HP Operations Orchestration Software

Software Version: 9.00

Purging OO Run Histories from MySQL Databases

Document Release Date: June 2010

Software Release Date: June 2010

Legal Notices

On the Web: Finding OO support and documentation

There are two Web sites where you can find support and documentation, including updates to OO Help systems, guides, and tutorials:

- The OO Support site
- BSA Essentials Network

Support

Documentation enhancements are a continual project at Hewlett-Packard Software. You can obtain or update the HP OO documentation set and tutorials at any time from the HP Software Product Manuals Web site. You will need an HP Passport to log in to the Web site.

To obtain HP OO documentation and tutorials

1. Go to the HP Software Product Manuals Web site (*http://support.openview.hp.com/selfsolve/manuals*).
2. Log in with your HP Passport user name and password.

   OR

   If you do not have an HP Passport, click **New users – please register** to create an HP Passport, then return to this page and log in.

   If you need help getting an HP Passport, see your HP OO contact.
3. In the **Product** list box, scroll down to and select **Operations Orchestration**.
4. In the **Product Version** list, click the version of the manuals that you're interested in.
5. In the **Operating System** list, click the relevant operating system.
6. Click the **Search** button.
7. In the **Results** list, click the link for the file that you want.

BSA Essentials Network

For support information, including patches, troubleshooting aids, support contract management, product manuals and more, visit the following site: *http://www.hp.com/go/bsaessentialsnetwork*

This is the **BSA Essentials Network** Web page. To sign in:

1. Click **Login Now**.
2. On the **HP Passport sign-in** page, enter your HP Passport user ID and password and then click **Sign-in**.
3. If you do not already have an HP Passport account, do the following:
   a. On the **HP Passport sign-in** page, click **New user registration**.
   b. On the **HP Passport new user registration** page, enter the required information and then click **Continue**.
   c. On the confirmation page that opens, check your information and then click **Register**.
   d. On the **Terms of Service** page, read the Terms of use and legal restrictions, select the **Agree** button, and then click **Submit**.
4. On the **BSA Essentials Network** page, click **Operations Orchestration Community.**

The Operations Orchestration Community page contains links to announcements, discussions, downloads, documentation, help, and support.

Note: Contact your OO contact if you have any difficulties with this process.

In OO: How to find Help, PDFs, and tutorials

The HP Operations Orchestration software (HP OO) documentation set is made up of the following:

- Help for Central

  Central Help provides information to the following:

  - Finding and running flows
  - For HP OO administrators, configuring the functioning of HP OO
  - Generating and viewing the information available from the outcomes of flow runs

  The Central Help system is also available as a PDF document in the HP OO home directory, in the \Central\docs subdirectory.

- Help for Studio

  Studio Help instructs flow authors at varying levels of programming ability.

  The Studio Help system is also available as a PDF document in the HP OO home directory, in the \Studio\docs subdirectory.

- Animated tutorials for Central and Studio

  HP OO tutorials can each be completed in less than half an hour and provide basic instruction on the following:

  - In Central, finding, running, and viewing information from flows
  - In Studio, modifying flows

  The tutorials are available in the Central and Studio subdirectories of the HP OO home directory.

- Self-documentation for operations and flows in the Accelerator Packs and ITIL folders

  Self-documentation is available in the descriptions of the operations and steps that are included in the flows.

Table of Contents

About deleting run histories

This document is designed to provide a method for pruning old run history data for Central administrators and DBAs involved in the management of the data stored by Central systems.

This document is divided into three main sections:

1. Descriptions of the tables involved in storing historical run data in the *OO database*.
2. The procedure for *physically deleting old run history data*.
3. *Appendices* that contain information such as a diagram of the tables in the 7.60 **Run** schema, how to upgrade older schemas, and performance implications.

   The code examples shown in the appendices and the script that calls the pruning process are included in text form in the file **MySQL_Run_History_Purge.zip** (available on the Web site where you downloaded this document). The code files are:

   - To call the schema update process—

     mysql_oo_upgrade_history_schema_call.sql
   - To call the pruning process—

     mysql_oo_prune_run_history_call.sql
   - For *Appendix B: Upgrading older schemas*—

     mysql_oo_upgrade_history_schema.sql
   - For *Appendix C: Example cleanup stored procedure*—

     mysql_oo_prune_run_history.sql
   - For *Appendix D: Example scheduling scripts*—

     mysql_oo_prune_job.sh

Before deciding whether to implement the procedures in this document, read the entire document including *Appendix E: Performance implications*.

Required knowledge

MySQL database knowledge is required.

About the OO database tables

The tables involved in capturing run history information belong to the OO database. See *Appendix A: Tables diagram* for a diagram of the tables in the schema. The tables in the **Run** schema are:

- The *run table*
- The *run_history table*
- The *runstep_history* table
- The *property_history* table
- The *log_record* table
- The *flow_metrics* table

The run table

The **run** table stores information about flows that have not yet finished running. Every time a run performs a checkpoint, its current frame stack (including context variables) is placed into a binary object and written to a row in this table. The primary key of the **run** table is the **run id**. As soon as a run finishes, the entry in the **run** table is removed and placed in the **run_history** table.

There are no foreign keys between this table and any other table.

The run_history table

The **run_history** table stores run information that is used in reporting. There is one row in this table stored for every execution of a flow. The table stores general information about the run, such as its start time, end time, the number of its steps, and how the run ended.

Important Deleting data from the **run_history** table causes the loss of reporting information. However, if storage space is critical, you can delete data from this table. Just be aware that flows deleted from the **run_history** table will no longer be visible in any reports.

The runstep_history table

The **runstep_history** table stores reporting information for each step. There is a one-to-many relationship between the **run_history** table and the **runstep_history** table, enforced by a foreign key relationship between the **runstep_history.run_history_id** and **run.oid** fields, which uses cascading deletes.

Important Deleting data from the **runstep_history** table causes the loss of reporting information for each step of a flow, but the general flow information is still available for reporting. You will not however, be able to "drill down" into the steps which were executed by a flow that has been pruned. However, if storage space is critical, you can delete data from this table. Deleting data from the **runstep_history** table also deletes any related records from the **property_history** table.

Note: OO versions older than 7.20 require schema altering in order to properly support cascading deletes. See *Appendix B: Upgrading older schemas*.

The property_history table

The **property_history** table stores a row for each input of a step. There is a foreign key relationship between the fields **property_history.runstep_hist_id** and **runstep_history.oid**, with cascading deletes.

The log_record table

The **log_record** table stores a row for each step input that was designated to be recorded for reporting under a domain-term name. Essentially, it stores a subset of the data in the **property_history** table, but there is no foreign key relationship to the **runstep_history** table. If a **run_history** row is deleted, rows will also be deleted from the **runstep_history** and **property_history** tables, but the **log_record** table is left intact.

The data in the **log_record** table is used to plot dashboard charts, so deleting data from it will result in loss of dashboard information. This may or may not be a problem depending on how often you prune data. Since dashboard charts are meant to give a more "real-time" picture of what's going on with OO, deleting data from the **log_record** table for a period past where the data is useful for dashboards should be fine

The flow_metrics table

The **flow_metrics** table stores flow outcome counters. There is one entry for each flow, with counters broken down into **Resolved**, **Error**, **Diagnosed**, **No Action Taken**, and **Failed** outcomes, as well as the cumulative time taken by the flows.

This table is used to create the flow metrics bar::

Physically deleting data

To delete run histories, use the following approach

1. Upgrade the database schema if necessary (see *Appendix B: Upgrading older schemas*).
2. Establish a timestamp (date and time) when run histories older than it are deleted.
3. Determine how many run histories should be deleted.
4. Divide these run histories into batches to minimize the transaction size.
5. Starting with the oldest batch, delete the batches using one transaction per batch as follows:
    a. Begin the transaction.
    b. Delete data from the **run_history** table, if required.
    c. Update the **flow_metrics** table to reflect the deleted rows, if run histories were deleted.
    d. Delete data from the **runstep_history** table if data was not removed from the **run_history** table.
    e. Delete the rows for the deleted run steps from the **log_record** table, if necessary.
    f. Commit the transaction.

These steps, excluding the first one (upgrading), can be performed on a periodic basis from a scheduled job. An example stored procedure is provided in *Appendix C: Example cleanup stored procedure*.

You can schedule the cleanup job, as explained in *Appendix D: Example scheduling script*.

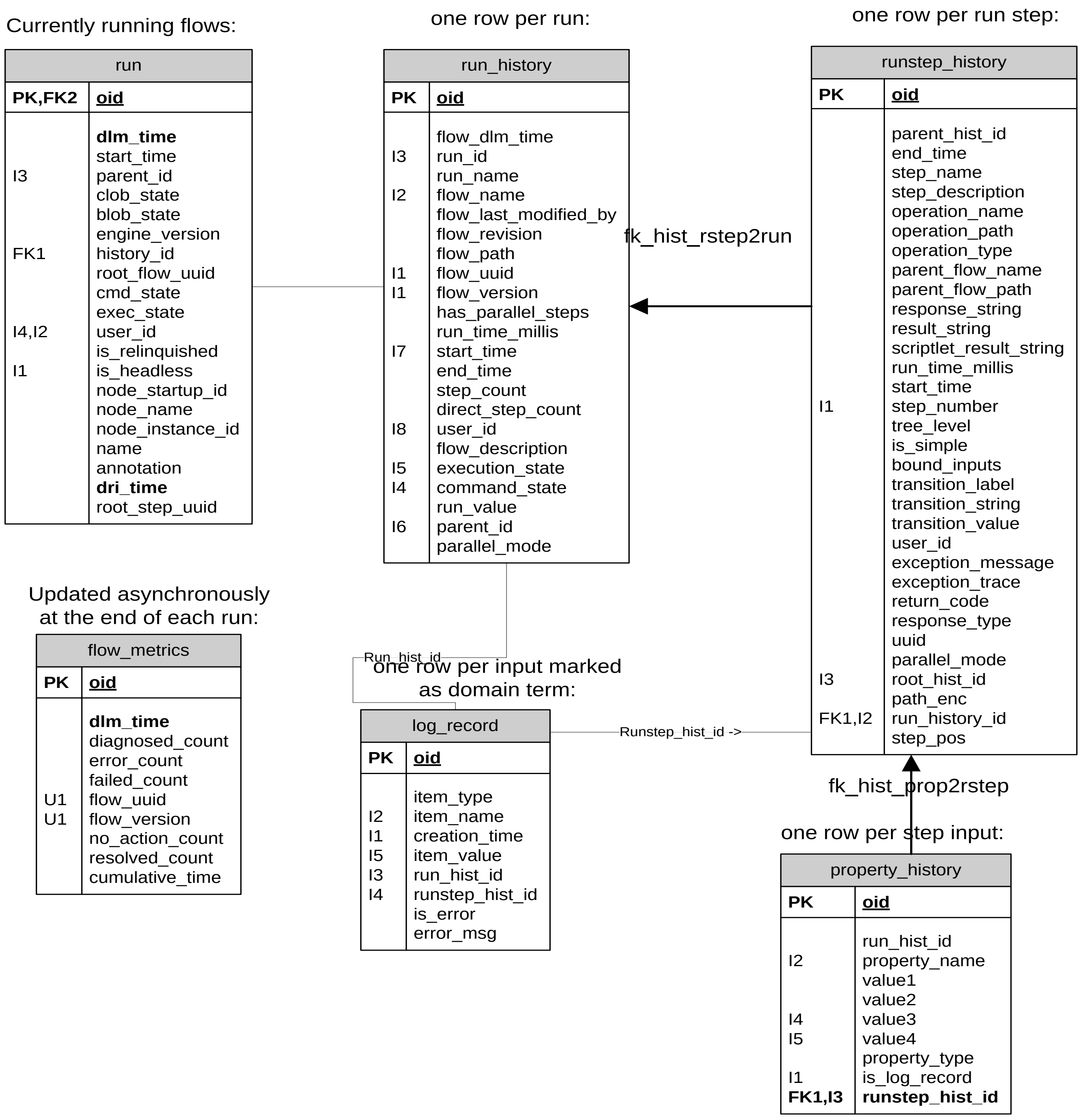
Appendices

The appendices in this section are meant to help you perform the necessary tasks involved in deleting run histories.

- *Appendix A. Tables diagram*
- *Appendix B. Upgrading older schemas*
- *Appendix C. Example cleanup stored procedure*
- *Appendix D. Example scheduling scripts*
- *Appendix E. Performance implications*

Appendix A: Tables diagram

7.50 Run Schema



Appendix B: Upgrading older schemas

This appendix contains the following examples:

- A stored procedure named **mysql_oo_upgrade_history_schema.sql**
- A script to call the procedure named **mysql_oo_upgrade_history_schema_call.sql**

The mysql_oo_upgrade_history_schema.sql stored procedure

The following stored procedure detects older versions of the schema (OO versions 7.0 and earlier) and alters the appropriate tables to support cascading deletes. We recommend that you use the text copy of this stored procedure contained in the file **mysql_oo_upgrade_history_schema.sql** instead of copying the code below, which has line breaks to make reading easier.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS `upgrade_history_schema` $$
CREATE DEFINER=`dharma_user`@`localhost` PROCEDURE `upgrade_history_schema`()
BEGIN

  /* find out the build version so we know if we need to do some schema altering */

  SET @need_alters = 0 ;

  SELECT build_info.version
  INTO @current_version
  FROM build_info
  WHERE dri_time IN (SELECT max(dri_time) FROM build_info);

  SELECT CASE WHEN (@current_version LIKE '7.0%') OR (@current_version LIKE '7.10%')
                   THEN 1
                   ELSE 0
           END
  INTO @need_alters;

  /* only do this if version is < 7.11 !!! */
  IF @need_alters = 1 THEN

      SELECT CONCAT('Schema is at version ', @current_version, '. Updating.. (This
may take some time)') AS "Message";

      /* drop constraint, if it exists*/
      IF EXISTS (SELECT NULL FROM information_schema.TABLE_CONSTRAINTS
                 WHERE CONSTRAINT_SCHEMA = DATABASE() AND CONSTRAINT_NAME =
'fk_hist_rstep2parent') THEN

        ALTER TABLE runstep_history DROP FOREIGN KEY fk_hist_rstep2parent;
      END IF;

      /** create index if not there already */
      IF EXISTS (SELECT NULL FROM information_schema.statistics
                 WHERE INDEX_SCHEMA = DATABASE() AND index_name =
'idx_hist_prop_runhist_id') THEN

          ALTER TABLE property_history
                DROP INDEX idx_hist_prop_runhist_id;

      END IF;

      CREATE INDEX idx_hist_prop_runhist_id
             ON property_history(run_hist_id);

      /* replace some of the foreign keys generated by hibernate
        with the same foreign keys, but with DELETE CASCADE */
      IF EXISTS (SELECT NULL FROM information_schema.TABLE_CONSTRAINTS
                 WHERE CONSTRAINT_SCHEMA = DATABASE() AND CONSTRAINT_NAME =
'fk_hist_rstep2run') THEN
         ALTER TABLE runstep_history
               DROP FOREIGN KEY fk_hist_rstep2run;
      END IF;

      ALTER TABLE runstep_history
            ADD CONSTRAINT fk_hist_rstep2run
            FOREIGN KEY (run_history_id)
            REFERENCES run_history(oid)
            ON DELETE CASCADE;

      IF EXISTS (SELECT NULL FROM information_schema.TABLE_CONSTRAINTS
                 WHERE CONSTRAINT_SCHEMA = DATABASE() AND CONSTRAINT_NAME =
'fk_hist_prop2rstep') THEN

         ALTER TABLE property_history
               DROP FOREIGN KEY fk_hist_prop2rstep ;
      END IF;

      ALTER TABLE property_history
            ADD CONSTRAINT fk_hist_prop2rstep
            FOREIGN KEY (runstep_hist_id)
            REFERENCES runstep_history(oid)
            ON DELETE CASCADE;
  ELSE
    SELECT CONCAT('Schema is at version ' , @current_version, '. No update is
required.') as "Message";
```

```
  END IF;
END $$

DELIMITER ;
```

The mysql_oo_upgrade_history_schema_call.sql script

You can use the following script to call the above stored procedure. We recommend that you use the text copy of this script contained in the file **mysql_oo_upgrade_history_schema_call.sql** instead of copying the code below, which has line breaks to make reading easier.

```
/* mysql_oo_upgrade_history_schema_call.sql
 *
 * example script to run upgrade_history_schema
 */

/* there are no options to this procedure */

call upgrade_history_schema();
```

To run this script

- Use the **mysql** utility as follows:

```
mysql -u database_user -p=password database_name <
      mysql_oo_upgrade_history_schema_call.sql
```

Appendix C: Example cleanup stored procedure

This appendix contains the following examples:

- A stored procedure named **mysql_oo_prune_run_history.sql.**
- A script to call the procedure named **mysql_oo_prune_run_history_call.sql.**

The mysql_oo_prune_run_history.sql stored procedure

The following stored procedure does the actual pruning from the database. Before you use this procedure, review *Appendix E: Performance implications* to choose suitable parameters for your system.

We recommend that you use the text copy of this example contained in the file **mysql_oo_prune_run_history.sql** instead of copying the code below, which has line breaks to make reading easier.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS dharma.prune_oo_data $$
CREATE PROCEDURE dharma.`prune_oo_data`(hoursToKeep int -- default 2160
                              , prune_batch_size int -- default 1000
                        , prune_run_history varchar(5) -- default 'false'
                              , prune_dashboards varchar(5) -- default
'true'
                              , verbose int -- default 1
                                )
    MODIFIES SQL DATA
BEGIN

  declare lastRunDate datetime;
  declare deleteOlderThan datetime;

  declare deleteFromIndex int;
  declare deleteToIndex int;
  declare deleteRowCount int;
  declare lastPruneTableIndex int;

  SELECT max(start_time)
  INTO lastRunDate
  FROM run_history;


  IF verbose > 0 THEN
    SELECT CONCAT('Last entry in the run_history table occurred on ',

            lastRunDate) AS "INFO";
  END IF;
  SET hoursToKeep = hoursToKeep * -1;
  SET deleteOlderThan = TIMESTAMPADD(HOUR, hoursToKeep, lastRunDate);


  IF verbose > 0 THEN
    SELECT CONCAT('Deleting entries older than ', deleteOlderThan) AS
"INFO";
  END IF;

  -- drop the temp table just in case it's still around from a failed run
  DROP TEMPORARY TABLE IF EXISTS oo_prune_table;
  CREATE TEMPORARY TABLE oo_prune_table
         (
            `oid` bigint(20) NOT NULL AUTO_INCREMENT,
            `run_hist_id` bigint(20) NOT NULL,
            `execution_state` int(11) DEFAULT NULL,
            `flow_uuid` varchar(255) DEFAULT NULL,
            `flow_version` bigint(20) DEFAULT NULL,
            `run_time_millis` bigint(20) DEFAULT NULL,
            PRIMARY KEY (`oid`),
            KEY `idx_oo_prune_table_run_hist_id` (`run_hist_id`)
         )
         ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;


  -- get the info for the records to delete, making sure not to remove
  -- anything that's still in the run table.
  INSERT INTO oo_prune_table(run_hist_id,
                                             execution_state,
                                             flow_uuid,
                                             flow_version,
                                             run_time_millis)
  SELECT      oid,
                    execution_state,
                    flow_uuid,
                    flow_version,
                    CAST(run_time_millis as UNSIGNED)
  FROM        run_history AS h
  WHERE       (  start_time < deleteOlderThan
                  AND
                 oid NOT IN (SELECT history_id FROM run)
              )
  ORDER BY h.oid ASC;
  -- get the first and last indexes from oo_prune_table
  SELECT count(*), min(oid), max(oid)
  INTO deleteRowCount, deleteFromIndex, lastPruneTableIndex
  FROM oo_prune_table;



  -- loop through oo_prune_table, stepping by batch_size, and delete the
data
  IF verbose > 0 THEN
    SELECT CONCAT('Pruning information for ', deleteRowCount,
            ' flow runs from the database. This may take a while...') AS
"INFO";
  END IF;

  WHILE deleteFromIndex <= lastPruneTableIndex DO

    -- calculate the end of the delete range
    SET deleteToIndex = deleteFromIndex + prune_batch_size;

    -- on the off chance that a batch ends on the end of the batch table,
    -- we would end up in an infinite loop without this check.
    IF deleteToIndex = deleteFromIndex THEN
      SET deleteFromIndex = deleteFromIndex + 1;
    END IF;



    IF verbose > 1 THEN
      SET @msg = CONCAT('Deleting chunk: ',
                        deleteFromIndex, ' to ',
                        deleteToIndex);
      SELECT @msg as "INFO";
    END IF;

    START TRANSACTION;

            -- delete dashboard data from log_record if requested
            IF (LOWER(prune_dashboards) = 'true') THEN

                  IF (verbose > 1) THEN
                    SELECT 'Deleting dashboard data...' AS "INFO";
                  END IF;
                  DELETE l
                  FROM log_record l
                    INNER JOIN oo_prune_table p
                                ON ((p.oid BETWEEN deleteFromIndex AND
deleteToIndex)
                                AND (l.run_hist_id = p.run_hist_id));

                  ELSE
                  -- notify the user that we're not deleting dashboards
                  IF (verbose > 1) THEN
                    SELECT 'Not deleting dashboard data...' AS "INFO";
                  END IF;
            END IF;

            -- check to see if we want to delete rows from run_history
            IF (LOWER(prune_run_history) = 'true') THEN

                  -- delete all data from run_history table
                  -- this requires recalculation of flow_metrics as well
                  IF (verbose > 1) THEN
                        SELECT 'Deleting run history data...' AS "INFO";
                  END IF;

                  -- delete rows from run_history
                  DELETE r
                  FROM run_history AS r
                  INNER JOIN oo_prune_table as p
                                    ON
                                          p.oid >= deleteFromIndex AND
                                          p.oid < deleteToIndex    AND
                                          r.oid = p.run_hist_id;



                  IF (verbose > 1) THEN
                        SELECT 'Updating flow metrics...' AS "INFO";
                        SELECT 'BEFORE:' AS "INFO";
                        SELECT * FROM flow_metrics;

                  END IF;

                  -- now recalculate the totals for flow_metrics
                  -- (this only needs to be done if we delete from
run_history)
                  UPDATE flow_metrics AS f
                        INNER JOIN (SELECT flow_uuid,
                                          flow_version,
                                          sum(if(execution_state = 0, 1, 0))
                                                            AS
diagnosedCount,
                                          sum(if(execution_state = 1, 1, 0))
                                                            AS
resolvedCount,
                                          sum(if(execution_state = 2, 1, 0))
                                                            AS
noActionCount,
                                          sum(IF(execution_state = 3, 1, 0))
                                                            AS errorCount,
                                          sum(IF(execution_state =
2147483647, 1, 0))
                                                            AS failedCount,
                                          sum(run_time_millis) AS
cumulativeTime
                                    FROM oo_prune_table
                                    WHERE oid BETWEEN deleteFromIndex AND
deleteToIndex
                                    GROUP BY flow_uuid, flow_version
                              ) AS d
                        ON f.flow_uuid = d.flow_uuid AND f.flow_version =
d.flow_version
                  SET f.diagnosed_count = f.diagnosed_count -
d.diagnosedCount,
                              f.resolved_count = f.resolved_count -
d.resolvedCount,
                              f.failed_count = f.failed_count -
d.failedCount,
                              f.no_action_count = f.no_action_count -
d.noActionCount,
                              f.resolved_count = f.resolved_count -
d.resolvedCount,
                              f.cumulative_time = f.cumulative_time -
d.cumulativeTime,
                              f.dlm_time = NOW();

                  -- now delete the metrics for those flows that are
                  -- left with 0 counts across the board
                  DELETE FROM flow_metrics
                  WHERE diagnosed_count = 0
                        AND failed_count = 0
                        AND no_action_count = 0
                        AND resolved_count = 0
                        AND error_count = 0
                        AND EXISTS (SELECT 1 FROM oo_prune_table p
                                                            WHERE oid
BETWEEN deleteFromIndex
```

```
        AND deleteToIndex
                                                                  AND
flow_uuid = p.flow_uuid);


                  IF (verbose > 1) THEN
                        SELECT 'AFTER:' AS "INFO";
                        SELECT * FROM flow_metrics;

                  END IF;

            ELSE
                  -- we are not deleting from run_history.
                  -- we just need to delete rows from runstep_history.
                  IF (verbose > 1) THEN
                        SELECT 'Deleting step details' AS "INFO";
                  END IF;

                  DELETE r
                  FROM runstep_history AS r
                  INNER JOIN oo_prune_table as p
                                    ON
                                          (p.oid BETWEEN deleteFromIndex AND
deleteToIndex)
                                          AND
                                          (r.run_history_id =
p.run_hist_id);

            END IF;

    COMMIT;

    -- move the start index to start one after the last index deleted.
    SET deleteFromIndex = deleteToIndex;

  END WHILE;

  IF verbose > 0 THEN
    SELECT 'Pruning complete.' AS "INFO";
  END IF;

  -- drop the temp table to free up resources.
  DROP TEMPORARY TABLE oo_prune_table;

END $$
```

```
DELIMITER ;
```

The mysql_oo_prune_run_history_call.sql script

You can use the following script to call the above stored procedure. We recommend that you use the text copy of this script contained in the file **mysql_oo_prune_run_history_call.sql** instead of copying the code below, which has line breaks to make reading easier.

```
/*
* sample script to call prune_oo_data
*
*   USAGE EXAMPLE:
*       mysql -u database_user database < mysql_oo_execute_prune_run_history.sql
*/
/*******************************************************************************
** modify parameters below to suit your needs.
**
** See "Appendix E: Performance Implications" of the documentation
** for guidlines
*******************************************************************************/

/* The number of hours to keep in run_history. Anything older than this many
   hours will be removed from the database.
*/
SET @keep_this_many_hours = 336; -- keep 2 weeks’ worth of data

/* batch size. deletes will be committed to the database for this many rows */
SET @batch_size = 1000;

/* prune run history. If set to 'true', records will be removed from the
 * run_history table. If set to false, the default value, records will no
 * be removed from the run_history table, and data will only be removed
 * from the runstep_history table.
 * Please see "About the OO 7.60 Run schema and tables" in the
 * documentation for further details. And be sure to understand all
 * implications before setting this to true
 */
SET @prune_run_history = 'false'
```

```
/* prune dashboards. If set to 'true', information will be removed from the
 * log_record table. See "About the OO 7.60 Run schema and tables" in the
 * documentation for further details.
*/
SET @prune_dashboards = 'true';

/* verbosity. Higher numbers will produce more detailed output. 2 is the
 * highest level at the moment
*/
SET @verbosity = 2;

/***************************************************************************/


call prune_oo_data(@keep_this_many_hours, @batch_size, @prune_dashboards,
@verbosity);
```

Appendix D: Example scheduling scripts

The preferred method to schedule a pruning job is to use your operating system's scheduling facility. In a UNIX environment, you can place a cron script like the one shown below under /etc/cron.daily or use it in a custom schedule as desired, as shown in the following script. In a Windows system, you can achieve similar results using Microsoft Windows Scheduler and a .bat file modeled after the following script.

We recommend that you use the text copy of this script contained in the file **mysql_oo_prune_job.sh** instead of copying the code below, which has line breaks to make reading easier.

See *Appendix E: Performance implications* for performance considerations and make sure that the file **mysql_oo_prune_run_history_call.sql** has been tailored to your needs before running this script. For more information on setting parameters in **mysql_oo_prune_run_history_call.sql,** see *Appendix C: Example cleanup stored procedure*.

```
#!/bin/sh

# example shell script to call prune_oo_data.
# edit the values below to match your system configuration

# enter your database information below
#   NOTE: entering passwords here might be a security issue. please
#                                be sure you understand the implication before you do
so.
db_user="root"
db_password="roots_password"
db_name="oo_database_name"

# change this to the location of your mysql scripts.
script_dir="/your_script_location"



cd $script_dir

/usr/bin/mysql -u $db_user --password=$db_password $db_name <
mysql_oo_prune_run_history_call.sql > hp_oo_prune_log_`/bin/date +"%Y-%m-%d"` 2>&1
```

Appendix E: Performance implications

Here are some recommendations for using the pruning code:

- Choose a pruning set size that is appropriate to your particular situation. This is important for maintaining the well being of your OO system. The number of hours retained should be calculated so that the pruning stored procedure deletes small amounts of history while allowing Central to make progress in running flows.
- The stored procedure uses a temporary table which is usually allocated in memory, but should be placed on disk if there is not enough memory for it. Make sure there is enough free space for this temporary table.
- In general, it is better to run the pruning procedure more often with small batches, than less frequently with larger batches. This helps both Central and MySQL's throughput, as the pruning jobs can be interleaved with normal processing jobs.
- Although beyond the scope of this document, note that proper allocation of disk space is important when considering the performance of the database. Having separate physical drives for the database file and the transaction log (separate from the operating system) is a good start.