Oracle BPEL 10g Purging Strategies

An Oracle White Paper
August 2010

Table of Contents

Oracle 10g BPEL purging strategies

Introduction

In many customer Oracle 10g Fusion Middleware (FMW) SOA production environments, SOA administrators have asked Oracle about the optimal strategy to purge their dehydration store DB. With growing data volumes, production load and business transactions; managing the dehydration store data set is quickly becoming a hot topic. Choosing a correct purging strategy that maps to an application's performance goals and meets the system availability requirements is not an easy task. The problem is that each application is different, and applying a "one size fits all" approach may not meet business requirements.

The objective of this white paper is to provide the tools and an approach required to assist SOA administrators in choosing an optimal purging strategy and tool sets to best manage dehydration store requirements. The goal is to provide a methodology to profile the dehydration store usage (based on # incoming flow of BPEL processes, # outflow of BPEL processes, etc) and then use the current 10g purging toolkit to create a strategy that best fits the dehydration store usage profile. The benefits of conducting such an exercise are:

- The profiling will provide a way to forecast DB disk usage and further disk space requirement
- Assess the impact of future increase in production load on the system and dehydration store
- It provides better timelines on system down requirements and maintenance windows
- Improved forecasting for system hardware requirements
- Increased purge rates and an optimally tuned system
- Overall better system and application performance

The intended audience of this white paper are 10g FMW SOA administrators, database administrators (DBA), system and solution architects and BPEL developers.

The document begins by reviewing the current 10g purging tools and scripts available today as part of the product. It is followed by an approach to create a formula of inflow and outflow of BPEL processes, to create a DB profile (small-medium-large datasets) to capture BPEL dehydration store space requirements. The white paper concludes by providing multiple strategies to various dehydration store profiles (DSP), while considering retention policies and long running BPEL processes requirements.

Oracle 10g FMW purging solutions and scripts

Currently Oracle 10g FMW supports three types of purging solutions that are used in various production environments:

1. The out of the box SOA purge scripts
2. The Create Table as Select (CTAS) script
3. Using various range-hash partitioning solutions

SOA purge scripts

The Oracle 10g SOA Suite supports multiple flavours of the out of the box purge scripts that are shipped with the product. The main goal of the purge script is to delete rows from the dehydration store schema[1]of completed BPEL processes. The CUBE_INSTANCE table is used as the driving table for the scripts where any instances that are in the complete state (state >= 5) and fall within the date range specified can be deleted. Any instances that are still running will not be deleted even if certain activities of the BPEL processes may have completed (like callback or invoke activities).

1.1 Batch-Looping construct

Oracle FMW SOA 10.1.3.5[2] *(MLR#2, tracking bug 9841044)* provides a new version of the purge script – one that is optimised to run in a loop using a batch construct – called ***single threaded looped purge procedure***[3]. The script accepts a batch size, which is the maximum number of rows to delete before a commit is issued. This approach keeps the redo log segments from growing too large when purge is applied to a large number of rows. Included in the loop is also a maximum run time check that will break the loop after the maximum run time has exceeded. The maximum run time check is a fail-safe that allows administrators an opportunity to schedule these runs either during production hours or during maintenance windows. Another advantage of this script is, it takes minimal system resources and can be used as continuous purge mechanism (with properly selected input parameters). For example this script could be scheduled to run for 15 minutes every hour, controlling the DB growth during business hours i.e. while the system is in production.

The purge script initially selects CIKEY of closed instances and inserts them into a temporary table and then selects corresponding callback/invokes messages and workflow instances in the remaining temporary tables. These four temporary tables are then used as driver tables to delete rows from the actual dehydration schema and they are then truncated at end of each loop. Our internal and client tests have shown that the batching of records saves time on redo logs, DB scans, and overall DML activity. The results are a faster purge, a better estimation of time required and more control over the whole purging cycle.

The following high-level code sample shows how we utilize the looping construct to purge BPEL instances.

WHILE (ci_flag AND sysdate < v_stoptime) LOOP.

[1] Appendix A – BPEL dehydration schema
[2] These scripts have been backported to 10.1.3.4 MLR8 as well. Please use same bug numbers as mentioned here.
[3] Please see note 1110833.1 on Metalink

```
// populate temp_cube_instance table with cikey of closed instances
which are older than retention period.
...
// Populate temp_invoke_message/temp_dlv_message and temp_wf_instance
table with corresponding cikey from earlier step
...
//Join tables with their corresponding temp table and delete batch
size records using rownum

//purge records from xml_document table using temp_xml_document table

//purge records from cube_instance table.

// truncate the temp tables
...
END LOOP;
```

For the XML_DOCUMENT table deletion, the script keeps collecting purgeable DOCKEYS in the TEMP_XML_DOCUMENT table, by joining different related tables (DOCUMENT_DLV_MSG_REF, DOCUMENT_CI_REF, INVOKE_MESSAGE and DLV_MESSAGE). The records are then purged in a single delete statement from XML_DOCUMENT. The CUBE_INSTANCE table is deleted at the very end. This is to ensure that the script does not delete anything from the main driver table (CUBE_INSTANCE) until all other data has been successfully purged without any errors. This avoids unreferenced data due to premature termination of purge due to some error.

The stop time check is performed at the start of each loop to ensure that we do not exceed the given time for purge run. Appendix B provides an example BPEL purge script that uses the loop-batching construct. For the official version of this script please request a patch for bug **9460162**.

1.2 Degree of Parallelism construct

Another variant of the Batch-Looping purge script is to add calls to "dbms_scheduler" in the above purge scripts to run multiple instances of the script for greater performance. The idea is to make the purge script "multi-threaded" [4] to delete data from the dehydration store in faster more performant way. This script is called ***multithreaded looped purge*** - an example of the degree of parallelism (DOP) is shown below:

```
LOOP
      -- exit loop when DOP jobs have been started

      EXIT WHEN p_DOP=v_thread;
      v_jobname := 'ORABPEL_DELETE'||v_thread;

      INSERT INTO job_flow_control (job_thread) values (v_thread);
      COMMIT;

      dbms_scheduler.create_job (v_jobname, 'STORED_PROCEDURE',
      'MULTI_THREADED_LOOPED_PURGE.PURGE_TABLES_JOB',4);
```

[4] Please see note 1110833.1 on Metalink

```
        dbms_scheduler.set_job_argument_value
        (v_jobname,1,to_char(p_DOP));

        dbms_scheduler.set_job_argument_value
        (v_jobname,2,to_char(v_thread));

        dbms_scheduler.set_job_argument_value
        (v_jobname,3,to_char(p_rownum));

        dbms_scheduler.set_job_argument_value
        (v_jobname,4,to_char(p_chunksize));

        dbms_scheduler.enable (v_jobname);
        v_thread := v_thread +1;

END LOOP;
```

In the BPEL PL/SQL code snippet above, the PURGE_TABLES_JOB job is being executed by **p_DOP** threads. The **v_thread** parameter is the number for the thread being scheduled. The loop will start 'v_thread' threads, where each thread will be working on a slice of the table. The rows that comprise the slice are selected based on the mod function on the CIKEY column. Again, the CUBE_INSTANCE (CIKEY) table is used as the driving table for all deletes.

Examples shown below:

```
IF f2_flag THEN
     f2_flag:=false;
     DELETE FROM cube_scope
        WHERE cikey IN (
           SELECT tpic.cikey
           FROM temp_cube_instance tpic
           WHERE mod (tpic.cikey, p_DOP)=p_THREAD)
        AND rownum < p_CHUNKS;
     IF SQL%FOUND THEN
        f2_flag:=true;
        v_deleted := true;
     END IF;
   END IF;

   COMMIT;
```

As per the example above the script is selecting a slice of rows for a particular thread (identified by p_THREAD) by using mod (tpic.cikey, p_DOP)=p_THREAD).

The JOB_FLOW_CONTROL table is used to keep record of how many threads have started and how many threads have finished. It also keeps record of any error messages that any purge thread might have thrown during execution.

A complete example of the above purge script can be found in Appendix B. For official version of this script please refer to bug **9219019**.

Please note that for LOB related tables, the purge script only removes the rows, it does not coalesce the space. Please see Appendix B for examples on how to claim back the disk space.

CTAS script

The CTAS script is a highly performant alternative to the purge script described above[5]. The general approach is to copy the rows in the source table that should be kept (that is, can still being operated on by the runtime engine) over to a clone of the table, after which the source table can be dropped and the clone table renamed to the original source table name. Instead of deleting rows, CTAS will only copy the subset of data that is still required (i.e. BPEL processes are still running or processes that cannot be deleted due to business reasons) in a single transaction. This is a great approach to "trim" the data set and "purge" the tables to create a more manageable schema while claiming disk space and reducing DB overheads like defragmentation and re-indexing.

The benefits of running CTAS include:

- Fast purging of tables (due to reduced number of rows operating over and no logging incurred).
- Implicit rebuild of indexes (as the indexes must be recreated on the clone table after the CTAS operation is complete).
- Implicit defragmentation of LOB tables (as the clone table has been newly constructed).
- Can be used to convert non-partitioned dehydration store into partitioned one as partitioning could be done during CTAS.
- Cleans up any unreferenced data (caused by premature deletion of BPEL processes) and could not be deleted during normal purge script.

However the following points must be taken into consideration when running CTAS:

- Mid-tiers must be shutdown as no rows in the source table can be modified (or new rows inserted) while the copy operation is in progress.
- A backup of the tables must be performed before and after the CTAS script due to the no-logging aspect of the operation.
- The run time for the CTAS script is indeterminate, unlike the traditional purge script (as the CTAS operation is creating a new table, the script can be aborted at any time, with no data impact).
- Testing is a prerequisite to make sure that there is no data loss when running CTAS. Testing is also required to determine a long enough maintenance window to transition the data.
- Sufficient disk space must be present for the copied rows to be inserted into the clone table.
- Referential integrity constraints must be disabled for the session and re-enabled on the target table after the CTAS operation.

There are three phases in the CTAS script:

1. Create the temp tables where the data will be moved to
2. Copy the data over that satisfies the retention criteria i.e. only save data that is required

[5] Please see note 1110833.1 on Metalink

3. Drop the original tables and rename temp tables to the original names and recreate all indexes and constraints

The PL/SQL snippet below illustrates how CTAS statements work:

```
// Create CTAS statement for cube_instance (master table)
//
vstmt := 'create table temp_ctas_cube_instance parallel 16 nologging
storage ( freelists 20 ) as ' ||
        ' (select /*+ parallel(ci,16) */ * from cube_instance ci
where ci.state < 5 OR ' ||
        ' ci.modify_date >= TO_TIMESTAMP(''RETENTION_PERIOD'')  )';

vstmt := REPLACE(vstmt,'RETENTION_PERIOD',p_older_than);

// Execute the CTAS statement
//
execute immediate vstmt;

// Drop the original table, rename the cloned table to the original
//
execute immediate 'drop table cube_instance';
execute immediate 'alter table temp_ctas_cube_instance rename to
cube_instance';

// Recreate the constraints on the new table
//
execute immediate 'alter table cube_instance add constraint ci_pk
primary key( cikey )';

// Execute the CTAS statement for a dependent table
//
execute immediate 'create table temp_ctas_work_item parallel 16
nologging storage ( freelists 20 ) as ' ||
                  '(select /*+ parallel(ci, 16) full(ci) */ wi.* from
work_item wi, cube_instance ci ' ||
                  ' where ci.cikey=wi.cikey )' ;

// Drop the original table, rename the cloned table to the original
//
execute immediate 'drop table work_item';
execute immediate 'alter table temp_ctas_work_item rename to
work_item';

// Recreate the constraints on the new table
//
execute immediate 'alter table work_item add constraint wi_pk primary
key( cikey, node_id, scope_id, count_id )';
execute immediate 'alter table work_item modify ( exception default 0,
exp_flag default 0, ' ||
                    ' idempotent_flag default 0, execution_type default
0 )';
```

To improve performance of the CTAS script we recommend using a hash partition version of the CTAS script.[6] With hash partitioning in place, CTAS can leverage DOP to work on LOB segmented tables and copy the data faster. For official version of this script please refer to bug **9315108**.

Partitioning

DB partitioning using range partitioning and hash partitioning is a supported solution for BPEL dehydration store purging. Partitioning by definition means storing data in multiple tables to reduce bigger data-sets into smaller much manageable data-set. One important requirement that customers must meet prior to configuring their tables for partitioning is to ensure that the database hardware has sufficient resources to handle the demands of Oracle Database Partitioning. If pre-production testing has indicated that the installation will be large, Oracle will expect that the customer will have sized their environment (CPU, memory and disk space) correctly to take advantage of the partitioning features.

With partitioning enabled, administrators are recommended to allow rows within a partition to transition to their completed states, verify that the partition is safe to drop, and then finally drop the partition. Since the data is stored in a much smaller table, the table and all its underlying data can be easily dropped without any impact to the DB or BPEL system. This is by far the most efficient way to delete data from the dehydration store. For most components, range partitions are the logical choice, but range-hash partitions (that is, range partition with a hash sub-partition) are recommended over range partitions for the following reasons:

- Range partition to facilitate drop partition (maintenance).
- Range partition for partition pruning capabilities (performance).
- Hash partition to improve primary key lookups (performance).

Hash sub-partitions are especially helpful for tables with LOB segments as this should assist with reducing High Watermark contentions.

[6] Assumption that proper DB Partitioning licenses have been procured

BPEL DB profiling methodology

The first step on deciding for an optimal purge strategy is to first determine a system's BPEL DB profile. The main steps in deriving the profile are:

1. Measure the inflow of BPEL processes and their impact on the DB data i.e. the # of BPEL processes stored in the DB, how many rows are inserted on a daily basis and the corresponding table space requirement
2. Measure the outflow of BPEL processes i.e. the amount of data purged, rows deleted, and table space reclaimed
3. Determine the retention policy i.e. how long to hold processes in the DB before purging
4. How much disk-space is available

So to put this in a formula:

DB Profile = Inflow of Data + Retention Policy + Amount of available disk-space

Daily Space Requirements = Inflow of data – Outflow of data

We have classified the BPEL DB usage into four profiles: small, medium, large and RetentionOnly. Table 1 describes the profiles:

DB Profile	Description
Small	A system that persists less than 10 GB of data per day, retains less than 100 GB of data[7] and has between 1-1.5 TB of diskspace
Medium	A system that persists between 10 GB-30 GB of data per day, retains between 100 GB-300GB of data[8] and has between 1.5 -5 TB of disk space
Large	A system that persists over 30 GB of data per day, retains over 300 GB of data and has over 5 TB of disk-space (no limitation)
RetentionOnly	A system that has very big retention requirements (>=500 GB) for very long period of times (>21 days) regardless of their data inflow. This system may or may not have disk-space limitations.

Table 1: DB profile descriptions

Conducting a thorough analysis to create a system's DB profile should be one of main points of a DBA and system administrator. The DB profile provides a lot of information about how the dehydration store behaves, provides a very good prognosis of space requirements and allows the DBA and system administrator to choose a DB purging strategy that fits their requirements. At a minimum all SOA/BPEL systems should go through this exercise – the information gained from studying your system can be very useful down the road.

[7] Average retention policy of 5-21 business days

Inflow of data

The first thing to do in assessing a systems DB profile is measure the rate of inflow of data into the dehydration store. A way of measuring the inflow is to capture the following data points over a five day period to produce a daily average rate of inflow:

1. Number of BPEL instances produced daily – either by looking at the amount of daily incoming requests or by querying the CUBE_INSTANCE table to get a count of BPEL instances. The main idea is to understand how many BPEL durable process instances are created daily.
2. How many rows in the DB are populated by the number of BPEL instances produced daily? Although this may not be an easy task for all tables it would be helpful to start with a baseline view (amount of rows before starting), and then conduct a row count on a daily basis to get an estimate of how many rows are populated for the whole BPEL schema.
3. The most important point is to calculate the table space (disk space) requirements – measure how much disk space is used up by the durable BPEL processes on a daily basis. Again start with a baseline view and then measure the daily growth in disk space on the dehydration store.

Once this data has been collected, determine the average inflow rate per day for eg: 15,000 BPEL instances day, map to 20,000 rows and 10 GB of disk space **a day**. The proper way to read this is: **on average 15,000 BPEL processes take up 10 GB of disk space a day** – (for now we will ignore the row count) it is more important to understand how much space is required for daily operations.

The next step is to breakdown the space requirements by tables. Review how much diskspace is occupied by each table to get an assessment of the distribution. 80% of the disk space "should be" occupied by tables that have a LOB column in it namely AUDIT_TRAIL, CUBE_SCOPE and XML_DOCUMENT. This distribution will be specific to systems environment but it will provide invaluable information on how the disk space is distributed across the BPEL dehydration. This information will be very useful in choosing a purge strategy later on.

One thing to note here is that even if a DBA or system administrator does not choose a strategy listed here, the information gathered in this section can still be used to devise a new strategy to best meet an application's purging requirements.

Outflow of data

The second step is to determine how much data is deleted when a purge is run. The task is to measure the amount of data (rows) that are deleted and disk space that is reclaimed after the deletion. Measure this statistic for five days to get an average rate of outflow by capturing the following data points:

1. Number of BPEL processes deleted in every purge. Usually this statistic will be dependent on the retention policy and date range, but its a good idea to get an average number of BPEL processes that are deleted every time.
2. Average number of rows deleted per purge. As stated above start with a baseline and monitor this for every purge across the dehydration store schema to get a number of average rows deleted per purge cycle.
3. Calculate the disk space that is claimed back after running the purge. Start with a baseline number and measure the space claimed over five days to get an average number.

So an example of outflow of rate would be:

A purge cycle deletes 10,000 BPEL instances, 15,000 rows and claims 8 GB **every cycle** or if the purge is run every **day** then we can state: **on average a deletion of 10,000 BPEL instances frees up 8 GB of disk space a day.**

The next step is to look at the tables and review the distribution again. The objective is to measure the difference in the distribution pre and post merge to determine the table(s) with maximum change in distribution. These tables will need to be constantly monitored since this is where most of the purge time will be spent[9].

Retention Policy

For the purging strategy to work, it is important to understand how long to retain the data in the DB. Factors that drive the retention policy are legal requirements, line of business requirements or overall company policy on retention of data. The longer the retention policy, the greater the volume of data that needs to stored and correspondingly, higher disk capacity requirements. The goal is to understand how much data will be retained on an average basis between purge cycles. Using the above use case an example of a retention policy would be:

Retention policy of 15 days → 10 GB (rate of inflow) x 15 days = 150 GB of retention data (on average). This is how much space will be required for retaining data over a 15 day period in for an inflow rate of 10 GB/day in the BPEL DB.

Disk space availability

The last task in our DB profiling exercise is to assess disk space availability for a given system. A lot of databases instance share their disk space with other DB instances to optimize hardware usage. There are only 2 types of disk options:

1. Infinite – an environment has no limitations and can add extra hard disk at any time
2. Finite – an environment has space limitations and needs to work within these boundaries

However what is most often neglected is the growth requirement that is often present with an ever increasing data set and pressures on the system with larger retention periods. Based on the above a system administrators need to define how much space is freely available to them. They need to take proper measures to line up extra hard disk if required, for systems which are on the brink of space limitations. We have observed many cases where disk space was ignored, leaving the operational team scrambling at the last minute to accommodate the growing business load requirements.

Summary

The three most important parts of this exercise are measuring the inflow rate, determining the retention policy and the retention dataset and figuring out disk space requirements. These metrics need to be gathered to truly understand which profile to choose and what strategy to apply to get the best results.

DB Profile = Inflow of Data + Retention Policy + Amount of available disk-space

[9] In our internal tests the same three tables, defined above, took the most time to delete due to the size of LOB columns.

The outflow rate, although important, only provides an insight into how many BPEL instances are deleted and how much table space is reclaimed but it does not play a role when picking a profile. It is however very useful in providing sizing perspectives and space requirements from an inflow outflow perspective and should be used to give administrative information on how table spaces are behaving.

Daily Space Requirements = Inflow of data – Outflow of data

The daily space requirements can be used as a guideline when choosing how much disk space to procure for a BPEL dehydration store database.

BPEL purging strategies

This section will highlight some of the strategies that can be deployed using the 10g purging toolsets after choosing a DB profile for the BPEL schema. Please note that there may be many combination of strategies, but we will only highlight the most relevant and optimal ones, we cannot list all possible permutations here. We recommend however that regardless of which purging strategy is chosen, it needs to be followed up with proper testing against a production like dataset. Purge testing cannot be left as an afterthought and needs to be become a part of the performance exercise and thorough testing is recommended to complete this cycle.

1. Small DB Profile

For an organization that has a small DB profile we would recommend using the BPEL purge scripts that come out of the box. The organization can choose to run either the multi-threaded looped or single-threaded looped purge procedure.

1.1 **Multi-threaded purge script:** If purge is done on a daily basis at the end of the day then the optimal choice is to use the multi-threaded script to spawn multiple jobs and shorten the time required to purge. This procedure is designed to purge large BPEL dehydration stores housed on high-end database nodes with multiple CPUs and good IO sub-system. It is recommended that this procedure is executed during non-business hours as it acquires a lot of resources and may contend with normal online operations.

1.2 **Single-threaded purge script:** If however an organization decides to purge during business hours, then we recommend using the single-threaded approach so that no DB cycles are taken away from the production system while the system is active.

2. Medium DB Profile

For an organization with a medium DB profile there are two options:

2.1 **Use purge scripts (recommended) :** Use either the multi-threaded or single threaded purge script to delete completed BPEL instances. Please see section 1.1

2.2 **Use range partitioning:** As an alternative, DB partitioning can be used as a purging strategy. The strategy would be to range partition the whole schema and drop partitions when instances complete.

- 2.2.1 If component instances tend to complete within a short period of time and a particular partition consists of only completed instances, drop the partition and rebuild global indexes (recommended scenario).
- 2.2.2 If component instances are very long-lived (weeks-months) and total retained data exceeds 1TB in size, run purge script (2.1) within partition to reclaim space[10]. As of now there are no special scripts that can be used to trim the partitions. Please use the regular single or multi threaded scripts as described above to reclaim space.
- 2.2.3 If purge script (2.1) has not been run for a long period of time (months) and some partitions still contain active instances/rows, consider running CTAS script during a maintenance window to trim the partitions.
- 2.2.4 If 11g database is being used to host the BPEL dehydration store then database compression is an option to consider for managing partitions with long running BPEL processes

[10] See section on long running BPEL processes

3. Large DB Profile

For organizations with large DB profiles we recommend using range-hash partitioning as the purging solution. Although the purge scripts can still be applied here, partitioning will provide the most options for DB manageability, administration and ease of use.

3.1 **Use multithreaded purge scripts:** Use either the multi-threaded script to delete completed BPEL instances. Please see section 1.1

3.2 **Use range/hash partitioning (recommended):** The strategy would be to range-hash partition the whole schema and drop partitions when instances complete.

- 3.2.1 If component instances tend to complete within a short period of time and a particular partition consists of only completed instances, drop the partition and rebuild global indexes (recommended scenario).
- 3.2.2 If component instances are very long-lived (weeks-months) and total retained data exceeds 1TB in size, run purge script (2.1) within partition to reclaim space.
- 3.2.3 If purge script (2.1) has not been run for a long period of time (months) and some partitions still contain active instances/rows, consider running CTAS script during a maintenance window to trim the partitions.
- 3.2.4 If 11g database is being used to host the BPEL dehydration store then database compression is an option to consider for managing partitions with long running BPEL processes

4. RetentionOnly DB Profile

For systems with RetentionOnly DB profile there are two options:

4.1 **Data Warehousing (recommended):** If disk space is not an issue then a Data Warehousing solution is most optimal where data can be archived as needed in a variety of partitions. The data can be kept for a very long time and discarded upon completion of retention policies

4.2 **Use Archiving and Compression (recommended):** If Data Warehousing is not an option then database archiving and compression methodologies can be deployed. The data can be compressed and stored in archives for long retention periods.

4.3 **Use range/hash partitioning:** If disk space is an issue then another strategy would be to range-hash partition the whole schema and drop partitions when instances complete.

- 4.3.1 If component instances tend to complete within a short period of time and a particular partition consists of only completed instances, drop the partition and rebuild global indexes (recommended scenario).
- 4.3.2 If component instances are very long-lived (weeks-months) and total retained data exceeds 1TB in size, run purge script (2.1) within partition to reclaim space.
- 4.3.3 If purge script (2.1) has not been run for a long period of time (months) and some partitions still contain active instances/rows, consider running CTAS script during a maintenance window to trim the partitions.
- 4.3.4 If 11g database is being used to host the BPEL dehydration store then database compression is an option to consider for managing partitions with long running BPEL processes

5. Long Running BPEL processes

A long running BPEL (LRB) process is one that is active and running for many weeks, months or even years. Usually these are long running business transactions that require durable BPEL processes which can be reactivated at given intervals of the business process. A SOA system can expect a maximum of 15% of its BPEL processes to be long running and may require special treatment. If the overall purging strategy is a partitioning strategy and since a partition cannot be dropped if there is an active instance present, it is advisable to either not drop the partition (adopt a data warehousing solution) or use the purge scripts to trim the partition. Since there should be no more than one or two active instances in a partition, running the purge scripts[11] will reclaim space while still keeping the partition active. Once the instances have completed the partition can be dropped as well. However, if partitioning is not an option then using the purge scripts with enough disk space to accommodate LRB processes will be the most prudent choice.

The ratio of long-running BPEL processes to short lived BPEL instances can be used as the fifth element to help determine your DB profile and subsequent purging strategy. If the ratio is high then consider using a RetentionOnly solution and if the ratio is low then small or medium level solutions.

6. 24x7 Operations

If the application is a 24x7 operation with little to no downtime then running a concurrent purge (either single or multi-threaded) may not be feasible. The concurrent purge will take away cycles from the production DB and will impact both performance and purge. For these kinds of systems we would recommend a RetentionOnly strategy using partitioning as the optimal purging mechanism.

7. Hybrid Solution

Another solution that has been used at some client sites is a hybrid solution – a purging strategy that uses a combination of partitioning and purge scripts. This solution is not the same as using the scripts to trim the partition as described above. Instead the strategy encompasses creating partitions for only LOB related tables while using purge scripts for the remainder. Here is how this solution works:

1. Create partition(s) for LOB related tables only. The user can choose to partition all three tables[12] or only one or two of them.
2. Use the purge script to delete rows from the rest of the schema
3. Create a temp table that stores the CIKEY (BPEL instance ID) for all the instances that were deleted. This should be done during the above purge.
4. Use this temp table to check if the partition can be dropped → this is the driver table that decides whether a partition can be dropped or not

The above approach allows the system administrator to purge efficiently saving time on deletion of LOB related tables by using partitioning instead. It also reduces the overhead of creating partitions on the full BPEL schema. Alternatively instead of creating partitions the same three LOB related tables can be purged asynchronously i.e. they can be purged later when there is much less load on the system using the same temp table approach with purge scripts.

This solution is not provided out of the box by Oracle – an example is included in Appendix B for review. Please note that for both of these hybrid solutions, minor changes will be required to the out of the box purging scripts and should be done by a certified DBA.

[11] The purge scripts will only delete completed instances
[12] XML_DOCUMENT, AUDIT_TRAIL, CUBE_SCOPE

Summary

Our objective through this white paper has been to provide a methodology to profile the BPEL DB by analysing the dehydration schema and its behaviour under load. We have provided four classifications of the profile: Small, Medium, Large and RetentionOnly which can be used as a means to determine an optimal purging strategy to meet a system's requirements. This DB profile should also be used as a way to conduct forecast future sizing requirements and provide some predictability into how the system will behave with new load requirements.

We would like to point out that purging cannot be just another after thought. It is important to test out each chosen strategy – these tests should become part of the performance testing cycle to provide the best results for a given system. Through this whitepaper we have tried to capture the most important purging strategies to fully use the current purging toolset provided by the product. Hopefully our readers can customize these strategies (if required) to meet their requirements, test them accordingly and benefit from a properly purged BPEL dehydration store.

Appendix A – BPEL Schema

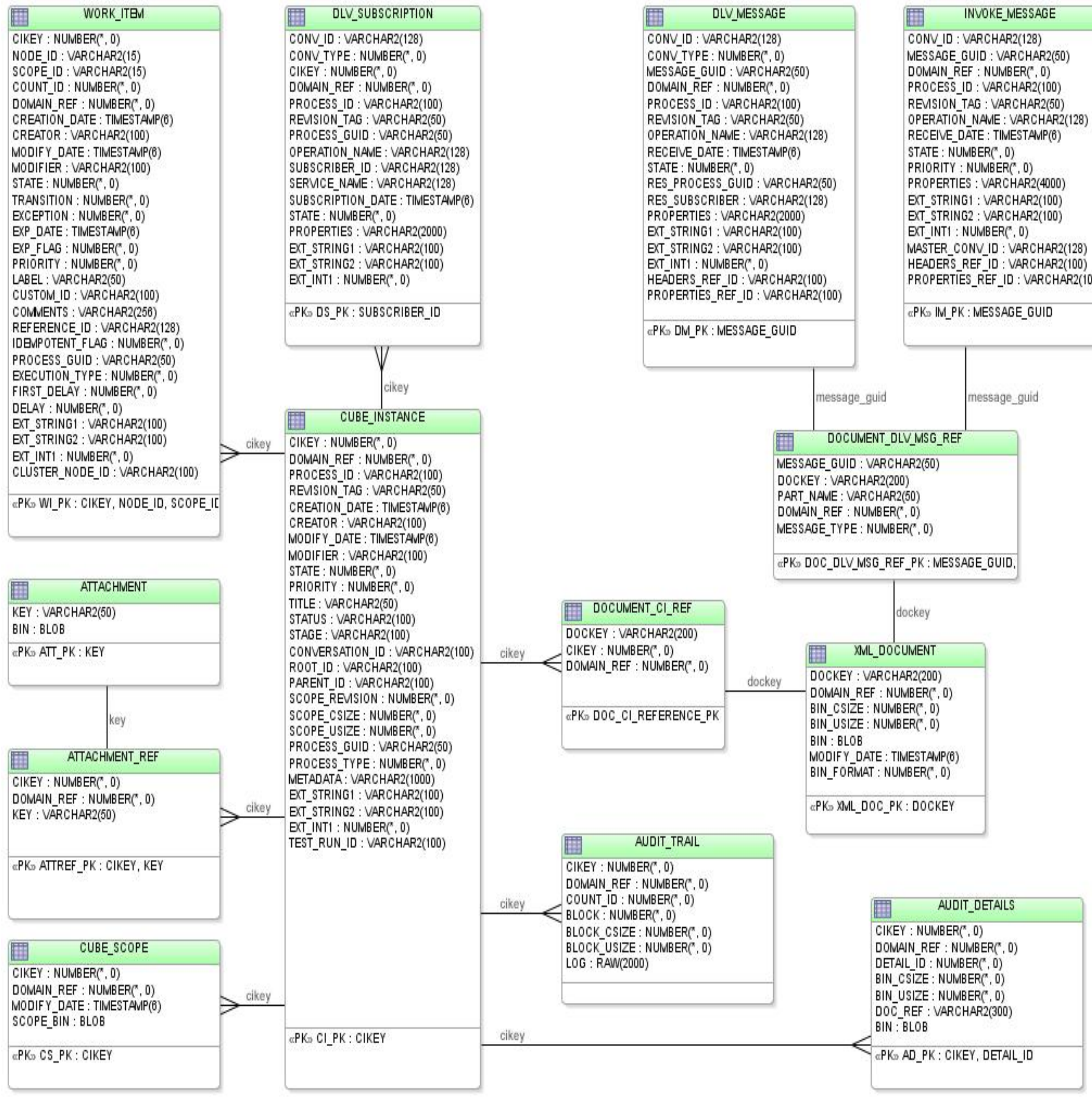
WORK_ITEM
CIKEY : NUMBER(*, 0)
NODE_ID : VARCHAR2(15)
SCOPE_ID : VARCHAR2(15)
COUNT_ID : NUMBER(*, 0)
DOMAIN_REF : NUMBER(*, 0)
CREATION_DATE : TIMESTAMP(6)
CREATOR : VARCHAR2(100)
MODIFY_DATE : TIMESTAMP(6)
MODIFIER : VARCHAR2(100)
STATE : NUMBER(*, 0)
TRANSITION : NUMBER(*, 0)
EXCEPTION : NUMBER(*, 0)
EXP_DATE : TIMESTAMP(6)
EXP_FLAG : NUMBER(*, 0)
PRIORITY : NUMBER(*, 0)
LABEL : VARCHAR2(50)
CUSTOM_ID : VARCHAR2(100)
COMMENTS : VARCHAR2(256)
REFERENCE_ID : VARCHAR2(128)
IDEMPOTENT_FLAG : NUMBER(*, 0)
PROCESS_GUID : VARCHAR2(50)
EXECUTION_TYPE : NUMBER(*, 0)
FIRST_DELAY : NUMBER(*, 0)
DELAY : NUMBER(*, 0)
EXT_STRING1 : VARCHAR2(100)
EXT_STRING2 : VARCHAR2(100)
EXT_INT1 : NUMBER(*, 0)
CLUSTER_NODE_ID : VARCHAR2(100)
«PK» WI_PK : CIKEY, NODE_ID, SCOPE_II
DLV_SUBSCRIPTION
CONV_ID : VARCHAR2(128)
CONV_TYPE : NUMBER(*, 0)
CIKEY : NUMBER(*, 0)
DOMAIN_REF : NUMBER(*, 0)
PROCESS_ID : VARCHAR2(100)
REVISION_TAG : VARCHAR2(50)
PROCESS_GUID : VARCHAR2(50)
OPERATION_NAME : VARCHAR2(128)
SUBSCRIBER_ID : VARCHAR2(128)
SERVICE_NAME : VARCHAR2(128)
SUBSCRIPTION_DATE : TIMESTAMP(6)
STATE : NUMBER(*, 0)
PROPERTIES : VARCHAR2(2000)
EXT_STRING1 : VARCHAR2(100)
EXT_STRING2 : VARCHAR2(100)
EXT_INT1 : NUMBER(*, 0)
«PK» DS_PK : SUBSCRIBER_ID
DLV_MESSAGE
CONV_ID : VARCHAR2(128)
CONV_TYPE : NUMBER(*, 0)
MESSAGE_GUID : VARCHAR2(50)
DOMAIN_REF : NUMBER(*, 0)
PROCESS_ID : VARCHAR2(100)
REVISION_TAG : VARCHAR2(50)
OPERATION_NAME : VARCHAR2(128)
RECEIVE_DATE : TIMESTAMP(6)
STATE : NUMBER(*, 0)
RES_PROCESS_GUID : VARCHAR2(50)
RES_SUBSCRIBER : VARCHAR2(128)
PROPERTIES : VARCHAR2(2000)
EXT_STRING1 : VARCHAR2(100)
EXT_STRING2 : VARCHAR2(100)
EXT_INT1 : NUMBER(*, 0)
HEADERS_REF_ID : VARCHAR2(100)
PROPERTIES_REF_ID : VARCHAR2(100)
«PK» DM_PK : MESSAGE_GUID
INVOKE_MESSAGE
CONV_ID : VARCHAR2(128)
MESSAGE_GUID : VARCHAR2(50)
DOMAIN_REF : NUMBER(*, 0)
PROCESS_ID : VARCHAR2(100)
REVISION_TAG : VARCHAR2(50)
OPERATION_NAME : VARCHAR2(128)
RECEIVE_DATE : TIMESTAMP(6)
STATE : NUMBER(*, 0)
PRIORITY : NUMBER(*, 0)
PROPERTIES : VARCHAR2(4000)
EXT_STRING1 : VARCHAR2(100)
EXT_STRING2 : VARCHAR2(100)
EXT_INT1 : NUMBER(*, 0)
MASTER_CONV_ID : VARCHAR2(128)
HEADERS_REF_ID : VARCHAR2(100)
PROPERTIES_REF_ID : VARCHAR2(100)
«PK» IM_PK : MESSAGE_GUID
cikey
message_guid
message_guid
CUBE_INSTANCE
CIKEY : NUMBER(*, 0)
DOMAIN_REF : NUMBER(*, 0)
PROCESS_ID : VARCHAR2(100)
REVISION_TAG : VARCHAR2(50)
CREATION_DATE : TIMESTAMP(6)
CREATOR : VARCHAR2(100)
MODIFY_DATE : TIMESTAMP(6)
MODIFIER : VARCHAR2(100)
STATE : NUMBER(*, 0)
PRIORITY : NUMBER(*, 0)
TITLE : VARCHAR2(50)
STATUS : VARCHAR2(100)
STAGE : VARCHAR2(100)
CONVERSATION_ID : VARCHAR2(100)
ROOT_ID : VARCHAR2(100)
PARENT_ID : VARCHAR2(100)
SCOPE_REVISION : NUMBER(*, 0)
SCOPE_CSIZE : NUMBER(*, 0)
SCOPE_USIZE : NUMBER(*, 0)
PROCESS_GUID : VARCHAR2(50)
PROCESS_TYPE : NUMBER(*, 0)
METADATA : VARCHAR2(1000)
EXT_STRING1 : VARCHAR2(100)
EXT_STRING2 : VARCHAR2(100)
EXT_INT1 : NUMBER(*, 0)
TEST_RUN_ID : VARCHAR2(100)
«PK» CI_PK : CIKEY
cikey
DOCUMENT_DLV_MSG_REF
MESSAGE_GUID : VARCHAR2(50)
DOCKEY : VARCHAR2(200)
PART_NAME : VARCHAR2(50)
DOMAIN_REF : NUMBER(*, 0)
MESSAGE_TYPE : NUMBER(*, 0)
«PK» DOC_DLV_MSG_REF_PK : MESSAGE_GUID,
dockey
ATTACHMENT
KEY : VARCHAR2(50)
BIN : BLOB
«PK» ATT_PK : KEY
DOCUMENT_CI_REF
DOCKEY : VARCHAR2(200)
CIKEY : NUMBER(*, 0)
DOMAIN_REF : NUMBER(*, 0)
«PK» DOC_CI_REFERENCE_PK
cikey
dockey
XML_DOCUMENT
DOCKEY : VARCHAR2(200)
DOMAIN_REF : NUMBER(*, 0)
BIN_CSIZE : NUMBER(*, 0)
BIN_USIZE : NUMBER(*, 0)
BIN : BLOB
MODIFY_DATE : TIMESTAMP(6)
BIN_FORMAT : NUMBER(*, 0)
«PK» XML_DOC_PK : DOCKEY
key
ATTACHMENT_REF
CIKEY : NUMBER(*, 0)
DOMAIN_REF : NUMBER(*, 0)
KEY : VARCHAR2(50)
«PK» ATTREF_PK : CIKEY, KEY
cikey
AUDIT_TRAIL
CIKEY : NUMBER(*, 0)
DOMAIN_REF : NUMBER(*, 0)
COUNT_ID : NUMBER(*, 0)
BLOCK : NUMBER(*, 0)
BLOCK_CSIZE : NUMBER(*, 0)
BLOCK_USIZE : NUMBER(*, 0)
LOG : RAW(2000)
cikey
AUDIT_DETAILS
CIKEY : NUMBER(*, 0)
DOMAIN_REF : NUMBER(*, 0)
DETAIL_ID : NUMBER(*, 0)
BIN_CSIZE : NUMBER(*, 0)
BIN_USIZE : NUMBER(*, 0)
DOC_REF : VARCHAR2(300)
BIN : BLOB
«PK» AD_PK : CIKEY, DETAIL_ID
CUBE_SCOPE
CIKEY : NUMBER(*, 0)
DOMAIN_REF : NUMBER(*, 0)
MODIFY_DATE : TIMESTAMP(6)
SCOPE_BIN : BLOB
«PK» CS_PK : CIKEY
cikey
cikey

Figure 1: BPEL Schema

Appendix B – Claim LOB Segments

The purge only remove the rows from the extents and but does not coalesce the space.

To shrink the space:
1. Enable row movement for the table.
SQL> ALTER TABLE xml_document ENABLE ROW MOVEMENT;

Options:

2.1 Shrink table but don't want to shrink HWM (High Water Mark).
SQL> ALTER TABLE xml_document SHRINK SPACE COMPACT;

2.2 Shrink table and HWM too.
SQL> ALTER TABLE xml_document SHRINK SPACE;

2.3. Shrink table and all dependent index too.
SQL> ALTER TABLE xml_document SHRINK SPACE CASCADE;

3. Disable row movement for the table.
ALTER TABLE xml_document DISABLE ROW MOVEMENT;

Appendix C – Hybrid solution examples

Example 1:

An example of purging the AUDIT_TRAIL using a range partition. Only this table is partitioned, the rest of the tables are purged using the multi-threaded looped purge. Data to be deleted from the AUDIT_TRAIL is collected in a temp table on a weekly basis and then dropped at the end of every week – the rest of the tables are purged daily using the purging scripts. Please see SQL scripts below:

add_audit_trail_partitions.sql   p_purge_audit_data.sql

Please make sure that all references to AUDIT_TRAIL are commented out in the purge scripts. To enable weekly collection of purgeable CIKEYS please add the following code to the purge script to populate the temp_cikey_audit_trail:

--Added to caputre all cikeys during weekday for weekend purge!

INSERT into temp_cikey_audit_trail select * from temp_cube_instance ;

commit;

<u>Please note:</u> The attached SQL scripts are not part of the product and hence bugs cannot be filed against them. This is an example of how to use the hybrid approach. Please conduct appropriate testing before using this on a production instance.

Example 2:

An example of purging the XML_DOCUMENT using range partitioning, the Oracle verify scripts and multi-looped purged scripts.

For this option to work do the following:

1. For either the single-looped purge or multi-threaded looped purge script comment out all references to the XML_DOCUMENT including the temp_xml_document tables. Since we are using partitioning for XML_DOCUMENT we will not be deleting anything via the purge scripts.
2. Review the BPEL partitioning PDF on OTN[13] and download patch 8682344. This patch has the verify scripts required to check if a partition can be dropped.
3. Run the purge scripts – the purge will delete everything from all table except XML_DOCUMENT
4. Run the verify scripts for XML_DOCUMENT → review the generated report to verify if the partition can be dropped or not.
5. Drop partition.
6. Please note that if you have partitions that cannot be dropped due to long running BPEL process, then you have to adapt a data warehousing strategy or trim the partitions to claim space.

[13] http://www.oracle.com/technetwork/middleware/bpel/documentation/bpel10gpartitioning-133743.pdf

Oracle BPEL 10g Purging Strategies
August 2010
Deepak Arora
Yogesh Kumar, Rajeev Misra, Michael Bousamra

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com