The function constructs a histogram for a column by randomly selecting from the collected data. If the percent value you specify equates to less than 128K rows, the function rounds the number to at least 128K rows, so enough data to sample is read from disk. If a column has less than 128K rows, `ANALYZE_HISTOGRAM` reads all rows from disk and analyzes the entire column.

Note: The sample data collected in a sample range is not indicative of how data should be distributed.

Following are some examples of different size columns with the `ANALYZE_HISTOGRAM` `percent` parameter set to the different values:

| Column Size | Percent | Read from disk | Sampled rows |
|----------------|---------|----------------|---------------|
| <128K rows | 20 | All rows | Entire column |
| 400,000 rows | 10 | 128,000 | 128K rows |
| 4,000,000 rows | 10 | 400,000 | 128K rows |

Note: If the column that you specify for the `ANALYZE_HISTOGRAM` function is first in a projection's sort order, the function reads all data from disk to avoid a biased sample.

Examples

In this example, the `ANALYZE_STATISTICS()` function reads 10 percent of the disk data. This is the static default value for this function. The function returns 0 for success:

```
=> SELECT ANALYZE_STATISTICS('shipping_dimension.shipping_key');
ANALYZE_STATISTICS
-----
                        0
(1 row)
```

This example uses `ANALYZE_HISTOGRAM ()` without specifying a percentage value. Since this function has a default value of 10 percent, it returns the identical data as the `ANALYZE_STATISTICS()` function, and returns 0 for success:

```
=> SELECT ANALYZE_HISTOGRAM('shipping_dimension.shipping_key');  
ANALYZE_HISTOGRAM  
-----  
0  
(1 row)
```

This example uses `ANALYZE_HISTOGRAM()`, specifying its percent parameter as 100, indicating it will read the entire disk to gather data. After the function performs a full column scan, it returns 0 for success:

```
=> SELECT ANALYZE_HISTOGRAM('shipping_dimension.shipping_key', 100);  
ANALYZE_HISTOGRAM  
-----  
0  
(1 row)
```

In this command, only 0.1% (1/1000) of the disk is read:

```
=> SELECT ANALYZE_HISTOGRAM('shipping_dimension.shipping_key', 0.1);  
ANALYZE_HISTOGRAM  
-----  
0  
(1 row)
```

How Statistics Are Computed

HP Vertica does not compute statistics incrementally or update full statistics during load operations.

For large tables that exceed 250,000 rows, **histograms** for minimum, maximum, and column value distribution are calculated on a sampled subset of rows. The default maximum number of samples for each column is approximately 2^{17} (131702) samples, or the number of rows that fit in 1GB of memory, whichever is smaller. For example, there could be fewer samples used for large VARCHAR columns.

HP Vertica does not provide a configuration setting to change the number of samples for analysis. However, you can decide between a faster or more accurate data sampling by specifying what percentage of data to read from disk, from 1 to 100 (a full table scan).

See [ANALYZE_HISTOGRAM\(\)](#) in the SQL Reference Manual and [How Statistics Are Collected](#) for information about using the `ANALYZE_HISTOGRAM` function.

How Statistics Are Reported

HP Vertica supplies hints about statistics in a couple ways:

- The EXPLAIN plan is annotated with a status. See [Reacting to Stale Statistics](#).
- The last time ANALYZE_STATISTICS() was run for a table is recorded, so that subsequent calls to the function are optimized. This is useful during the database design process because if the Database Designer does not collect statistics when adding design tables, it generates a warning indicating that statistics are old. You can then decide whether to run ANALYZE_STATISTICS before you proceed with the design.

Two columns in the [V_CATALOG.PROJECTION_COLUMNS](#) system table capture statistical information, as follows:

- STATISTICS_TYPE—Returns the type of statistics the column contains (NONE, ROWCOUNT or FULL).
- STATISTICS_COLLECTION_TIME—Returns the last time statistics were collected in this table.

Determining When Statistics Were Last Updated

The [V_CATALOG.PROJECTION_COLUMNS](#) system table returns information about projection columns, including the type of statistics, and the the time at which column statistics were last updated.

The following example illustrates how you can examine the run status for statistics on your tables.

On a single-node cluster, the following sample schema defines a table named trades, which groups the highly-correlated columns bid and ask and stores the stock column separately:

```
=> CREATE TABLE trades (stock CHAR(5), bid INT, ask INT);
=> CREATE PROJECTION trades_p (stock ENCODING RLE, GROUPED(bid ENCODING
    DELTAVAL, ask)) AS (SELECT * FROM trades) ORDER BY stock, bid;
=> INSERT INTO trades VALUES('acme', 10, 20);
=> COMMIT;
```

Query the PROJECTION_COLUMNS table for table trades:

```
=> \x
Expanded display is on.
=> SELECT * FROM PROJECTION_COLUMNS WHERE table_name = 'trades';
```

Notice that the statistics_type column returns NONE for all three columns in the trades table. Also, there is no value in the statistics_updated_timestamp field because statistics have not yet been run on this table.

```
-[ RECORD 1 ]-----+-----
projection_id      | 45035996273718838
projection_name     | trades_p
```

```
projection_column_name | stock
column_position        | 0
sort_position          | 0
column_id              | 45035996273718840
data_type              | char(5)
encoding_type          | RLE
access_rank            | 0
group_id               | 0
table_schema           | public
table_id               | 45035996273718836
table_name              | trades
table_column_id        | 45035996273718836-1
table_column_name      | stock
statistics_type         | NONE
statistics_updated_timestamp |
-[ RECORD 2 ]-----+-----
projection_id          | 45035996273718838
projection_name         | trades_p
projection_column_name | bid
column_position        | 1
sort_position          | 1
column_id              | 45035996273718842
data_type              | int
encoding_type          | DELTAVAL
access_rank            | 0
group_id               | 45035996273718844
table_schema           | public
table_id               | 45035996273718836
table_name              | trades
table_column_id        | 45035996273718836-2
table_column_name      | bid
statistics_type         | NONE
statistics_updated_timestamp |
-[ RECORD 3 ]-----+-----
projection_id          | 45035996273718838
projection_name         | trades_p
projection_column_name | ask
column_position        | 2
sort_position          |
column_id              | 45035996273718846
data_type              | int
encoding_type          | AUTO
access_rank            | 0
group_id               | 45035996273718844
table_schema           | public
table_id               | 45035996273718836
table_name              | trades
table_column_id        | 45035996273718836-3
table_column_name      | ask
statistics_type         | NONE
statistics_updated_timestamp |
```

Now run statistics on the stock column:

```
=> SELECT ANALYZE_STATISTICS('trades.stock');
```

The system returns 0 for success:


```
-[ RECORD 1 ]-----+--
ANALYZE_STATISTICS | 0
```

Now query PROJECTION_COLUMNS again:

```
=> SELECT * FROM PROJECTION_COLUMNS where table_name = 'trades';
```

This time, `statistics_type` changes to `FULL` for the `trades.stock` column (representing full statistics were run), and the `statistics_updated_timestamp` column returns the time the stock columns statistics were updated. The timestamp for the `bid` and `ask` columns have not changed because statistics were not run on those columns. Also, the `bid` and `ask` columns changed from `NONE` to `ROWCOUNT`. This is because HP Vertica automatically updates `ROWCOUNT` statistics from time to time. The statistics are created by looking at existing catalog metadata.

```
-[ RECORD 1 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | stock
column_position       | 0
sort_position         | 0
column_id             | 45035996273718840
data_type             | char(5)
encoding_type         | RLE
access_rank           | 0
group_id              | 0
table_schema          | public
table_id              | 45035996273718836
table_name            | trades
table_column_id       | 45035996273718836-1
table_column_name     | stock
statistics_type        | FULL
statistics_updated_timestamp | 2012-12-08 13:52:04.178294-05
-[ RECORD 2 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | bid
column_position       | 1
sort_position         | 1
column_id             | 45035996273718842
data_type             | int
encoding_type         | DELTAVAL
access_rank           | 0
group_id              | 45035996273718844
table_schema          | public
table_id              | 45035996273718836
table_name            | trades
table_column_id       | 45035996273718836-2
table_column_name     | bid
statistics_type        | ROWCOUNT
statistics_updated_timestamp | 2012-12-08 13:51:20.016465-05
-[ RECORD 3 ]-----+-----
projection_id          | 45035996273718838
projection_name        | trades_p
projection_column_name | ask
```

```

column_position      | 2
sort_position       |
column_id           | 45035996273718846
data_type           | int
encoding_type       | AUTO
access_rank         | 0
group_id            | 45035996273718844
table_schema        | public
table_id            | 45035996273718836
table_name          | trades
table_column_id     | 45035996273718836-3
table_column_name   | ask
statistics_type     | ROWCOUNT
statistics_updated_timestamp | 2012-12-08 13:51:20.016475-05

```

If you run statistics on the `bid` column and then query this system table again, only RECORD 2 is updated:

```

=> SELECT ANALYZE_STATISTICS('trades.bid');
-[ RECORD 1 ]-----+--
ANALYZE_STATISTICS | 0
=> SELECT * FROM PROJECTION_COLUMNS where table_name = 'trades';
-[ RECORD 1 ]-----+-----
projection_id       | 45035996273718838
projection_name     | trades_p
projection_column_name | stock
column_position    | 0
sort_position      | 0
column_id          | 45035996273718840
data_type          | char(5)
encoding_type      | RLE
access_rank        | 0
group_id           | 0
table_schema       | public
table_id           | 45035996273718836
table_name         | trades
table_column_id    | 45035996273718836-1
table_column_name  | stock
statistics_type    | FULL
statistics_updated_timestamp | 2012-12-08 13:52:04.178294-05
-[ RECORD 2 ]-----+-----
projection_id       | 45035996273718838
projection_name     | trades_p
projection_column_name | bid
column_position    | 1
sort_position      | 1
column_id          | 45035996273718842
data_type          | int
encoding_type      | DELTAVAL
access_rank        | 0
group_id           | 45035996273718844
table_schema       | public
table_id           | 45035996273718836
table_name         | trades
table_column_id    | 45035996273718836-2
table_column_name  | bid

```

```

statistics_type           | FULL
statistics_updated_timestamp | 2012-12-08 13:53:23.438447-05
-[ RECORD 3 ]-----+-----
projection_id            | 45035996273718838
projection_name          | trades_p
projection_column_name   | ask
column_position         | 2
sort_position           |
column_id               | 45035996273718846
data_type               | int
encoding_type          | AUTO
access_rank            | 0
group_id               | 45035996273718844
table_schema           | public
table_id               | 45035996273718836
table_name             | trades
table_column_id        | 45035996273718836-3
table_column_name      | ask
statistics_type        | ROWCOUNT
statistics_updated_timestamp | 2012-12-08 13:51:20.016475-05

```

You can quickly query just the timestamp column to see when the columns were updated:

```

=> \x
Expanded display is off.
=> SELECT ANALYZE_STATISTICS('trades');
ANALYZE_STATISTICS
-----
                                0
(1 row)
=> SELECT projection_column_name, statistics_type,
       statistics_updated_timestamp
       FROM PROJECTION_COLUMNS where table_name = 'trades';
projection_column_name | statistics_type | statistics_updated_timestamp
-----+-----+-----
stock                 | FULL           | 2012-12-08 13:54:27.428622-05
bid                   | FULL           | 2012-12-08 13:54:27.428632-05
ask                   | FULL           | 2012-12-08 13:54:27.428639-05
(3 rows)

```

See [V_CATALOG.PROJECTION_COLUMNS](#) in the SQL Reference Manual for more information.

Reacting to Stale Statistics

During predicate selectivity estimation, the query optimizer can identify when **histograms** are not available or are out of date. If the value in the predicate is outside the histogram's maximum range, the statistics are stale. If no histograms are available, then no statistics are available to the plan.

When the optimizer detects stale or no statistics, such as when it encounters a column predicate for which it has no histogram, the optimizer takes these actions:

- Generates a message (and log), recommending that you run [ANALYZE_STATISTICS\(\)](#).
- Annotates [EXPLAIN](#) plans with a statistics entry.

- Ignores the stale statistics when it generates a query. Here, the optimizer executes queries using other considerations, such as FK-PK constraints, when available.

The following EXPLAIN fragment shows no statistics (histograms unavailable):

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 604, Rows: 10K (NO STATISTICS)]
```

The following EXPLAIN fragment shows that the predicate falls outside the histogram range:

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 35, Rows: 1 (PREDICATE VALUE OUT-OF-RANGE)]
```

You can get information about which table column has no statistics by querying a system table; for example, view the timestamp for when statistics were last run by querying [V_CATALOG.PROJECTION_COLUMNS](#).

Example

First run full statistics on table 'trades':

```
=> SELECT ANALYZE_STATISTICS('trades'); ANALYZE_STATISTICS
-----
                                0
(1 row)
```

Next, query the `projection_column_name`, `statistics_type`, and `statistics_updated_timestamp` columns:

```
=> SELECT projection_column_name, statistics_type, statistics_updated_timestamp
FROM PROJECTION_COLUMNS where table_name = 'trades';
projection_column_name | statistics_type | STATISTICS_UPDATED_TIMESTAMP
-----+-----
stock                  | FULL           | 2011-03-31 13:39:16.968177-04
bid                    | FULL           | 2011-03-31 13:39:16.96885-04
ask                    | FULL           | 2011-03-31 13:39:16.968883-04
(3 rows)
```

You can also query the [V_CATALOG.PROJECTIONS.HAS_STATISTICS](#) column, which returns true only when all non-epoch columns for a table have full statistics. Otherwise the column returns false.

See Also

- [Analyzing Workloads](#)
-
-

Canceling Statistics Collection

To cancel statistics collection mid analysis, execute CTRL-C on **vsq** or call the [INTERRUPT_STATEMENT\(\)](#) function.

If you want to remove statistics for the specified table or type, call the [DROP_STATISTICS\(\)](#) function.

Caution: After you drop statistics, it can be time consuming to regenerate them.

Best Practices for Statistics Collection

The query optimizer requires representative statistics in order to choose the best query plan. For most applications, statistics need not be accurate to the minute. The ANALYZE ROW COUNT operation automatically collects partial statistics and supplies sufficient data for many optimizer choices. You can also invoke this operation by calling the [DO_TM_TASK \(\)](#) function and passing it the 'analyze_row_count' argument. For example, the following command analyzes the row count on the [Vmart Schema](#) database:

```
vmart=> SELECT DO_TM_TASK('analyze_row_count');
          DO_TM_TASK-----
-----
row count analyze for projection 'call_center_dimension_DBD_27_seg_temp_init_temp_init'
row count analyze for projection 'call_center_dimension_DBD_28_seg_temp_init_temp_init'
row count analyze for projection 'online_page_dimension_DBD_25_seg_temp_init_temp_init'
row count analyze for projection 'online_page_dimension_DBD_26_seg_temp_init_temp_init'
row count analyze for projection 'online_sales_fact_DBD_29_seg_temp_init_temp_init'
row count analyze for projection 'online_sales_fact_DBD_30_seg_temp_init_temp_init'
row count analyze for projection 'customer_dimension_DBD_1_seg_temp_init_temp_init'
row count analyze for projection 'customer_dimension_DBD_2_seg_temp_init_temp_init'
row count analyze for projection 'date_dimension_DBD_7_seg_temp_init_temp_init'
row count analyze for projection 'date_dimension_DBD_8_seg_temp_init_temp_init'
row count analyze for projection 'employee_dimension_DBD_11_seg_temp_init_temp_init'
row count analyze for projection 'employee_dimension_DBD_12_seg_temp_init_temp_init'
row count analyze for projection 'inventory_fact_DBD_17_seg_temp_init_temp_init'
row count analyze for projection 'inventory_fact_DBD_18_seg_temp_init_temp_init'
row count analyze for projection 'product_dimension_DBD_3_seg_temp_init_temp_init'
row count analyze for projection 'product_dimension_DBD_4_seg_temp_init_temp_init'
row count analyze for projection 'promotion_dimension_DBD_5_seg_temp_init_temp_init'
row count analyze for projection 'promotion_dimension_DBD_6_seg_temp_init_temp_init'
row count analyze for projection 'shipping_dimension_DBD_13_seg_temp_init_temp_init'
row count analyze for projection 'shipping_dimension_DBD_14_seg_temp_init_temp_init'
row count analyze for projection 'vendor_dimension_DBD_10_seg_temp_init_temp_init'
row count analyze for projection 'vendor_dimension_DBD_9_seg_temp_init_temp_init'
row count analyze for projection 'warehouse_dimension_DBD_15_seg_temp_init_temp_init'
row count analyze for projection 'warehouse_dimension_DBD_16_seg_temp_init_temp_init'
row count analyze for projection 'store_dimension_DBD_19_seg_temp_init_temp_init'
row count analyze for projection 'store_dimension_DBD_20_seg_temp_init_temp_init'
row count analyze for projection 'store_orders_fact_DBD_23_seg_temp_init_temp_init'
row count analyze for projection 'store_orders_fact_DBD_24_seg_temp_init_temp_init'
row count analyze for projection 'store_sales_fact_DBD_21_seg_temp_init_temp_init'
```

```
row count analyze for projection 'store_sales_fact_DBD_22_seg_temp_init_temp_init'  
(1 row)
```

Running full ANALYZE_STATISTICS on a table is an efficient but potentially long-running operation that analyzes each unique column exactly once across all projections. You can run it concurrently with queries and loads in a production environment.

When to Gather Full Statistics

Because statistics gathering consumes resources (CPU and memory) from queries and loads, HP recommends that you run full ANALYZE_STATISTICS() under the following conditions:

- The table is first loaded (see [Bulk Loading Data](#)).
- A new projection using a newly-loaded table is created and [refreshed](#). Projections that have no data never have full statistics. Use the [PROJECTION_STORAGE](#) system table to see if your projection contains data.
- The number of rows in the table changes by 50%.
- The minimum/maximum values in the table's columns change by 50%.
- New primary key values are added to tables with referential integrity constraints. When this occurs, both the primary key and foreign key tables should be reanalyzed.
- The relative table size, compared to tables it is being joined to, changes materially. For example, a table becomes only five times larger than the other, when it was previously 50 times larger.
- There is a significant deviation in the distribution of data, which necessitates recalculating **histograms**. For example, an event causes abnormally high levels of trading for a particular stock. This is application specific.
- There is a down-time window when the database is not in active use.

Tip: You can analyze statistics on a single table column, rather than on the entire table. Running statistics on a single important column (such as the predicate column) is useful for large tables, which could take a long time to compute. It's also a good idea to run statistics on a column after you use ALTER TABLE to add or change a column.

Also consider using the [ANALYZE_HISTOGRAM\(\)](#) function. Whereas ANALYZE_STATISTICS uses a fixed value of 10 percent for the proportion of disk reads, ANALYZE_HISTOGRAM lets you specify what percent of the disk to read. You can also diagnose and resolve many statistics-related issues by calling the [ANALYZE_WORKLOAD\(\)](#) function, which returns tuning recommendations.

If you update statistics and find that the query still performs poorly, run your query through the Database Designer and choose incremental as the design type. See [Creating an Incremental Design Using the](#) .

Save Statistics

Once your system is running well, HP recommends that you save exported statistics for all tables. In the unlikely scenario that statistics changes impact optimizer plans, particularly after an upgrade, you can always revert back to the exported statistics. See [Importing, exporting and modifying statistics](#) for details.

The following topics in the SQL Reference Manual:

See Also

- [Analyzing Workloads](#)
- [Collecting Database Statistics](#)
- [Determining When Statistics Were Last Updated](#)
- [Reacting to Stale Statistics](#)
-
-
-
-
-

Using Diagnostic Tools

HP provides several diagnostic tools. In this section, you'll learn how to identify which version of HP Vertica you are running, use the diagnostics tools, and export a catalog and profiling data.

Determining Your Version of HP Vertica

To determine which version of HP Vertica is installed on a host, log in to that host and type:

```
$ rpm -qa | grep vertica
```

The command returns the name of the installed package, which contains the version and build numbers. The following example indicates that both HP Vertica 7.0.x and Management Console 7.0.x are running on the targeted host:

```
[dbadmin@myhost01 ~]$ rpm -qa | grep vertica  
vertica-7.0.0-20131125.x86_64  
vertica-console-7.0.0-0.x86_64
```

When you are logged in to your Vertica Analytics Platform database, you can also run a query for the version only, by running the following command:

```
dbadmin=> SELECT version();  
          version  
-----  
Vertica Analytic Database v7.0.0-20131124  
(1 row)
```

Collecting Diagnostics (scrutinize Command)

The diagnostics script, `scrutinize`, collects a broad range of diagnostic information from all the nodes in the Vertica Analytics Platform environment, such as:

- Host diagnostics
- Log files from the installation process, the database, and the administration tools (e.g., `vertica.log`, `dbLog`, `admintools.log`)
- Unstructured HP Vertica logs
- Structured database event information
- License size
- System table information, such as queries executed and their run time

- Catalog metadata, such as statistics

Note: Although `scrutinize` might collect individual values stored in statistics, the script does not gather samples of actual data in the database.

- Host configuration data
- Run-time state (number of nodes up or down)
- Database schema
- Backup information
- Important error messages

Syntax

```
/opt/vertica/bin/scrutinize [ arguments ... ]
```

Command Line Arguments

The following table provides both a long form and short form for most arguments.

| Arguments | Description |
|--|--|
| <code>--help,</code> <code>-h</code> | Outputs all <code>scrutinize</code> arguments to the console. |
| <code>--local_diags,</code> <code>-s</code> | [Default all hosts] If you use this argument, <code>scrutinize</code> gathers diagnostics for the specified host. Query the V_MONITOR.NODE_RESOURCES system table for information about hosts in your cluster. |

| Arguments | Description |
|---|---|
| <pre>--database dbname, -d dbname</pre> | <p>Gathers diagnostics for the specified database. If your database has a password, you must also supply the <code>-P PASSWORD</code> argument. If you omit the <code>--database</code> argument, <code>scrutinize</code> collects diagnostics on the running database; otherwise:</p> <ul style="list-style-type: none"> • If no database is running but that database is defined in the Administration Tools metadata, <code>scrutinize</code> collects diagnostics on the stopped database. • If more than one database is defined but none is running, <code>scrutinize</code> returns an error. You must supply the <code>--database</code> argument for the appropriate database. • If multiple databases are running (unsupported in production environments), <code>scrutinize</code> returns an error. Only one database can be running during diagnostics collection. <p>Tip: Always use the <code>--database</code> argument if you have more than one database defined on the cluster.</p> |
| <pre>--hosts HOST_LIST, -n HOST_LIST</pre> | <p>[Default all cluster hosts] Gathers diagnostics for the specified hosts.</p> |
| <pre>--message MESSAGE, -m MESSAGE</pre> | <p>[Default no message] Include a user-supplied message in the diagnostics output file, such as the reason for gathering/submitting diagnostics, along with a support-supplied case number, or anything that might help your Vertica Analytics Platform support contact troubleshoot your case. There is no character limit. The message argument accepts the following inputs:</p> <ul style="list-style-type: none"> • <code>--message "my message"</code> • <code>--message PROMPT</code> • <code>--message "/path/to/file"</code> <p>See scrutinize Examples for different inputs using the <code>--message</code> argument.</p> |
| <pre>--output_dir OUTPUT_DIR, -o OUTPUT_DIR</pre> | <p>[Default current directory] Redirects output to a location that is different from the current directory.</p> |

| Arguments | Description |
|--|---|
| --user USERNAME, -U USERNAME | [Default current user] Specifies the database user, which you must pass to the scrutinize command if the Vertica Analytics Platform administrative user is different from the operating system dbadmin user (e.g., you specified a name different from the default dbadmin user when you installed Vertica Analytics Platform). |
| --password PASSWORD -P PASSWORD | [Default no password] Specifies the database password, which is required if your database has a password. If you omit this argument on a password-protected database, scrutinize fails with a run-time error. Note: You must include the --password argument if the Vertica Analytics Platform administrator account (default dbadmin, set during an Vertica Analytics Platform installation) has a non-blank password. You can omit the --database argument if the database is running or if there is only one database defined on the cluster. Otherwise, you must also include the --database argument with the --password argument. |
| --type TYPE, -t TYPE | Specifies the type of diagnostics collection to perform. Options are: <ul style="list-style-type: none"> • basic – [default] omit profiling data • profiling – gather profiling data • context – gather important data |
| --tasks TASKS, -T TASKS | [Experts only] In collaboration with your Vertica Analytics Platform technical support contact, the --tasks argument instructs scrutinize to gather diagnostics for one or more specified tasks in a file or JSON list. |
| --tmpdir TMP_DIR | [Default /tmp] Specifies the tmp directory to use on all nodes. |
| --exclude-tasks EXCLUDE_TASKS -X EXCLUDE_TASKS [all n] | [Experts only] In collaboration with your Vertica Analytics Platform technical support contact, instructs scrutinize to exclude diagnostics gathering from one or more specified tasks. Using 'all' excludes all default tasks. |
| --url URL -U URL | [Default none (do not upload)] Posts output of diagnostics collection to an Vertica Analytics Platform support-provided URL, which can be http or ftp. |

| Arguments | Description |
|--|---|
| <code>--include_gzlogs NUMGZ</code> <code>-z</code> <code>[all n]</code> | [Default 1] Includes <i>n</i> number of gzipped <code>vertica.log-*.gz</code> files within the <code>VerticaScrutinize</code> file. To collect all logs, specify 'all.' |

Privileges

The database administrator runs `scrutinize`. If you use `root` when the `dbadmin` user exists, the system returns an error like the following:

```
[root@host01 ~]# /opt/vertica/bin/scrutinize
Root user is not allowed to use this tool.
Try again as the DB administrative user.
```

The `root` user can run `scrutinize` if the `dbadmin` account has not yet been created and you need to collect diagnostics on the HP Vertica installation process.

How to Run Scrutinize

Although the database does not need to be running, `scrutinize` gathers more comprehensive information if the database is up, which it does with minimal disruption to operations.

To collect general diagnostics about your Vertica Analytics Platform environment, run `scrutinize` without arguments:

```
$ /opt/vertica/bin/scrutinize
```

In this scenario, the script collects a broad range of diagnostic information from all the nodes in the cluster, described in [Collecting Diagnostics \(scrutinize Command\)](#). While such output can diagnose most issues, it intentionally lacks minute profiling data in order to reduce the upload size.

If you want to limit the information `scrutinize` collects, pass one or more arguments to the script. Typically, your Vertica Analytics Platform support contact will provide you with arguments, including the upload URL.

How Scrutinize Collects and Packages Diagnostics

When you run `scrutinize`, the script performs the following operations:

1. Collects information from nodes in the database cluster
2. Stages diagnostics data on individual nodes
3. Moves diagnostics data from cluster nodes to the initiator node (the server where you ran the script)
4. Packages individual files on the initiator node into the final, tree-structured

VerticaScrutinize*.zip file, which is saved to the current directory, although you can specify a different directory with the `--output_dir` argument.

Files take the following name, where the `<timestamp>` portion is denoted in `yyyymmddh24mms` format, a naming convention that changes with each subsequent scrutinize run; for example:

```
VerticaScrutinize.<timestamp>.zip  
VerticaScrutinize.20130118132609.zip  
VerticaScrutinize.20130131143508.zip
```

While Scrutinize Runs

As scrutinize runs, it sends the following information to the terminal window:

- User
- Database
- Location of the diagnostics output file
- Other information, depending on the arguments you pass to the script

The following example output is similar to what you might see, depending on the arguments you provide:

```
Vertica Scrutinize Report  
-----  
Result Dir:           /home/dbadmin/VerticaScrutinize.20131126081105  
User:                 dbadmin  
Database:             mcdb  
Running remote worker on v_mcdb_node0001 @ /tmp/VerticaScrutinizeLocal.20131126081105/v_m  
cdb_node0001  
Running remote worker on v_mcdb_node0003 @ /tmp/VerticaScrutinizeLocal.20131126081105/v_m  
cdb_node0003  
Running remote worker on v_mcdb_node0004 @ /tmp/VerticaScrutinizeLocal.20131126081105/v_m  
cdb_node0004  
Cleaning up on node v_mcdb_node0001 (10.20.300.003)  
Cleaning up on node v_mcdb_node0003 (10.20.300.004)  
Cleaning up on node v_mcdb_node0004 (10.20.300.005)  
Gathered diagnostics for  
Customer: Vertica Systems, Inc.  
Database designation: FAMOUS GOLD  
Timestamp: 20131126081105  
All results are stored in /home/dbadmin/VerticaScrutinize.20131126081105.zip
```

After Scrutinize Finishes Running

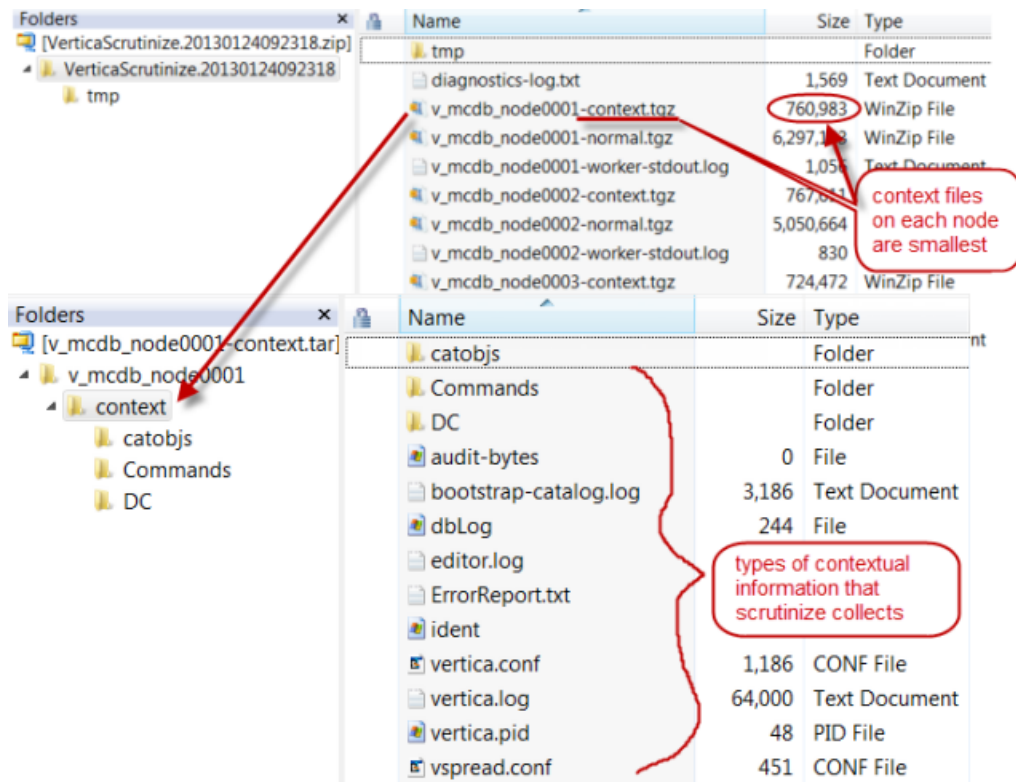
At the end of diagnostics collection, scrutinize displays a "All results are stored in `<path>`" message. This path is the location of the VerticaScrutinize*.zip file. If you passed the `--url` upload argument to the script, each node that is participating in diagnostics collection will upload

diagnostics information directly through that URL. See [How to Upload Scrutinize Results to Support](#).

Example

The following image illustrates the tree-like structure of the VerticaScrutinize*.zip file for full diagnostics collection (no arguments). Note that the file was saved to and opened on a Windows system.

- The top panel shows the top-level contents of the VerticaScrutinize*.zip file, including file names and size. The <node>-context.tgz files, which are on each node in the database cluster, are kept small to optimize upload speed. For example, if you specify the --url upload argument, each node posts the files directly to the url, sending the smaller context file first.
- The bottom panel provides an expanded view of a <node>-context.tgz file to show the types of information scrutinize collects for context.



How to Upload Scrutinize Results to Support

In most cases, your technical support contact will provide arguments for you to run with the scrutinize script, including the --url (upload) value.

Before you can run scrutinize with --url:

- The `cURL` program must be installed and in the path for the database administrator user who is running `scrutinize`
- Each node in the cluster must be able to make an `http` or `ftp` connection directly to the Internet

When you use the `--url` argument:

- `Scrutinize` avoids collecting results into a single `VerticaScrutinize<timestamp>.zip` file
- Each node posts the files directly to the url, where a smaller "context" file is posted first, which means support might be able to look at high-level information while waiting for the larger, more complete, download to finish. See [How Scrutinize Collects and Packages Diagnostics](#) for an example of a context file.

How to Refer to Your Database Cluster

Vertica Analytics Platform provides a way for you to refer to the cluster you're gathering diagnostics on when you contact support.

Toward the end of `scrutinize` output to the terminal window is a section called "Gathered diagnostics for", and within this section is a "Database designation" notation. For example, in the sample output that follows, the database designation is called `FAMOUS GOLD`. This title is randomly generated and assigned based on your HP Vertica license and the database creation time.

```
Vertica Scrutinize Report
-----
Result Dir:           /home/dbadmin/VerticaScrutinize.20131126081105
...
Gathered diagnostics for
Customer: Vertica Systems, Inc.
Database designation: FAMOUS GOLD
Timestamp: 20131126081105
...
```

The database designation gives you and technical support a point of reference, so you can quickly identify and refer to the database cluster being analyzed. The naming convention is particularly useful if you have multiple databases on the same cluster or same-name databases on different clusters.

The database designation name remains the same until you drop the database. Therefore, use the same name when you contact technical support. If you create a new database and run `scrutinize` on it, the script assigns a new randomly-generated designation name for the new database.

Example upload script

The following command is a generic upload example, which gathers all diagnostics on the cluster and posts output from the collection to the support-provided URL:

```
$ /opt/vertica/bin/scrutinize --password mypassword --url "ftp://user:password@customers.
vertica.com/"
```

For additional examples, see [scrutinize Examples](#)

Examples for the Scrutinize Command

This section provides additional examples for running the scrutinize command.

How to Include Gzipped Log Files

The following command gathers diagnostics on the mcdb database and instructs the script to include the three most recent gzipped log files:

```
$ /opt/vertica/bin/scrutinize --database mcdb --password dbpassword
--include_gzlogs 3
```

The sample mcdb database has a password, so the scrutinize command must include the `--password` argument. If you omit `--password` on password-protected databases, Vertica Analytics Platform returns an error message like the following:

```
No vertica process appears to be queryable with given settings:vsq1:
FATAL 3781: Invalid username or password
```

How to Include a Message in the Diagnostics Package

The following message-type commands use the `--message` argument to illustrate different ways you can include message text in your diagnostics file.

Note: Always include your support case number in the message file.

- Use `--message` with double-quoted text to include that message in the VerticaScrutinize output file; for example:

```
/opt/vertica/bin/scrutinize --message "my support case number and message"
```

- Use `--message` with the `PROMPT` keyword, and scrutinize reads input until you type a period [.] on a line by itself to end the message. Scrutinize then writes the message to the VerticaScrutinize file; for example:

```
/opt/vertica/bin/scrutinize --message PROMPT
Enter reason for collecting diagnostics; end with '.' on a line by itself:
```



```
Query performance degradation noticed around 9AM EST on Saturday
.
Vertica Scrutinize Report
-----
Result Dir:                /home/dbadmin/VerticaScrutinize.20131126083311
...
```

- Use `--message` with a double-quoted path to include contents of the specified message file in VerticaScrutinize file output; for example:

```
/opt/vertica/bin/scrutinize --message "/path/to/file/mycaseno-messagetosupport"
```

How to Send Results to Support

See [How to Upload Scrutinize Results to Support](#)

Collecting Diagnostics (diagnostics Command)

When you run the diagnostics utility with one or more parameters, information about your database and host configuration is exported to a `.zip` file.

Syntax

```
/opt/vertica/bin/diagnostics [ argument ... ]
```

Arguments

| | |
|---|---|
| <code>-h --help</code> | Shows a help message that contains all commands for the diagnostics utility and exits. |
| <code>-l --listdbs</code> | Lists running and non-running databases. |
| <code>-s --local_diags</code> | Gathers diagnostics for local host only. |
| <code>-d dbname--database dbname</code> | Gathers information about the specified database. |
| <code>-o OUTPUT_DIR--output_dir OUTPUT_DIR</code> | Redirects output to a location other than the default <code>/opt/vertica/log</code> . |
| <code>-n HOST_LIST,--hosts HOST_LIST</code> | Gathers diagnostics for the specified hosts. |
| <code>-z [all n]</code> | Specifies how many <code>vertica.log.*.gz</code> files to include in diagnostics. The default is 1. |

Using the Diagnostics Utility

Running `/opt/vertica/bin/diagnostics` without arguments gathers information on all databases and all nodes in the HP Vertica cluster.

When the diagnostics utility finishes running, it reports a message with the location of the .zip file, which is in `/opt/vertica/log/HP_VerticaDiagnostics.<date>.zip`. The `<date>` variable is automatically assigned a unique ID that represents the date and time you ran diagnostics. This ID changes with each diagnostics run; for example, here is the command and resulting filename for a diagnostics run on the `mddb` database:

```
$ /opt/vertica/bin/diagnostics -d mddb  
VerticaDiagnostics.20130117075507.zip
```

If you make multiple calls to `diagnostics`, the utility creates a new file with a different `<date>` field so that it does not overwrite results from the previous call. For example, if I ran the same command a few moments later, the filename is as follows:

```
VerticaDiagnostics.20130117075900.zip
```

Both the `-d <dbname>` and `-n <host list>` parameters are required to gather the HP Vertica log from a specific node. The `-d` parameter is useful if you have multiple databases on the same cluster. If you omit `-d` from the command, `diagnostics` doesn't know which database you want to examine and gathers host-level information only.

To include all `*.gz` files in diagnostics (including all databases on all hosts), use the `-z all` parameter; otherwise specify the number of files you want (for example, the last 5 `*.gz` files). The gzipped log files are included in the order in which they were last modified, with the most recent first.

The diagnostics utility uses a `diagnostics-<username>.log` file instead of the `adminTools-<username>.log` file for logging to allow for improved readability of log files collected during diagnostics. You will find both the `diagnostics-<username>.log` and `adminTools-<username>.log` files in the `/opt/vertica/log/` directory.

Tip: If you are having trouble with an installation, run the diagnostics utility as root or sudo. See [Running Diagnostics Utility for failed Installation](#). For other situations, run the diagnostics utility as the database administrator.

Examples

The following command uses the `-l` parameter, which returns a list of running and non-running databases and notifies you where you can find the output file:

```
[dbadmin@host01 ~]$ /opt/vertica/bin/diagnostics -l  
Running Diagnostics as user: dbadmin  
Vertica Diagnostics Report
```

```
-----  
Using VSQL:          /opt/vertica/bin/vsql  
Result Dir:         /opt/vertica/log/VerticaDiagnostics.20130116111121  
Listing databases...  
  Running database:  mcdb  
  Non Running database: vmart  
  Non Running database: myotherdb  
  Non Running database: onenode  
All results are stored in /opt/vertica/log/VerticaDiagnostics.20130116111121.zip
```

The following command gathers diagnostics information for the PROD01 database on host01 and host02:

```
$ /opt/vertica/bin/diagnostics -n host01.acme.com,host02.acme.com -d PROD01
```

This command includes all vertica*.gz files in the diagnostics output for the PROD01 database:

```
$ /opt/vertica/bin/diagnostics -d PROD01 -z all
```

This command includes only the last three .gz files for the PROD01 database in the diagnostics:

```
$ /opt/vertica/bin/diagnostics -d PROD01 -z 3
```

Exporting a Catalog

When you export a catalog you can quickly move a catalog to another cluster. Exporting a catalog transfers schemas, tables, constraints, projections, and views. System tables are not exported.

Exporting catalogs can also be useful for support purposes.

See the [EXPORT_CATALOG](#) function in the SQL Reference Manual for details.

Exporting Profiling Data

The diagnostics audit script gathers system table contents, design, and planning objects from a running database and exports the data into a file named `./diag_dump_<timestamp>.tar.gz`, where `<timestamp>` denotes when you ran the script.

If you run the script without parameters, you will be prompted for a database password.

Syntax

```
/opt/vertica/scripts/collect_diag_dump.sh [ command... ]
```

Parameters

| | | |
|----------------|----|---|
| <i>command</i> | -U | User name, typically the database administrator account, dbadmin. |
| | -w | Database password. |
| | -c | Includes a compression analysis, resulting in a longer script execution time. |

Example

The following command runs the audit script with all arguments:

```
$ /opt/vertica/scripts/collect_diag_dump.sh -U dbadmin -w password -c
```

Understanding Query Plans

A query plan is a sequence of step-like **paths** that the HP Vertica cost-based query optimizer selects to access or alter information in your HP Vertica database.

HP Vertica can execute a query in many different ways to achieve the same results. The query optimizer evaluates the possible plans it can use and returns what it considers to be the best alternative, which is usually a query plan with the lowest cost.

Cost

Cost is an estimate of the resources that the query plan will use for its execution strategy, such as data distribution statistics, CPU, disk, memory, network, data segmentation across cluster nodes, and so on. Although such resources correlate to query run time, they are not an *estimate* of run time. For example, if Plan1 costs more than Plan2, the optimizer estimates that Plan2 will take less time to run. Cost does not mean that if Plan1 costs two times more than Plan2, the optimizer estimates that Plan2 will take half the time to run.

The optimizer does not compute cost across queries. For example, if you run a plan for Query1 and a plan for Query2, a higher cost in Query2 does not indicate that Query2 will take longer than Query1.

Statistics

Many important optimizer decisions rely on statistics, which the query optimizer uses to determine the final plan to execute a query. Therefore, it is important that statistics be up to date. Without reasonably accurate statistics, the optimizer could choose a suboptimal plan, which might affect query performance.

HP Vertica uses the following statistics to calculate the lowest query plan cost and to create plan candidates:

- Number of rows in the table
- Number of distinct values of each column (**cardinality**)
- Minimum/maximum values of each column
- A **histogram** of the distribution of values in each column
- Disk space that the column occupies

The optimizer also considers the following:

- The access path with the fewest expected I/O operations and lowest CPU, memory, and network usage
- Multiple eligible [projections](#) to answer the query

- Join types of differing algorithms (hash join/merge join or group by hash/group by pipelined)
- The order in which to perform joins
- Query predicates
- Data re-distribution algorithms (broadcast and re-segmentation) across nodes in the cluster

HP Vertica provides hints about statistics through the query plan, which is annotated with a statistics status. See [Viewing statistics query plan output](#).

See Also

[Collecting Database Statistics](#)

How to Get Query Plan Information

You can get information about query plans in three ways:

- Run the EXPLAIN command.

To view query plan information, preface the query with the [EXPLAIN](#) command, as in the following example:

```
=> EXPLAIN SELECT customer_name, customer_state FROM customer_dimension
      WHERE customer_state in ('MA','NH') AND customer_gender = 'Male'
      ORDER BY customer_name LIMIT 10;
```

The output from a query plan is presented in a tree-like structure, where each step (**path**) represents a single operation in the database that the optimizer uses for its execution strategy. The following example output is based on the previous query:

```
----- QUERY PLAN DESCRIPTION: -----
EXPLAIN SELECT
customer_name,
customer_state
FROM customer_dimension
WHERE customer_state in ('MA','NH')
AND customer_gender = 'Male'
ORDER BY customer_name
LIMIT 10;
Access Path:
+-SELECT LIMIT 10 [Cost: 370, Rows: 10] (PATH ID: 0)
|  Output Only: 10 tuples
|  Execute on: Query Initiator
| +---> SORT [Cost: 370, Rows: 544] (PATH ID: 1)
| |   Order: customer_dimension.customer_name ASC
| |   Output Only: 10 tuples
```

```
| |      Execute on: Query Initiator
| | +---> STORAGE ACCESS for customer_dimension [Cost: 331, Rows: 544] (PATH ID: 2)
| | |      Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_des
ign_node0001
| | |      Materialize: customer_dimension.customer_state, customer_dimension.custome
r_name
| | |      Filter: (customer_dimension.customer_gender = 'Male')
| | |      Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))
| | |      Execute on: Query Initiator
```

- Query the `QUERY_PLAN_PROFILING` system table.

To observe the real-time flow of data through the plan, query the `V_MONITOR.QUERY_PLAN_PROFILES` system table. For details, see [Profiling query plans](#).

- Use the Management Console **Explain** option.

The Management Console can create query plans and profile query plans. For details, see [Viewing EXPLAIN output in Management Console](#) and [Viewing Profile Data in Management Console](#).

See Also

- [About EXPLAIN Output](#)
- [EXPLAIN](#)
- [QUERY_PLAN_PROFILES](#)

How to save query plan information

To save query plan information to a file, use the `vsq! \o` command by following these steps:

1. Turn on saving output to a file using the `\o` command.

```
vmartdb=> \o /home/dbadmin/my-plan-output
```

2. Run the query using the EXPLAIN command.

```
vmartdb=> EXPLAIN SELECT * FROM customer_dimension;
```

3. Turn off saving output to a file using the `\o` command.

```
vmartdb=> \o
```

If you do not turn off the `\o` command, HP Vertica continues to save output to the file that you specify. New output is appended to the previous output. The `\o` command captures all EXPLAIN output in the file until you issue the `\o` command again.

Viewing EXPLAIN output in Management Console

Management Console allows you to view the EXPLAIN output for a single query, allowing you to:

- Review the EXPLAIN output in an easy-to-read format.
- Display details about the projections and columns accessed by the query directly from the EXPLAIN output.

After you select a database, there are two ways to view the EXPLAIN path using Management Console.

To focus on specific areas of the database activity, such as spikes in CPU usage, take these steps:

1. For the database you have selected, click **Activity** at the bottom of the Management Console window.
2. Select **Queries** from the drop-down list at the top of the page.

You can also access EXPLAIN plans for **User Sessions** and **User Query Phases** data by selecting that item on the drop-down list.

3. Click the spot on the activity graph for which you want to view the query plan.

The activity detail window appears.

4. In the **View Plan** column, click **Explain** next to the command for which you want to view the query plan. Only certain queries, like SELECT, INSERT, DELETE, and UPDATE, have EXPLAIN plans.

The **Explain** window opens with the query you selected in the text box. HP Vertica reruns the query and when it completes, the EXPLAIN output for that query loads in the window, below the query.

To review the EXPLAIN output for a specific query, take these steps:

1. Select the database for the query whose query plan you want to see.
2. Click **Explain** at the bottom of the Management Console window.

The **Explain** window opens. The text box is empty because you have not selected a query.

3. Type or paste the query text into that box. Do not enter the word EXPLAIN before the query. If

you do enter the EXPLAIN keyword, a syntax error occurs.

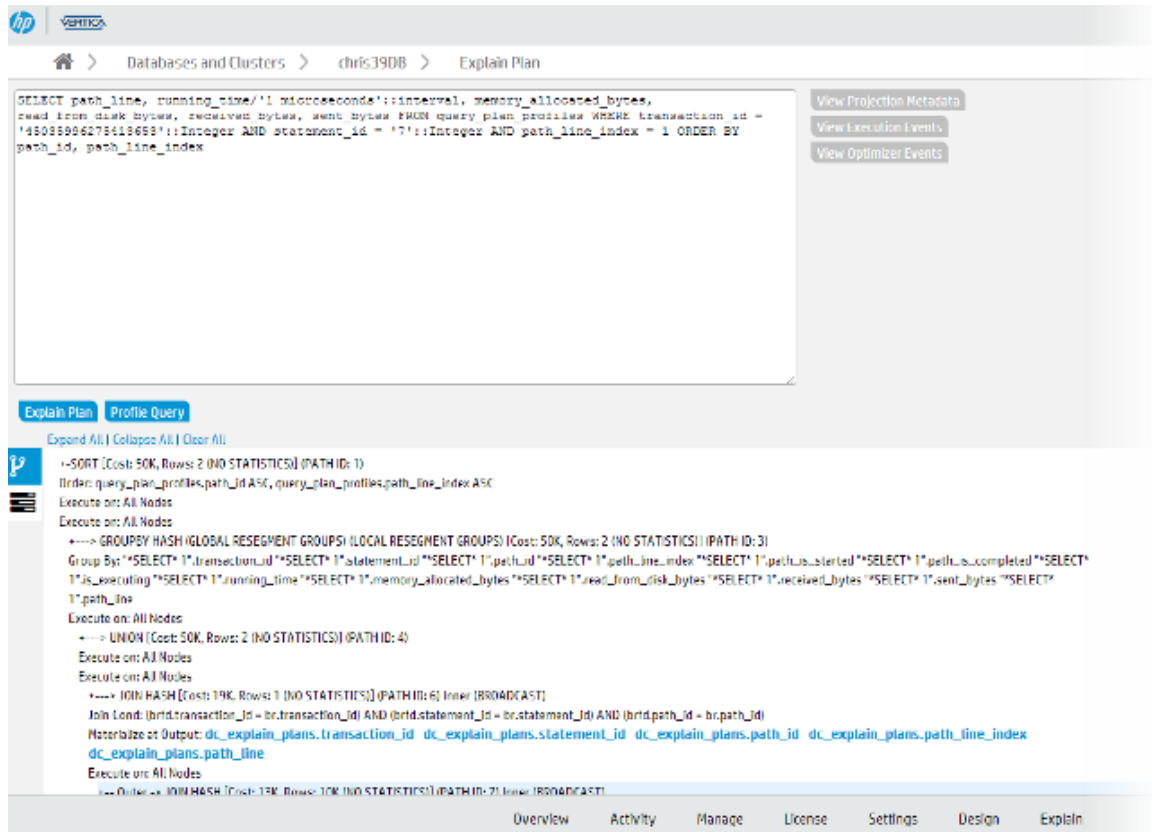
If your query uses incorrect syntax or if you enter more than one query, an error occurs.

4. Click **Explain Plan**.

HP Vertica reruns the query and when it completes, the query plan displays below the query text box.

About the EXPLAIN Plan in Management Console

When you select a query on the activity details page or enter a query manually after you click **Explain Plan**, the window appears with the query text and EXPLAIN output, as in the following image.



On this window, you can perform the following tasks related to the query you entered:

- [Expanding and collapsing query paths in EXPLAIN output](#)
- [Viewing projection and column metadata in EXPLAIN output](#)
- [Clearing query data](#)
- [Viewing Profile Data in Management Console](#)

Expanding and collapsing query paths in EXPLAIN output

When you have a query on the EXPLAIN window, the full EXPLAIN plan displays in the lower-left half of the window. The EXPLAIN path output can be lengthy, so collapse path information that is uninteresting, or expand only those paths that you want to focus on.

- To collapse all the query paths in order to see a summary of each path in the EXPLAIN output, click **Collapse All**.
- To expand all the query paths to display the path details, click **Expand All**.
- To expand an individual query path so you can see details about that path, click the first line of the path information. Click the first line again to collapse the path data.

For details about the EXPLAIN output, see [About EXPLAIN Output](#).

Clearing query data

When you have finished reviewing the current query data, click **Clear All** to clear the query text and data. Alternatively, to display information about another query, enter the query text and click **Explain Plan** or **Profile Query**.

Viewing projection and column metadata in EXPLAIN output

In the Management Console on the EXPLAIN page, when the query paths are expanded, the **Projection** line contains a projection name and the **Materialize** line contains one or more column names.

To view metadata for a projection or a column, click the object name. A pop-up window displays the metadata. The following image on the left shows example projection metadata and the image on the right shows example column metadata.

Note: Most system tables do not have metadata.

| Details [x] | |
|--------------------------|------------------|
| name | online_sales_fac |
| id | 450359962765! |
| owner | uidbadmin |
| schema | online_sales |
| node | null |
| anchor_table | online_sales_fac |
| isprejoin | false |
| create_type | DESIGNER |
| verified_fault_tolerance | 0 |
| isuptodate | true |
| hasstatistics | true |
| issegmented | true |
| issuperprojection | true |

| Details [x] | |
|---------------|------------|
| name | cc_name |
| max | Southwest |
| position | 3 |
| min | California |
| ndv | 11 |
| sort_order | 4 |

When you are done viewing the metadata, close the pop-up window.

Viewing EXPLAIN Output in vsql

To view the query plan in vsql, preface the SQL command with the EXPLAIN keyword, as in the following example:

```
=> EXPLAIN SELECT customer_name, customer_state
-> FROM customer_dimension
-> WHERE customer_state IN ('MA','NH')
-> ORDER BY customer_name;
```

QUERY PLAN

QUERY PLAN DESCRIPTION:

```
EXPLAIN SELECT customer_name, customer_state
FROM customer_dimension
WHERE customer_state IN ('MA','NH')
ORDER BY customer_name;
```

Access Path:

```
+--SORT [Cost: 2K, Rows: 50K (NO STATISTICS)] (PATH ID: 1)
| Order: customer_dimension.customer_name ASC
| Execute on: All Nodes
| +---> STORAGE ACCESS for customer_dimension [Cost: 235, Rows: 50K (NO STATISTICS)] (PATH ID: 2)
| | Projection: public.customer_dimension_b0
| | Materialize: customer_dimension.customer_state, customer_dimension.customer_name
| | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))
| | Execute on: All Nodes
```

About EXPLAIN Output

The EXPLAIN command returns the execution strategy for the optimizer's query plan and provides information that lets you see the optimizer decisions.

EXPLAIN output documents the choices the optimizer has made. If you think your query is not performing as it should, run the query through the Database Designer.

Note: For more information see:

- [Incremental Design](#)
- [Reducing Run-time of Queries](#)

Textual output of query plans

Textual output from a query plan is presented in a tree-like structure, where each step (**path**) represents a single operation in the database that the optimizer uses for its execution. Depending on the query and database schema, the output provides the following information:

- Tables referenced by the statement
- The estimated cost of the optimization
- Estimated row cardinality
- The Path ID, an integer that links to error messages and profiling counters, making it easier for you to troubleshoot performance issues
- Data operations such as SORT, FILTER, LIMIT, and GROUP BY
- The chosen projections
- Information about statistics, such as if they are current or out of range
- Algorithms chosen for operations into the query, such as HASH/MERGE or GROUPBY HASH/GROUPBY PIPELINED
- Data re-distribution (broadcast, segmentation) across nodes in the cluster

Example

In the example EXPLAIN plan output that follows, the optimizer processes the query in three steps, which are identified by path IDs:

1. Path ID 2: STORAGE ACCESS and FILTER
2. Path ID 1: SORT

3. Path ID 0: LIMIT

QUERY PLAN DESCRIPTION:

```
EXPLAIN SELECT
customer_name,
customer_state
FROM customer_dimension
WHERE customer_state in ('MA','NH')
AND customer_gender = 'Male'
```

```
ORDER BY customer_name
```

```
LIMIT 10;
```

```
Access Path:
+-SELECT LIMIT 10 [Cost: 370, Rows: 10] (PATH ID: 0)
| Output Only: 10 tuples
| Execute on: Query Initiator
| +----> SORT [Cost: 370, Rows: 544] (PATH ID: 1)
| | Order: customer_dimension.customer_name ASC
| | Output Only: 10 tuples
| | Execute on: Query Initiator
| | +----> STORAGE ACCESS for customer_dimension [Cost: 331, Rows: 544] (PATH ID: 2)
| | | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
| | | Materialize: customer_dimension.customer_state, customer_dimension.customer_name
| | | Filter: (customer_dimension.customer_gender = 'Male')
| | | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))
| | | Execute on: Query Initiator
```

Note: A STORAGE ACCESS operation can scan more than the columns in the select list; for example, columns referenced in WHERE clause, and so on.

The following table shows which portions of the query the three steps align with:

| Path | Associated part of query |
|--------------------------|---|
| STORAGE ACCESS FILTER | SELECT customer_name, customer_state WHERE customer_state in ('MA','NH') AND customer_gender = 'Male' |
| SORT | ORDER BY customer_name |
| LIMIT | LIMIT 10; |

Viewing statistics query plan output

If table statistics are stale or you haven't collected statistics, the optimizer might not choose the best plan for a query. HP Vertica recommends that you gather full statistics on a table whenever:

- The table is first bulk loaded.
- A new projection using that table is created and refreshed.

- The number of rows in the table changes by 50%.
- The minimum/maximum values in the table's columns change by 50%.
- New primary key values are added to tables with referential integrity constraints. Both the primary key and foreign key tables should be reanalyzed.
- Relative size of a table, compared to tables it is being joined to, has changed materially; for example, one table is now only five times larger than the other table when previously it was 50 times larger.
- There is a significant deviation in the distribution of data, which would necessitate recalculating **histograms**. For example, there is an event that caused abnormally high levels of trading for a particular stock.
- There is a down-time window when the database is not in active use.

HP Vertica provides hints about statistics through the query plan, which is annotated with a status of either NO STATISTICS or STALE STATISTICS. For example, the following EXPLAIN fragment indicates that histograms are unavailable (NO STATISTICS):

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 604, Rows: 10K (NO STATISTICS)]
```

The next EXPLAIN fragment indicates that the predicate has fallen outside the histogram range (STALE STATISTICS):

```
| | +-- Outer -> STORAGE ACCESS for fact [Cost: 35, Rows: 1 (STALE STATISTICS)]
```

Notes

- You can resolve many issues related to statistics by calling the ANALYZE_STATISTICS() function for the tables involved.
- If you update statistics and find that the query still performs suboptimally, run your query through Database Designer, choosing incremental design as the design type.
- Projections that have no data never have full statistics. Query the PROJECTION_STORAGE system table to see if your projection contains data. You can also query the PROJECTIONS table, the HAS_STATISTICS field.
- Once your system is running well, save exported statistics for all tables. In the unlikely scenario that statistics modifications impact optimizer plans, particularly after an upgrade, you can always revert back to the exported statistics.

The following topics in the SQL Reference Manual:

See Also

- [Collecting Database Statistics](#)
- [Determining When Statistics Were Last Updated](#)
- [Reacting to Stale Statistics](#)
- [Creating an Incremental Design Using Database Designer](#)
- [Optimizing Query Performance](#)
- [ANALYZE_STATISTICS](#)
- [PROJECTION_COLUMNS](#)
- [PROJECTION_STORAGE](#)
- [PROJECTIONS](#)

Viewing cost and rows path

One of the first items you'll see in query plan output is **Cost**.

The following EXPLAIN output shows the **Cost** operator:

```
Access Path: +-SELECT LIMIT 10 [Cost: 370, Rows: 10] (PATH ID: 0)
| Output Only: 10 tuples
| Execute on: Query Initiator
| +---> SORT [Cost: 370, Rows: 544] (PATH ID: 1)
| | Order: customer_dimension.customer_name ASC
| | Output Only: 10 tuples
| | Execute on: Query Initiator
| | +---> STORAGE ACCESS for customer_dimension [Cost: 331, Rows: 544] (PATH ID: 2)
| | | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_desig
n_node0001
| | | Materialize: customer_dimension.customer_state, customer_dimension.customer_n
ame
| | | Filter: (customer_dimension.customer_gender = 'Male')
| | | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))
| | | Execute on: Query Initiator
```

The **Row** operator is the number of rows the optimizer estimates the query will return. Letters after numbers refer to the units of measure (K=thousand, M=million, B=billion, T=trillion), so the output for the following query indicates that the number of rows to return is *50 thousand*.

```
=> EXPLAIN SELECT customer_gender FROM customer_dimension;
Access Path:
+-STORAGE ACCESS for customer_dimension [Cost: 17, Rows: 50K (3 RLE)] (PATH ID: 1)
| Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node000
1
```

```
| Materialize: customer_dimension.customer_gender  
| Execute on: Query Initiator
```

The reference to (3 RLE) in the STORAGE ACCESS path means that the optimizer estimates that the storage access operator returns 50K rows. Because the column is run-length encoded (**RLE**), the real number of RLE rows returned is only three rows:

- 1 row for female
- 1 row for male
- 1 row that represents unknown (NULL) gender

Note: See [Understanding Query Plans](#) for additional information about how the optimizer estimates cost.

Viewing projection path

You can see which **projections** the optimizer chose for the query plan by looking at the **Projection** path in the textual output:

```
EXPLAIN SELECT  
  customer_name,  
  customer_state  
FROM customer_dimension  
WHERE customer_state in ('MA','NH')  
AND customer_gender = 'Male'  
ORDER BY customer_name  
LIMIT 10;  
Access Path:  
+-SELECT LIMIT 10 [Cost: 370, Rows: 10] (PATH ID: 0)  
| Output Only: 10 tuples  
| Execute on: Query Initiator  
| +---> SORT [Cost: 370, Rows: 544] (PATH ID: 1)  
| | Order: customer_dimension.customer_name ASC  
| | Output Only: 10 tuples  
| | Execute on: Query Initiator  
| | +---> STORAGE ACCESS for customer_dimension [Cost: 331, Rows: 544] (PATH ID: 2)  
| | | Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001  
| | | Materialize: customer_dimension.customer_state, customer_dimension.customer_n  
ame  
| | | Filter: (customer_dimension.customer_gender = 'Male')  
| | | Filter: (customer_dimension.customer_state = ANY (ARRAY['MA', 'NH']))  
| | | Execute on: Query Initiator
```

The query optimizer automatically picks the best projections, but without reasonably accurate statistics, the optimizer could choose a suboptimal projection or join order for a query. For details, see [Collecting Statistics](#).

HP Vertica considers which projection to choose for a plan by considering the following aspects:

- How columns are joined in the query
- How the projections are grouped or sorted
- Whether SQL analytic operations applied
- Any column information from a projection's storage on disk

As HP Vertica scans the possibilities for each plan, projections with the higher initial costs could end up in the final plan because they make joins cheaper. For example, a query can be answered with many possible plans, which the optimizer considers before choosing one of them. For efficiency, the optimizer uses sophisticated algorithms to prune intermediate partial plan fragments with higher cost. The optimizer knows that intermediate plan fragments might initially look bad (due to high storage access cost) but which produce excellent final plans due to other optimizations that it allows.

If your statistics are up to date but the query still performs poorly, run the query through the Database Designer. For details, see [Incremental Design](#)

Tips

- To test different projections, refer to a segmented projection by name in the query.
- If you query an unsegmented projection by name, it changes the plan because then data is used from one node only, where unsegmented projection names include a specific node name.
- For optimal performance, write queries so the columns are sorted the same way that the projection columns are sorted.

See Also

- [Reducing Query Run-Time](#)
- [Maximizing Projection Performance](#)
- [Creating Custom Designs](#)
- [Physical Schema](#)

Viewing join path

Just like a join query, which references two or more tables, the `Join` step in query plans has two input branches:

- The left input, which is the outer table of the join
- The right input, which is the inner table of the join

In the following query, table T1 is the left input because it is on the left side of the JOIN keyword, and table T2 is the right input, because it is on the right side of the JOIN keyword:

```
SELECT * FROM T1 JOIN T2 ON T1.x = T2.x;
```

Outer versus inner join

Query performance is better if the small table is used as the inner input to the join. The query optimizer automatically reorders the inputs to joins to ensure that this is the case unless the join in question is an outer join.

The following example shows a query and its plan for a left outer join:

```
=> EXPLAIN SELECT CD.annual_income,OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> LEFT OUTER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key;
Access Path:
+-JOIN HASH [LeftOuter] [Cost: 4K, Rows: 5M] (PATH ID: 1)
| Join Cond: (CD.customer_key = OSI.customer_key)
| Materialize at Output: OSI.sale_date_key
| Execute on: All Nodes
| +- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
| | Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_de
sign
| | Materialize: OSI.customer_key
| | Execute on: All Nodes
| +- Inner -> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
| | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_n
ode0001
| | Materialize: CD.annual_income, CD.customer_key
| | Execute on: All Nodes
```

The following example shows a query and its plan for a full outer join:

```
=> EXPLAIN SELECT CD.annual_income,OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> FULL OUTER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key;
Access Path:
+-JOIN HASH [FullOuter] [Cost: 18K, Rows: 5M] (PATH ID: 1) Outer (RESEGMENT) Inner (FILTE
R)
| Join Cond: (CD.customer_key = OSI.customer_key)
| Execute on: All Nodes
| +- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
| | Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_de
sign
| | Materialize: OSI.sale_date_key, OSI.customer_key
| | Execute on: All Nodes
| +- Inner -> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
| | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_n
ode0001
| | Materialize: CD.annual_income, CD.customer_key
| | Execute on: All Nodes
```

Hash and merge joins

HP Vertica has two join algorithms to choose from: merge join and hash join. The optimizer automatically chooses the most appropriate algorithm, given the query and projections in a system.

For the following query, the optimizer chooses a hash join.

```
=> EXPLAIN SELECT CD.annual_income,OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> INNER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key;
Access Path:
+-JOIN HASH [Cost: 4K, Rows: 5M] (PATH ID: 1)
|   Join Cond: (CD.customer_key = OSI.customer_key)
|   Materialize at Output: OSI.sale_date_key
|   Execute on: All Nodes
|   +- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
|   |   Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_de
sign
|   |   Materialize: OSI.customer_key
|   |   Execute on: All Nodes
|   +- Inner -> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
|   |   Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_n
ode0001
|   |   Materialize: CD.annual_income, CD.customer_key
|   |   Execute on: All Nodes
```

Tip: If you get a hash join when you are expecting a merge join, it means that at least one of the projections is not sorted on the join column (for example `customer_key` in the preceding query). To facilitate a merge join, you might need to create different projections that are sorted on the join columns.

In the next example, the optimizer chooses a merge join. The optimizer's first pass performs a merge join because the inputs are presorted, and then it performs a hash join.

```
=> EXPLAIN SELECT count(*) FROM online_sales.online_sales_fact OSI
-> INNER JOIN customer_dimension CD ON CD.customer_key = OSI.customer_key
-> INNER JOIN product_dimension PD ON PD.product_key = OSI.product_key;
Access Path:
+-GROUPBY NOTHING [Cost: 8K, Rows: 1] (PATH ID: 1)
|   Aggregates: count(*)
|   Execute on: All Nodes
|   +--> JOIN HASH [Cost: 7K, Rows: 5M] (PATH ID: 2)
|   |   Join Cond: (PD.product_key = OSI.product_key)
|   |   Materialize at Input: OSI.product_key
|   |   Execute on: All Nodes
|   +- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 4K, Rows: 5M] (PATH ID: 3)
|   |   |   Join Cond: (CD.customer_key = OSI.customer_key)
|   |   |   Execute on: All Nodes
|   +- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 4)
|   |   |   Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb
b_design
|   |   |   Materialize: OSI.customer_key
```

```
| | | |      Execute on: All Nodes
| | | +-- Inner -> STORAGE ACCESS for CD [Cost: 132, Rows: 50K] (PATH ID: 5)
| | | |      Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_desi
gn_node0001
| | | |      Materialize: CD.customer_key
| | | |      Execute on: All Nodes
| | +-- Inner -> STORAGE ACCESS for PD [Cost: 152, Rows: 60K] (PATH ID: 6)
| | |      Projection: public.product_dimension_DBD_2_rep_vmartdb_design_vmartdb_design_
node0001
| | |      Materialize: PD.product_key
| | |      Execute on: All Nodes
```

Inequality joins

HP Vertica processes joins with equality predicates very efficiently. The EXPLAIN plan shows equality join predicates as join condition (**Join Cond**).

```
=> EXPLAIN SELECT CD.annual_income, OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> INNER JOIN customer_dimension CD
-> ON CD.customer_key = OSI.customer_key;
Access Path:
+-JOIN HASH [Cost: 4K, Rows: 5M] (PATH ID: 1)
|   Join Cond: (CD.customer_key = OSI.customer_key)
|   Materialize at Output: OSI.sale_date_key
|   Execute on: All Nodes
| +-- Outer -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 2)
| |       Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_de
sign
| |       Materialize: OSI.customer_key
| |       Execute on: All Nodes
| +-- Inner -> STORAGE ACCESS for CD [Cost: 264, Rows: 50K] (PATH ID: 3)
| |       Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_n
ode0001
| |       Materialize: CD.annual_income, CD.customer_key
| |       Execute on: All Nodes
```

However, inequality joins are treated like cross joins and can run less efficiently, which you can see by the change in cost between the two queries:

```
=> EXPLAIN SELECT CD.annual_income, OSI.sale_date_key
-> FROM online_sales.online_sales_fact OSI
-> INNER JOIN customer_dimension CD
-> ON CD.customer_key < OSI.customer_key;
Access Path:
+-JOIN HASH [Cost: 98M, Rows: 5M] (PATH ID: 1)
|   Join Filter: (CD.customer_key < OSI.customer_key)
|   Materialize at Output: CD.annual_income
|   Execute on: All Nodes
| +-- Outer -> STORAGE ACCESS for CD [Cost: 132, Rows: 50K] (PATH ID: 2)
| |       Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_n
ode0001
| |       Materialize: CD.customer_key
```

```
| |      Execute on: All Nodes
| |-- Inner -> STORAGE ACCESS for OSI [Cost: 3K, Rows: 5M] (PATH ID: 3)
| |      Projection: online_sales.online_sales_fact_DBD_12_seg_vmartdb_design_vmartdb_de
sign
| |      Materialize: OSI.sale_date_key, OSI.customer_key
| |      Execute on: All Nodes
```

Event series joins

Event series joins are denoted by the INTERPOLATED path.

```
=> EXPLAIN SELECT * FROM hTicks h FULL OUTER JOIN aTicks a -> ON (h.time INTERPOLATE PREV
IOUS
Access Path:
+-JOIN (INTERPOLATED) [FullOuter] [Cost: 31, Rows: 4 (NO STATISTICS)] (PATH ID: 1)
  Outer (SORT ON JOIN KEY) Inner (SORT ON JOIN KEY)
  | Join Cond: (h."time" = a."time")
  | Execute on: Query Initiator
  |-- Outer -> STORAGE ACCESS for h [Cost: 15, Rows: 4 (NO STATISTICS)] (PATH ID: 2)
  | |      Projection: public.hTicks_node0004
  | |      Materialize: h.stock, h."time", h.price
  | |      Execute on: Query Initiator
  |-- Inner -> STORAGE ACCESS for a [Cost: 15, Rows: 4 (NO STATISTICS)] (PATH ID: 3)
  | |      Projection: public.aTicks_node0004
  | |      Materialize: a.stock, a."time", a.price
  | |      Execute on: Query Initiator
```

Viewing path ID

The PATH ID is a unique identifier that HP Vertica assigns to each operation (path) within a query plan. The same ID is shared among

[EXPLAIN plans](#)

[Join error messages](#)

[EXECUTION_ENGINE_PROFILES](#) system table

[QUERY_PLAN_PROFILES](#) system table

This allows you to trace issues to their root cause.

Note: For more information, see [Linking EXPLAIN Plans to Error Messages and Profiling Information](#) for more information.

Here's an example of EXPLAIN output, showing the PATH ID for each path in the optimizer's query plan.

```
=> EXPLAIN SELECT * FROM fact JOIN dim ON x=y JOIN ext on y=z;
Access Path:
```

```
+--JOIN MERGEJOIN(inputs presorted) [Cost: 815, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| Join Cond: (dim.y = ext.z)
| Materialize at Output: fact.x
| Execute on: All Nodes
| +- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 408, Rows: 10K (NO STATISTICS)]
(PATH ID: 2)
| | Join Cond: (fact.x = dim.y)
| | Execute on: All Nodes
| | +- Outer -> STORAGE ACCESS for fact [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH
ID: 3)
| | | Projection: public.fact_super
| | | Materialize: fact.x
| | | Execute on: All Nodes
| | +- Inner -> STORAGE ACCESS for dim [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID:
4)
| | | Projection: public.dim_super
| | | Materialize: dim.y
| | | Execute on: All Nodes
| +- Inner -> STORAGE ACCESS for ext [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH ID: 5
)
| | Projection: public.ext_super
| | Materialize: ext.z
| | Execute on: All Nodes
```

Viewing filter path

The **Filter** step evaluates predicates on a single table. It accepts a set of rows, eliminates some of them (based on the criteria you provide in your query), and returns the rest. For example, the optimizer can filter local data of a join input that will be joined with another re-segmented join input.

The following statement queries the `customer_dimension` table and uses the `WHERE` clause to filter the results only for male customers in Massachusetts and New Hampshire.

```
EXPLAIN SELECT
  CD.customer_name,
  CD.customer_state,
  AVG(CD.customer_age) AS avg_age,
  COUNT(*) AS count
FROM customer_dimension CD
WHERE CD.customer_state in ('MA','NH') AND CD.customer_gender = 'Male'
GROUP BY CD.customer_state, CD.customer_name;
```

The query plan output is as follows:

```
Access Path:
+-GROUPBY HASH [Cost: 378, Rows: 544] (PATH ID: 1)
| Aggregates: sum_float(CD.customer_age), count(CD.customer_age), count(*)
| Group By: CD.customer_state, CD.customer_name
| Execute on: Query Initiator
| +--> STORAGE ACCESS for CD [Cost: 372, Rows: 544] (PATH ID: 2)
| | Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_n
ode0001
| | Materialize: CD.customer_state, CD.customer_name, CD.customer_age
```

```
| | Filter: (CD.customer_gender = 'Male')  
| | Filter: (CD.customer_state = ANY (ARRAY['MA', 'NH']))  
| | Execute on: Query Initiator
```

Viewing the GROUP BY paths

A GROUP BY operation has three algorithms:

- GROUPBY PIPELINED requires that inputs be presorted on the columns specified in the group, which means that HP Vertica need only retain data in the current group in memory. GROUPBY PIPELINED operations are preferred because they are generally faster and require less memory than GROUPBY HASH. GROUPBY PIPELINED is especially useful for queries that process large numbers of high-cardinality group by columns or DISTINCT aggregates.
- GROUPBY HASH input is not sorted by the group columns, so HP Vertica builds a hash table on those group columns in order to process the aggregates and group by expressions.
- Partially sorted GROUPBY leverages the sort order of one or more GROUP BY columns, which may or may not include DISTINCT aggregate columns, such as COUNT(DISTINCT ...) or SUM(DISTINCT ...). This algorithm uses smaller hash tables, which reduces the possibility of hash tables spilling to temporary files on disk.

The optimizer chooses the faster GROUP BY PIPELINED over GROUP BY HASH if the certain conditions are met.

Note: For details, see [Avoiding GROUPBY HASH with Projection Design](#) in the Programmer's Guide.

For examples of EXPLAIN plans that identify the GROUP BY algorithm, see:

- [GROUPBY HASH EXPLAIN plan example](#)
- [GROUPBY PIPELINED EXPLAIN plan example](#)
- [Partially sorted GROUPBY EXPLAIN plan example](#)

GROUPBY HASH EXPLAIN plan example

Here's an example of how GROUPBY HASH operations look in EXPLAIN output.

```
=> EXPLAIN SELECT COUNT(DISTINCT annual_income)  
FROM customer_dimension  
WHERE customer_region='NorthWest';
```

The output shows that the optimizer chose the less efficient GROUPBY HASH path, which means the projection was not presorted on the annual_income column. If such a projection is available, the optimizer would choose the GROUPBY PIPELINED algorithm.

```
Access Path:
+-GROUPBY NOTHING [Cost: 256, Rows: 1 (NO STATISTICS)] (PATH ID: 1)
|   Aggregates: count(DISTINCT customer_dimension.annual_income)
|   +---> GROUPBY HASH (LOCAL RESEGMENT GROUPS) [Cost: 253, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
|   |   Group By: customer_dimension.annual_income
|   |   +---> STORAGE ACCESS for customer_dimension [Cost: 227, Rows: 50K (NO STATISTICS)] (PATH ID: 3)
|   |   |   Projection: public.customer_dimension_super
|   |   |   Materialize: customer_dimension.annual_income
|   |   |   Filter: (customer_dimension.customer_region = 'NorthWest'
|   |   ...
```

GROUPBY PIPELINED EXPLAIN plan example

If you have a projection that is already sorted on the `customer_gender` column, the optimizer chooses the faster `GROUPBY PIPELINED` operation:

```
=> EXPLAIN SELECT COUNT(distinct customer_gender) from customer_dimension;
Access Path:
+-GROUPBY NOTHING [Cost: 22, Rows: 1] (PATH ID: 1)
|   Aggregates: count(DISTINCT customer_dimension.customer_gender)
|   Execute on: Query Initiator
|   +---> GROUPBY PIPELINED [Cost: 20, Rows: 10K] (PATH ID: 2)
|   |   Group By: customer_dimension.customer_gender
|   |   Execute on: Query Initiator
|   |   +---> STORAGE ACCESS for customer_dimension [Cost: 17, Rows: 50K (3 RLE)] (PATH ID: 3)
|   |   |   Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
|   |   |   Materialize: customer_dimension.customer_gender
|   |   |   Execute on: Query Initiator
```

Similarly, the use of an equality predicate, such as in the following query, preserves `GROUPBY PIPELINED`:

```
=> EXPLAIN SELECT COUNT(DISTINCT annual_income) FROM customer_dimension
WHERE customer_gender = 'Female';

Access Path: +-GROUPBY NOTHING [Cost: 161, Rows: 1] (PATH ID: 1)
|   Aggregates: count(DISTINCT customer_dimension.annual_income)
|   +---> GROUPBY PIPELINED [Cost: 158, Rows: 10K] (PATH ID: 2)
|   |   Group By: customer_dimension.annual_income
|   |   +---> STORAGE ACCESS for customer_dimension [Cost: 144, Rows: 47K] (PATH ID: 3)
|   |   |   Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_design_node0001
|   |   |   Materialize: customer_dimension.annual_income
|   |   |   Filter: (customer_dimension.customer_gender = 'Female')
```

Tip: If `EXPLAIN` reports `GROUPBY HASH`, you can modify the projection design to force it to use `GROUPBY PIPELINED`.

Partially sorted GROUPBY EXPLAIN plan example

The following example shows the EXPLAIN query plan that uses partially sorted GROUPBY. To try this example, take these steps to create and populate tables in the VMart database and then generate the query plan:

```
$ cd /opt/Vertica/examples/VMart_Schema
$ ls vmart_gen
$ vsql
-- create and populate the example tables from the .tbl files
=>\i vmart_create_schema.sql
=>\i vmart_load_data.sql
-- verify table population
=> SELECT COUNT(*) FROM store.store_sales_fact;
COUNT
-----
5000000
(1 row)
=> CREATE PROJECTION store.store_sales_fact_by_store_and_date AS
SELECT * FROM store.store_sales_fact f
ORDER BY f.store_key, f.date_key;
=> SELECT START_REFRESH(); -- wait a few seconds before running queries
```

The query plan shows that optimizer uses the partially sorted GROUPBY optimization because the GROUP BY column, `store_key`, is one of the sort columns in the projection, `store.store_sales_fact_by_store_and_date`, and contains more than one call to a COUNT or SUM function with the DISTINCT keyword. The partially sorted GROUPBY optimization can have a significant impact on the performance of such queries.

```
VMart=> EXPLAIN SELECT COUNT(distinct customer_key) AS cntd_cust,
store_key,
COUNT(DISTINCT product_key) AS cntd_prod,
COUNT(DISTINCT promotion_key) AS cntd_promo,
SUM(sales_dollar_amount) AS sum_sales_dollar,
SUM(cost_dollar_amount) AS sum_cost_dollar
FROM store.store_sales_fact
GROUP BY store_key
ORDER BY cntd_cust DESC
LIMIT 25;

Access Path:

+-SELECT LIMIT 25 [Cost: 43K, Rows: 25 (NO STATISTICS)] (PATH ID: 0)
| Output Only: 25 tuples
| +---> SORT [TOPK] [Cost: 43K, Rows: 10K (NO STATISTICS)] (PATH ID: 1)
| | Order: "Sqry$_1".cntd_cust DESC
| | Output Only: 25 tuples
| | +---> JOIN MERGEJOIN(inputs presorted) [Cost: 43K, Rows: 10K (NO STATISTICS)] (PATH I
D: 2)
| | | Join Cond: ("Sqry$_2".store_key <=> "Sqry$_3".store_key)
| | | +- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 32K, Rows: 10K (NO STATISTIC
S)] (PATH ID: 3)
| | | | Join Cond: ("Sqry$_1".store_key <=> "Sqry$_2".store_key)
| | | | +- Outer -> SELECT [Cost: 22K, Rows: 10K (NO STATISTICS)] (PATH ID: 4)
```

```

| | | | | +---> GROUPBY HASH (SORT OUTPUT) (LOCAL RESEGMENT GROUPS) [Cost: 22K, Rows: 10K
(NO STATISTICS)] (PATH ID: 5)
| | | | |     Aggregates: count(DISTINCT store_sales_fact.customer_key), sum(<SVAR>), su
m(<SVAR>)
| | | | |     Group By: store_sales_fact.store_key
| | | | |     Partially sorted keys: 1
| | | | | +---> GROUPBY HASH (LOCAL RESEGMENT GROUPS) [Cost: 22K, Rows: 10K (NO STATIST
ICS)] (PATH ID: 6)
| | | | |     Aggregates: sum(store_sales_fact.sales_dollar_amount), sum(store_sale
s_fact.cost_dollar_amount)
| | | | |     Group By: store_sales_fact.store_key, store_sales_fact.customer_key
| | | | |     Partially sorted keys: 1
| | | | | +---> STORAGE ACCESS for store_sales_fact [Cost: 18K, Rows: 5M (NO STATISTI
CS)] (PATH ID: 7)
...

```

Viewing sort path

The **Sort** operator sorts the data according to a specified list of columns. The **EXPLAIN** plan indicates the sort expressions and if the sort order is ascending (ASC) or descending (DESC).

For example, the following query plan shows the column list nature of the **Sort** operator:

```

EXPLAIN SELECT
  CD.customer_name,
  CD.customer_state,
  AVG(CD.customer_age) AS avg_age,
  COUNT(*) AS count
FROM customer_dimension CD
WHERE CD.customer_state in ('MA','NH')
  AND CD.customer_gender = 'Male'
GROUP BY CD.customer_state, CD.customer_name
ORDER BY avg_age, customer_name;
Access Path:
+-SORT [Cost: 422, Rows: 544] (PATH ID: 1)
| Order: (<SVAR> / float8(<SVAR>)) ASC, CD.customer_name ASC
| Execute on: Query Initiator
| +---> GROUPBY HASH [Cost: 378, Rows: 544] (PATH ID: 2)
| | Aggregates: sum_float(CD.customer_age), count(CD.customer_age), count(*)
| | Group By: CD.customer_state, CD.customer_name
| | Execute on: Query Initiator
| | +---> STORAGE ACCESS for CD [Cost: 372, Rows: 544] (PATH ID: 3)
| | | Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001
| | | Materialize: CD.customer_state, CD.customer_name, CD.customer_age
| | | Filter: (CD.customer_gender = 'Male')
| | | Filter: (CD.customer_state = ANY (ARRAY['MA', 'NH']))
| | | Execute on: Query Initiator

```

If you change the sort order to descending, the change appears in the plan:

```

EXPLAIN SELECT
  CD.customer_name,
  CD.customer_state,
  AVG(CD.customer_age) AS avg_age,

```

```
    COUNT(*) AS count
FROM customer_dimension CD
WHERE CD.customer_state in ('MA','NH')
    AND CD.customer_gender = 'Male'
GROUP BY CD.customer_state, CD.customer_name
ORDER BY avg_age DESC, customer_name;
Access Path:
+-SORT [Cost: 422, Rows: 544] (PATH ID: 1)
|   Order: (<SVAR> / float8(<SVAR>)) DESC, CD.customer_name ASC
|   Execute on: Query Initiator
| +---> GROUPBY HASH [Cost: 378, Rows: 544] (PATH ID: 2)
| |       Aggregates: sum_float(CD.customer_age), count(CD.customer_age), count(*)
| |       Group By: CD.customer_state, CD.customer_name
| |       Execute on: Query Initiator
| | +---> STORAGE ACCESS for CD [Cost: 372, Rows: 544] (PATH ID: 3)
| | |       Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001
| | |       Materialize: CD.customer_state, CD.customer_name, CD.customer_age
| | |       Filter: (CD.customer_gender = 'Male')
| | |       Filter: (CD.customer_state = ANY (ARRAY['MA', 'NH']))
| | |       Execute on: Query Initiator
```

Viewing limit path

The **LIMIT** path restricts the number of result rows based on the **LIMIT** clause in the query. Using the **LIMIT** clause in queries with thousands of rows could increase query performance.

The optimizer pushes the **LIMIT** operation down as far as possible in queries. A single **LIMIT** clause in the query could generate multiple **Output Only** annotations on the plan.

```
=> EXPLAIN SELECT COUNT(DISTINCT annual_income) FROM customer_dimension LIMIT 10;
Access Path:
+-SELECT LIMIT 10 [Cost: 161, Rows: 10] (PATH ID: 0)
|   Output Only: 10 tuples
| +---> GROUPBY NOTHING [Cost: 161, Rows: 1] (PATH ID: 1)
| |       Aggregates: count(DISTINCT customer_dimension.annual_income)
| |       Output Only: 10 tuples
| | +---> GROUPBY HASH (SORT OUTPUT) [Cost: 158, Rows: 10K] (PATH ID: 2)
| | |       Group By: customer_dimension.annual_income
| | | +---> STORAGE ACCESS for customer_dimension [Cost: 132, Rows: 50K] (PATH ID: 3)
| | | |       Projection: public.customer_dimension_DBD_1_rep_vmartdb_design_vmartdb_desi
| | | |       gn_node0001
| | | |       Materialize: customer_dimension.annual_income
```

Viewing data redistribution path

The optimizer broadcasts or resegments data, as needed.

Broadcasting sends a complete copy of an intermediate result to all nodes in the cluster. Broadcast is used for joins when

- One table is very small (usually the inner table) compared to the other
- HP Vertica can avoid other large upstream resegmentation operations

- Outer join or subquery semantics require one side of the join to be replicated

Resegmentation takes an existing projection or intermediate relation and segments the data evenly to each node in the cluster. At the end of the resegmentation operation, every row from the input relation is on exactly one node. Resegmentation is the operation used most often for distributed joins in HP Vertica if the data is not already segmented for local joins. See [Using Identically Segmented Projections](#) in the Programmer's Guide.

Resegment example

```
=> CREATE TABLE T1 (a INT, b INT) SEGMENTED BY HASH(a) ALL NODES;
=> CREATE TABLE T2 (x INT, y INT) SEGMENTED BY HASH(x) ALL NODES;
=> EXPLAIN SELECT * FROM T1 JOIN T2 ON T1.a = T2.y;

----- QUERY PLAN DESCRIPTION: -----
Access Path:
+-JOIN HASH [Cost: 639, Rows: 10K (NO STATISTICS)] (PATH ID: 1) Inner (RESEGMENT)
|  Join Cond: (T1.a = T2.y)
|  Materialize at Output: T1.b
|  Execute on: All Nodes
|  +-- Outer -> STORAGE ACCESS for T1 [Cost: 151, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
|  |      Projection: public.T1_b0
|  |      Materialize: T1.a
|  |      Execute on: All Nodes
|  +-- Inner -> STORAGE ACCESS for T2 [Cost: 302, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
|  |      Projection: public.T2_b0
|  |      Materialize: T2.x, T2.y
|  |      Execute on: All Nodes
```

Broadcast example

```
=> EXPLAIN SELECT * FROM T1 LEFT JOIN T2 ON T1.a > T2.y;
Access Path:
+-JOIN HASH [LeftOuter] [Cost: 40K, Rows: 10K (NO STATISTICS)] (PATH ID: 1) Inner (BROADCAST)
|  Join Filter: (T1.a > T2.y)
|  Materialize at Output: T1.b
|  Execute on: All Nodes
|  +-- Outer -> STORAGE ACCESS for T1 [Cost: 151, Rows: 10K (NO STATISTICS)] (PATH ID: 2)
|  |      Projection: public.T1_b0
|  |      Materialize: T1.a
|  |      Execute on: All Nodes
|  +-- Inner -> STORAGE ACCESS for T2 [Cost: 302, Rows: 10K (NO STATISTICS)] (PATH ID: 3)
|  |      Projection: public.T2_b0
|  |      Materialize: T2.x, T2.y
|  |      Execute on: All Nodes
```

Viewing analytic function path

HP Vertica attempts to optimize multiple SQL-99 [Analytic Functions](#) from the same query by grouping them together under the `Analytical Groups` area that appears on the `ORDER BY` and `PARTITION BY` clauses.

For each analytical group, HP Vertica performs a distributed sort and resegment of the data, if necessary.

You can tell how many sorts and resegments are required based on the query plan.

For example, the following `EXPLAIN` plan shows that the `FIRST_VALUE()` and `LAST_VALUE()` functions are in the same analytic group because their `OVER` clause is the same. In contrast, `ROW_NUMBER()` has a different `ORDER BY` clause, so it is in a different analytic group. Since both groups share the same `PARTITION BY deal_stage`, the data does not need to be resegmented between groups :

```
EXPLAIN SELECT
  first_value(deal_size) OVER (PARTITION BY deal_stage
    ORDER BY deal_size),
  last_value(deal_size) OVER (PARTITION BY deal_stage
    ORDER BY deal_size),
  row_number() OVER (PARTITION BY deal_stage
    ORDER BY largest_bill_amount)
FROM customer_dimension;

Access Path:
+-ANALYTICAL [Cost: 1K, Rows: 50K] (PATH ID: 1)
| Analytic Group
| Functions: row_number()
| Group Sort: customer_dimension.deal_stage ASC, customer_dimension.largest_bill_amount
t ASC NULLS LAST
| Analytic Group
| Functions: first_value(), last_value()
| Group Filter: customer_dimension.deal_stage
| Group Sort: customer_dimension.deal_stage ASC, customer_dimension.deal_size ASC NULL
S LAST
| Execute on: All Nodes
| +---> STORAGE ACCESS for customer_dimension [Cost: 263, Rows: 50K]
| (PATH ID: 2)
| | Projection: public.customer_dimension_DBD_1_rep_vmart_vmart_node0001
| | Materialize: customer_dimension.largest_bill_amount,
| | customer_dimension.deal_stage, customer_dimension.deal_size
| | Execute on: All Nodes
```

See Also

[The Window OVER\(\) Clause](#)

Viewing node down information

HP Vertica provides performance optimization when cluster nodes fail by distributing the work of the down nodes uniformly among available nodes throughout the cluster.

When a node in your cluster is down, the query plan identifies which node the query will execute on. To help you quickly identify down nodes on large clusters, EXPLAIN output lists up to a total of six nodes, if the number of running nodes is less than or equal to six, and lists only the down nodes if the number of running nodes is more than six.

Note: The node that is executing a query for a down node is not necessarily node each time you run the query.

The following table provides more detail:

| Node state | EXPLAIN output |
|--|--|
| If all nodes are up, EXPLAIN outputs the label "All Nodes" | Execute on: All Nodes |
| If fewer than 6 nodes are up, EXPLAIN lists the running nodes, up to a total of 6. | Execute on: [node_list]. |
| If more than 6 nodes are up, EXPLAIN lists only the non-running nodes | Execute on: All Nodes Except [node_list] |
| If the node list contains non-ephemeral nodes, EXPLAIN outputs the label "All Permanent Nodes" | Execute on: All Permanent Nodes |
| If the path is being run on the query initiator, EXPLAIN outputs the label "Query Initiator" | Execute on: Query Initiator |

Examples

In the following example, the down node is v_vmart_node0005, and the node v_vmart_node0006 will execute this run of the query.

```
=> EXPLAIN SELECT * FROM test;
QUERY PLAN

-----
QUERY PLAN DESCRIPTION:
-----
EXPLAIN SELECT * FROM my1table;
Access Path:
+-STORAGE ACCESS for my1table [Cost: 10, Rows: 2] (PATH ID: 1)
 | Projection: public.my1table_b0
 | Materialize: my1table.c1, my1table.c2
 | Execute on: All Except v_vmart_node0005
+-STORAGE ACCESS for my1table (REPLACEMENT FOR DOWN NODE) [Cost: 66, Rows: 2]
 | Projection: public.my1table_b1
 | Materialize: my1table.c1, my1table.c2
 | Execute on: v_vmart_node0006
```

The All Permanent Nodes output in the following example fragment denotes that the node list is for permanent (non-ephemeral) nodes only:

```
=> EXPLAIN SELECT * FROM my2table;
Access Path:
+-STORAGE ACCESS for my2table [Cost: 18, Rows:6 (NO STATISTICS)] (PATH ID: 1)
| Projection: public.my2tablee_b0
| Materialize: my2table.x, my2table.y, my2table.z
| Execute on: All Permanent Nodes
```

Viewing MERGE Path

By default, HP Vertica prepares an optimized query plan for a MERGE statement if the statement and its tables meet the criteria described in [Optimized Versus Non-Optimized MERGE](#).

How to use EXPLAIN to look for an optimized query plan for MERGE

Run MERGE with the [EXPLAIN](#) keyword. After the plan output displays, scroll past the GraphViz format and look for the [Semi] path, as shown in the following sample fragment. Semi means HP Vertica will prepare an optimized query plan for the MERGE statement.

```
...
Access Path:
+-DML DELETE [Cost: 0, Rows: 0]
| Target Projection: public.A_b1 (DELETE ON CONTAINER)
| Target Prep:
| Execute on: All Nodes
| +---> JOIN MERGEJOIN(inputs presorted) [Semi] [Cost: 6, Rows: 1 (NO STATISTICS)] (PATH ID: 1)
| | Inner (RESEGMENT)
| | | Join Cond: (A.a1 = VAL(2))
| | | Execute on: All Nodes
| | +-- Outer -> STORAGE ACCESS for A [Cost: 2, Rows: 2 (NO STATISTICS)] (PATH ID: 2)
...
```

When HP Vertica prepares a non-optimized execution plan for a MERGE statement, a RightOuter path appears in the plan's textual output, such as in the following fragment.

```
...
Access Path: +-DML MERGE
| Target Projection: public.locations_b1
| Target Projection: public.locations_b0
| Target Prep:
| Execute on: All Nodes
| +---> JOIN MERGEJOIN(inputs presorted) [RightOuter] [Cost: 28, Rows: 3 (NO STATISTIC S)] (PATH ID: 1) Outer (RESEGMENT) Inner (RESEGMENT)
| | | Join Cond: (locations.user_id = VAL(2)) AND (locations.location_x = VAL(2)) AND (locations.location_y = VAL(2))
| | | Execute on: All Nodes
| | +-- Outer -> STORAGE ACCESS for <No Alias> [Cost: 15, Rows: 2 (NO STATISTICS)] (PATH ID: 2)
...
```

How to use *grep* to look for optimized query plan for **MERGE**

To see if MERGE will run with optimization, *grep* for 'Semi' before you run the EXPLAIN statement and look for the [Semi] path in the output, as in the following fragment.

```
> \o | grep 'Semi'
> EXPLAIN MERGE INTO a USING c ON a1 = c1
  WHEN MATCHED THEN UPDATE SET a1 = c1, a2=c2,a3=c3
  WHEN NOT MATCHED THEN INSERT(a1,a2,a3) VALUES(c1,c2,c3); | +---> JOIN MERGEJOIN(inputs
presorted) [Semi] [Cost: 6, Rows: 1 (NO STATISTICS)] (PATH ID: 1)
  Inner (RESEGMENT)
> \o
```

Another way to see if HP Vertica will prepare an optimized query plan for MERGE is to *grep* for the *non-optimized* MERGE path, 'RightOuter'. A null result after the EXPLAIN statement, such as in the following example, means the MERGE statement will likely run with an optimized plan.

```
> \o | grep 'RightOuter'
> EXPLAIN MERGE INTO a USING c ON a1 = c1
  WHEN MATCHED THEN UPDATE SET a1 = c1, a2=c2,a3=c3
  WHEN NOT MATCHED THEN INSERT (a1,a2,a3) VALUES(c1,c2,c3);
> \o
```

Linking EXPLAIN Plans to Error Messages and Profiling Information

The `PATH ID` is a unique identifier that HP Vertica assigns to each operation (path) within a query plan. The same ID is shared among:

- [EXPLAIN](#) plans
- Join error messages
- [EXECUTION_ENGINE_PROFILES](#) system table
- [QUERY_PLAN_PROFILES](#) system table

This allows you to quickly trace issues to their root cause.

If a query returns a join error similar to the following examples, you can preface the query with EXPLAIN and look for `PATH ID n` in the output to see which join in the query had the problem.

```
ERROR: Join inner did not fit in memory ((B x A)using B_sp and A_sp (PATH ID: 1))
```

```
ERROR: Nonexistent foreign key value detected in FK-PK join
```



```
Hash-Join(public.fact x public.dim)  
using subquery and dim_p (PATH ID: 1); value 15
```

```
ERROR: Join ((public.ask x public.bid) using ask_super and bid_super  
(PATH ID: 2)) inner partition did not fit in memory; value Null
```

Example

In the following series of commands, EXPLAIN returns the PATH ID for each plan, PROFILE profiles the query, and the EXECUTION_ENGINE_PROFILES system table returns operating and profiling counters, along with the PATH_ID that links back to EXPLAIN:

1. Run **EXPLAIN** <query>. The command in the following example output returns five paths, each identified by a distinct PATH ID:

```
=> EXPLAIN SELECT * FROM fact JOIN dim ON x=y JOIN ext on y=z;  
Access Path:  
+-JOIN MERGEJOIN(inputs presorted) [Cost: 815, Rows: 10K (NO STATISTICS)] (PATH ID:  
1)  
| Join Cond: (dim.y = ext.z)  
| Materialize at Output: fact.x  
| Execute on: All Nodes  
| +- Outer -> JOIN MERGEJOIN(inputs presorted) [Cost: 408, Rows: 10K (NO STATISTIC  
S)] (PATH ID: 2)  
| | Join Cond: (fact.x = dim.y)  
| | Execute on: All Nodes  
| | +- Outer -> STORAGE ACCESS for fact [Cost: 202, Rows: 10K (NO STATISTICS)] (PAT  
H ID: 3)  
| | | Projection: public.fact_super  
| | | Materialize: fact.x  
| | | Execute on: All Nodes  
| | +- Inner -> STORAGE ACCESS for dim [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH  
ID: 4)  
| | | Projection: public.dim_super  
| | | Materialize: dim.y  
| | | Execute on: All Nodes  
| +- Inner -> STORAGE ACCESS for ext [Cost: 202, Rows: 10K (NO STATISTICS)] (PATH I  
D: 5)  
| | Projection: public.ext_super  
| | Materialize: ext.z  
| | Execute on: All Nodes
```

2. Run **PROFILE** <query> to save execution counters to the EXECUTION_ENGINE_PROFILES table.

```
=> PROFILE SELECT * FROM fact JOIN dim ON x=y JOIN ext on y=z;  
NOTICE: Statement is being profiled.  
HINT: select * from v_monitor.execution_engine_profiles  
where transaction_id=45035996273743212 and statement_id=2;
```

```
NOTICE: Initiator memory for query: [on pool sysquery: 16384 KB, minimum: 6020 KB]  
NOTICE: Total memory required by query: [16384 KB]
```

3. Query the `EXECUTION_ENGINE_PROFILES` system table for the join operations and PATH ID. The output refers to PATH IDs 1 and 2 in the EXPLAIN output.

```
=> SELECT node_name, operator_name, counter_name, path_id  
FROM execution_engine_profiles  
WHERE operator_name LIKE 'Join%' AND counter_name LIKE '%rows%';  
node_name | operator_name | counter_name | path_id  
-----+-----+-----+-----  
e0        | JoinManyFewMerge | rows produced | 1  
e0        | JoinManyFewMerge | rle rows produced | 1  
e0        | JoinManyFewMerge | estimated rows produced | 1  
e0        | JoinManyFewMerge | rows produced | 2  
e0        | JoinManyFewMerge | rle rows produced | 2  
e0        | JoinManyFewMerge | estimated rows produced | 2  
e1        | JoinManyFewMerge | rows produced | 1  
e1        | JoinManyFewMerge | rle rows produced | 1  
e1        | JoinManyFewMerge | estimated rows produced | 1  
e1        | JoinManyFewMerge | rows produced | 2  
e1        | JoinManyFewMerge | rle rows produced | 2  
e1        | JoinManyFewMerge | estimated rows produced | 2  
initiator | JoinManyFewMerge | rows produced | 1  
initiator | JoinManyFewMerge | rle rows produced | 1  
initiator | JoinManyFewMerge | estimated rows produced | 1  
initiator | JoinManyFewMerge | rows produced | 2  
initiator | JoinManyFewMerge | rle rows produced | 2  
initiator | JoinManyFewMerge | estimated rows produced | 2  
(18 rows)
```

Using the `QUERY_PLAN_PROFILES` table

The `QUERY_PLAN_PROFILES` system table contains real-time status for each path in a query plan, where each PATH ID provides the following information:

- Clock time the query spent on a particular path in the plan
- CPU resources used on that path
- Memory used
- Disk/network I/O done
- Path status (started/completed)
- Whether the query is running or finished running

Profiling Database Performance

To determine where time is spent during query execution, profile the queries. Unlike the [EXPLAIN](#) plan, where the cost and row counts are estimates, the counters and plans from query profile reflect what really happened and let you consider some of the following:

- Projection design issues such as segmentation and sort order
- If the query is network bound
- How much memory each operator allocated
- If a query rewrite might speed up the query
- How many threads are executing for each operator
- How the data has flowed through each operator at different points in time over the life of the query

Real-time profiling is always "on"; you do not need to explicitly enable it.

Profile data is available for:

- Any query that has been explicitly profiled
- Any query that is currently executing

Profile data is saved in the following system tables:

- [EXECUTION_ENGINE_PROFILES](#)—Provides detailed information about the execution run of each query.
- [QUERY_PROFILES](#) and [QUERY_PLAN_PROFILES](#)—Provides general information about queries that have executed, such as the query strings used and the duration of the queries.
- [SESSION_PROFILES](#)—Provides basic session parameters and lock timeout data.

If profiling is not enabled, HP Vertica does not save the data in [EXECUTION_ENGINE_PROFILES](#), but some data is saved in [QUERY_PROFILES](#), [QUERY_PLAN_PROFILES](#), and [SESSION_PROFILES](#).

HP Vertica continues to make enhancements to the [EXECUTION_ENGINE_PROFILES](#) system table and the **Workload Analyzer**, but the profiling methods described in this section are still a valuable tool for gaining insight into what happens during execution time.

Use the following functions to get query profile data:

- [Profiling Functions](#)
- [SET_CONFIG_PARAMETER\(\)](#)

How to Determine If Profiling Is Enabled

To determine if profiling is enabled, use the following command:

```
=> SELECT SHOW_PROFILING_CONFIG()  
SHOW_PROFILING_CONFIG  
-----  
Session Profiling: Local off, Global on  
EE Profiling:      Local off, Global on  
Query Profiling:   Local off, Global on  
(1 row)
```

You can enable profiling in the following ways:

- Set local profiling using the [Profiling Functions](#).
- Set global profiling using the [SET_CONFIG_PARAMETER\(\)](#) function.

How to Enable Profiling

To enable profiling for the current session, call the [ENABLE_PROFILING](#) function and pass it the profile type, which can be one of the following:

- *session*—Enables profiling basic session parameters and lock timeout data.
- *query*—Enables profiling for general information about queries that ran, such as the query strings used and the duration of queries.
- *ee*—Enables profiling for query execution runs.

To enable profiling for all sessions on all nodes, call the [SET_CONFIG_PARAMETER\(\)](#) function, as in the following command:

```
=> SELECT SET_CONFIG_PARAMETER( 'global-type', 1 )
```

global-type can be one of the following:

- *GlobalSessionProfiling*—Enables profiling for sessions.
- *GlobalQueryProfiling*—Enables profiling for queries.
- *GlobalEEProfiling*—Enables profiling for query execution runs.

Examples

The following statement enables profiling for the execution run of each query:

```
=> SELECT ENABLE_PROFILING('ee');
```

```
ENABLE_PROFILING
-----
EE Profiling Enabled
(1 row)
```

The following statement enables query profiling for all sessions and nodes:

```
=> SELECT SET_CONFIG_PARAMETER('GlobalSessionProfiling',1);
```

How to Disable Profiling

To disable profiling for the current session, call the [DISABLE_PROFILING](#) function:

```
=> SELECT DISABLE_PROFILING( 'profiling-type' )
```

profiling-type can be one of the following:

- *session*—Disables profiling basic session parameters and lock time out data.
- *query*—Disables profiling for general information about queries that ran, such as the query strings used and the duration of queries.
- *ee*—Disables profiling for query execution runs.

To disable profiling for all sessions on all nodes, call the [SET_CONFIG_PARAMETER](#) function:

```
=> SELECT SET_CONFIG_PARAMETER( 'global-type', 0 )
```

global-type can be one of the following:

- *GlobalSessionProfiling*—Disables profiling for sessions.
- *GlobalEEProfiling*—Disables profiling for query execution runs.
- *GlobalQueryProfiling*—Disables profiling for queries.

Example

The following command disables profiling on query execution runs:

```
=> SELECT DISABLE_PROFILING('ee');
DISABLE_PROFILING
-----
EE Profiling Disabled
(1 row)
```

The following command disables query profiling for all sessions and nodes:

```
=> SELECT SET_CONFIG_PARAMETER('GlobalSessionProfiling',0); SET_CONFIG_PARAMETER
-----
Parameter set successfully
(1 row)
```

About Real-Time Profiling

Real-time profiling provides a way to monitor long-running queries while they are executing.

Real-time profiling counters are available for all currently executing statements—including internal operations such as **mergeout**, **recovery**, and **refresh**—but only while the statements are executing. Unless you explicitly enabled profiling using the keyword **PROFILE** or [ENABLE_PROFILING](#), profiling counters are not available after the statement completes.

Tip: To view real-time profiling data, queries need, at a minimum, the `transaction_id` for the transaction of interest. If multiple statements have been executed within the transaction, you also need the `Statement_id`. The transaction IDs for internal operations are in the `vertica.log` files.

About profiling counters

The [EXECUTION_ENGINE_PROFILES](#) system table contains available profiling counters for internal operations and user statements. Some of the most useful counters are:

- Execution time (μ s)
- Rows produced
- Total merge phases
- Completed merge phases
- Current size of temp files (bytes)

About query plan profiles

If you want to know how much time a query spent on a particular operation in a query plan, you can observe the real-time flow of data through the plan by querying the [QUERY_PLAN_PROFILES](#) system table. See [Profiling query plans](#).

Example

To monitor a refresh operation that was initiated on node0001, find the “`select start_refresh()`” entry in the `vertica.log` file on node0001. You'll see something similar to the following log fragment, where the transaction ID for this recovery operation is `a000000000227`:

```
2011-04-21 13:34:56.494 Refresh:0xb9ab5e0 [Refresh] <INFO> Refreshing projection public.fact_p from buddy
2011-04-21 13:34:56.494 Refresh:0xb9ab5e0 [Refresh] <INFO> Refreshing projection public.fact_p from buddy, historical epochs 0-12, oid 45035996273713164
2011-04-21 13:34:56.497 nameless:0xb972330 [Txn] <INFO> Begin Txn: a000000000227 'Refresh through recovery'
2011-04-21 13:34:56.497 nameless:0xb972330 [Command] <INFO> Performing refresh through recovery on projection fact_p (45035996273713164) 0-12
2011-04-21 13:34:56.497 nameless:0xb972330 <LOG> @v_db_node0001: 00000: Recover alterSpec 45035996273713164 0-12
2011-04-21 13:34:56.500 nameless:0xb972330 [Command] <INFO> (a000000000227) Executing the recovery plan
```

To monitor the profiling counters, you can run a command like the following:

```
=> SELECT * FROM execution_engine_profiles
WHERE TO_HEX(transaction_id)='a000000000227'
AND counter_name = 'execution time (us)'
ORDER BY node_name, counter_value DESC;
```

In this example, find the operators with the largest execution time on each node:

```
=> SELECT node_name, operator_name, counter_value execution_time_us
FROM v_monitor.execution_engine_profiles
WHERE counter_name='execution time (us)'
ORDER BY node_name, counter_value DESC;
```

You can use the Linux watch command to monitor long-running queries with one-second updates:

```
WATCH -n 1 -d "vsq1 -c \"select node_name, operator_name, counter_value execution_time_us ... \""
```

See Also

- [Profiling query plans](#)

System tables with profile data

The system tables for profiling are as follows:

| Virtual Table | Description |
|---|--|
| EXECUTION_ENGINE_PROFILES | Provides profile information for query execution runs. |
| QUERY_PLAN_PROFILES | Provides real-time status for each path in a query plan. |
| QUERY_PROFILES | Provides profile information for queries. |
| SESSION_PROFILES | Provides profile information for sessions. |

What to look for in query profiles

Profile data can show data skew, when some nodes are processing more data than others. The `rows produced` counter in the `EXECUTION_ENGINE_PROFILES` table shows how many rows have been processed by each operator. Comparing the `rows produced` across all nodes for a given operator reveals if there is a data skew issue.

Note: Some of the profiling operators shown in `EXECUTION_ENGINE_PROFILES` are generic, such as `Join`. The [EXPLAIN](#) plan includes more details about the specific join that is executing.

Viewing Profile Data in Management Console

Management Console allows you to view profile data about a single query, allowing you to:

- Review the profile data in an easy-to-read format.
- View details about projections metadata, execution events, and optimizer events.
- Quickly identify how much time was spent in each phase of query execution and which phases took the most amount of time.

After you select the database you want to work with, there are two ways to view the profile data using Management Console:

To focus on specific areas of the database activity, such as spikes in CPU usage, take these steps:

1. For the database you have selected, click **Activity** at the bottom of the Management Console window.
2. Select **Queries** from the drop-down list at the top of the page.

You can also access the profile data for **User Sessions** and **User Query Phases** data if you select those options on the drop-down list.

3. Click the spot on the graph for which you want to view the query plan.

The activity detail window appears.

4. in the **View Plan** column, click **Profile** next to the command for which you want to view the query plan. Only certain queries, like `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, have profile data.

The **Explain** window opens. The query you selected appears in the text box. HP Vertica reruns the query and when it completes, the profile data and metrics for that query loads in the window, below the query.

To review the profile data for a specific query, take these steps:

2. Select the database for the query you want to profile.
3. Click **Explain** at the bottom of the Management Console window.

The **Explain** window opens. The text box is empty because you have not selected a query.

4. Type or paste the query text into the text box.
5. Click **Profile Query**.

HP Vertica reruns the query. An error occurs if:

- The query is invalid.
- The query begins with a comment.
- The query contains hints.

If you enter more than one query, Management Console profiles only the first query.

Note: While HP Vertica is re-executing and profiling the query, a **Cancel Profiling** button is enabled for a short time, allowing you to cancel the profiling task. If the **Cancel Profiling** button is disabled, the Management Console does not have the proper information to cancel the query or the query is no longer running in the database. You can cancel the profiling task only when the **Cancel Profiling** button is enabled.

When processing completes, the profile data and metrics display below the text box. The pie chart shows data for each phase of the query plan, and metrics for each query path appear next to the query path in the query plan.

Expanding and collapsing query path profile data

When you have a query on the EXPLAIN window, the profile data displays in the right-hand side of the lower half of the window. The query path information can be lengthy, so you can collapse path information that is uninteresting, or expand paths that you want to focus on.

- To collapse all the query paths, click **Collapse All**.
- To expand all the query paths, click **Expand All**.
- To expand an individual query path so you can see details about that step in processing the query, click the first line of the path information. Click the first line again to collapse the path data.

For information about what the profile data means, see [About Profile Data in Management Console](#).

About Profile Data in Management Console

After you request EXPLAIN and profile data for a specific query, the Management Console Explain page looks as follows:

The screenshot displays the Management Console interface for an 'EXPLAIN' query. At the top, there's a navigation bar with 'Databases and Clusters', 'chris390B', and 'Explain Plan'. Below this is a text area containing a complex SQL query. To the right of the query is a 'Query Phase Durations' pie chart with a legend showing: Plan (7.3%), InitPlan (0.1%), SerializePlan (0.8%), PopulateVirtualProjection (1.8%), PreparePlan (8.4%), and CompilePlan (1.3%). Below the query and chart are buttons for 'View Projection Metadata', 'View Execution Events', and 'View Optimizer Events'. The main part of the page is the 'EXPLAIN' output, which includes a tree of query operations like 'JOIN HASH [RightOuter]', 'GROUPBY HASH', 'SORT', and 'UNION ALL'. To the right of the EXPLAIN text are five columns of progress bars labeled 'Disk', 'Memory', 'Sent', 'Received', and 'Time'. At the bottom, there's a navigation bar with 'Overview', 'Activity', 'Manage', 'License', 'Settings', 'Design', and 'Explain'.

There are several kinds of profile data you can review on the Management Console **Explain** page:

- [Projection metadata](#)
- [Query phase duration](#)
- [Profile metrics](#)
- [Execution events](#)
- [Optimizer events](#)

The projection metadata, execution events, and optimizer events data are only available after you profile the query.

Projection metadata

To view projection metadata for a specific projection, click the projection name in the EXPLAIN output. Metadata for that projection opens in a pop-up window.

To view projection data for all projections accessed by that query, click the **View Projection Metadata** button at the top of the page. The metadata for all projections opens in a new browser window.

Note: If the **View Projection Metadata** button is not enabled, click **Profile Query** to retrieve the profile data, including the projection metadata.

The projection metadata includes the following information:

- Projection ID
- Schema name
- Whether or not it is a superprojection
- Whether or not it is a pre-join projection.
- Whether or not it is segmented
- Whether or not it is up to date
- Whether or not it has statistics
- IDs of the nodes the projection is stored on
- Sort columns
- Owner name
- Anchor table name

To display a SQL script that can recreate the projection on a different cluster, click **Click to get export data** to the right of any projection. This script is identical to the output of the [EXPORT_OBJECTS](#) function. The SQL script opens in a pop-up window.

Copy and paste the command from this window, and click **Close** when finished. When you finish reviewing the projection metadata, close the browser window.

Query phase duration

The pie chart in the upper-right corner of the **Explain** window shows what percentage of total query processing was spent in each phase of processing the query.

The phases included in the pie chart (when applicable) are:

- Plan
- InitPlan
- SerializePlan
- PopulateVirtualProjection
- PreparePlan

- CompilePlan
- ExecutePlan
- AbandonPlan

Hover over the slices on the pie chart or over the names of the phases in the box to get the approximate number of milliseconds (ms) used during each phase.

Note: The time data in the profile metrics might not match the times in the query phase duration because the query phase duration graph uses the longest execution time for a given phase from all the nodes. Further noise may be added through network latency, which is not taken into account in these calculations.

Profile metrics

The area to the right of each query path contains profile metrics for that path.

- **Disk**—Bytes of data accessed from disks by each query path. If none of the query paths accessed the disk data, all the values are 0.
- **Memory**—Bytes of data accessed from memory by each query path.
- **Sent**—Bytes of data sent across the cluster by this query path.
- **Received**—Bytes of data received across the cluster by this query path.
- **Time**—Number of milliseconds (ms) that this query path took to process on a given node. The sum of this data does not match the total time it tooks to execute the query because many tasks are executed in parallel on different nodes.

Note: The time data in the profile metrics might not match the times in the [Query phase duration](#) data because the query phase duration graph uses the longest execution time for a given phase from all the nodes. Further noise may be added through network latency, which is not taken into account in these calculations.

Execution events

To help you monitor your database system, HP Vertica logs significant events that affect database performance and functionality. Click **View Execution Events** to see information about the events that took place while the query was executed.

Note: If the **View Execution Events** button is not enabled, click **Profile Query** to retrieve the profile data, including the execution events.

The arrows on the header of each column allow you to sort the table in ascending or descending order of that column.

The execution events are described in the following table.

| Event characteristic | Details |
|----------------------|--|
| Time | Clock time when the event took place. |
| Node name | Name of the node for which information is listed |
| Session ID | Identifier of the session for which profile information is captured |
| User ID | Identifier of the user who initiated the query |
| Request ID | Unique identifier of the query request in the user session. |
| Event type | <p>Type of event processed by the execution engine. Examples include but are not limited to:</p> <ul style="list-style-type: none"> • PREDICATE OUTSIDE HISTOGRAM • NO HISTOGRAM • MEMORY LIMIT HIT • GROUP_BY_SPILLED • JOIN_SPILLED • PARTITIONS_ELIMINATED • MERGE_CONVERTED_TO_UNION |
| Event description | Generic description of the event |
| Operator name | <p>Name of the Execution Engine component that generated the event. Examples include but are not limited to:</p> <ul style="list-style-type: none"> • DataSource • DataTarget • NetworkSend • NetworkRecv • StorageUnion <p>Values from the Operator name and Path ID columns let you tie a query event back to a particular operator in the EXPLAIN plan. If the event did not come from a specific operator, this column is NULL.</p> |

| | |
|------------------|--|
| Path ID | Unique identifier that HP Vertica assigns to a query operation or a path in a query plan. If the event did not come from a specific operator, this column is NULL. |
| Event OID | A unique ID that identifies the specific event. |
| Event details | A brief description of the event and details pertinent to the specific situation. |
| Suggested action | Recommended actions (if any) to improve query processing. |

Optimizer events

To help you monitor your database system, HP Vertica logs significant events that affect database performance and functionality. Click **View Optimizer Events** to see information about the events that took place while the optimizer was planning the query.

Note: If the **View Optimizer Events** button is not enabled, click **Profile Query** to retrieve the profile data, including the optimizer events.

The arrows on the header of each column allow you to sort the table in ascending or descending order of that column.

The optimizer events are described in the following table.

| Event characteristic | Details |
|----------------------|---|
| Time | Clock time when the event took place. |
| Node name | Name of the node for which information is listed. |
| Session ID | Identifier of the session for which profile information is captured. |
| User ID | Identifier of the user who initiated the query. |
| Request ID | Unique identifier of the query request in the user session. |
| Event type | Type of event processed by the optimizer. |
| Event description | Generic description of the event. |
| Event OID | A unique ID that identifies the specific event. |
| Event details | A brief description of the event and details pertinent to the specific situation. |
| Suggested action | Recommended actions (if any) to improve query processing. |

Clearing query data

When you have finished reviewing the current query data, click **Clear All** to clear the query text and data. Alternatively, to display information about another query, enter the query text and click **Explain Plan** or **Profile Query**.

Viewing Profile Data in vsql

There are several ways to view profile data using vsql commands:

- [How to profile a single statement](#)
- [How to save query plan information](#)
- [Sample views for counter information](#)
- [About Real-Time Profiling](#)
- [How to label queries for profiling](#)

How to profile a single statement

To profile a single statement, add the **PROFILE** keyword to the beginning of the statement, which saves profiling information for future analysis:

```
=> PROFILE SELECT customer_name, annual_income
FROM public.customer_dimension
WHERE (customer_gender, annual_income) IN (
  SELECT customer_gender, MAX(annual_income)
  FROM public.customer_dimension
  GROUP BY customer_gender);
```

While a query is executing, a notice and hint display in the terminal window. For example, the preceding query returns the following:

```
NOTICE 4788: Statement is being profiled
HINT: Select * from v_monitor.execution_engine_profiles where transaction_id=45035996273
726389 and statement_id=10;
NOTICE 3557: Initiator memory for query: [on pool general: 732168 KB, minimum: 451218 K
B]
NOTICE 5077: Total memory required by query: [732168 KB]
 customer_name | annual_income
-----+-----
James M. McNulty | 999979
Emily G. Vogel | 999998
(2 rows)
```

Tip: Use the statement returned by the hint as a starting point for reviewing the query's profiling data, such as to see what counters are available.

Real-Time Profiling Example

The following sample statement requests the operators with the largest execution time on each node:

```
=> SELECT node_name, operator_name, counter_valueexecution_time_us
      FROM v_monitor.execution_engine_profiles
      WHERE counter_name='execution time (us)'
      ORDER BY node_name, counter_value DESC;
```

How to Use the Linux watch Command

You can use the Linux watch command to monitor long-running queries with one-second updates; for example:

```
WATCH -n 1 -d "vsq1-c \"select node_name, operator_name, counter_valueexecution_time_u
s... \""
```

How to Find Out Which Counters are Available

HP Vertica keeps track of the following counters during query execution:

```
=> SELECT DISTINCT(counter_name) FROM EXECUTION_ENGINE_PROFILES;
      counter_name
-----
estimated rows produced
bytes spilled
rle rows produced
join inner current size of temp files (bytes)
request wait (us)
start time
intermediate rows to process
producer wait (us)
rows segmented
consumer stall (us)
bytes sent
rows sent
join inner completed merge phases
encoded bytes received
cumulative size of raw temp data (bytes)
end time
bytes read from cache
total merge phases
rows pruned by valindex
cumulative size of temp files (bytes)
output queue wait (us)
rows to process
input queue wait (us)
rows processed
memory allocated (bytes)
```



```
join inner cumulative size of temp files (bytes)
current size of temp files (bytes)
join inner cumulative size of raw temp data (bytes)
bytes received
file handles
bytes read from disk
join inner total merge phases
completed merge phases
memory reserved (bytes)
clock time (us)
response wait (us)
network wait (us)
rows received
encoded bytes sent
execution time (us)
producer stall (us)
buffers spilled
rows produced
(43 rows)
```

Sample views for counter information

The EXECUTION_ENGINE_PROFILES table contains the data for each profiling counter as a row within the table. For example, the execution time (us) counter is in one row, and the rows produced counter is in a second row. Since there are many different profiling counters, many rows of profiling data exist for each operator. Some sample views are installed by default to simplify the viewing of profiling counters.

Running scripts to create the sample views

The following script creates the v_demo schema and places the views in that schema.

```
/opt/vertica/scripts/demo_eeprof_view.sql
```

Viewing counter values using the sample views

There is one view for each of the profiling counters to simplify viewing of a single counter value. For example, to view the execution time for all operators, issue the following command from the database:

```
=> SELECT * FROM v_demo.eeprof_execution_time_us;
```

To view all counter values available for all profiled queries:

```
=> SELECT * FROM v_demo.eeprof_counters;
```

To select all distinct operators available for all profiled queries:

```
=> SELECT * FROM v_demo.eeprof_operators;
```

Combining sample views

These views can be combined:

```
=> SELECT * FROM v_demo.eeprof_execution_time_us  
NATURAL LEFT OUTER JOIN v_demo.eeprof_rows_produced;
```

To view the execution time and rows produced for a specific transaction and `statement_id` ranked by execution time on each node:

```
=> SELECT * FROM v_demo.eeprof_execution_time_us_rank  
WHERE transaction_id=45035996273709699  
AND statement_id=1  
ORDER BY transaction_id, statement_id, node_name, rk;
```

To view the top five operators by execution time on each node:

```
=> SELECT * FROM v_demo.eeprof_execution_time_us_rank  
WHERE transaction_id=45035996273709699  
AND statement_id=1 AND rk<=5  
ORDER BY transaction_id, statement_id, node_name, rk;
```

Viewing real-time profile data

You can query real-time profiling data during execution runs for a long-running query or other operation. The demo view `v_demo.eeprof_execution_time_us_rank` is helpful for viewing the current top five operators by execution time prior to the query completing:

```
=> SELECT * FROM v_demo.eeprof_execution_time_us_rank  
WHERE transaction_id=45035996273709699 AND statement_id=1 AND rk<=5  
ORDER BY transaction_id, statement_id, node_name, rk;
```

The Linux `watch` command is useful for long-running queries or long-running internal operations by observing which part of the query plan is currently active:

```
=> watch -d -n 2 "vsq1 -c \  
=> SELECT * FROM v_demo.eeprof_execution_time_us_rank  
WHERE transaction_id=45035996273709699 AND statement_id=1 AND rk<=5  
ORDER BY transaction_id, statement_id, node_name, rk;\" "
```

This `watch` command executes the query every two seconds and highlights the differences between successive updates.

Tip: Using `watch` is a convenient way to monitor the currently executing operators within the

plan on each nodes in the HP Vertica cluster. `watch` is also a convenient way to monitor workloads that might be unbalanced between nodes. For example, if some nodes become idle while other nodes are still active. Such imbalances could be caused by data skews or by hardware issues.

See Also

- [Profiling query plans](#)

How to label queries for profiling

To quickly identify queries for profiling and debugging, you can pass a user-defined label to an HP Vertica query as a hint.

Query labels must have the following characteristics:

- All characters must be alphanumeric with any number of underscores (`_`) and dollar signs (`$`)
- The maximum label length is 128 **octets**
- The label cannot contain space characters
- The first character cannot be numeric

Label syntax

Labels take the following form:

```
/*+label(Label_name)*/
```

You can use optional spaces before and after the plus sign in label hints (between the `/*` and the `+`). For example, HP Vertica accepts each of the following directives:

```
/*+label(label_name)*/  
/* + label(label_name)*/  
/*+ label(label_name)*/  
/*+label(label_name) */
```

Example

Here's an example of some simple label hints and their associated queries:

```
SELECT /*+label(myselectquery)*/ COUNT(*) FROM t;  
INSERT /*+label(myinsertquery)*/ INTO t VALUES(1);  
UPDATE /*+label(myupdatequery)*/ t SET a = 2 WHERE a = 1;  
DELETE /*+label(mydeletequery)*/ FROM t WHERE a = 1;  
SELECT /*+ label(abc123) */table_name FROM system_tables LIMIT 5;
```

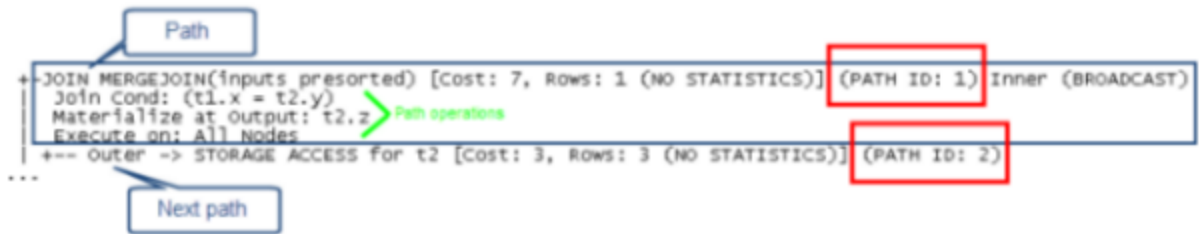
After you added a label to one or more queries, query the `QUERY_PROFILES` system table to see which queries ran with your supplied labels.

The `QUERY_PROFILES` system table `IDENTIFIER` column returns the user-defined label that you previously assigned to a query. Here's an example of the table's putput using the query lables

```
=> SELECT identifier, query FROM query_profiles;
  identifier | query
-----+-----
myselectquery | SELECT /*+label(myselectquery)*/ COUNT(*) FROM t;
myinsertquery | INSERT /*+label(myinsertquery)*/ INTO t VALUES(1);
myupdatequery | UPDATE /*+label(myupdatequery)*/ t SET a = 2 WHERE a = 1;
mydeletequery | DELETE /*+label(mydeletequery)*/ FROM t WHERE a = 1;
              | SELECT identifier, query from query_profiles;
(5 rows)
```

Profiling query plans

To monitor real-time flow of data through a query plan, query the `V_MONITOR.QUERY_PLAN_PROFILES` system table. Information returned by this table is useful for letting you know *what* a query did *when*, which occurs throughout a plan in a series of steps, called **paths**. Each path has an associated PATH ID, as illustrated in the following `EXPLAIN` command fragment.



The same PATH ID is shared among `EXPLAIN` plans, join error messages, and `EXECUTION_ENGINE_PROFILES` and `QUERY_PLAN_PROFILES` system tables, making it easy to quickly trace issues to their root cause.

For each PATH ID, the `QUERY_PLAN_PROFILES` system table provides a high-level summary of:

- The time a particular query operation took to execute
- How much memory that path's operation consumed
- Size of data sent/received over the network
- Whether the path operation is executing

For example, you might observe that a `GROUP BY HASH` operation executed in 0.2 seconds using 100MB of memory.

Note: Real-time status is available in `QUERY_PLAN_PROFILES` system table output only; it is not available in `EXPLAIN` plan output.

What you need for query plan profiling

In order to view real-time profiling data, you need, at a minimum, the `transaction_id` for the transaction you want to monitor. If multiple statements executed in the same transaction, you also need the `statement_id`. You can get both statement and transaction IDs by issuing the `PROFILE <query>` command, as well as by querying the `CURRENT_SESSION` system table, as in the following example:

```
=> SELECT transaction_id, statement_id from current_session;
  transaction_id | statement_id
-----+-----
45035996273955001 | 4
```

(1 row)

Transaction IDs for internal operations, such as **mergeout**, **recovery**, and **refresh**, are stored in the `vertica.log` files.

See Also

- [Understanding Query Plans](#)
- [Collecting Database Statistics](#)
- [Analyzing Workloads](#)

How to get query plan status for small queries

Real-time profiling counters, stored in the `EXECUTION_ENGINE_PROFILES` system table, are available for all currently executing statements—including internal operations, such as a **mergeout**.

Profiling counters are available after query execution has completed if any of the following conditions are true:

- The query was run via the `PROFILE <query>` command
- Systemwide profiling has been enabled through the `ENABLE_PROFILING()` function
- The query took longer than two seconds to run

Profiling counters are saved in the `EXECUTION_ENGINE_PROFILES` system table until the storage quota has been exceeded.

Here's an example:

1. Profile the query to get the `transaction_id` and `statement_id` from from `EXECUTION_ENGINE_PROFILES`; for example:

```
=> PROFILE SELECT * FROM t1 JOIN t2 ON t1.x = t2.y;
NOTICE 4788: Statement is being profiled
HINT: Select * from v_monitor.execution_engine_profiles where transaction_id=4503599
6273955065 and statement_id=4;
NOTICE 3557: Initiator memory for query: [on pool general: 248544 KB, minimum: 24854
4 KB]
NOTICE 5077: Total memory required by query: [248544 KB]
 x | y | z
-----+-----
 3 | 3 | three
(1 row)
```

2. Query the `QUERY_PLAN_PROFILES` system table.

Note: For best results, sort on the `transaction_id`, `statement_id`, `path_id`, and `path_line_index` columns.

```
=> SELECT ... FROM query_plan_profiles
      WHERE transaction_id=45035996273955065 and statement_id=4;
      ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

How to get query plan status for large queries

Real-time profiling is designed to monitor large (long-running) queries. Take the following steps to monitor plans for large queries:

1. Get the statement and transaction IDs for the query plan you want to profile by querying the `CURRENT_SESSION` system table:

```
=> SELECT transaction_id, statement_id from current_session;
transaction_id | statement_id
-----+-----
45035996273955001 | 4
(1 row)
```

2. Run the query:

```
=> SELECT * FROM t1 JOIN t2 ON x=y JOIN ext on y=z;
```

3. Query the `QUERY_PLAN_PROFILES` system table, and sort on the `transaction_id`, `statement_id`, `path_id`, and `path_line_index` columns.

```
=> SELECT ... FROM query_plan_profiles WHERE transaction_id=45035996273955001 and statement_id=4
      ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

You can also use the Linux `watch` command to monitor long-running queries with two-second updates. See [Viewing real-time profile data](#).

Example

The following series of commands creates a table for a long-running query and then runs the `QUERY_PLAN_PROFILES` system table:

```
=> CREATE TABLE longq(x int);
CREATE TABLE
=> COPY longq FROM STDIN;
```

```
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 1
>> 2
>> 3
>> 4
>> 5
>> 6
>> 7
>> 8
>> 9
>> 10
>> \.
=> INSERT INTO longq SELECT f1.x+f2.x+f3.x+f4.x+f5.x+f6.x+f7.x
      FROM longq f1
      CROSS JOIN longq f2
      CROSS JOIN longq f3
      CROSS JOIN longq f4
      CROSS JOIN longq f5
      CROSS JOIN longq f6
      CROSS JOIN longq f7;
      OUTPUT
-----
10000000
(1 row)
=> COMMIT;
COMMIT
```

Suppress query output on the terminal window by using the **vsq! \o** command:

```
=> \o /home/dbadmin/longQprof
```

Query the new table:

```
=> SELECT * FROM longq;
```

Get the transaction and statement IDs:

```
=> SELECT transaction_id, statement_id from current_session;
  transaction_id | statement_id
-----+-----
45035996273955021 |          4
(1 row)
```

Turn off the **\o** command so that HP Vertica continues to save query plan information to the file you specified. Alternatively, leave it on and examine the file after you query the **QUERY_PLAN_PROFILES** system table.

```
=> \o
```

Query the **QUERY_PLAN_PROFILES** system table;


```
=> SELECT
    transaction_id,
    statement_id,
    path_id,
    path_line_index,
    is_executing,
    running_time,
    path_line
FROM query_plan_profiles
WHERE transaction_id=45035996273955021 AND statement_id=4
ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

Improving the readability of QUERY_PLAN_PROFILES output

Output from the QUERY_PLAN_PROFILES table can be very wide because of the path_line column, so for best results and optimal readability, when you query the QUERY_PLAN_PROFILES table, consider using one or more of the following methods:

- Sort the system table output by transaction_id, statement_id, path_id, and path_line_index, as in this example:

```
=> SELECT ... FROM query_plan_profiles
    WHERE ...
    ORDER BY transaction_id, statement_id, path_id, path_line_index;
```

- Consider using column aliases to decrease column width, as in the following sample query:

```
=> SELECT statement_id AS sid, path_id AS id, path_line_index AS order,
    is_started AS start, is_completed AS end, is_executing AS exe,
    running_time AS run, memory_allocated_bytes AS mem,
    read_from_disk_bytes AS read, received_bytes AS rec,
    sent_bytes AS sent, FROM query_plan_profiles
WHERE transaction_id=45035996273910558 AND statement_id=3
ORDER BY transaction_id, statement_id, path_id, path_line_index;;
```

- Suppress output from the screen and save profiling data to a file by using the **vsq| \o** command. For example:

- a. Turn on the \o command.

```
=> \o /home/dbadmin/long-queries
```

- b. Run the query using the EXPLAIN command.

```
=> EXPLAIN SELECT * FROM customer_dimension;
```

- c. Turn off the `\o` command.

```
=> \o
```

See [How to save query plan information](#) for details.

Managing query profile data

HP Vertica retains data for queries until the storage quota for the table is exceeded, when it automatically purges the oldest queries to make room for new ones. You can clear profiled data by manually calling one of the following functions:

- [CLEAR_PROFILING\(\)](#) clears profiled data from memory. For example, the following command clears profiling for general query-run information, such as the query strings used and the duration of queries.

```
=> SELECT CLEAR_PROFILING('query');
```

- [CLEAR_DATA_COLLECTOR\(\)](#) clears all memory and disk records on the Data Collector tables and functions and resets collection statistics in the `V_MONITOR.DATA_COLLECTOR` system table.
- [FLUSH_DATA_COLLECTOR\(\)](#) waits until memory logs are moved to disk and then flushes the Data Collector, synchronizing the DataCollector log with the disk storage.

Configuring data retention policies

HP Vertica retains the historical data it gathers based on retention policies, which a **superuser** can configure. See [Retaining Monitoring Information](#).

Reacting to suboptimal query plans

If profiling uncovers a suboptimal query, invoking one of the following functions might help:

- [ANALYZE_WORKLOAD\(\)](#) analyzes system information held in system tables and provides tuning recommendations that are based on a combination of statistics, system and **data collector** events, and database-table-projection design.
- [ANALYZE_STATISTICS\(\)](#) and [ANALYZE_HISTOGRAM\(\)](#) both collect and aggregates data samples and storage information from all nodes that store projections associated with the specified table or column with one primary difference:
 - `ANALYZE_STATISTICS` uses a fixed-size statistical data sampling (10% per disk). This function returns results quickly, but it is less accurate than using `ANALYZE_HISTOGRAM` to get a larger sampling of disk data.

- `ANALYZE_HISTOGRAM` takes a user-specified percentage of disk data sampling (from 1 - 100). If you analyze more than 10 percent data per disk, this function returns more accurate data than `ANALYZE_STATISTICS`, but it takes proportionately longer to return statistics.

You can also run your query through the Database Designer. See [Creating an Incremental Design Using the](#) .

About Locales

HP Vertica supports the following internationalization features:

Unicode Character Encoding: UTF-8 (8-bit UCS/Unicode Transformation Format)

UTF-8 is an abbreviation for Unicode Transformation Format-8 (where 8 equals 8-bit) and is a variable-length character encoding for Unicode created by Ken Thompson and Rob Pike. UTF-8 can represent any universal character in the Unicode standard, yet the initial encoding of byte codes and character assignments for UTF-8 is coincident with ASCII (requiring little or no change for software that handles ASCII but preserves other values).

All input data received by the database server is expected to be in UTF-8, and all data output by HP Vertica is in UTF-8. The ODBC API operates on data in UCS-2 on Windows systems, and normally UTF-8 on Linux systems. (A UTF-16 ODBC driver is available for use with the DataDirect ODBC manager.) JDBC and ADO.NET APIs operate on data in UTF-16. The client drivers automatically convert data to and from UTF-8 when sending to and receiving data from HP Vertica using API calls. The drivers do not transform data loaded by executing a [COPY](#) or [COPY LOCAL](#) statement.

Locales

The locale is a parameter that defines the user's language, country, and any special variant preferences, such as collation. HP Vertica uses the locale to determine the behavior of various string functions as well for collation for various SQL commands that require ordering and comparison; for example, GROUP BY, ORDER BY, joins, the analytic ORDER BY clause, and so forth.

By default, the locale for the database is `en_US@collation=binary` (English US). You can establish a new default locale that is used for all sessions on the database, as well as override individual sessions with different locales. Additionally the locale can be set through [ODBC](#), [JDBC](#), and [ADO.net](#).

See the following topics in the Administrator's Guide

- [Implement Locales for International Data Sets](#)
- [Supported Locales](#)
- [Appendix](#)

Notes

- Projections are always collated using the `en_US@collation=binary` collation regardless of the session collation. Any locale-specific collation is applied at query time.

- The maximum length parameter for VARCHAR and CHAR data type refers to the number of **octets** (bytes) that can be stored in that field and not number of characters. When using multi-byte UTF-8 characters, the fields must be sized to accommodate from 1 to 4 bytes per character, depending on the data.
- When the locale is non-binary, the collation function is used to transform the input to a binary string which sorts in the proper order.

This transformation increases the number of bytes required for the input according to this formula:

```
result_column_width = input_octet_width * CollationExpansion + 4
```

CollationExpansion defaults to 5.

Locale Aware String Functions

HP Vertica provides string functions to support internationalization. Unless otherwise specified, these string functions can optionally specify whether VARCHAR arguments should be interpreted as octet (byte) sequences, or as (locale-aware) sequences of characters. You do this by adding "USING OCTETS" and "USING CHARACTERS" (default) as a parameter to the function. The following is the full list of string functions that are locale aware:

- **BTRIM** removes the longest string consisting only of specified characters from the start and end of a string.
- **CHARACTER_LENGTH** returns an integer value representing the number of characters or octets in a string.
- **GREATEST** returns the largest value in a list of expressions.
- **GREATESTB** returns its greatest argument, using binary ordering, not UTF-8 character ordering.
- **INITCAP** capitalizes first letter of each alphanumeric word and puts the rest in lowercase.
- **INSTR** searches string for substring and returns an integer indicating the position of the character in string that is the first character of this occurrence.
- **LEAST** returns the smallest value in a list of expressions.
- **LEASTB** returns its least argument, using binary ordering, not UTF-8 character ordering.
- **LEFT** returns the specified characters from the left side of a string.
- **LENGTH** takes one argument as an input and returns returns an integer value representing the number of characters in a string.

- **LTRIM** returns a VARCHAR value representing a string with leading blanks removed from the left side (beginning).
- **OVERLAY** returns a VARCHAR value representing a string having had a substring replaced by another string.
- **OVERLAYB** returns an octet value representing a string having had a substring replaced by another string.
- **REPLACE** replaces all occurrences of characters in a string with another set of characters.
- **RIGHT** returns the *length* right-most characters of string.
- **SUBSTR** returns a VARCHAR value representing a substring of a specified string.
- **SUBSTRB** returns a byte value representing a substring of a specified string.
- **SUBSTRING** given a value, a position, and an optional length, returns a value representing a substring of the specified string at the given position.
- **TRANSLATE** replaces individual characters in *string_to_replace* with other characters.
- **UPPER** returns a VARCHAR value containing the argument converted to uppercase letters.

UTF-8 String Functions

Starting in Release 5.1, the following string functions treat the the VARCHAR arguments as UTF-8 strings (when "USING OCTETS" is not specified) regardless of the locale setting.

- **LOWER** returns a VARCHAR value containing the argument converted to lowercase letters.
- **UPPER** returns a VARCHAR value containing the argument converted to uppercase letters.
- **INITCAP** capitalizes first letter of each alphanumeric word and puts the rest in lowercase.
- **INSTR** searches string for substring and returns an integer indicating the position of the character in string that is the first character of this occurrence.
- **SPLIT_PART** splits string on the delimiter and returns the location of the beginning of the given field (counting from one).
- **POSITION** returns an integer value representing the character location of a specified substring with a string (counting from one).
- **STRPOS** returns an integer value representing the character location of a specified substring within a string (counting from one).

Locale Specification

The locale is a parameter that defines the user's language, country, and any special variant preferences, such as collation. HP Vertica uses the locale to determine the behavior of various string functions as well for collation for various SQL commands that require ordering and comparison; for example, GROUP BY, ORDER BY, joins, the analytic ORDER BY clause, and so forth.

By default, the locale for the database is `en_US@collation=binary` (English US). You can establish a new default locale that is used for all sessions on the database, as well as override individual sessions with different locales. Additionally the locale can be set through [ODBC](#), [JDBC](#), and [ADO.net](#).

HP Vertica locale specifications follow a subset of the [Unicode LDML](#) standard as implemented by the ICU library.

Locales are specified using [long](#) or [short](#) forms.

Long Form

The long form uses full keyname pair/value names.

Syntax

```
[language][_script][_country][_variant][@keyword=type[;keyword=type]...]
```

Note: Only collation-related keywords are supported by HP Vertica 4.0.

Parameters

| | |
|-----------------------|---|
| <code>language</code> | A two- or three-letter lowercase code for a particular language. For example, Spanish is "es", English is "en" and French is "fr". The two-letter language code uses the ISO-639 standard. |
| <code>_script</code> | An optional four-letter script code that follows the language code. If specified, it should be a valid script code as listed on the Unicode ISO 15924 Registry. |
| <code>_country</code> | A specific language convention within a generic language for a specific country or region. For example, French is spoken in many countries, but the currencies are different in each country. To allow for these differences among specific geographical, political, or cultural regions, locales are specified by two-letter, uppercase codes. For example, "FR" represents France and "CA" represents Canada. The two letter country code uses the ISO-3166 standard. |

| | |
|-----------------------|---|
| <code>_variant</code> | <p>Differences may also appear in language conventions used within the same country. For example, the Euro currency is used in several European countries while the individual country's currency is still in circulation. To handle variations inside a language and country pair, add a third code, the variant code. The variant code is arbitrary and completely application-specific. ICU adds "<code>_EURO</code>" to its locale designations for locales that support the Euro currency. Variants can have any number of underscored key words. For example, "<code>EURO_WIN</code>" is a variant for the Euro currency on a Windows computer.</p> <p>Another use of the variant code is to designate the Collation (sorting order) of a locale. For instance, the "<code>es__TRADITIONAL</code>" locale uses the traditional sorting order which is different from the default modern sorting of Spanish.</p> |
|-----------------------|---|

| | |
|---------|---|
| keyword | <p>Use optional keywords and their values to specify collation order and currency instead of variants (as desired). If used, keywords must be unique, but their order is not significant. If a keyword is unknown or unsupported an error is reported. Keywords and values are not case sensitive.</p> <p>HP Vertica supports the following keywords:</p> |
|---------|---|

| Keyword | Short form | Description |
|-----------|------------|--|
| collation | K | <p>If present, the collation keyword modifies how the collation service searches through the locale data when instantiating a collator. Collation supports the following values:</p> <ul style="list-style-type: none"> • big5han — Pinyin ordering for Latin, big5 charset ordering for CJK characters (used in Chinese). • dict — For a dictionary-style ordering (such as in Sinhala). • direct — Hindi variant. • gb2312/gb2312han — Pinyin ordering for Latin, gb2312han charset ordering for CJK characters (used in Chinese). • phonebook — For a phonebook-style ordering (such as in German). • pinyin — Pinyin ordering for Latin and for CJK characters; that is, an ordering for CJK characters based on a character-by-character transliteration into a pinyin (used in Chinese). • reformed — Reformed collation (such as in Swedish). • standard — The default ordering for each language. For root it is [UCA] order; for each other locale it is the same as UCA (Unicode Collation Algorithm) ordering except for appropriate modifications to certain characters for that language. The following are additional choices for certain locales; they have effect only in certain locales. • stroke — Pinyin ordering for Latin, stroke order for CJK characters (used in Chinese) not supported. • traditional — For a traditional-style ordering (such as in Spanish). • unihan — Pinyin ordering for Latin, Unihan radical-stroke ordering for CJK characters (used in Chinese) not supported. • binary — the HP Vertica default, providing UTF-8 octet ordering, compatible with HP Vertica 3.5. |
| • | | |

Collation Keyword Parameters

The following parameters support the collation keyword:

| Parameter | Short form | Description |
|--------------|------------|--|
| colstrength | S | <p>Sets the default strength for comparison. This feature is locale dependant.</p> <p>Values can be any of the following:</p> <ul style="list-style-type: none">• 1 (primary) — Ignores case and accents. Only primary differences are used during comparison. For example, "a" versus "z".• 2 (secondary) — Ignores case. Only secondary and above differences are considered for comparison. For example, different accented forms of the same base letter ("a" versus "\u00E4").• 3 (tertiary) — Is the default. Only tertiary and above differences are considered for comparison. Tertiary comparisons are typically used to evaluate case differences. For example "Z" versus "z".• 4 (quaternary) — Used with Hiragana, for example.• 5 (identical) — All differences are considered significant during comparison. |
| colAlternate | A | <p>Sets alternate handling for variable weights, as described in UCA.</p> <p>Values can be any of the following:</p> <ul style="list-style-type: none">• Non-ignorable (short form N or D)• Shifted (short form S) |

| Parameter | Short form | Description |
|-----------------------|------------|---|
| colBackwards | F | <p>For Latin with accents, this parameter determines which accents are sorted. It sets the comparison for the second level to be backwards.</p> <p>Note: colBackwards is automatically set for French accents.</p> <p>If on (short form O), then the normal UCA algorithm is used. If off (short form X), then all strings that are in Fast C or D Normalization Form (FCD http://unicode.org/notes/tn5/#FCD-Test) sort correctly, but others do not necessarily sort correctly. So it should only be set off if the strings to be compared are in FCD.</p> |
| colNormalization | N | <p>If on (short form O), then the normal UCA algorithm is used. If off (short form X), all strings that are in [FCD] sort correctly, but others won't necessarily sort correctly. So it should only be set off if the strings to be compared are in FCD.</p> |
| colCaseLevel | E | <p>If set to on (short form O), a level consisting only of case characteristics is inserted in front of tertiary level. To ignore accents but take cases into account, set strength to primary and case level to on. If set to off (short form X), this level is omitted.</p> |
| colCaseFirst | C | <p>If set to upper (short form U), causes upper case to sort before lower case. If set to lower (short form L), lower case sorts before upper case. This is useful for locales that have already supported ordering but require different order of cases. It affects case and tertiary levels.</p> <p>If set to off (short form X), tertiary weights are not affected.</p> |
| colHiraganaQuaternary | H | <p>Controls special treatment of Hiragana code points on quaternary level. If turned on (short form O), Hiragana codepoints get lower values than all the other non-variable code points. The strength must be greater or equal than quaternary for this attribute to take effect. If turned off (short form X), Hiragana letters are treated normally.</p> |
| colNumeric | D | <p>If set to on, any sequence of Decimal Digits (General_Category = Nd in the [UCD]) is sorted at a primary level with its numeric value. For example, "A-21" < "A-123".</p> |
| variableTop | B | <p>If set to on, any sequence of Decimal Digits (General_Category = Nd in the [UCD]) is sorted at a primary level with its numeric value. For example, "A-21" < "A-123".</p> |

Notes

- Locale specification strings are case insensitive. The following are all equivalent: en_us, EN_US, and En_uS.
- You can substitute underscores with hyphens. For example: [-script]
- The ICU library works by adding options, such as S=1 separately after the long-form locale has been accepted. HP Vertica has extended its long-form processing to accept options as keywords, as suggested by the Unicode Consortium.
- Collations may default to root, the ICU default collation.
- Incorrect locale strings are accepted if the prefix can be resolved to a known locale version.

For example, the following works because the language can be resolved:

```
\locale en_XXINFO: Locale: 'en_XX'  
INFO: English (XX)  
INFO: Short form: 'LEN'
```

- The following does not work because the language cannot be resolved:

```
\locale xx_XXxx_XX: invalid locale identifier
```

- Invalid values of the collation keyword and its synonyms do not cause an error. For example, the following does not generate an error. It simply ignores the invalid value:

```
\locale en_GB@collation=xyzINFO: Locale: 'en_GB@collation=xyz'  
INFO: English (United Kingdom, collation=xyz)  
INFO: Short form: 'LEN'
```

- POSIX-type locales, such as en_US.UTF-8 work to some extent in that the encoding part "UTF-8" is ignored.
- HP Vertica 4.0 uses the icu4c-4_2_1 library to support basic locale/collation processing with some extensions as noted here. This does not yet meet the current standard for locale processing, <http://tools.ietf.org/html/rfc5646>.
- To learn more about collation options, consult http://www.unicode.org/reports/tr35/#Locale_Extension_Key_and_Type_Data.

Examples

The following specifies a locale for english as used in the United States:

```
en_US
```

The following specifies a locale for english as used in the United Kingdom:

```
en_GB
```

The following specifies a locale for German as used in Deutschland and uses phonebook-style collation.

```
\locale de_DE@collation=phonebookINFO:  Locale: 'de_DE@collation=phonebook'  
INFO:   German (Germany, collation=Phonebook Sort Order)  
INFO:   Deutsch (Deutschland, Sortierung=Telefonbuch-Sortierregeln)  
INFO:   Short form: 'KPHONEBOOK_LDE'
```

The following specifies a locale for German as used in Deutschland. It uses phonebook-style collation with a strength of secondary.

```
\locale de_DE@collation=phonebook;colStrength=secondaryINFO:  Locale: 'de_DE@collation=ph  
onebook'  
INFO:   German (Germany, collation=Phonebook Sort Order)  
INFO:   Deutsch (Deutschland, Sortierung=Telefonbuch-Sortierregeln)  
INFO:   Short form: 'KPHONEBOOK_LDE_S2'
```

Short Form

HP Vertica accepts locales in short form. You can use the short form to specify the locale and keyname pair/value names.

Determining the Short Form of a Locale

To determine the short form for a locale, type in the long form and view the last line of INFO, as follows:

```
\locale frINFO:  Locale: 'fr'  
INFO:   French  
INFO:   franÃ§ais  
INFO:   Short form: 'LFR'
```

Specifying a Short Form Locale

The following example specifies the *en* (English) locale:

```
\locale LENINFO:  Locale: 'en'  
INFO:   English  
INFO:   Short form: 'LEN'
```

The following example specifies a locale for German as used in Deutschland, and it uses phonebook-style collation.

```
\locale LDE_KPHONEBOOKINFO: Locale: 'de@collation=phonebook'  
INFO: German (collation=Phonebook Sort Order)  
INFO: Deutsch (Sortierung=Telefonbuch-Sortierregeln)  
INFO: Short form: 'KPHONEBOOK_LDE'
```

The following example specifies a locale for German as used in Deutschland. It uses phonebook-style collation with a strength of secondary (see "Collation Keyword Parameters" in [Long Form](#)).

```
\locale LDE_KPHONEBOOK_S2INFO: Locale: 'de@collation=phonebook'  
INFO: German (collation=Phonebook Sort Order)  
INFO: Deutsch (Sortierung=Telefonbuch-Sortierregeln)  
INFO: Short form: 'KPHONEBOOK_LDE_S2'
```

Supported Locales

The following are the supported locale strings for HP Vertica. Each locale can optionally have a list of key/value pairs (see [Long Form](#)).

| Locale Name | Language or Variant | Region |
|-------------|---------------------|----------------------|
| af | Afrikaans | |
| af_NA | Afrikaans | Namibian Afrikaans |
| af_ZA | Afrikaans | South Africa |
| am | Ethiopic | |
| am_ET | Ethiopic | Ethiopia |
| ar | Arabic | |
| ar_AE | Arabic | United Arab Emirates |
| ar_BH | Arabic | Bahrain |
| ar_DZ | Arabic | Algeria |
| ar_EG | Arabic | Egypt |
| ar_IQ | Arabic | Iraq |
| ar_JO | Arabic | Jordan |
| ar_KW | Arabic | Kuwait |
| ar_LB | Arabic | Lebanon |
| ar_LY | Arabic | Libya |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|---------------------|
| ar_MA | Arabic | Morocco |
| ar_OM | Arabic | Oman |
| ar_QA | Arabic | Qatar |
| ar_SA | Arabic | Saudi Arabia |
| ar_SD | Arabic | Sudan |
| ar_SY | Arabic | Syria |
| ar_TN | Arabic | Tunisia |
| ar_YE | Arabic | Yemen |
| as | Assamese | |
| as_IN | Assamese | India |
| az | Azerbaijani | |
| az_Cyrl | Azerbaijani | Cyrillic |
| az_Cyrl_AZ | Azerbaijani | Azerbaijan Cyrillic |
| az_Latn | Azerbaijani | Latin |
| az_Latn_AZ | Azerbaijani | Azerbaijan Latin |
| be | Belarusian | |
| be_BY | Belarusian | Belarus |
| bg | Bulgarian | |
| bg_BG | Bulgarian | Bulgaria |
| bn | Bengali | |
| bn_BD | Bengali | Bangladesh |
| bn_IN | Bengali | India |
| bo | Tibetan | |
| bo_CN | Tibetan | PR China |
| bo_IN | Tibetan | India |
| ca | Catalan | |
| ca_ES | Catalan | Spain |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|---------------------------|
| cs | Czech | |
| cs_CZ | Czech | Czech Republic |
| cy | Welsh | |
| cy_GB | Welsh | United Kingdom |
| da | Danish | |
| da_DK | Danish | Denmark |
| de | German | |
| de_AT | German | Austria |
| de_BE | German | Belgium |
| de_CH | German | Switzerland |
| de_DE | German | Germany |
| de_LI | German | Liechtenstein |
| de_LU | German | Luxembourg |
| el | Greek | |
| el_CY | Greek | Cyprus |
| el_GR | Greek | Greece |
| en | English | |
| en_AU | English | Australia |
| en_BE | English | Belgium |
| en_BW | English | Botswana |
| en_BZ | English | Belize |
| en_CA | English | Canada |
| en_GB | English | United Kingdom |
| en_HK | English | Hong Kong S.A.R. of China |
| en_IE | English | Ireland |
| en_IN | English | India |
| en_JM | English | Jamaica |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|--------------------------|
| en_MH | English | Marshall Islands |
| en_MT | English | Malta |
| en_NA | English | Namibia |
| en_NZ | English | New Zealand |
| en_PH | English | Philippines |
| en_PK | English | Pakistan |
| en_SG | English | Singapore |
| en_TT | English | Trinidad and Tobago |
| en_US_POSIX | English | United States Posix |
| en_VI | English | U.S. Virgin Islands |
| en_ZA | English | Zimbabwe or South Africa |
| en_ZW | English | Zimbabwe |
| eo | Esperanto | |
| es | Spanish | |
| es_AR | Spanish | Argentina |
| es_BO | Spanish | Bolivia |
| es_CL | Spanish | Chile |
| es_CO | Spanish | Columbia |
| es_CR | Spanish | Costa Rica |
| es_DO | Spanish | Dominican Republic |
| es_EC | Spanish | Ecuador |
| es_ES | Spanish | Spain |
| es_GT | Spanish | Guatemala |
| es_HN | Spanish | Honduras |
| es_MX | Spanish | Mexico |
| es_NI | Spanish | Nicaragua |
| es_PA | Spanish | Panama |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|---------------|
| es_PE | Spanish | Peru |
| es_PR | Spanish | Puerto Rico |
| es_PY | Spanish | Paraguay |
| es_SV | Spanish | El Salvador |
| es_US | Spanish | United States |
| es_UY | Spanish | Uruguay |
| es_VE | Spanish | Venezuela |
| et | Estonian | |
| et_EE | Estonian | Estonia |
| eu | Basque | Spain |
| eu_ES | Basque | Spain |
| fa | Persian | |
| fa_AF | Persian | Afghanistan |
| fa_IR | Persian | Iran |
| fi | Finnish | |
| fi_FI | Finnish | Finland |
| fo | Faroese | |
| fo_FO | Faroese | Faroe Islands |
| fr | French | |
| fr_BE | French | Belgium |
| fr_CA | French | Canada |
| fr_CH | French | Switzerland |
| fr_FR | French | France |
| fr_LU | French | Luxembourg |
| fr_MC | French | Monaco |
| fr_SN | French | Senegal |
| ga | Gaelic | |

| Locale Name | Language or Variant | Region |
|----------------|---------------------|-----------------|
| ga_IE | Gaelic | Ireland |
| gl | Gallegan | |
| gl_ES | Gallegan | Spain |
| gsw | German | |
| gsw_CH | German | Switzerland |
| gu | Gujurati | |
| gu_IN | Gujurati | India |
| gv | Manx | |
| gv_GB | Manx | United Kingdom |
| ha | Hausa | |
| ha_Latn | Hausa | Latin |
| ha_Latn_GH | Hausa | Ghana (Latin) |
| ha_Latn_NE | Hausa | Niger (Latin) |
| ha_Latn_NG | Hausa | Nigeria (Latin) |
| haw | Hawaiian | Hawaiian |
| haw_US | Hawaiian | United States |
| he | Hebrew | |
| he_IL | Hebrew | Israel |
| hi | Hindi | |
| hi_IN | Hindi | India |
| hr | Croatian | |
| hr_HR | Croatian | Croatia |
| hu | Hungarian | |
| hu_HU | Hungarian | Hungary |
| hy | Armenian | |
| hy_AM | Armenian | Armenia |
| hy_AM_REVISSED | Armenian | Revised Armenia |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|-----------------------|
| id | Indonesian | |
| id_ID | Indonesian | Indonesia |
| ii | Sichuan | |
| ii_CN | Sichuan | Yi |
| is | Icelandic | |
| is_IS | Icelandic | Iceland |
| it | Italian | |
| it_CH | Italian | Switzerland |
| it_IT | Italian | Italy |
| ja | Japanese | |
| ja_JP | Japanese | Japan |
| ka | Georgian | |
| ka_GE | Georgian | Georgia |
| kk | Kazakh | |
| kk_Cyrl | Kazakh | Cyrillic |
| kk_Cyrl_KZ | Kazakh | Kazakhstan (Cyrillic) |
| kl | Kalaallisut | |
| kl_GL | Kalaallisut | Greenland |
| km | Khmer | |
| km_KH | Khmer | Cambodia |
| kn | Kannada | |
| kn-IN | Kannada | India |
| ko | Korean | |
| ko_KR | Korean | Korea |
| kok | Konkani | |
| kok_IN | Konkani | India |
| kw | Cornish | |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|----------------|
| kw_GB | Cornish | United Kingdom |
| lt | Lithuanian | |
| lt_LT | Lithuanian | Lithuania |
| lv | Latvian | |
| lv_LV | Latvian | Latvia |
| mk | Macedonian | |
| mk_MK | Macedonian | Macedonia |
| ml | Malayalam | |
| ml_IN | Malayalam | India |
| mr | Marathi | |
| mr_IN | Marathi | India |
| ms | Malay | |
| ms_BN | Malay | Brunei |
| ms_MY | Malay | Malaysia |
| mt | Maltese | |
| mt_MT | Maltese | Malta |
| nb | Norwegian Bokml | |
| nb_NO | Norwegian Bokml | Norway |
| ne | Nepali | |
| ne_IN | Nepali | India |
| ne_NP | Nepali | Nepal |
| nl | Dutch | |
| nl_BE | Dutch | Belgium |
| nl_NL | Dutch | Netherlands |
| nn | Norwegian nynorsk | |
| nn_NO | Norwegian nynorsk | Norway |
| om | Oromo | |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|-------------------|
| om_ET | Oromo | Ethiopia |
| om_KE | Oromo | Kenya |
| or | Oriya | |
| or_IN | Oriya | India |
| pa | Punjabi | |
| pa_Arab | Punjabi | Arabic |
| pa_Arab_PK | Punjabi | Pakistan (Arabic) |
| pa_Guru | Punjabi | Gurmukhi |
| pa_Guru_IN | Punjabi | India (Gurmukhi) |
| pl | Polish | |
| pl_PL | Polish | Poland |
| ps | Pashto | |
| ps_AF | Pashto | Afghanistan |
| pt | Portuguese | |
| pt_BR | Portuguese | Brazil |
| pt_PT | Portuguese | Portugal |
| ro | Romanian | |
| ro_MD | Romanian | Moldavia |
| ro_RO | Romanian | Romania |
| ru | Russian | |
| ru_RU | Russian | Russia |
| ru_UA | Russian | Ukraine |
| si | Sinhala | |
| si_LK | Sinhala | Sri Lanka |
| sk | Slovak | |
| sk_SK | Slovak | Slovakia |
| sl | Slovenian | |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|-----------------------------------|
| sl_SL | Slovenian | Slovenia |
| so | Somali | |
| so_DJ | Somali | Djibouti |
| so_ET | Somali | Ethiopia |
| so_KE | Somali | Kenya |
| so_SO | Somali | Somalia |
| sq | Albanian | |
| sq_AL | Albanian | Albania |
| sr | Serbian | |
| sr_Cyrl | Serbian | Cyrillic |
| sr_Cyrl_BA | Serbian | Bosnia and Herzegovina (Cyrillic) |
| sr_Cyrl_ME | Serbian | Montenegro (Cyrillic) |
| sr_Cyrl_RS | Serbian | Serbia (Cyrillic) |
| sr_Latn | Serbian | Latin |
| sr_Latn_BA | Serbian | Bosnia and Herzegovina (Latin) |
| sr_Latn_ME | Serbian | Montenegro (Latin) |
| sr_Latn_RS | Serbian | Serbia (Latin) |
| sv | Swedish | |
| sv_FI | Swedish | Finland |
| sv_SE | Swedish | Sweden |
| sw | Swahili | |
| sw_KE | Swahili | Kenya |
| sw_TZ | Swahili | Tanzania |
| ta | Tamil | |
| ta_IN | Tamil | India |
| te | Telugu | |
| te_IN | Telugu | India |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|--------------------------------------|
| th | Thai | |
| th_TH | Thai | Thailand |
| ti | Tigrinya | |
| ti_ER | Tigrinya | Eritrea |
| ti_ET | Tigrinya | Ethiopia |
| tr | Turkish | |
| tr_TR | Turkish | Turkey |
| uk | Ukrainian | |
| uk_UA | Ukrainian | Ukraine |
| ur | Urdu | |
| ur_IN | Urdu | India |
| ur_PK | Urdu | Pakistan |
| uz | Uzbek | |
| uz_Arab | Uzbek | Arabic |
| uz_Arab_AF | Uzbek | Afghanistan (Arabic) |
| uz_Cryl | Uzbek | Cyrillic |
| uz_Cryl_UZ | Uzbek | Uzbekistan (Cyrillic) |
| uz_Latin | Uzbek | Latin |
| us_Latin_UZ | | Uzbekistan (Latin) |
| vi | Vietnamese | |
| vi_VN | Vietnamese | Vietnam |
| zh | Chinese | |
| zh_Hans | Chinese | Simplified Han |
| zh_Hans_CN | Chinese | China (Simplified Han) |
| zh_Hans_HK | Chinese | Hong Kong SAR China (Simplified Han) |
| zh_Hans_MO | Chinese | Macao SAR China (Simplified Han) |
| zh_Hans_SG | Chinese | Singapore (Simplified Han) |

| Locale Name | Language or Variant | Region |
|-------------|---------------------|---------------------------------------|
| zh_Hant | Chinese | Traditional Han |
| zh_Hant_HK | Chinese | Hong Kong SAR China (Traditional Han) |
| zh_Hant_MO | Chinese | Macao SAR China (Traditional Han) |
| zh_Hant_TW | Chinese | Taiwan (Traditional Han) |
| zu | Zulu | |
| zu_ZA | Zulu | South Africa |

Locale Restrictions and Workarounds

The following list contains the known restrictions for locales for international data sets.

Session related:

- The locale setting is session scoped and applies to queries only (no DML/DDDL) run in that session. You cannot specify a locale for an individual query.
- The default locale for new sessions can be set using a configuration parameter

Query related:

The following restrictions apply when queries are run with locale other than the default `en_US@collation=binary`:

- Multicolumn NOT IN subqueries are not supported when one or more of the left-side NOT IN columns is of CHAR or VARCHAR data type. For example:

```
=> CREATE TABLE test (x VARCHAR(10), y INT);
=> SELECT ... FROM test WHERE (x,y) NOT IN (SELECT ...);
      ERROR: Multi-expression NOT IN subquery is not supported because a left hand expression could be NULL
```

Note: An error is reported even if columns `test.x` and `test.y` have a "NOT NULL" constraint.

- Correlated HAVING clause subqueries are not supported if the outer query contains a GROUP BY on a CHAR or a VARCHAR column. In the following example, the GROUP BY `x` in the outer query causes the error:

```
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
```

```
=> SELECT COUNT(*) FROM test t GROUP BY x HAVING x
      IN (SELECT x FROM test WHERE t.x||'a' = test.x||'a' );
ERROR: subquery uses ungrouped column "t.x" from outer query
```

- Subqueries that use analytic functions in the HAVING clause are not supported. For example:

```
=> DROP TABLE test CASCADE;
=> CREATE TABLE test (x VARCHAR(10));
=> SELECT MAX(x)OVER(PARTITION BY 1 ORDER BY 1)
      FROM test GROUP BY x HAVING x IN (SELECT MAX(x) FROM test);
ERROR: Analytics query with having clause expression that involves aggregates and subquery is not supported
```

DML/DDDL related:

- SQL identifiers (such as table names, column names, and so on) can use UTF-8 Unicode characters. For example, the following CREATE TABLE statement uses the ß (German eszett) in the table name:

```
=> CREATE TABLE straÙe(x int, y int);
CREATE TABLE
```

- Projection sort orders are made according to the default en_US@collation=binary collation. Thus, regardless of the session setting, issuing the following command creates a projection sorted by col1 according to the binary collation:

```
=> CREATE PROJECTION p1 AS SELECT * FROM table1 ORDER BY col1;
```

Note that in such cases, *straÙe* and *strasse* would not be near each other on disk.

Sorting by binary collation also means that [sort optimizations](#) do not work in locales other than binary. HP Vertica returns the following warning if you create tables or projections in a non-binary locale:

```
WARNING: Projections are always created and persisted in the default HP Vertica locale. The current locale is de_DE
```

- When creating pre-join projections, the projection definition query does not respect the locale or collation setting. This means that when you insert data into the fact table of a pre-join projection, referential integrity checks are not locale or collation aware.

For example:

```
\locale LDE_S1 -- German
=> CREATE TABLE dim (col1 varchar(20) primary key);
=> CREATE TABLE fact (col1 varchar(20) references dim(col1));
=> CREATE PROJECTION pj AS SELECT * FROM fact JOIN dim
    ON fact.col1 = dim.col1 UNSEGMENTED ALL NODES;
=> INSERT INTO dim VALUES('ß');
=> COMMIT;
```

The following INSERT statement fails with a "nonexistent FK" error even though 'ß' is in the dim table, and in the German locale 'SS' and 'ß' refer to the same character.

```
=> INSERT INTO fact VALUES('SS');
    ERROR: Nonexistent foreign key value detected in FK-PK join (fact x dim)
    using subquery and dim_node0001; value SS
=> => ROLLBACK;
=> DROP TABLE dim, fact CASCADE;
```

- When the locale is non-binary, the collation function is used to transform the input to a binary string which sorts in the proper order.

This transformation increases the number of bytes required for the input according to this formula:

```
result_column_width = input_octet_width * CollationExpansion + 4
```

CollationExpansion defaults to 5.

- CHAR fields are displayed as fixed length, including any trailing spaces. When CHAR fields are processed internally, they are first stripped of trailing spaces. For VARCHAR fields, trailing spaces are usually treated as significant characters; however, trailing spaces are ignored when sorting or comparing either type of character string field using a non-BINARY locale.

Appendix: Binary File Formats

Creating Native Binary Format Files

Using COPY to load data with the NATIVE parser requires that the input data files adhere to the format described below. All NATIVE files must contain:

1. A file signature.
2. A set of column size definitions.
3. The rows of data.

Note: You cannot mix Binary and ASCII source files in the same COPY statement.

File Signature

The first part of a NATIVE binary file consists of a file signature. The contents of the signature are fixed, and listed in the following table.

| | | | | | | | | | | | |
|----------------------|----|----|----|----|----|----|-------|---------|-------|-------|---------|
| Byte Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Hex Value | 4E | 41 | 54 | 49 | 56 | 45 | 0A | FF | 0D | 0A | 00 |
| Text Literals | N | A | T | I | V | E | E'\n' | E'\317' | E'\r' | E'\n' | E'\000' |

The purpose of the required signature is to ensure that the file has neither been corrupted by a non-8-bit file transfer, nor stripped of carriage returns, linefeeds, or null values. If the signature is intact, HP Vertica determines that the file has not been corrupted.

Column Definitions

Following the file signature, the file must define the widths of each column in the file as follows.

| Byte Offset | Length (bytes) | Description | Comments |
|--------------------|-----------------------|--------------------|---|
| 11 | 4 | Header area length | 32-bit integer in little-endian format that contains the length in bytes of remaining in the header, not including itself. This is the number of bytes from the end of this value to the start of the row data. |

| Byte Offset | Length (bytes) | Description | Comments |
|-------------|--|---------------------|---|
| 15 | 2 | NATIVE file version | 16-bit integer in little-endian format containing the version number of the NATIVE file format. The only valid value is currently 1. Future changes to the format could be assigned different version numbers to maintain backward compatibility. |
| 17 | 1 | Filler | Always 0. |
| 18 | 2 | Number of columns | 16-bit integer in little-endian format that contains the number of columns in each row in the file. |
| 20+ | 4 bytes for each column of data in the table | Column widths | Array of 32-bit integers in little-endian format that define the width of each column in the row. Variable-width columns have a value of -1 (0xFF 0xFF 0xFF 0xFF). |

Note: All integers in NATIVE files are in **little-endian format** (least significant byte first).

The width of each column is determined by the data type it contains. The following table explains the column width needed for each data type, along with the data encoding.

| Data Type | Length (bytes) | Column Content |
|-----------|----------------|--|
| INTEGER | 1, 2, 4, 8 | 8-, 16-, 32-, and 64-bit integers are supported. All multi-byte values are stored in little-endian format. Note: All values for a column must be the width you specify here. If you set the length of an INTEGER column to be 4 bytes, then all of the values you supply for that column must be 32-bit integers. |
| BOOLEAN | 1 | 0 for false, 1 for true. |
| FLOAT | 8 | Encoded in IEEE-754 format. |
| CHAR | User-specified | <ul style="list-style-type: none"> Strings shorter than the specified length must be right-padded with spaces (E'\040'). Strings are not null-terminated. Character encoding is UTF-8. UTF-8 strings can contain multi-byte characters. Therefore, number of characters in the string may not equal the number of bytes. |

| Data Type | Length (bytes) | Column Content |
|-------------|--------------------------------|--|
| VARCHAR | 4-byte integer (length) + data | <p>The column width for a VARCHAR column is always -1 to signal that it contains variable-length data.</p> <ul style="list-style-type: none"> • Each VARCHAR column value starts with a 32-bit integer that contains the number of bytes in the string. • The string must not be null-terminated. • Character encoding must be UTF-8. • Remember that UTF-8 strings can contain multi-byte characters. Therefore, number of characters in the string may not equal the number of bytes. |
| DATE | 8 | 64-bit integer in little-endian format containing the Julian day since Jan 01 2000 (J2451545) |
| TIME | 8 | 64-bit integer in little-endian format containing the number of microseconds since midnight in the UTC time zone. |
| TIMETZ | 8 | <p>64-bit value where</p> <ul style="list-style-type: none"> • Upper 40 bits contain the number of microseconds since midnight. • Lower 24 bits contain time zone as the UTC offset in microseconds calculated as follows: Time zone is logically from -24hrs to +24hrs from UTC. Instead it is represented here as a number between 0hrs to 48hrs. Therefore, 24hrs should be added to the actual time zone to calculate it. <p>Each portion is stored in little-endian format (5 bytes followed by 3 bytes).</p> |
| TIMESTAMP | 8 | 64-bit integer in little-endian format containing the number of microseconds since Julian day: Jan 01 2000 00:00:00. |
| TIMESTAMPTZ | 8 | A 64-bit integer in little-endian format containing the number of microseconds since Julian day: Jan 01 2000 00:00:00 in the UTC timezone. |
| INTERVAL | 8 | 64-bit integer in little-endian format containing the number of microseconds in the interval. |

| Data Type | Length (bytes) | Column Content |
|-----------|---|---|
| BINARY | User-specified | Similar to CHAR. The length should be specified in the file header in the <i>Field Lengths</i> entry for the field. The field in the record must contain <i>length</i> number of bytes. If the value is smaller than the specified length, the remainder should be filled with nulls (E'\000'). |
| VARBINARY | 4-byte integer + data | Stored just like VARCHAR but data is interpreted as bytes rather than UTF-8 characters. |
| NUMERIC | (precision, scale) (precision ÷ 19 + 1) × 8 rounded up | <p>A constant-length data type. Length is determined by the precision, assuming that a 64-bit unsigned integer can store roughly 19 decimal digits. The data consists of a sequence of 64-bit integers, each stored in little-endian format, with the most significant integer first. Data in the integers is stored in base 2⁶⁴. 2's complement is used for negative numbers.</p> <p>If there is a scale, then the numeric is stored as numeric × 10^{scale}; that is, all real numbers are stored as integers, ignoring the decimal point. It is required that the scale matches that of the target column in the database table. Another option is to use FILLER columns to coerce the numeric to the scale of the target column.</p> |

Row Data

Following the file header is a sequence of records that contain the data for each row of data. Each record starts with a header:

| Length (bytes) | Description | Comments |
|----------------|-------------|--|
| 4 | Row length | <p>A 32-bit integer in little-endian format containing the length of the row's data in bytes. It includes the size of data only, not the header.</p> <p>Note: The number of bytes in each row can vary not only because of variable-length data, but also because columns containing NULL values do not have any data in the row. If column 3 has a NULL value, then column 4's data immediately follows the end of column 2's data. See the next</p> |

| Length (bytes) | Description | Comments |
|---|----------------------|--|
| Number of columns ÷ 8 rounded up (CEILING (NumFields / (sizeof (uint8) * 8));) | Null value bit field | A series of bytes whose bits indicate whether a column contains a NULL. The most significant bit of the first byte indicates whether the first column in this row contains a NULL, the next most significant bit indicates whether the next column contains a NULL, and so on. If a bit is 1 (true) then the column contains a NULL, and there is no value for the column in the data for the row. |

Following the record header is the column values for the row. There is no separator characters for these values. Their location in the row of data is calculated based on where the previous column's data ended. Most data types have a fixed width, so their location is easy to determine. Variable-width values (such as VARCHAR and VARBINARY) start with a count of the number of bytes the value contains.

See the table in the previous section for details on how each data type's value is stored in the row's data.

Example

The example below demonstrates creating a table and loading a NATIVE file that contains a single row of data. The table contains all possible data types.

```
=> CREATE TABLE allTypes (INTCOL INTEGER,                                FLOATCOL FLOAT,
                           CHARCOL CHAR(10),
                           VARCHARCOL VARCHAR,
                           BOOLCOL BOOLEAN,
                           DATECOL DATE,
                           TIMESTAMPCOL TIMESTAMP,
                           TIMESTAMPTZCOL TIMESTAMPTZ,
                           TIMECOL TIME,
                           TIMETZCOL TIMETZ,
                           VARBINCOL VARBINARY,
                           BINCOL BINARY,
                           NUMCOL NUMERIC(38,0),
                           INTERVALCOL INTERVAL
                           );
=> COPY allTypes FROM '/home/dbadmin/allTypes.bin' NATIVE DIRECT;
=> \pset expanded
Expanded display is on.
=> SELECT * from allTypes;
-[ RECORD 1 ]--+-+-----
INTCOL          | 1
FLOATCOL        | -1.11
CHARCOL          | one
VARCHARCOL      | ONE
BOOLCOL         | t
DATECOL         | 1999-01-08
TIMESTAMPCOL    | 1999-02-23 03:11:52.35
TIMESTAMPTZCOL | 1999-01-08 07:04:37-05
TIMECOL         | 07:09:23
```

```
TIMETZCOL | 15:12:34-04
VARBINCOL | \253\315
BINCOL    | \253
NUMCOL    | 1234532
INTERVALCOL | 03:03:03
```

The content of the allTypes.bin file appears below as a raw hex dump:

```
4E 41 54 49 56 45 0A FF 0D 0A 00 3D 00 00 00 01 00 00 0E 0008 00 00 00 08 00 00 00 0A 00
00 00 FF FF FF FF 01 00 00 00
08 00 00 00 08 00 00 00 08 00 00 00 08 00 00 00 08 00 00 00
FF FF FF FF 03 00 00 00 18 00 00 00 08 00 00 00 73 00 00 00
00 00 01 00 00 00 00 00 00 00 C3 F5 28 5C 8F C2 F1 BF 6F 6E
65 20 20 20 20 20 20 03 00 00 00 4F 4E 45 01 9A FE FF FF
FF FF FF FF 30 85 B3 4F 7E E7 FF FF 40 1F 3E 64 E8 E3 FF FF
C0 2E 98 FF 05 00 00 00 D0 97 01 80 F0 79 F0 10 02 00 00 00
AB CD AB CD 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 64 D6 12 00 00 00 00 00 C0 47 A3 8E 02 00 00 00
```

The following table breaks this file down into each of its components, and describes the values it contains.

| Hex Values | Description | Value |
|--|------------------------------|----------------------------|
| 4E 41 54 49 56 45 0A FF 0D 0A 00 | Signature | NATIVE\n\317\r\n\000 |
| 3D 00 00 00 | Header area length | 61 bytes |
| 01 00 | Native file format version | Version 1 |
| 00 | Filler value | 0 |
| 0E 00 | Number of columns | 14 columns |
| 08 00 00 00 | Width of column 1 (INTEGER) | 8 bytes |
| 08 00 00 00 | Width of column 2 (FLOAT) | 8 bytes |
| 0A 00 00 00 | Width of column 3 (CHAR(10)) | 10 bytes |
| FF FF FF FF | Width of column 4 (VARCHAR) | -1 (variable width column) |

| Hex Values | Description | Value |
|-------------|--|---|
| 01 00 00 00 | Width of column 5 (BOOLEAN) | 1 bytes |
| 08 00 00 00 | Width of column 6 (DATE) | 8 bytes |
| 08 00 00 00 | Width of column 7 (TIMESTAMP) | 8 bytes |
| 08 00 00 00 | Width of column 8 (TIMESTAMPTZ) | 8 bytes |
| 08 00 00 00 | Width of column 9 (TIME) | 8 bytes |
| 08 00 00 00 | Width of column 10 (TIMETZ) | 8 bytes |
| FF FF FF FF | Width of column 11 (VARBINARY) | -1 (variable width column) |
| 03 00 00 00 | Width of column 12 (BINARY) | 3 bytes |
| 18 00 00 00 | Width of column 13 (NUMERIC) | 24 bytes. The size is calculated by dividing 38 (the precision specified for the numeric column) by 19 (the number of digits each 64-bit chunk can represent) and adding 1. $38 \div 19 + 1 = 3$. then multiply by eight to get the number of bytes needed. $3 \times 8 = 24$ bytes. |
| 08 00 00 00 | Width of column 14 (INTERVAL). last portion of the header section. | 8 bytes |
| 73 00 00 00 | Number of bytes of data for the first row. this is the start of the first row of data. | 115 bytes |
| 00 00 | Bit field for the null values contained in the first row of data | The row contains no null values. |

| Hex Values | Description | Value |
|----------------------------------|--|--|
| 01 00 00 00 00 00 00 00 | Value for 64-bit INTEGER column | 1 |
| C3 F5 28 5C 8F C2 F1 BF | Value for the FLOAT column | -1.11 |
| 6F 6E 65 20 20 20 20 20 20 20 | Value for the CHAR(10) column | "one " (padded With 7 spaces to fill the full 10 characters for the column) |
| 03 00 00 00 | The number of bytes in the following VARCHAR value. | 3 bytes |
| 4F 4E 45 | The value for the VARCHAR column | "one" |
| 01 | The value for the BOOLEAN column | True |
| 9A FE FF FF FF FF FF FF | The value for the DATE column | 1999-01-08 |
| 30 85 B3 4F 7E E7 FF FF | The value for the TIMESTAMP column | 1999-02-23 03:11:52.35 |
| 40 1F 3E 64 E8 E3 FF FF | The value for the TIMESTAMPTZ column | 1999-01-08 07:04:37-05 |
| C0 2E 98 FF 05 00 00 00 | The value for the TIME column | 07:09:23 |
| D0 97 01 80 F0 79 F0 10 | The value for the TIMETZ column | 15:12:34-05 |
| 02 00 00 00 | The number of bytes in the following VARBINARY value | 2 bytes |

| Hex Values | Description | Value |
|--|------------------------------------|--|
| AB CD | The value for the VARBINARY column | Binary data (\253\315 as octal values) |
| AB CD 00 | The value for the BINARY column | Binary data (\253\315\000 as octal values) |
| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 64 D6 12 00 00 00 00 00 | The value for the NUMERIC column | 1234532 |
| C0 47 A3 8E 02 00 00 00 | The value for the INTERVAL column | 03:03:03 |

See Also

-

We appreciate your feedback!

If you have comments about this document, you can [contact the documentation team](#) by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Administrator's Guide (Vertica Analytics Platform 7.0.x)

Just add your feedback to the email and click send.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to vertica-docfeedback@hp.com.