Archiving and Viewing archived data

How to use RDBMS tools to archive and view archived data

Introduction

This document describes how to archive and directly view archived data using RDBMS methods rather than the unload method using the purge/archive utility. The benefit of using the RDBMS to archive and view archived data is more flexibility when accessing the data and faster access, while still keeping the main data tables small for fast I/O access.

This document concentrates on how to archive Incident Management data, but the steps can be applied to any Service Manager module.

Prerequisites

You must have extensive knowledge of Service Manager and the underlying RDBMS.

Archiving Service Manager Data in the RDBMS

This chapter describes how to archive data directly into an Oracle database. The solution can be adapted to other databases like Microsoft SQL Server with the assistance of a Database Administrator.

The following steps are necessary to archive data:

1. Identify tables to be included in the archiving procedure
2. Build and configure the archiving table space
3. Implement the archiving procedure

There are two very different ways to archive using the RDBMS:

1. Archive into a different Service Manager instance
2. Archive into the same Service Manager instance

Archiving into a different Service Manager instance follows the same RDBMS procedures of selecting and moving the data as described in this document, with the exception that the same table names will be used in the archive as in the productive system, since they are both independent copies of Service Manager. The archive system will start out as a copy of the productive Service Manager system and will continually be filled with additional data. Depending on the archiving requirements, data older than 5 or 10 years can be permanently deleted from the archive system. User access to the archive system should be limited to power users, if possible. Read-only requirements should be followed by setting the IO condition of all archived tables to false. The steps described in section “Viewing archived data from the RDBMS” are not necessary using this archiving option.

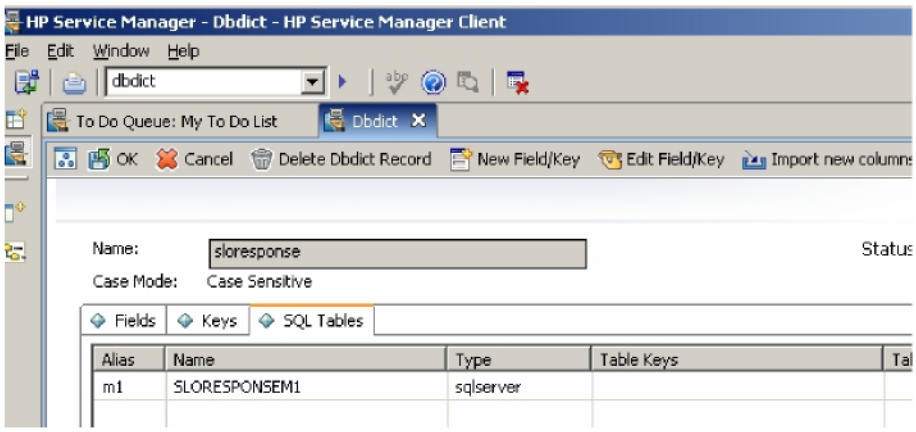
Archiving into the same Service Manager instance also involves tailoring and configuration of the Service Manager instance, which is discussed in this paper.

Identify tables to be included in the archiving procedure

The first step is to identify which tables need to be archived per module. The RDBMS table names are stored in the dbdict record. Which dbdicts need to be involved in the archiving procedure, depends on the module. For a sample list, please refer to Appendix A. The list may be different in a customer implementation and has to be adjusted as needed.

Follow these steps to determine the RDBMS table names to use in the archive procedure:

- Go into the dbdict utility via Tailoring > Database Dictionary or by entering “dbdict” in the Service Manager command line
- Search for the file you want to archive, for example sloresponse for SLO records
- Click on the SQL tables tab to see the database tables linked to the dbdict



Build and configure archiving table space

There are three options to create a database space for archived data that accomplish the archiving requirements of improving database maintenance and performance by reducing data load:

1. two separate databases on different hardware
2. two separate databases on the same hardware
3. two separate table spaces on the same database

The following table describes the advantages and disadvantages for each option.

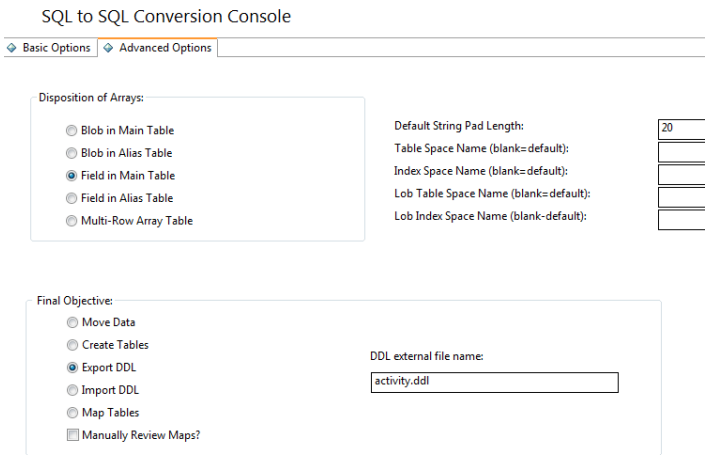
Option	**Advantages**	**Disadvantages**
two separate databases on different hardware	relief of productive tables independent shutdown and restart	cost for second hardware second hardware to maintain own cluster for high availability
two separate databases on same hardware	relief of productive tables no new cluster for high availability	no independent shutdown and restart
two separate table spaces on the same database	relief of productive tables no new cluster for high availability	no independent shutdown and restart

In this paper, we will discuss option 3 – two table spaces in the same database.

Setup of the archiving table space

The database administrator will create a new table space in the Oracle database together with a new user that uses that table space as their default table space.

Next the archive tables for each of the source tables (see Appendix A) need to be created. These tables need to have a structure identical to the tables in the source table space. To retrieve the DDL to create the tables identical to their source tables, log in to the Service Manager Windows client and go to Tailoring > SQL Utilities > Move Files from SQL to SQL and export the DDL as shown in the below picture, or ask your database administrator to generate the DDL from the source system:



Below a sample DDL to create the table ARCHSLORESPONSEM1:

```
1 CREATE TABLE ARCHSLORESPONSEM1
2  (
3  FOREIGN_FILENAME              VARCHAR2(70 BYTE) ,
4  FOREIGN_KEY                   VARCHAR2(70 BYTE) ,
5  SLO_ID                        FLOAT(126),
6  SLO_NAME                      VARCHAR2(110 BYTE),
7 INITIAL_STATE                  VARCHAR2(70 BYTE) ,
8 FINAL_STATE                    VARCHAR2(70 BYTE) ,
9 EXPIRATION_TIME                DATE,
10 RUNNING                       CHAR(1 BYTE),
11 SUSPENDED                     CHAR(1 BYTE),
12 BREACHED                      CHAR(1 BYTE),
13 CURRENT_STATUS                FLOAT(126),
14 ACTIVE                        CHAR(1 BYTE) ,
15 START_TIME                    DATE,
16 END_TIME                      DATE,
17 AGREEMENTID                   FLOAT(126),
18 TOTAL_TIME                    DATE,
19 SYSMODTIME                    DATE,
20 SYSMODUSER                    VARCHAR2(60 BYTE) ,
21 SYSMODCOUNT                   FLOAT(126)
22)
```

Implement archiving procedure

As soon as the productive table space and the archived table space are configured a procedure can be implemented to copy the data from the productive table space to the archived table space. Appendix B has a sample Oracle procedure that includes writing log messages.

To archive Incident Management records, follow these steps:

1. Decide on the condition to select the incidents to be archived, for example the incident is closed and the close date is older than 1 year.
2. Using a select statement against the PROBSUMMARYM1 table, use this condition to retrieve the list of Incident numbers to archive
3. Based on this list, retrieve the records from all tables listed for Incident Management in Appendix A, and copy them to the archive table
4. Once all data was successfully copied, delete them from the source table.

It is recommended to process each record sequentially with error checking to avoid data inconsistency. As soon as one of the select and move functions fails the transaction has to be rolled

back. We strongly recommend directing the output of the procedure used to move the data to a log file. This log file needs to be analyzed by the administrator after each archiving run.

NOTE: A common error is missing access rights. Therefore the correct privileges have to be set to allow insert statements to the destination tables. Use a statement like the following:

```
1 grant insert on [DB ARCHIVING USER].[TABLENAME] to [DB PRODUCTIVE USER]
```

Option 1 – using SQL statements manually

This option is independent of the RDBMS, but needs to be run manually for error control. The Database Administrator may want to translate these statements into a procedure that can be scheduled with added error handling.

NOTE: Ensure to test that this option will move all LOBs correctly and check for errors after each commit. For better performance, it is important to not run this option against too many rows, so it may be necessary to start with incidents older than 5 years, then 4 years, then 3 years, then 2 years, then 1 year and run it in separate statements when archiving for the first time.

```
INSERT PROBSUMMARYARCHIVEM1 SELECT * FROM PROBSUMMARYM1 WHERE (CLOSE_TIME
< '01/01/2010');
COMMIT;

INSERT PROBSUMMARYARCHIVEA1
   (SELECT * FROM PROBSUMMARYA1 WHERE (NUMBER ISIN ( SELECT NUMBER FROM
PROBSUMMARYM1 WHERE (CLOSE_TIME < '01/01/2010')))

INSERT PROBSUMMARYARCHIVEA2
   (SELECT * FROM PROBSUMMARYA2 WHERE (NUMBER ISIN ( SELECT NUMBER FROM
PROBSUMMARYM1 WHERE (CLOSE_TIME < '01/01/2010')))


INSERT ACTIVITYARCHIVEM1
(
 SELECT * FROM ACTIVITYM1
  WHERE
  ( NUMBER ISIN
   (
     SELECT NUMBER FROM PROBSUMMARYM1 WHERE ( CLOSE_TIME < '01/01/2010' )
   )
  )
)

DELETE FROM ACTIVITYM1 WHERE (NUMBER ISIN ( SELECT NUMBER FROM
PROBSUMMARYM1 WHERE (CLOSE_TIME < '01/01/2010'  ) ) )
COMMIT;

INSERT SLORESPONSEARCHIVEM1
(
 SELECT * FROM SLORESPONSEM1
  WHERE
  ( FOREIGN_KEY ISIN
   (
     SELECT NUMBER FROM PROBSUMMARYM1 WHERE (CLOSE_TIME < '01/01/2010' )
   )
  )
)

DELETE FROM SLORESPONSEM1 WHERE (FOREIGN_KEY ISIN ( SELECT NUMBER FROM
PROBSUMMARYM1 WHERE (CLOSE_TIME < '01/01/2010'  ) ) )
```

```
COMMIT;

INSERT SYSATTACHARCHIVEM1
(
 SELECT * FROM SYSATTACHMENTSM1
  WHERE
  ( TOPIC ISIN
   (
     SELECT NUMBER FROM PROBSUMMARYM1 WHERE (CLOSE_TIME < '01/01/2010' )
   )
  )
)

DELETE FROM SYSATTACHMENTSM1 WHERE ( TOPIC ISIN ( SELECT number FROM
PROBSUMMARYM1 WHERE (CLOSE_TIME < '01/01/2010'  ) ) )
COMMIT;


DELETE FROM PROBSUMMARYM1 WHERE (CLOSE_TIME < '01/01/2010')
DELETE FROM PROBSUMMARYA1 WHERE (CLOSE_TIME < '01/01/2010')
DELETE FROM PROBSUMMARYA2 WHERE (CLOSE_TIME < '01/01/2010')
COMMIT;
```

Option 2 – calling a stored procedure with error handling

The implementation can be stored as a procedure in the database table space. For scheduling, the procedure can be called using a batch file.

The SQL command to call the procedure listed in Appendix B is:

```
1 spool process_tables.log
2 begin
3   pkg_Archive_Incidents.Process_Tables (12);
4 end;
```

This example executes the function Process_Tables in the package pkg_Archive_Incidents. This package takes as parameter the number of months the incident has to be closed, in this example 12 month or a year.

Viewing archived data from the RDBMS

This chapter describes how to view the archived data from within Service Manager.

The following steps are necessary to view the archived data:

1. Create and configure the archiving dbdicts in Service Manager
2. Configure archiving dbdicts to be accessible in Service Manager

Create and configure archiving objects in Service Manager

The new archive dbdicts need to be configured to connect to the archived data of the archiving table space. Since the data should be only viewed, but not updated, additional steps are needed.

There are two options to access the archived data from within Service Manager. The first is by adding a second database section to the sm.ini, the second by creating a view that the productive Service Manager user uses to access the archive data by the archive database user.

Option 1: Adding a second database section

In your sm.ini, locate the following section:

```
[oracle10]
sqldb:<prod table space>
sqllogin:<prod user>/<password>
```

Copy this section to represent the archive table space:

```
[oracle10arch]
sqldb:<arch table space>
sqllogin:<arch user>/<password>
```

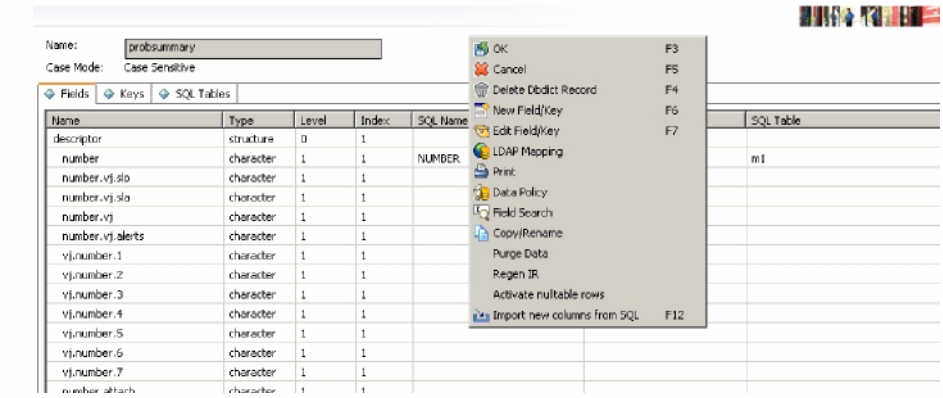
It is necessary to restart the Service Manager server after changing this section of the sm.ini.

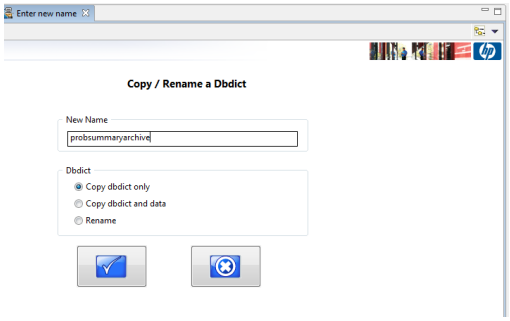
NOTE: Ensure that the <prod user> has create table rights in the archive table space for these steps to work.

```
1 grant create table to [DB PRODUCTIVE USER]
```

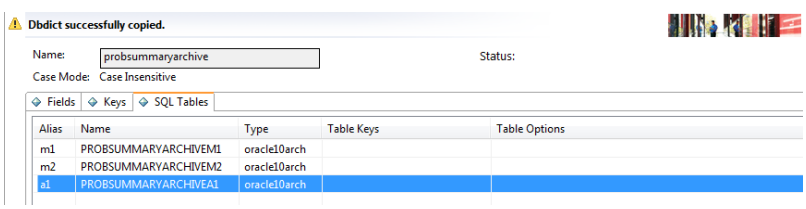
After the restart, follow these steps to create the archiving dbdict and table in the archive table space:

1. Go into the dbdict utility via Tailoring > Database Dictionary or by entering “dbdict” in the Service Manager command line
2. Search for the source filename, e.g. probsummary
3. Go to Options > Copy / Rename
4. Enter a unique name for the new dbdict, e.g. probsummaryarch
5. Select “copy dbdict only”





Once the dbdict was created, immediately go to the SQL Tables tab and change the Type column for all tables from oracle10 to oracle10arch as pictured below:



Click OK to move the tables from the productive table space to the archive table space.

Once the tables are created, change the granted rights of the SM archive user to allow for viewing of the data on the RDBMS only. Use the productive user to fill the archive tables with data. Service Manager will use only the archive user, based on the sm.ini settings related to the Type on the SQL tables tab, to access the data in the archive tables.

Option 2: Creating a View

Start with copying the dbdict as described in Option 1 as well.

1. Go into the dbdict utility via Tailoring > Database Dictionary or by entering "dbdict" in the Service Manager command line
2. Search for the source filename, e.g. probsummary
3. Go to Options > Copy / Rename
4. Enter a unique name for the new dbdict, e.g. probsummaryarch
5. Select "copy dbdict only"

Next, delete the RDBMS tables that were automatically generated when the new dbdict record was added. Use the following command for each of the defined database tables. Please be aware to use the database table names.

```
drop table [TABLENAME];
```

Instead of a database table create a view with the same name and same specification of fields and fieldnames. The view will be directed to the archiving table space to access the archived data. The following statement is an example of how to create a view for the archived attachments.

```
CREATE OR REPLACE FORCE VIEW [DB PRODUCTIVE USER]. archiveattachmentsm1 (
"application", "topic", "type", "segment", "dataprgn", "sysmodcount",
"sysmoduser", "sysmodtime", "mimetype", "sizeprgn ", "filename", "uidprgn
", "compressed", "compressed size")
AS
SELECT
sysattach.APPLICATION, sysattach.TOPIC, sysattach."TYPE",
sysattach."SEGMENT", sysattach.DATAPRGN, sysattach.SYSMODCOUNT,
sysattach.SYSMODUSER,sysattach.SYSMODTIME, sysattach.MIMETYPE,
sysattach.SIZEPRGN, sysattach.FILENAME, sysattach.UIDPRGN,
sysattach.COMPRESSED, sysattach.COMPRESSED SIZE
 FROM [DB ARCHIVING USER].SYSATTACHMEM1 sysattach;
```

A create view statement is needed for each table which was dropped in the step before. Service Manager automatically uses the database view to access the data, if the same name is used for the view that was used for the table name before. It also controls that no update rights are available for Service Manager when using the database view to display the data.

Configure archiving objects to be accessible in Service Manager

The user has only read-only access to the archive data if the tables or views were set up correctly. To ensure that Service Manager displays and treats the archive records as read-only as well, follow the steps outlined in this section.

Implement view of archived incidents

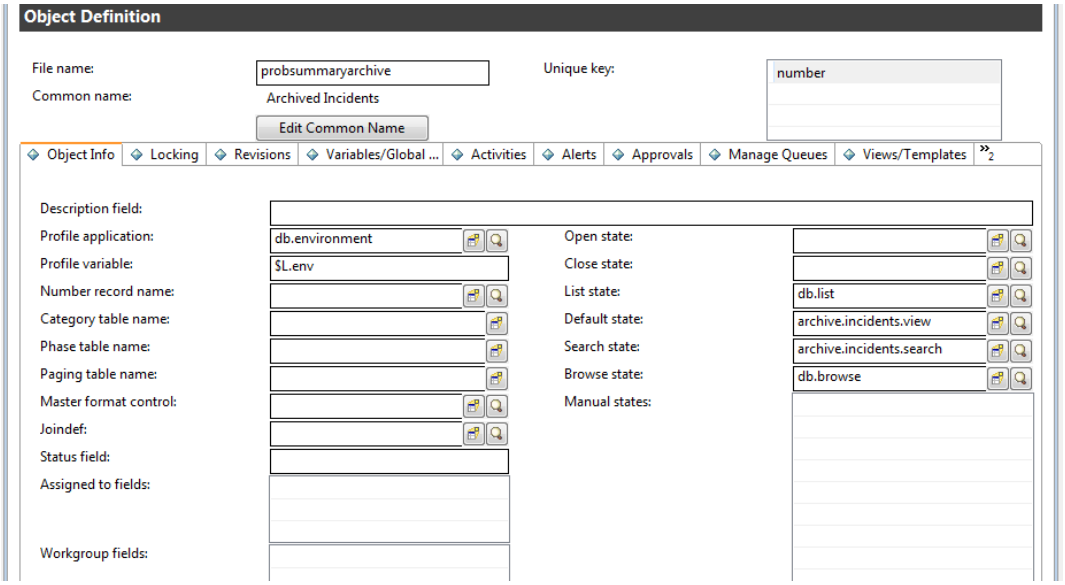
Create the following Service Manager elements to view archived incidents:

- Object archiveprobsummary
- Search Configuration archiveprobsummary
- Search State archive.incident.search
- Default State archive.incident.view
- Search Displayscreen and Displayoptions archive.incident.search
- Default Displayscreen and Displayoptions archive.incident.view

- Search Form advFind.archive.incident.search
- Default Form archive.incident.view.g
- Link archiveprobsummary

Object

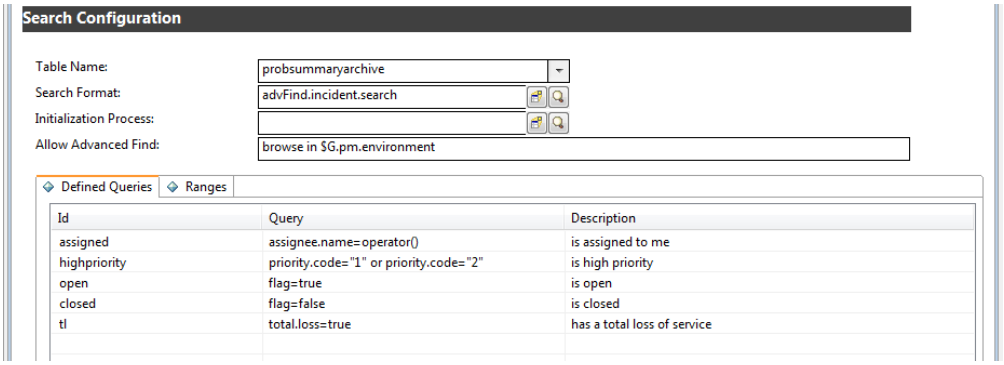
First, create the Object record archiveprobsummary as a copy of the DEFAULT object. To do so go to Tailoring > Document Engine > Objects and search for the DEFAULT object. Change the name to the name of your archive dbdict and click the Add button. Next, click on Edit Common name to set the common name to Archived Incidents.



Set the States to the following values:

- Open State: <empty>
- Close State: <empty>
- List State: <db.list>
- Default State: archive.incidents.view
- Search State: archive.incident.search
- Browse State: db.browse

To add a search configuration, go to Database Manager, table SearchConfig, search for Table Name = probsummary. Change the table name to archiveprobsummary and click on Add. Check in the archiveprobsummary Object record to ensure that the search configuration was assigned correctly.



States

Create the State record archive.incident.view as a copy of the db.view State record. To do so go to Tailoring > Document Engine > States and search for the db.view State. Change the name to

archive.incident.view and click the Add button. Enter the following information in the newly created State record:

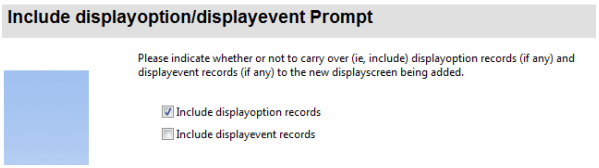
- Displayscreen: archive.incident.view
- Default Format: IM.update.incident
- Input condition: false

Next create the archive.incident.search State by copying the db.search State. Enter the following information in the newly created State record:

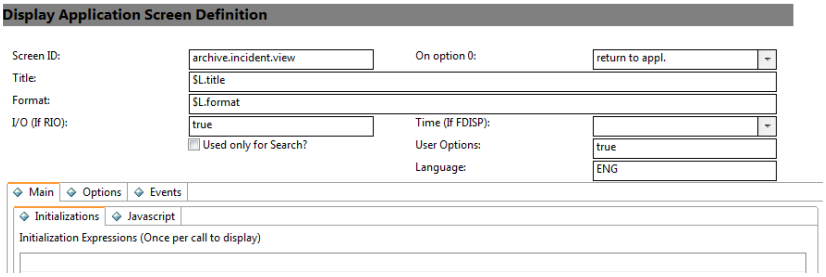
- Displayscreen: archive.incident.search

Displayscreens and -options

Create the displayscreen archive.incident.view as a copy of the displayscreen db.view by going to Tailoring > Tailoring Tools > Display screens, selecting the db.view displayscreen, changing the name to archive.incident.view and clicking on the Add button. In the wizard select to Include displayoption records and click Next.



The I/O condition needs to be set to false.



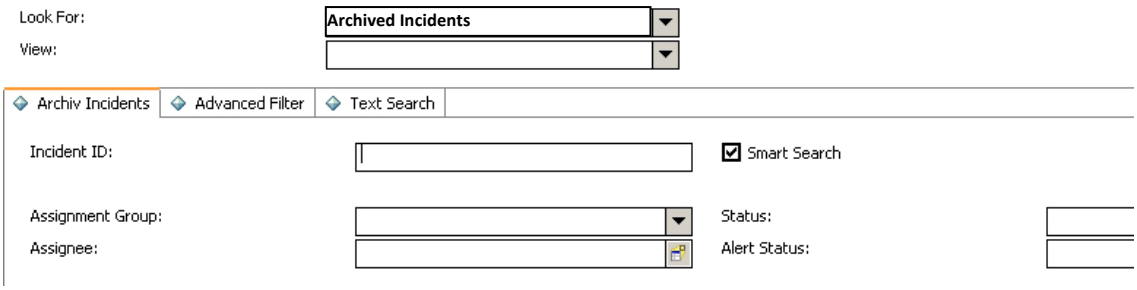
Next, delete all displayoption records related to the new displayscreen that require update access:

- OK
- Add
- Save
- Delete
- Mass Add
- Mass Update
- Mass Delete

Create the displayscreen archive.incident.search as a copy of the displayscreen db.search as described above. Also include the displayoptions when adding the new displayscreen record.

Delete the Add displayoption in the archive.incident.search displayscreen, since add is not allowed in the archived data set.

A user can search for archived incidents by changing to Archived Incidents in the Look For drop down box in the search screen.



Links

Copy the probsummary link to archiveprobsummary(or the name of your archive dbdict) to be able to use the find and fill (only in search mask) functions and to display the virtual joins. To copy the link, go to Tailoring > Tailoring Tools > Links and select the probsummary link. Change the name from probsummary to archiveprobsummary and click Add.

Next some of the link lines have to be adjusted to point to the archived data sets for activities, SLOs, attachments and related records.

Viewing archived activities

To view archived activities, the link line for activities in the probsummaryarchive link has to point to the archived version of activities, and a new form has to be created for the archiveactivity table to be able to view these activities.

Locate the link lines for number.vj and thenumber in the probsummaryarchive link that go against the activity table and change the target file name for both to archiveactivity.

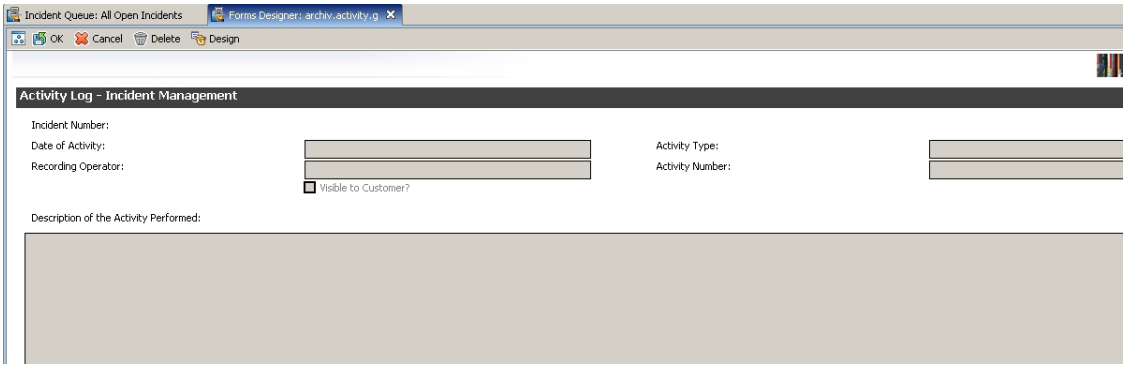
Link File

Name: probsummaryarchive System:

Description:

Source Field Name	Target File Name	Target Format Name	Target Field Name	Add Query	Comments
number	schedule		pm.number		VJ only
agreement.id	sla		agreement.id		
number.vj	archivactivity		number		Used for Virtual Join only
thenumber	archivactivity		thenumber	$query	executes when a line in the activity VJ is clicked

Next copy the activities.g form and name it archiveactities.g. Change the filename association for the new form to archiveactivity. This form is used when a user clicks on a specific activity entry for viewing.



Viewing archived SLOs

For viewing archived SLOs, the link line starting with number.vj.slo in the archiveprobsummary link has to be changed to have the target file name of archivesloresponse instead of sloresponse.

Viewing archived attachments

To view attachments from a table other than SYSATTACHMENTS, several steps are necessary.

Background Information regarding view of attachment

Normally attachments are stored in the SYSATTACHMENT table. When viewing incidents, problems or other records a java applet is used to display the attachments in the form's attachment element. The java applet also manages the functions to add, view and delete attachments. Unfortunately the java applet is programmed to only work with the SYSATTACHMENT table. When archiving attachments the name of the attachment table changes and new logic needs to be implemented to be able to view attachments. Implementing add or delete functions is not necessary since archived objects are not edited.

There are three options on how to handle archiving attachments.

Option1: To view an attachment the attachment data has to be read, written to a temporary file and opened. The best option to write and read the temporary file is the java script function writeFile(). This function can process binary data, but has the disadvantage that the file is written to the Service Manager server machine only.

Note: Option 1 requires the uncompress() method that is not supported for attachments in binary versions higher than SM 7.02. This method was replaced by SCFile.getAttachment() and related methods that will work only against the SYSATTACHMENT table.

Option 2: Copy the SYSATTACHMENTARCHIVE record to the SYSATTACHMENT table when initializing the incident for viewing. The attachment information will then be automatically available and will stay in the SYSATTACHMENT table until removed by an administrator.

Option 3: Keep all information in the SYSATTACHMENT table, but change the application from probsummary to probsummaryarchive.

Pros and cons of each option

	Pro	Con
Option 1	All archived data stays in the archive table	Will only work for attachments that are small (1 segment), since uncompress will not work for multi-segment binary data. Will only work in SM 7.02 and older
Option 2	Attachments are readily and consistently available	Administrator needs to write a purge script to remove SYSATTACHMENT records that have no matching incident record in the probsummary table
Option 3	Fast and efficient archiving, no tailoring necessary to view archived attachments	All data stays in SYSATTACHMENT

Option 1

To view the attachments, the linkline number.vj.alerts in the link archiveprobsummary is used to point to the archiveattachment table. In the original probsummary link this line is used to select alerts but alerts are not used for archived incidents. Change the number.vj.alerts link line to point to the target file name of archiveattachment and the target field name of topic.

Create the displayoption archive.incident.view_open_attach for the display screen archive.incident.view to open a selected attachment out of the table archiveattachments.

Enter the following in the RAD tab, Pre RAD Expressions:

```
$filename=cursor.field.contents()
```

In the pre JavaScript section the attachment is selected from the table archiveattachments and a physical file is created on the Server. The data and the filename are read from the attachment record and used to create the physical file. The us.launch.external RAD application is called to execute a share to the server and open the file. See the below code excerpt for an example. The network share drive and temp drive need to be adjusted for each customer system.

```
1 var  attachFilename = system.vars.$filename;
2
3 print( "start processing for file " + attachFilename ) ;
4
5 //Get attachments from archiveattachment table
6 var fAttachment s = new SCFile ( "archiveattachments" ) ;
7 var rc = fAttachments.doSelect("application=\"probsummary \" and
topic=\""+
8 system.vars.$L.filed.number +"\" and filename=\"" + attachFilename
+"\"" ) ;
9 v a r bCompressed ;
10
11 // decompress attachment data
12 if ( rc == RC_SUCCESS ) {
13 var binaryStr = "";
14
14 rc = fAttachments.getFirst( ) ;
15 bCompressed = fAttachments.compressed ;
16
17 while ( rc == RC_SUCCESS) {
18 binaryStr += fAttachments.data ;
19 rc = fAttachments.getNext( ) ;
20 }
21
22 if ( bCompressed == true ) {
23 var attachData = uncompress( binaryStr ) ;
24 } else {
25 var attachData = binaryStr ;
26 }
27
28 print( "writing file to temp folder on server " ) ;
29
30 // set file name
31 var fileName = "D:\\temp\\ " + attachFilename ;
32 var outputEmpty = writeFile( fileName , "b" , attachData ) ;
33
34 // execute bat file to create net share
35 system.functions.rtecall( "callrad",rc,"us.launch.external",["name"] ,
36 [ "cmd;/Q /C net use x: \\\\servername\\temp"] , true ) ;
37
38 // open file
39 var netShareFileName = "x:\\" + attachFilename ;
40 system.functions.rtecall("callrad",rc,"us.launch.external",["name"],
41 [netShareFileName],true) ;
42 }
```

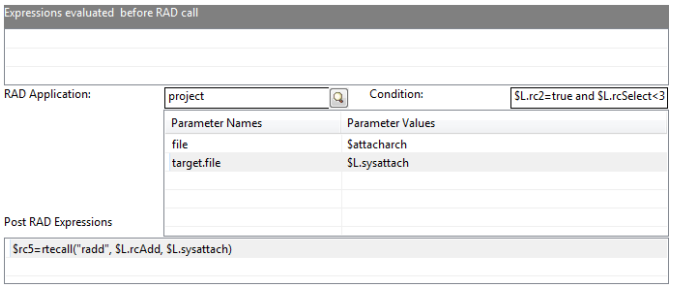
Option 2

To copy the attachment from the SYSATTACHMENTARCHIVE table to SYSATTACHMENT on viewing the archived incident create an initial process archive.incident.view.init for the archive.incident.view State.

Within that process, enter the following initial expression:

```
$L.rc1=rtecall("rinit", $L.rcinit1, $L.sysattach, "SYSATTACHMENTS")
if (nullsub($nextRec, false)=false) then ($L.rc2=rtecall("rinit",
$L.rcinit2, $attacharch, "SYSATTACHMENTSARCH"))
if ($L.rc2=true and nullsub($nextRec, false)=false) then
($L.rc3=rtecall("select", $L.rcSelect, $attacharch, "topic=\""+number in
$L.file+"\""))
```

On the RAD tab, enter the following:



RAD application: `project`

Condition: `$L.rc2=true and $L.rcSelect<3`

Parameter names	Parameter Values
file	$attacharch
target.file	$L.sysattach

Post RAD expressions:

```
$L.rc5=rtecall("radd", $L.rcAdd, $L.sysattach)
```

On the Final Expressions tab enter:

```
$L.rc6=rtecall("next", $L.rcNext, $attacharch)
if ($L.rcNext~=1) then ($nextRec=true) else ($nextRec=false)
```

On the next Process tab enter:

```
archive.incidents.view.init          $nextRec=true
```

To periodically clean up the records pulled back into the SYSATTACHMENTS table, run the following JavaScript code from the ScriptLibrary:

```
function CleanUpAttach()
{
   var attach = new SCFile("SYSATTACHMENTS")
   var incidents = new SCFile("probsummary")

   var rcSel = attach.doSelect("topic # \"IM\"")
   while (rcSel == RC_SUCCESS)
   {
      rcSelInc=incidents.doSelect("number = \"" + attach.topic + "\"")
      if (rcSelInc != RC_SUCCESS)
      {
         print("topic " + attach.topic + " will be removed from
SYSATTACHMENTS")
         attach.doDelete()
```

```
        }
        rcSel = attach.getNext()
    }
}

CleanUpAttach()
```

Option 3

In this option, the archived attachment stays in the SYSATTACHMENT table with the modified application of "probsummaryarch".

The SQL syntax to include in the archiving steps for updating the application field rather than moving the data in SYSATTACHMENTSM1 is:

```
UPDATE SYSATTACHMENTSM1 SET APPLICATION = 'probsummaryarch'
WHERE(TOPIC IN (SELECT NUMBER FROM PROBSUMMARYM1 WHERE (CLOSE_TIME <
'01/01/2010')))
```

After the archive time frame is exceeded (e.g. after 10 years), use the following delete statement to delete all old SYSATTACHMENTSM1 records:

```
DELETE FROM SYSATTACHMENTSM1 WHERE ( APPLICATION = 'probsummaryarch' AND
TOPIC ISIN ( SELECT NUMBER FROM PROBSUMMARYARCHM1 WHERE (CLOSE_TIME <
'01/01/2000'  ) ) )
COMMIT;
```

Summary

The System Administrator needs to first decide on an archival strategy. Archiving into an independent system is less work to create the archive, but more work to retrieve the data, whereas archiving into the same Service Manager system requires some work to set up the archive, but makes it easier for end-users to access the data.

Archiving attachments needs special consideration, because viewing attachments is only possible through the SYSATTACHMENTS table. It is not possible to use JavaScript to copy the attachment data back and forth, because the JavaScript engine translates all information into UTF which causes the binary attachment data to become corrupt. In our examples, we point out 3 options, each with its sets of pro's and con's. We do not recommend using the first option, since it will only work for 1-segment attachments.

This document concentrates on Incident Management and was written for SQL Server and Oracle, but can be ported with slight modifications for other modules and other databases.

Appendix A

DBDICTS to archive per module in the out-of-box system

ServiceDesk

incidents, screlations, activityservicemgt, incdepends, SYSATTACHMENT, slaactive, sloresponse

Incident Management

probsummary, (problem), screlation, activity, SYSATTACHMENT, slaactive, sloresponse

Change Management

cm3r,(cm3rpage), cm3t, (cm3tpage), screlation, activitycm3r, SYSATTACHMENT, slaactive, sloresponse, ApprovalLog, (Approval not included, since there should be no active approvals on closed changes), AlertLog, dataModEvent (if unplanned changes and change verification are used)

Problem Management

rootcause, rootcausetask, knownerror, knownerrortask, screlation, activityproblem, activityproblemtask, activityknownerror, SYSATTACHMENT, slaactive, sloresponse, AlertLog

Request Management

ocmq, ocml, ocmo, screlation, ApprovalLog

Knowledge Management

kmdocument, kmadaptivelearning, kmfeedback, cm3r, SYSATTACHMENT

SLA, Service Catalog and Configuration Management not included, since they typically do not include "aging" data.

Appendix B

Sample Oracle package to move data to archive table space

```
define Source_Schema = SM
define Arc_Dest_Schema = SM9ARCH
define Records_In_Chunk = 100000
define Screen_Trace = true
define Analyze_Tables = true

create or replace package pkg_Archive_Incidents
as
    procedure Enable_Row_Movement;
    procedure Process_Tables (
        pnMonthsAgo   number default 12
    );
end pkg_Archive_Incidents;
/

create or replace package body pkg_Archive_Incidents
as
    glScreenTrace       boolean          := &&Screen_Trace;
    gnRecordsInChunk    binary_integer   := &&Records_In_Chunk;
```

```
    gdArcThresholdTime  date;
    gcLoggedUser        varchar2 (1024)  := substr (   sys_context
('USERENV', 'SESSION_USER')
                            || ' @ '
                            || sys_context ('USERENV', 'TERMINAL')
                            || ' ['
                            || sys_context ('USERENV', 'IP_ADDRESS')
                            || ']',
                            1,
                            1024
                            );

    type rTableInfo is record (
        cTableName              varchar2 (30),
        cAction                 varchar2 (32),
        cJoinField              varchar2 (30),
        cInsertJoinCondition    varchar2 (4096) := null,
        cDeleteJoinCondition    varchar2 (4096) := null,
        nRowsJustMoved          binary_integer  := 0,
        nRowsJustDeleted        binary_integer  := 0,
        nTotalMovedRows         binary_integer  := 0,
        nTotalDeletedRows       binary_integer  := 0
    );

    type trTableInfo is table of rTableInfo
        index by binary_integer;

    tTableInfo          trTableInfo;
    nI                  binary_integer;
    cSQLStatement       varchar2 (32767);

    procedure Say (
        pcMessageToScreen   in   varchar2
    )
    is
    begin
        if glScreenTrace
        then
            dbms_output.put_line (pcMessageToScreen);
        end if;
    end Say;

    procedure Log_Error (
        pcErrorMessage   in   varchar2
    )
    is
        pragma autonomous_transaction;
    begin
        insert into ARC_ERROR_LOG
                    (ERROR_MESSAGE,
                     LOGGED_USER
                    )
             values (pcErrorMessage,
                     gcLoggedUser
                    );
        commit;
    end Log_Error;

    procedure Init_Table_Info
    is
```

```
    begin
        nI := 1;
        tTableInfo (nI).cTableName := 'SCRELATIONA1';
        tTableInfo (nI).cAction := 'delete';
        tTableInfo (nI).cJoinField := 'NUMBERPRGN';
        tTableInfo (nI).cDeleteJoinCondition :=
         'rowid in ('
      || 'select P.rowid '
      || 'from &&Source_Schema..'
      || tTableInfo (nI).cTableName
      || ' P, '
      || 'table (:1) T '
      || 'where '
      || 'T.column_value = '
      || 'case '
      || 'when T.column_value like ''IM%'' then P."depend" '
      || 'when (T.column_value like ''PM%'' or T.column_value like
''SD%'') then P."source" '
      || 'else null '
      || 'end'
      || ')';
        nI := nI + 1;
        tTableInfo (nI) := tTableInfo (nI - 1);
        tTableInfo (nI).cTableName := 'SCRELATIONM1';
        tTableInfo (nI).cDeleteJoinCondition :=
         'rowid in ('
      || 'select P.rowid '
      || 'from &&Source_Schema..'
      || tTableInfo (nI).cTableName
      || ' P, '
      || 'table (:1) T '
      || 'where '
      || 'T.column_value = '
      || 'case '
      || 'when T.column_value like ''IM%'' then P."depend" '
      || 'when (T.column_value like ''PM%'' or T.column_value like
''SD%'') then P."source" '
      || 'else null '
      || 'end'
      || ')';
        nI := nI + 1;
        tTableInfo (nI) := tTableInfo (nI - 1);
        tTableInfo (nI).cTableName := 'SLORESPONSEM1';
        tTableInfo (nI).cAction := 'move';
        tTableInfo (nI).cInsertJoinCondition :=
                                    'P.FOREIGN_KEY = T.column_value';
        tTableInfo (nI).cDeleteJoinCondition :=
               'FOREIGN_KEY in ('
            || 'select column_value '
            || 'from table (:1)'
            || ')';
        nI := nI + 1;
        tTableInfo (nI) := tTableInfo (nI - 1);
        tTableInfo (nI).cTableName := 'ACTIVITYA1';
        tTableInfo (nI).cAction := 'move';
        tTableInfo (nI).cJoinField := 'NUMBERPRGN';
        tTableInfo (nI).cInsertJoinCondition :=
                                    'P.NUMBERPRGN = T.column_value';
        tTableInfo (nI).cDeleteJoinCondition :=
               'NUMBERPRGN in ('
```

```
            || 'select column_value '
            || 'from table (:1)'
            || ')';
        nI := nI + 1;
        tTableInfo (nI) := tTableInfo (nI - 1);
        tTableInfo (nI).cTableName := 'ACTIVITYM1';
         nI := nI + 1;
        tTableInfo (nI) := tTableInfo (nI - 1);
        tTableInfo (nI).cTableName := 'SYSATTACHMEM1';
        tTableInfo (nI).cInsertJoinCondition :=
               'P.rowid in ('
            || 'select /*+ full (M) index (R R(APPLICATION, TOPIC)) */
R.rowid '
            || 'from &&Source_Schema..'
            || tTableInfo (nI).cTableName
            || ' R, '
            || 'MasterIDs M '
            || 'where '
            || 'R.APPLICATION = ''probsummary'' '
            || 'and R.TOPIC = M.ID '
            || 'union '
            || 'select /*+ full (M) index (R R(APPLICATION, TOPIC)) */
R.rowid ';
            ||
        tTableInfo (nI).cDeleteJoinCondition :=
               'rowid in ('
            || ' with MasterIDs as '
            || '(select /*+ materialize */ column_value ID from table
(:1)) '
            || 'select /*+ full (M) index (R R(APPLICATION, TOPIC)) */
R.rowid '
            || 'from &&Source_Schema..'
            || tTableInfo (nI).cTableName
            || ' R, '
            || 'MasterIDs M '
            || 'where '
            || 'R.APPLICATION = ''probsummary'' '
            || 'and R.TOPIC = M.ID '
            || 'union '
            || 'select /*+ full (M) index (R R(APPLICATION, TOPIC)) */
R.rowid '
            ||
;
        nI := nI + 1;
        tTableInfo (nI) := tTableInfo (nI - 1);
        tTableInfo (nI).cTableName := 'PROBSUMMARYM1';
        tTableInfo (nI).cInsertJoinCondition :=
                                              'P."numberprgn" =
T.column_value';
        tTableInfo (nI).cDeleteJoinCondition :=
               '"numberprgn" in ('
            || 'select column_value '
            || 'from table (:1)'
            || ')';
    end Init_Table_Info;

    procedure Move_And_Purge_Tables
    is
        tNumberPrgn    sys.dbms_debug_vc2coll;
    begin
```

```
        tNumberPrgn := sys.dbms_debug_vc2coll ();

        loop
            select "numberprgn"
            bulk collect into tNumberPrgn
              from &&Source_Schema..PROBSUMMARYM1
             where "status" = 'closed'
               and "sysmodtime" < gdArcThresholdTime
               and rownum < gnRecordsInChunk;
            exit when tNumberPrgn.count = 0

            for nJ in tTableInfo.first .. tTableInfo.last
            loop
                nI := nJ;
                if tTableInfo (nI).cAction = 'move'
                then
                    if tTableInfo (nI).cTableName = 'SYSATTACHMEM1'
                    then
                        cSQLStatement :=
                          'insert /*+ append */ into &&Arc_Dest_Schema..'
                            || tTableInfo (nI).cTableName
                            || ' with MasterIDs as '
                            || '(select /*+ materialize */ column_value
ID from table (:1)) '
                            || ' select P.*'
                            || ' from &&Source_Schema..'
                            || tTableInfo (nI).cTableName
                            || ' P'
                            || ' where '
                            || tTableInfo (nI).cInsertJoinCondition;
                    else
                        cSQLStatement :=
                          'insert /*+ append */ into &&Arc_Dest_Schema..'
                            || tTableInfo (nI).cTableName
                            || ' select P.*'
                            || ' from &&Source_Schema..'
                            || tTableInfo (nI).cTableName
                            || ' P,'
                            || ' table (:1) T'
                            || ' where '
                            || tTableInfo (nI).cInsertJoinCondition;
                    end if;

                    if tTableInfo (nI).cJoinField = 'NUMBERPRGN'
                    then
                        execute immediate cSQLStatement
                                    using tNumberPrgn;



                    tTableInfo (nI).nRowsJustMoved := sql%rowcount;
                    Say (   'Table '
                         || tTableInfo (nI).cTableName
                         || ': just moved '
                         || to_char (tTableInfo (nI).nRowsJustMoved)
                         || ' rows.');
                end if;

                cSQLStatement :=
                       'delete &&Source_Schema..'
```

```
                    || tTableInfo (nI).cTableName
                    || ' where '
                    || tTableInfo (nI).cDeleteJoinCondition;

                if tTableInfo (nI).cJoinField = 'NUMBERPRGN'
                then
                    execute immediate cSQLStatement
                                using tNumberPrgn;

                tTableInfo (nI).nRowsJustDeleted := sql%rowcount;
                Say (   'Table '
                     || tTableInfo (nI).cTableName
                     || ': just deleted '
                     || to_char (tTableInfo (nI).nRowsJustDeleted)
                     || ' rows.');

            end loop;

            commit;

            for nJ in tTableInfo.first .. tTableInfo.last
            loop
                nI := nJ;
                tTableInfo (nI).nTotalDeletedRows :=
                      tTableInfo (nI).nTotalDeletedRows
                    + tTableInfo (nI).nRowsJustDeleted;
                tTableInfo (nI).nTotalMovedRows :=
                      tTableInfo (nI).nTotalMovedRows
                    + tTableInfo (nI).nRowsJustMoved;
            end loop;
        end loop;

        for nJ in tTableInfo.first .. tTableInfo.last
        loop
            nI := nJ;
            Say (   'Table '
                 || tTableInfo (nI).cTableName
                 || ': grand total of '
                 || to_char (tTableInfo (nI).nTotalDeletedRows)
                 || ' deleted rows.');
            Say (   'Table '
                 || tTableInfo (nI).cTableName
                 || ': grand total of '
                 || to_char (tTableInfo (nI).nTotalMovedRows)
                 || ' moved rows.');
        end loop;
    end Move_And_Purge_Tables;

    procedure Enable_Row_Movement
    is
        cRowMovement   varchar2 (32);
    begin
        Init_Table_Info ();

        for nJ in tTableInfo.first .. tTableInfo.last
        loop
            nI := nJ;
            Say (   'Enabling row movement for table '
                 || tTableInfo (nI).cTableName);
```

```
            execute immediate    'select row_movement from user_tables
where table_name = '''
                                || upper (tTableInfo (nI).cTableName)
                                || ''''
                           into cRowMovement;

            if cRowMovement = 'ENABLED'
            then
                Say (   'Row movement IS ALREADY ENABLED for table '
                     || tTableInfo (nI).cTableName);
            else
                execute immediate    'alter table '
                                  || tTableInfo (nI).cTableName
                                  || ' enable row movement';

                Say (   'Row movement has been enabled for table '
                     || tTableInfo (nI).cTableName);
            end if;
        end loop;
    end Enable_Row_Movement;

    procedure Shrink_Tables
    is
        cRowMovement   varchar2 (32);
    begin
        for nJ in tTableInfo.first .. tTableInfo.last
        loop
            nI := nJ;
            Say ('Shrinking ' || tTableInfo (nI).cTableName);

            execute immediate    'select row_movement from user_tables
where table_name = '''
                                || upper (tTableInfo (nI).cTableName)
                                || ''''
                           into cRowMovement;

            if cRowMovement = 'ENABLED'
            then
                execute immediate    'alter table '
                                  || tTableInfo (nI).cTableName
                                  || ' shrink space cascade';

                Say (   'Table '
                     || tTableInfo (nI).cTableName
                     || ' has been shrinked.');
            else
                Say (   'Cannot shrink table '
                     || tTableInfo (nI).cTableName
                     || ' because row movement IS NOT ENABLED.');
            end if;
        end loop;
    end Shrink_Tables;

    procedure Analyze_Tables
    is
    begin
        for nJ in tTableInfo.first .. tTableInfo.last
        loop
            nI := nJ;
            Say (   'Gathering statistics for table '
```

```
                || tTableInfo (nI).cTableName);
            dbms_stats.gather_table_stats
                    (ownname    => '&&Source_Schema',
                     tabname    => tTableInfo (nI).cTableName,
                     estimate_percent       => 100,
                     method_opt  => 'for all indexed columns size auto',
                           cascade               => true
                          );
            Say ('Table ' || tTableInfo (nI).cTableName
                 || ' has been analyzed.');
        end loop;
    end Analyze_Tables;

    procedure Process_Tables (
        pnMonthsAgo   number default 12
    )
    is
        cErrorMessage   varchar2 (32767);
    begin
        dbms_output.put_line
                    ('The archiving of Incident Data has been started @ '
                    || to_char (sysdate, 'dd-mon-yyyy hh24:mi:ss')
                    || ' by '
                    || sys_context ('USERENV', 'SESSION_USER'));

        gdArcThresholdTime := trunc (add_months (sysdate, -pnMonthsAgo));
        dbms_output.put_line (   'The threshold archiving date is '
                              || to_char (gdArcThresholdTime,
                                          'dd-mon-yyyy hh24:mi:ss'));
        Init_Table_Info ();

        if &&Analyze_Tables.
        then
            Analyze_Tables ();
        end if;

        Move_And_Purge_Tables ();
        Shrink_Tables ();
        dbms_output.put_line ( 'The archiving of Incident Data ended @ '
                    || to_char (sysdate, 'dd-mon-yyyy hh24:mi:ss'));
    exception
        when others
        then
            cErrorMessage :=
                   case
                       when tTableInfo.exists (nI)
                           then    'Probably @ table '
                                || tTableInfo (nI).cTableName
                                || ': '
                       else ''
                   end
                || sqlerrm
                || '; '
                || dbms_utility.format_error_backtrace;
            Log_Error (cErrorMessage);
            Say (cErrorMessage);
            rollback;
            raise;
    end Process_Tables;
end pkg_Archive_Incidents;
```

```
/

exit
/
```

For more information

Please visit the HP Software support Web site at:

www.hp.com/go/hpsoftwaresupport

This Web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued customer, you can benefit by being able to:

- Search for knowledge documents of interest
- Submit and track progress on support cases
- Submit enhancement requests online
- Download software patches
- Manage a support contract
- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Note: Most of the support areas require that you register as an HP Passport user and sign in. Many also require an active support contract.

To find more information about support access levels, go to the following URL:

www.hp.com/go/hpsoftwaresupport/new_access_levels

To register for an HP Passport ID, go to the following URL:

www.hp.com/go/hpsoftwaresupport/passport-registration

Technology for better business outcomes