HP Operations Orchestration Software

Software Version: 7.51

Purging OO Run Histories from Oracle Databases

Document Release Date: August 2009

Software Release Date: August 2009

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 Notices

Trademark Notices

All marks mentioned in this document are the property of their respective owners.

Finding or updating documentation on the Web

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.

Where to find Help, tutorials, and more

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.

Support

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.

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:

5. Descriptions of the tables involved in storing historical run data in the *OO database*.
6. The procedure for *physically deleting old run history data*.
7. *Appendices* that contain information such as a diagram of the tables in the 7.50 **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 **Oracle_Run_History_Purge.zip** (which also contains this document). The code files are:

   - To call the pruning process—
     oracle_oo_prune_run_history_call.sql
   - For *Appendix B: Upgrading older schemas*—
     oracle_oo_upgrade_history_schema.sql
   - For *Appendix C: Example cleanup stored procedure*—
     oracle_oo_prune_run_history_temp_tables.sql
     oracle_oo_prune_run_history.sql
     oracle_oo_prune_run_history_pkgb.sql
   - For *Appendix D: Example scheduling script*—
     oracle_oo_schedule_prune_run_history.sql

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

Required knowledge

Oracle 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 these tables. The tables 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*.

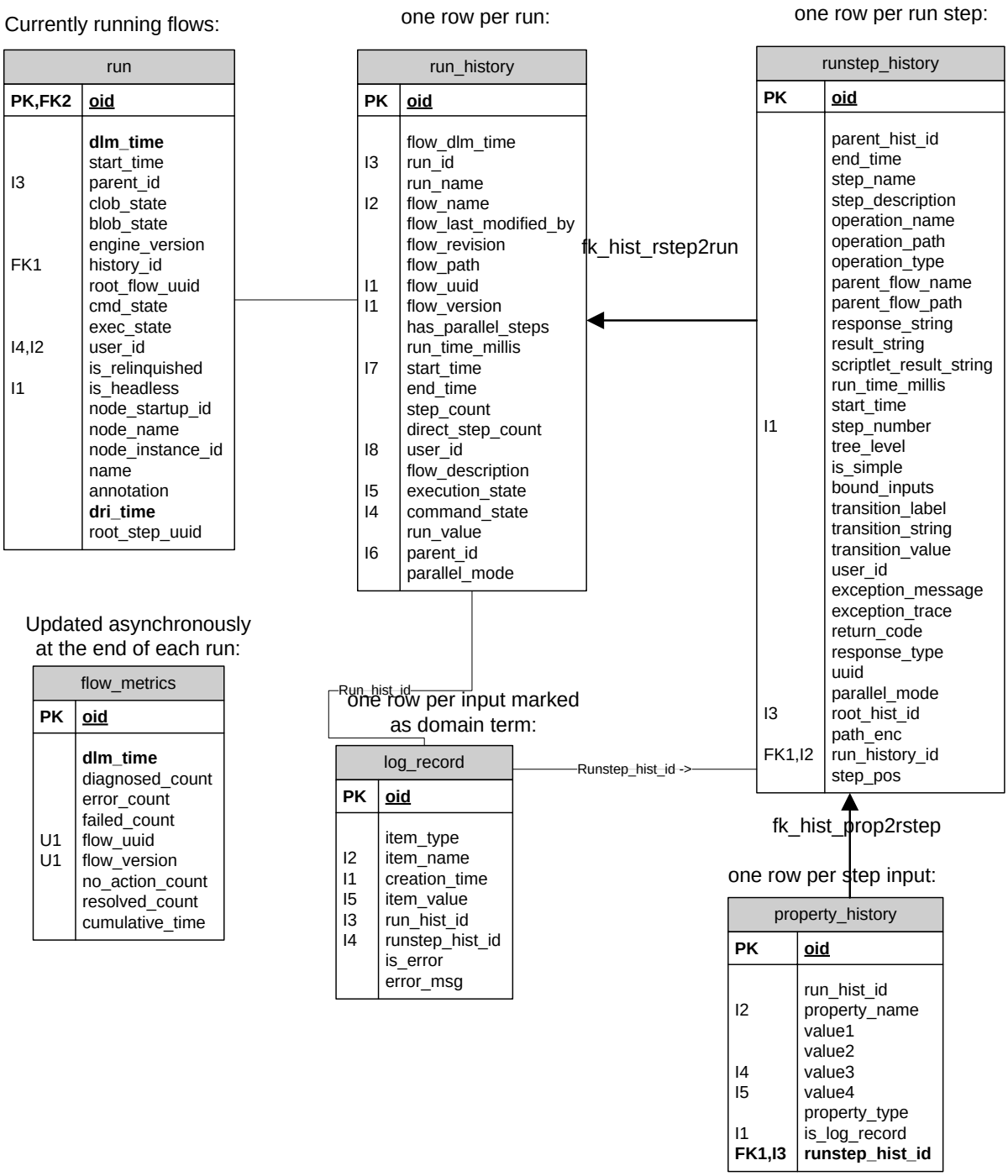
You can schedule the cleanup job, as explained in *Appendix D: Example scheduling script*.

Appendices

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

Appendix A: Tables diagram

7.50 Run Schema



Appendix B: Upgrading older schemas

The following script 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 script contained in the file **oracle_oo_upgrade_history_schema.sql** instead of copying the code below, which has line breaks to make reading easier.

```
set serveroutput on 20000
/
declare
  need_alters number := 0;
begin
  dbms_output.enable;

  select 1 into need_alters
   from build_info
  where dri_time = (select max(dri_time) from build_info)
   and ((version like '7.0%') or (version like '7.10%'));

  if (need_alters > 0) then
    begin
      dbms_output.put_line('Upgrade needed, '||
            'preparing schema for pruning...');

      begin
        execute immediate 'alter table runstep_history '||
            'drop constraint FK_HIST_RSTEP2PARENT';
      exception when others then
        if (SQLCODE = -2443) then
          dbms_output.put_line('ignoring exception, '||
            'constraint FK_HIST_RSTEP2PARENT does not exist');
          null;
        else
          raise;
        end if;
      end;

      begin
        execute immediate 'create index idx_hist_prop_runhist_id'||
            ' on property_history(run_hist_id)';
      exception when others then
        if (SQLCODE = -955) then
          dbms_output.put_line('ignoring exception, '||
            'index idx_hist_prop_runhist_id exists');
          null;
        else
```

```
          raise;
        end if;
      end;

      execute immediate 'alter table runstep_history '||
            'drop constraint FK_HIST_RSTEP2RUN';

      execute immediate 'alter table runstep_history ' ||
        'add constraint fk_hist_rstep2run '||
        'foreign key (run_history_id) '||
        'references run_history(oid) '||
        'on delete cascade';

      execute immediate 'alter table property_history '||
            'drop constraint FK_HIST_PROP2RSTEP';

      execute immediate 'alter table property_history '||
        'add constraint fk_hist_prop2rstep '||
        'foreign key (runstep_hist_id) '||
        'references runstep_history(oid) '||
        'on delete cascade';

      dbms_output.put_line('Upgrade done.');

    exception when others then
      dbms_output.put_line('Upgrade failed: '||SQLCODE||', '||SQLERRM);
    end;
  else
    dbms_output.put_line('Upgrade not needed!');
  end if;

end;
/
```

Appendix C: Example cleanup stored procedure

The following stored procedure illustrates the points made in the deletion algorithm. It consists of three components.

- A temporary table
- A package header for the stored procedure
- A package body for the stored procedure

This appendix includes examples of the above components. We recommend that you use the text copies of these examples contained in the following files instead of copying the code below, which has line breaks to make reading easier.

- **oracle_oo_prune_run_history_temp_tables.sql**
- **oracle_oo_prune_run_history_pkg.sql**
- **oracle_oo_prune_run_history_pkgb.sql**

Example temporary table for the stored procedure

```
drop table oo_prune_table;
/
create global temporary table OO_PRUNE_TABLE(
      OID NUMBER(19,0) NOT NULL ENABLE,
         RUN_HISTORY_ID NUMBER(19,0),
      FLOW_UUID VARCHAR2(255),
         FLOW_VERSION NUMBER(19,0),
      EXECUTION_STATE NUMBER(10,0),
      RUN_TIME_MILLIS NUMBER(19,0)
) ON COMMIT PRESERVE ROWS;
/
```

Example package header for the stored procedure

```
create or replace package hp_oo_prune
  authid current_user
is
  procedure prune_run_history( keep_this_many_hours in number default 2160
      , prune_batch_size in number default 1000
      , prune_run_history in varchar2 default 'false'
      , prune_dashboard in varchar2 default 'true'
      , verbose in number default 1
  );

end hp_oo_prune;
```

Example package body for the stored procedure

```
create or replace
package body hp_oo_prune
is

-- private
PROCEDURE update_flow_metrics(
   verbose in varchar,
   v_delete_start_row in number,
   prune_batch_size in number)
IS
BEGIN
  if (verbose > 1) then
     dbms_output.put_line('Updating flow metrics...');
  end if;

  MERGE INTO flow_metrics fm
     USING (
       SELECT flow_uuid,
              flow_version,
              sum(case when execution_state = 0 then 1 else 0 end)
                as diagnosedCount,
              sum(case when execution_state = 1 then 1 else 0 end)
                as resolvedCount,
              sum(case when execution_state = 2 then 1 else 0 end)
                as noActionCount,
              sum(case when execution_state = 3 then 1 else 0 end)
                as errorCount,
              sum(case when execution_state = 2147483647 then 1 else 0 end)
                as failedCount,
              sum(run_time_millis) as cumulativeTime
       FROM oo_prune_table
       WHERE oid >= v_delete_start_row
         AND
             oid < v_delete_start_row + prune_batch_size
       GROUP BY flow_uuid, flow_version
     ) d
     ON (fm.flow_uuid = d.flow_uuid and fm.flow_version = d.flow_version)
     WHEN MATCHED THEN
        UPDATE SET fm.diagnosed_count = fm.diagnosed_count -
d.diagnosedCount,
                   fm.resolved_count = fm.resolved_count - d.resolvedCount,
                   fm.no_action_count = fm.no_action_count -
d.noActionCount,
```

```
                  fm.error_count = fm.error_count - d.errorCount,
                  fm.failed_count = fm.failed_count - d.failedCount,
                  fm.cumulative_time = fm.cumulative_time -
d.cumulativeTime;

  -- delete the metrics for those flows that are left with 0's on all
counts.
  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 flow_uuid = p.flow_uuid
            AND
                oid >= v_delete_start_row
            AND
                oid < v_delete_start_row + prune_batch_size);

END update_flow_metrics;

-- private
PROCEDURE delete_batch(
  prune_batch_size in number,
  v_delete_start_row in number,
  prune_run_history in varchar2,
  prune_dashboard in varchar2,
  verbose in number)
IS
      v_batch_size number;
      v_min_oid number;
      v_max_oid number;
      v_delete_stop_row number;
BEGIN

     v_delete_stop_row := v_delete_start_row + prune_batch_size;

      select count(*), min(oid), max(oid)
        into v_batch_size, v_min_oid, v_max_oid
        FROM oo_prune_table
        WHERE oid >= v_delete_start_row
           AND
              oid < v_delete_start_row + prune_batch_size;
```

```
      if (v_batch_size = 0) then
        commit;
        return;
      end if;

      if verbose > 0 then
        DBMS_OUTPUT.PUT_LINE('Deleting next batch of size '
          || v_batch_size || ' from run_history ');
      end if;

      --PRUNE THE DASHBOARD INFO, IF REQUESTED
      IF prune_dashboard = 'true' THEN
         IF verbose > 1 THEN
            DBMS_OUTPUT.put_line('Deleting dashboard data...');
         END IF;

         DELETE
         FROM log_record l
         WHERE l.run_hist_id IN (SELECT run_history_id
                                 FROM oo_prune_table
                                 WHERE oid >= v_delete_start_row
                                    AND
                                       oid < v_delete_stop_row);
      END IF;

      IF prune_run_history = 'true' THEN

         -- NOW DELETE THE BATCH FROM run_history
         if (verbose > 1) then
           dbms_output.put_line('Deleting '||v_batch_size
             ||' run histories (min_oid='||v_min_oid
             ||', max_oid='||v_max_oid||')');
         end if;

         DELETE
         FROM run_history r
         WHERE r.oid IN (SELECT run_history_id
                         FROM oo_prune_table
                         WHERE oid >= v_delete_start_row
                            AND
                               oid < v_delete_stop_row);

         -- CALCULATE THE LOST FLOW_METRIC COUNTS AND CUMULATIVE_TIME,
         -- AND UPDATE FLOW_METRICS
         update_flow_metrics(verbose, v_delete_start_row, prune_batch_size);
```

```
      ELSE

         DELETE
         FROM runstep_history r
         WHERE r.run_history_id IN (SELECT run_history_id
                         FROM oo_prune_table
                         WHERE oid >= v_delete_start_row
                            AND
                               oid < v_delete_stop_row);

      END IF;

      COMMIT;

END;

-- public
PROCEDURE prune_run_history( keep_this_many_hours in number default 2160
  , prune_batch_size in number default 1000
  , prune_run_history in varchar2 default 'false'
  , prune_dashboard in varchar2 default 'true'
  , verbose in number default 1
  )
IS
  v_ts_last_run TIMESTAMP(6);
  v_ts_delete_older_than run_history.start_time%TYPE;
  v_total_rows_to_del run_history.oid%TYPE;
  v_oo_prune_table_size PLS_INTEGER;
  v_delete_start_row PLS_INTEGER;
  v_delete_rows_left PLS_INTEGER;
BEGIN

  SELECT MAX(start_time)
      INTO v_ts_last_run
      FROM run_history;

   v_ts_delete_older_than := v_ts_last_run - keep_this_many_hours/24;

   if (verbose > 0) then
    dbms_output.put_line('Preparing pruning table. '||
      'Will delete histories where start_time <= '||
v_ts_delete_older_than);
   end if;
```

```
   INSERT INTO oo_prune_table
          SELECT rownum, oid, flow_uuid, flow_version, execution_state,
                 cast(run_time_millis as number)
          FROM (SELECT oid, flow_uuid, flow_version, execution_state,
run_time_millis
                FROM run_history
                WHERE (start_time < v_ts_delete_older_than)
                     AND
                     oid NOT IN (SELECT history_id FROM run)
                ORDER BY oid
                );


   select count(*)
     into v_oo_prune_table_size
     from oo_prune_table;

   if (verbose > 0) then
    DBMS_OUTPUT.PUT_LINE('Total rows to delete: ' || v_oo_prune_table_size);
   end if;

   select min(oid)
      into v_delete_start_row
      from oo_prune_table;

   WHILE v_delete_start_row < v_oo_prune_table_size LOOP

      -- this is an autonomous transaction
      delete_batch(prune_batch_size, v_delete_start_row, prune_run_history,
                                                         prune_dashboard,
verbose);

      -- assuming everything went ok with the delete, we can calculate
      -- the rows left to delete
      v_delete_rows_left := v_oo_prune_table_size - v_delete_start_row
                            - prune_batch_size + 1;

      if (v_delete_rows_left < 0) then
         v_delete_rows_left := 0;
      end if;

      if (verbose > 0) then
        dbms_output.put_line(''||v_delete_rows_left
          ||' histories left to delete...');
      end if;
```

```
      v_delete_start_row := v_delete_start_row + prune_batch_size;

   END LOOP;

   DBMS_OUTPUT.PUT_LINE('rows deleted: ' || SQL%ROWCOUNT);

   EXECUTE IMMEDIATE 'TRUNCATE TABLE OO_PRUNE_TABLE';

END prune_run_history;

end hp_oo_prune;


/

show errors;
/
```

Appendix D: Example scheduling script

The following script creates a schedule and job to run the database pruning script on a recurring basis. Values should be selected for all parameters in the user configuration section for your particular needs. We recommend that you use the text copy of this script contained in the file **oracle_oo_schedule_prune_run_history.sql** instead of copying the code below, which has line breaks to make reading easier.

See *Appendix E: Performance implications* for performance considerations, which should be taken into account when setting these parameters. As noted in the comments, you must run this script as an OO user who has CREATE JOB system rights.

```
/*
*   this script will create a job to run prune_run_history on a recurring
*   basis. it must be run by DHARMA_USER, and DHARMA_USER must be granted
*   the right to create jobs:
*
*        GRANT CREATE JOB TO DHARMA_USER
*/

DECLARE

   v_prune_dashboard VARCHAR2(5);
   v_prune_run_history VARCHAR(5);
   v_prune_batch_size NUMBER;
   v_keep_this_many_hours NUMBER;
   v_verbose NUMBER;
   v_repeat_interval VARCHAR2(255);

BEGIN

      ------------------------------------------------------------------------
      -- CHANGE VALUES BELOW TO SUIT YOUR NEEDS
      ------------------------------------------------------------------------

      /* batch size. deletes will be commited to the database for this many rows */
      v_prune_batch_size := 1000;

      /* The number of hours to keep in run_history. Anything older than this many
         hours will be removed from the database.
      */
      v_keep_this_many_hours := 2;
```

```
   /* 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 7.50 Run schema and tables" in the
       * documentation for further details. And be sure to understand all
       * implications before setting this to true
       */
   v_prune_run_history := 'flase';

      /* prune dashboards. If set to 'true', information will be removed from the
       * log_record table. See "About the OO 7.50 Run schema and tables" in the
       * documentation for further details.
      */
      v_prune_dashboard := 'false';

   -- verbosity level. 0=terse, 1=normal, 2=verbose
   v_verbose := 2;

   -- v_repeat_interval defines when the job will be run
   --    FREQ is the minimum amount of time between runs (DAILY = once a day,
   --                                                     WEEKLY = once a week,
   --                                                     etc...
   --    INTERVAL is the number of periods of FREQ between runs
   --         i.e. if FREQ=DAILY and INTERVAL=2, then it runs every 2 days
   --    BYHOUR, BYMINUTE, and BYSECOND define the time at which the job is run
   --
   -- so the default below runs the job every day at 18:00
   v_repeat_interval := 'FREQ=DAILY;INTERVAL=1;BYHOUR=18;BYMINUTE=0;BYSECOND=0';

   ------------------------------------------------------------------------------
   -- END USER CONFIGURABLE PARAMETERS
   ------------------------------------------------------------------------------


   -- drop the program if it exists, ignore the exception if it doesn't
   BEGIN
      dbms_scheduler.drop_program('PRUNERUNHIST_PRG', TRUE);

   EXCEPTION
      WHEN OTHERS THEN
         NULL;
   END;
   -- create program
   dbms_scheduler.create_program(
```

```
      program_name=>'PRUNERUNHIST_PRG',
      program_action=>'HP_OO_PRUNE.PRUNE_RUN_HISTORY',
      program_type=>'STORED_PROCEDURE',
      number_of_arguments=>4,
      comments=>'call HP_OO_PRUNE.PRUNE_RUN_HISTORY',
      enabled=>FALSE);
-- add the four attributes
dbms_scheduler.define_program_argument(
      program_name => 'PRUNERUNHIST_PRG',
      argument_name => 'KEEP_THIS_MANY_HOURS',
      argument_position => 1,
      argument_type => 'NUMBER',
      default_value => v_keep_this_many_hours);
dbms_scheduler.define_program_argument(
      program_name => 'PRUNERUNHIST_PRG',
      argument_name => 'PRUNE_BATCH_SIZE',
      argument_position => 2,
      argument_type => 'NUMBER',
      default_value => v_prune_batch_size);
dbms_scheduler.define_program_argument(
      program_name => 'PRUNERUNHIST_PRG',
      argument_name => 'PRUNE_RUN_HISTORY',
      argument_position => 3,
      argument_type => 'VARCHAR2',
      default_value => v_prune_run_history);
dbms_scheduler.define_program_argument(
      program_name => 'PRUNERUNHIST_PRG',
      argument_name => 'PRUNE_DASHBOARD',
      argument_position => 4,
      argument_type => 'VARCHAR2',
      default_value => v_prune_dashboard);
dbms_scheduler.define_program_argument(
      program_name => 'PRUNERUNHIST_PRG',
      argument_name => 'VERBOSE',
      argument_position => 5,
      argument_type => 'NUMBER',
      default_value => v_verbose);

-- now that all the arguments are defined, we should be able to enable the
-- program
dbms_scheduler.enable('PRUNERUNHIST_PRG');



-- drop the schedule if it exists, ignore the exception if it doesn't
```

```
   BEGIN
      dbms_scheduler.drop_schedule('PRUNERUNHIST_SCHEDULE', TRUE);
   EXCEPTION
      WHEN OTHERS THEN
         NULL;
   END;

   dbms_scheduler.create_schedule(
         repeat_interval =>
            v_repeat_interval,
         comments =>
            'Schedule for periodic pruning of run_history',
         schedule_name => 'PRUNERUNHIST_SCHEDULE');


   -- drop the job if it exists, ignore the exception if it doesn't
   BEGIN
      dbms_scheduler.drop_job('PRUNERUNHIST_JOB', TRUE);

   EXCEPTION
      WHEN OTHERS THEN
         NULL;
   END;

   dbms_scheduler.create_job(
         job_name => 'PRUNERUNHIST_JOB',
         program_name => 'PRUNERUNHIST_PRG',
         schedule_name => 'PRUNERUNHIST_SCHEDULE',
         job_class => 'DEFAULT_JOB_CLASS',
         comments => 'periodically prune run_history',
         auto_drop => FALSE,
         enabled => TRUE);

END;
```

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 global temporary tables, allocated out of the temporary tablespace. The main pruning table contains IDs for the whole set size, not just for one individual batch. Make sure that there is enough space for it.
- 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 Oracle'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.