Oracle BPEL Process Manager 10g
Database Schema Partitioning
9 BPEL Purge Script................................................................. 23
10 CERTIFICATION .................................................................. 24
11 Purging and Partitioning Methodology .............................. 25
   11.1 Initial configuration ...................................................... 25
   11.2 Dropping the partitions ............................................... 26
Introduction

When the volume of data in the Oracle BPEL Process Manager dehydration store grows very large, maintaining the database can become difficult. To address this maintenance challenge, Oracle BPEL Process Manager 10.1.3.5 has been instrumented with partition keys that allow database administrators (DBAs) to take advantage of the Oracle RDBMS partitioning features and capabilities.

With the new instrumentation of the BPEL engine, the schema tables can be range partitioned on time intervals. This is arguably the most useful partitioning option when one needs to reduce the database maintenance window of large tables. (Though not discussed in this paper, this also opens up the possibility of archiving partitioned data.)

Conventions

Throughout this document, Oracle BPEL Process Manager is often. The following terms are synonymous: Oracle BPEL Process Manager, BPEL PM, BPEL engine, or simply BPEL.

Oracle BPEL PM utilizes a database to store metadata and instance data during runtime. This data is stored in what is known as the dehydration store, which is simply a database schema. Throughout this document the terms dehydration store, BPEL schema, BPEL tables, etc are all equivalent. Any references to database schema objects in this document pertain only to the BPEL dehydration store (i.e., BPEL’s metadata and instance data). This is separate and independent from any database objects used by your BPEL processes for storing application or business data.

For performance reasons, the BPEL schema does not utilize foreign keys and thus master-detail relationships are not obviously inferred from looking at the schema definition. Within this paper the term "dependent tables" will be used interchangeably with
"detail tables" (as in master-detail tables) and imply a master-detail relationship, even if that relationship is not obvious. These dependency relationships are maintained by the BPEL engine.

Before reading this paper…
Please be aware of the following:

- The intended audience for this whitepaper is DBAs.
- Database partitioning is not for everyone. BPEL PM ships with a database script for removing instance rows from the dehydration store:

  `purge_instances_oracle.sql`

  This purge script is adequate for the majority of BPEL PM installations. Partitioning need only be considered when your dehydration store grows so large that the purge script doesn’t meet your performance needs.

- The BPEL purge script remains independent of BPEL table partitioning but with some important caveats; for full details, please refer to Chapter 9 BPEL Purge Script (page 23) before you begin any partitioning.

- The task of partitioning the BPEL tables should be performed by an experienced DBA. Since partitioning tables is considered a core DBA skill, this paper does not provide detailed step-by-step instructions on how to partition tables. Rather, it serves as a guide in order to give a DBA knowledge and understanding of the BPEL schema and its associated scripts. Armed with this knowledge, the DBA is free to customize any partitioning scheme for their environment, as well as incorporate any tuning parameters in response to the performance of their database. Tuning is never a one-size-fits-all
proposition or a one-off configuration change, but an iterative process of monitoring and tuning.

• The partitioning schemes discussed in this paper can only be used with Oracle BPEL Process Manager 10gR3 10.1.3.5 or higher (within 10gR3). They also do not apply to BPEL 11g because of significant schema changes.

Feedback
Oracle are keen on hearing your feedback and comments on both this document, and your experience with partitioning the BPEL dehydration store. Please send your comments to Michael.Bousamra@oracle.com.
1 Introducing the core BPEL 10g schema tables

While the BPEL schema comprises many tables, only three main tables drive any consideration for database partitioning:

- CUBE_INSTANCE
- DLV_MESSAGE
- XML_DOCUMENT

(Appendix A The BPEL Schema describes these and other BPEL tables in more detail.)

These three main tables can be logically arranged into three groups with their dependent tables. Each group will be described in the sections ahead.

The following table lists each of the groups, their main driver table, and their associated group partition keys.

<table>
<thead>
<tr>
<th>GROUP NAME</th>
<th>DRIVER TABLE(S)</th>
<th>PARTITION KEY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td>CUBE_INSTANCE</td>
<td>CI_PARTITION_DATE</td>
</tr>
<tr>
<td>Group 2</td>
<td>DLV_MESSAGE</td>
<td>DLV_PARTITION_DATE</td>
</tr>
<tr>
<td>Group 3</td>
<td>XML_DOCUMENT</td>
<td>DOC_PARTITION_DATE</td>
</tr>
</tbody>
</table>

These groups were formed to provide a strategy to maintain referential integrity across the BPEL schema. More details on each group will be provided in the following sections, however it should prove useful to first discuss referential integrity and equi-partitioning.
2 Referential Integrity and Equi-Partitioning

For performance reasons the BPEL Schema has no foreign key constraints to police integrity. This fact also discounts the use of the new 11g RDBMS feature: referential partitioning. This feature provides significant benefits because it equi-partitions master and detail tables across foreign key constraints. As an example, the STATE (i.e., completed, faulted, etc) of each detail row in the equi-partition can be inferred from its associated master table row.

Although referential partitioning cannot be used with the BPEL schema, similar behavior can be mimicked in order to realize some of the same benefits. As of BPEL 10.1.3.5 the BPEL engine will ensure that the partition key of every detail table row is the same as the partition key of its master table row. Then to complete the setup, the DBA simply needs to ensure that the master and detail tables are range partitioned on the same intervals. Some examples are provided in the sections ahead.

NOTE: Some customers may decide that referential integrity of aged partitions is not a concern for their site. For example, the site may have ample disk space allowing data to significantly age; or they may have found no (apparent) adverse affects of allowing unreferenced data to be housed in the dependent tables. However there is no guarantee that unreferenced data will not cause issues in future releases.
3 BPEL Partition Key Overview

The following factors were considered when selecting the BPEL Schema partition keys:

1. Convey or imply state (e.g., completed) for referential integrity
2. Allow range partitioning on time intervals for maintenance operations
3. Be static to avoid row movement that may lead to unreferenced data
4. Be static to avoid row movement when table maintenance operations are performed
5. Provide performance benefits for BPEL console queries via partition pruning

The following sections list the range partition keys, by group, that can be used by the DBA to partition the BPEL schema.

3.1 Group 1 – CUBE_INSTANCE group

<table>
<thead>
<tr>
<th>TABLE</th>
<th>RANGE PARTITION KEY</th>
<th>HASH SUBPARTITION KEY</th>
<th>RELATIONSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUBE_INSTANCE</td>
<td>CREATION_DATE</td>
<td>CIKEY</td>
<td>Master</td>
</tr>
<tr>
<td>ATTACHMENT</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>ATTACHMENT_REF</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>AUDITDETAILS</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>AUDIT_TRAIL</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>CI_INDEXES</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>CUBE_SCOPE</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>DLV_SUBSCRIPTION</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>DOCUMENT_CI_REF</td>
<td>CI_PARTITION_DATE</td>
<td>DOCKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>WI_EXCEPTION</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>WI_FAULT</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
<tr>
<td>WORK_ITEM</td>
<td>CI_PARTITION_DATE</td>
<td>CIKEY</td>
<td>Dependent</td>
</tr>
</tbody>
</table>
3.2 Group 2 – DLV_MESSAGE group

<table>
<thead>
<tr>
<th>TABLE</th>
<th>RANGE PARTITION KEY</th>
<th>HASH SUBPARTITION KEY</th>
<th>RELATIONSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLV_MESSAGE</td>
<td>RECEIVE_DATE</td>
<td>MESSAGE_GUID</td>
<td>Master</td>
</tr>
<tr>
<td>INVOKE_MESSAGE</td>
<td>RECEIVE_DATE</td>
<td>MESSAGE_GUID</td>
<td>Dependent</td>
</tr>
<tr>
<td>DOCUMENT_DLV_MSG_REF</td>
<td>DLV_PARTITION_DATE</td>
<td>DOCKEY</td>
<td>Dependent</td>
</tr>
</tbody>
</table>

3.3 Group 3 – XML_DOCUMENT group

Unlike the other two groups, this group contains only the one table.

<table>
<thead>
<tr>
<th>TABLE</th>
<th>RANGE PARTITION KEY</th>
<th>HASH SUBPARTITION KEY</th>
<th>RELATIONSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML_DOCUMENT</td>
<td>DOC_PARTITION_DATE</td>
<td>DOCKEY</td>
<td>Dependent</td>
</tr>
</tbody>
</table>

3.4 Notes

- A hash subpartition is an option the DBA may want to explore especially for tables with LOB segments as this should assist with HighWater (HW) enqueue contention.
- A global hash index on primary keys that are monotonically increasing (like CIKEY) may relieve hot block contention. (Refer Chapter 8 Index Types and Maintenance on page 21.)
- BPEL console queries have been optimized to take advantage of the range partition keys described above. Refer to BUG 8614009 for the status of this optimization.
4  Deep Dive: Group 1 – CUBE_INSTANCE

This chapter builds on the partition key overview from Chapter 3 BPEL Partition Key Overview to provide more details on the CUBE_INSTANCE group.

In this group the only mandatory table to partition is CUBE_INSTANCE as it is the master table which holds the instance state of each row (e.g., completed). When considering which dependent tables to partition along with CUBE_INSTANCE, the DBA should note that any dependent tables not partitioned will probably need to be purged. (Refer to Chapter 9 BPEL Purge Script on page 23.)

As stated in the section in Chapter 2 Referential Integrity and Equi-Partitioning, to complete the equi-partitioning setup the DBA must partition the master table (CUBE_INSTANCE) and any detail tables on the same range partition interval, such as WORK_ITEM.

4.1 Example

This example shows how the CUBE_INSTANCE table may be partitioned, along with the dependent WORK_ITEM table.

4.1.1  CUBE_INSTANCE

Here the CUBE_INSTANCE table is range partitioned on CREATION_DATE with a hash subpartition on CIKEY:

```
TABLESPACE ORABPEL
PARTITION BY RANGE (CREATION_DATE)
SUBPARTITION BY HASH (CIKEY)
(
    PARTITION p01_2009 VALUES LESS THAN (TIMESTAMP'2009-04-01 00:00:00')
     SUBPARTITIONS 16,
    PARTITION p02_2009 VALUES LESS THAN (TIMESTAMP'2009-05-01 00:00:00')
     SUBPARTITIONS 16,
    PARTITION p03_2009 VALUES LESS THAN (TIMESTAMP'2009-06-01 00:00:00')
     SUBPARTITIONS 16,
    PARTITION p04_2009 VALUES LESS THAN (TIMESTAMP'2009-07-01 00:00:00')
     SUBPARTITIONS 16
)
```

4.1.2  WORK_ITEM

The WORK_ITEM table is range partitioned on CI_PARTITION_DATE with a hash subpartition on CIKEY. The CI_PARTITION_DATE partition key will be populated with the same value as the CREATION_DATE of the master CUBE_INSTANCE row. The partition
intervals (e.g., VALUES LESS THAN) must be the same as the master CUBE_INSTANCE table in the example above.

```
TABLESPACE ORABPEL
PARTITION BY RANGE (CI_PARTITION_DATE)
SUBPARTITION BY HASH (CIKEY)
(
  PARTITION p01_2009 VALUES LESS THAN (TIMESTAMP'2009-04-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p02_2009 VALUES LESS THAN (TIMESTAMP'2009-05-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p03_2009 VALUES LESS THAN (TIMESTAMP'2009-06-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p04_2009 VALUES LESS THAN (TIMESTAMP'2009-07-01 00:00:00')
    SUBPARTITIONS 16
)
```

### 4.1.3 Notes

- The examples, above, used a hash subpartition because the DBA may find improvements in disk I/O. However this depends on the site’s disk storage configuration and needs to be properly tested.
- The DBA can choose to add a MAXVALUE partition to the examples above.

### 4.2 Verification script

A verification script has been written so that the DBA can determine when a CUBE_INSTANCE partition and its equi-partitioned dependent table partitions can be dropped. The verification script will not make any changes, but will generate a report. (The DBA is responsible for partitioning the BPEL tables and creating the necessary drop scripts.)

The location of the verification scripts is:

```
ORACLE_HOME/bpel/system/database/script
```

The verification scripts for this group are:

- **CI_VERIFY.sql**
  Creates the verification package.

- **CI_EXEC_VERIFY.sql**
  Executes the CI_VERIFY verification package. Please read the comments in this script for setup and execution instructions.
This is the pseudo code for the CUBE_INSTANCE verification package:

1. This script accepts three parameters:
   1.1. Flag chk_tree (see Appendix C on page 35 for an explanation of this flag)
   1.2. Collection of CUBE_INSTANCE partitions to verify
   1.3. Collection of dependent tables that have been partitioned

2. For each CUBE_INSTANCE partition:
   2.1. Check that all rows (i.e., BPEL instances) in the partition have completed
   2.2. Check that all dependent tables are equi-partitioned:
       2.2.1. By the same upper bound (high_value)
       2.2.2. By the same lower bound (high_value of previous partition)
       2.2.3. By the same name

3. Generate a pass / failure report of whether or not the partition can be dropped

4.2.1 Notes

- This script will ensure that all rows in a CUBE_INSTANCE partition are in a completed state (i.e., state < 5). However if the chk_tree flag is set to true then the instance tree (parent and child instances) for each row in a partition must have completed. (Checking the parent and child BPEL instance tree is expensive, so be sure this requirement is necessary before enabling.)

Appendix C Relationships between instances on page 35 has a detailed explanation of the chk_tree flag and what it means.

- All dependent table partitions will be checked to ensure that they have the same upper bound, lower bound, and partition name as the CUBE_INSTANCE partition. If this check fails then it is possible that unreferenced data will occur when the partition is dropped. Therefore ensure to configure the CUBE_INSTANCE partition and the dependent table partitions with the exact same partition interval, as described earlier.

- It is assumed that only aged partitions (and not the current partition) will be verified and dropped. Therefore the verification scripts can be executed prior to the intended maintenance drop time (mid-week, say) and the partitions can actually be dropped later, on the weekend after backups, say. There is nothing stopping a DBA from dropping aged partitions while the system is running however please read Chapter 8 Index Types and Maintenance (page 21).

- A report file (CI_<partition_name>) is created for each CUBE_INSTANCE partition. The messages in the report file are self-explanatory and detail the pass or failure of the checks outlined in the pseudo code, above.
• The report files require the creation of a directory where they should be saved to, which is documented in the verification scripts. For example:

```sql
SQLPLUS> create directory PART_DIR as '/.........';
```

• The verification package does not drop any partitions; it just ensures that partitions are eligible to be dropped.
5 Deep Dive: Group 2 – DLV_MESSAGE

This chapter builds on the partition key overview from Chapter 3 BPEL Partition Key Overview to provide more details on the DLV_MESSAGE group.

The relationship between the DLV_MESSAGE, INVOKE_MESSAGE, and DOCUMENT_DLV_MSG_REF makes it mandatory to partition all three tables for equi-partitioning. The DLV_MESSAGE is considered the master table. (For those who have familiarity with the BPEL 10g schema, the INVOKE_MESSAGE table is also a candidate to be the master table. However in BPEL Process Manager 11g the two tables have been consolidated into DLV_MESSAGE, hence the decision was made to use this table as the master to be future-proof.)

As stated in the section in Chapter 2 Referential Integrity and Equi-Partitioning, to complete the equi-partitioning setup the DBA must partition DLV_MESSAGE and INVOKE_MESSAGE and DOCUMENT_DLV_MSG_REF on the same range partition interval.

5.1 Example

This example shows one way the three tables can be partitioned.

5.1.1 DLV_MESSAGE

Here the DLV_MESSAGE table is range partitioned on RECEIVE_DATE with a hash subpartition on MESSAGE_GUID:

```
TABLESPACE ORABPEL
PARTITION BY RANGE (RECEIVE_DATE)
SUBPARTITION BY HASH (MESSAGE_GUID)
(
  PARTITION p01_2009 VALUES LESS THAN (TIMESTAMP'2009-04-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p02_2009 VALUES LESS THAN (TIMESTAMP'2009-05-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p03_2009 VALUES LESS THAN (TIMESTAMP'2009-06-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p04_2009 VALUES LESS THAN (TIMESTAMP'2009-07-01 00:00:00')
    SUBPARTITIONS 16
)
```

5.1.2 INVOKE_MESSAGE

The INVOKE_MESSAGE table is range partitioned on RECEIVE_DATE with a hash subpartition on MESSAGE_GUID. The partition intervals (e.g., VALUES LESS THAN) must be the same as the DLV_MESSAGE master table in the example above:
TABLESPACE ORABPEL
PARTITION BY RANGE (RECEIVE_DATE)
SUBPARTITION BY HASH (MESSAGE_GUID)
(
  PARTITION p01_2009 VALUES LESS THAN (TIMESTAMP'2009-04-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p02_2009 VALUES LESS THAN (TIMESTAMP'2009-05-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p03_2009 VALUES LESS THAN (TIMESTAMP'2009-06-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p04_2009 VALUES LESS THAN (TIMESTAMP'2009-07-01 00:00:00')
    SUBPARTITIONS 16,
)

5.1.3 DOCUMENT_DLV_MSG_REF

The DOCUMENT_DLV_MSG_REF table is range partitioned on DLV_PARTITION_DATE with a hash subpartition on DOCKEY. The DLV_PARTITION_DATE partition key will be populated with the same value as the RECEIVE_DATE of either the DLV_MESSAGE or INVOKE_MESSAGE row it is associated with. The partition intervals (e.g., VALUES LESS THAN) must be the same as the DLV_MESSAGE master table in the example above:

TABLESPACE ORABPEL
PARTITION BY RANGE (DLV_PARTITION_DATE)
SUBPARTITION BY HASH (DOCKEY)
(
  PARTITION p01_2009 VALUES LESS THAN (TIMESTAMP'2009-04-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p02_2009 VALUES LESS THAN (TIMESTAMP'2009-05-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p03_2009 VALUES LESS THAN (TIMESTAMP'2009-06-01 00:00:00')
    SUBPARTITIONS 16,
  PARTITION p04_2009 VALUES LESS THAN (TIMESTAMP'2009-07-01 00:00:00')
    SUBPARTITIONS 16
)

5.1.4 Notes

- The examples, above, used a hash subpartition because the DBA may find improvements in disk I/O. However this depends on the site’s disk storage configuration and needs to be properly tested.
- The DBA can choose to add a MAXVALUE partition to the examples above.
5.2 Verification script

A verification script has been written so that the DBA can determine when a DLV_MESSAGE partition and its dependent table partitions can be dropped. The verification script will not make any changes, but will generate a report. (The DBA is responsible for partitioning the BPEL tables and creating the necessary drop scripts.)

The location of the verification scripts is:

```
ORACLE_HOME/bpel/system/database/script
```

The verification scripts for this group are:

- **DL_VERIFY.sql**
  Creates the verification package.

- **DL_EXEC_VERIFY.sql**
  Executes the DL_VERIFY verification package. Please read the comments in this script for setup and execution instructions.

This is the pseudo code for the DLV_MESSAGE verification package:

1. This script accepts two parameters:
   1.1. Collection of DLV_MESSAGE partitions to verify
   1.2. Collection of dependent tables that have been partitioned
2. For each DLV_MESSAGE partition:
   2.1. Check that all rows in the partition have been handled
   2.2. Check that all rows in the INVOKE_MESSAGE partition (with the same partition name) have been handled
   2.3. Check that all dependent tables are equi-partitioned:
      2.3.1. By the same upper bound (high_value)
      2.3.2. By the same lower bound (high_value of previous partition)
      2.3.3. By the same name
3. Generate a pass / failure report of whether or not the partition can be dropped

5.2.1 Notes

- This script will ensure that all rows in a DLV_MESSAGE partition are in a handled state (i.e., state > 1) and that they are not associated with any open CUBE_INSTANCE rows. This check is then repeated for the INVOKE_MESSAGE partition with the same partition name as the DLV_MESSAGE partition.
• All dependent table partitions will be checked to ensure that they have the same upper bound, lower bound, and partition name as the DLV_MESSAGE partition. If this check fails then it is possible that unreferenced data will occur when the partition is dropped. Therefore ensure to configure the DLV_MESSAGE partition and the dependent table partitions with the exact same partition interval, as described earlier.

• Although it is not essential, this script will execute faster if performed after the CUBE_INSTANCE partitions are dropped. Part of the verification logic ensures that any associated CUBE_INSTANCE rows have completed, therefore the script will still function correctly but should execute faster if it has fewer rows to parse in the CUBE_INSTANCE table.

• It is assumed that only aged partitions (and not the current partition) will be verified and dropped. Therefore the verification scripts can be executed prior to the intended maintenance drop time (mid-week, say) and the partitions can actually be dropped later, on the weekend after backups, say. There is nothing stopping a DBA from dropping aged partitions while the system is running however please read Chapter 8 Index Types and Maintenance (page 21).

• A report file (DL_<partition_name>), is created for each DLV_MESSAGE partition. The messages in the report file are self-explanatory and detail the pass or failure of the checks outlined in the pseudo code, above.

• The report files require the creation of a directory where they should be saved to, which is documented in the verification scripts. For example:

  `SQLPLUS> create directory PART_DIR as '/........';`

• The verification package does not drop any partitions; it just ensures that partitions are eligible to be dropped.
6  Deep Dive: Group 3 – XML_DOCUMENT

This chapter builds on the partition key overview from Chapter 3 BPEL Partition Key Overview to provide more details on the XML_DOCUMENT group.

Although it is referred to as a group, the XML_DOCUMENT table stands alone but has dependencies on the CUBE_INSTANCE and DLV_MESSAGE group of tables. There are no equi-partitioning requirements on this table and the state (e.g., completed) of its rows are determined by whether they are unreferenced. If the rows in a partition are not referenced in tables: DOCUMENT_CI_REF, DOCUMENT_DLV_MSG_REF, DLV_MESSAGE, INVOKE_MESSAGE, and AUDIT_DETAILS then they are considered to be complete and eligible for removal.

6.1 Example

This example shows how the XML_DOCUMENT table may be partitioned.

6.1.1 XML_DOCUMENT

Here the XML_DOCUMENT table is range partitioned on DOC_CREATION_DATE with a hash subpartition on DOCKEY. The DOC_PARTITION_DATE partition key is actually the creation date of the row.

```
TABLESPACE ORABPEL
PARTITION BY RANGE (DOC_PARTITION_DATE)
SUBPARTITION BY HASH (DOCKEY)
(
    PARTITION p01_2009 VALUES LESS THAN (TIMESTAMP'2009-04-01 00:00:00')
        SUBPARTITIONS 16,
    PARTITION p02_2009 VALUES LESS THAN (TIMESTAMP'2009-05-01 00:00:00')
        SUBPARTITIONS 16,
    PARTITION p03_2009 VALUES LESS THAN (TIMESTAMP'2009-06-01 00:00:00')
        SUBPARTITIONS 16,
    PARTITION p04_2009 VALUES LESS THAN (TIMESTAMP'2009-07-01 00:00:00')
        SUBPARTITIONS 16
)
```

6.1.2 Notes

- The examples, above, used a hash subpartition because the DBA may find improvements in disk I/O. However this depends on the site’s disk storage configuration and needs to be properly tested.
- The DBA can choose to add a MAXVALUE partition to the examples above.
6.2 Verification script

A verification script has been written so that the DBA can determine when a XML_DOCUMENT partition and its dependent table partitions can be dropped. The verification script will not make any changes, but will generate a report. (The DBA is responsible for partitioning the BPEL tables and creating the necessary drop scripts.)

The location of the verification scripts is:

```
ORACLE_HOME/bpel/system/database/script
```

The verification scripts for this group are:

- **DC_VERIFY.sql**
  Creates the verification package.
- **DC_EXEC_VERIFY.sql**
  Executes the DC_VERIFY verification package. Please read the comments in this script for setup and execution instructions.

This is the pseudo code for the XML_DOCUMENT verification package:

1. This script accepts one parameter:
   1.1. Collection of XML_DOCUMENT partitions to verify
2. For each XML_DOCUMENT partition:
   2.1. Check that no rows are referenced in table DOCUMENT_DLV_MSG_REF
   2.2. Check that no rows are referenced in table DLV_MESSAGE
   2.3. Check that no rows are referenced in table INVOKE_MESSAGE
   2.4. Check that no rows are referenced in table DOCUMENT_CI_REF
   2.5. Check that no rows are referenced in table AUDIT_DETAIL
3. Generate a pass / failure report of whether or not the partition can be dropped

### Notes

- This script will ensure that all rows in the XML_DOCUMENT partition are not referenced in tables DOCUMENT_CI_REF, DOCUMENT_DLV_MSG_REF, and AUDIT_DETAILS. If the XML_DOCUMENT partition rows are unreferenced then their states are considered closed and eligible to be dropped.
- For better performance, it is recommended to run the XML_DOCUMENT verification script after the CUBE_INSTANCE and DLV_MESSAGE group of table partitions have been dropped. As stated the XML_DOCUMENT verification script checks that the rows are
unreferenced so it makes little sense to run it before CUBE_INSTANCE and DLV_MESSAGE dependent tables have been dropped.

- Unless completed rows are removed from the association tables DOCUMENT_CI_REF and DOCUMENT_DLV_MSG_REF the XML_DOCUMENT verification script will be unable to state that a partition can be dropped. Therefore if these association tables are not partitioned and dropped, then they must be purged. Refer to section titled Chapter 9 BPEL Purge Script (page 23).

- It is assumed that only aged partitions (and not the current partition) will be verified and dropped. Therefore the verification scripts can be executed prior to the intended maintenance drop time (mid-week, say) and the partitions can actually be dropped later, on the weekend after backups, say. There is nothing stopping a DBA from dropping aged partitions while the system is running however please read Chapter 8 Index Types and Maintenance (page 21).

- A report file (DC_<partition_name>) is created for each XML_DOCUMENT partition. The messages in the report file are self-explanatory and detail the pass or failure of the checks outlined in the pseudo code, above.

- The report files require the creation of a directory where they should be saved to, which is documented in the verification scripts. For example:

  SQLPLUS> create directory PART_DIR as '/.........';

- The verification package does not drop any partitions; it just ensures that partitions are eligible to be dropped.
7 Migrating to BPEL 10.1.3.5 with existing data

The BPEL Process Manager 10.1.3.5 patchset is the first release to add the partition key columns discussed throughout this document. The BPEL engine will then automatically start populating these columns for any new rows.

However if data exists in the dehydration store prior to the application of the 10.1.3.5 patchset then the partition key for these rows will be NULL. It should be apparent that after the application of the 10.1.3.5 patchset there is a potential for a mixture of rows, either with or without populated partition keys.

When a DBA decides to partition the BPEL Schema then, it is mandatory that all partition keys be populated (for equi-partitioning to be honored). These partition keys need to be updated with a timestamp value that will ensure both detail and master rows are written to partitions, which are equi-partitioned.

This equi-partitioning requirement can be easily met by updating the NULL partition keys with a timestamp that will place ALL existing data in the first partition of every table. Therefore prior to partitioning the BPEL schema the DBA should update all NULL partition keys with a timestamp that is less than the HIGH_VALUE of the first partition.

This only ever needs to be done once and only needs to be performed if you intend to partition the BPEL schema.

7.1 Example

The following examples show how you can update the NULL partition key columns for the different partition groups.

7.1.1 CUBE_INSTANCE

For the CUBE_INSTANCE dependent table WI_FAULT:

```sql
UPDATE WI_FAULT
SET CI_PARTITION_DATE = TIMESTAMP'2009-03-01 00:00:00' 
WHERE CI_PARTITION_DATE is NULL;

COMMIT;
```

See TABLE 2 on page 6 for all CUBE_INSTANCE dependent tables.

Note: DO NOT update CREATION_DATE for CUBE_INSTANCE.

7.1.2 DLV_MESSAGE

For DLV_MESSAGE dependent table DOCUMENT_DLV_MSG_REF:
UPDATE DOCUMENT_DLV_MSG_REF
SET DLV_PARTITION_DATE = TIMESTAMP'2009-03-01 00:00:00'
WHERE DLV_PARTITION_DATE is NULL;

COMMIT;

See TABLE 3 on page 7 for all DLV_MESSAGE dependent tables.

Note: DO NOT update RECEIVE_DATE for the DLV_MESSAGE or INVOKE_MESSAGE tables.

7.1.3 CUBE_INSTANCE

For the XML_DOCUMENT table:

UPDATE XML_DOCUMENT
SET DOC_PARTITION_DATE = TIMESTAMP'2009-03-01 00:00:00'
WHERE DOC_PARTITION_DATE is NULL;

COMMIT;

This is the only table in this group so no other updates are needed.
8 Index Types and Maintenance

Oracle database partitioning provides additional choices on the index types that can be created for a table. The type of index will affect performance and maintenance windows. To help DBAs make informed decisions, the BPEL schema creation scripts contain some guidance, in the form of comments, on the types of indexes suitable to be created. The schema creation scripts are located here:

```
ORACLE_HOME/ bpel/system/database/scripts
```

The base script that creates the schema (including indexes) is:

```
domain_oracle.ddl
```

Sometimes patchsets or updates may modify the BPEL schema. If so there may be additional SQL files with the filename:

```
update_<from_version>_to_version>.sql
```

Typically only `domain_oracle.ddl` will contain any index definitions, but the "update" scripts should also be examined for thoroughness.

NOTE: On-site performance testing might suggest a different index than the one advised. In this case the DBA should go with the results of their performance testing.

As of BPEL Process Manager 10.1.3.5, the BPEL schema creation script has been modified to include comments that should assist a DBA when deciding what type of index to create when they have partitioned their schema. Each index will have one of the following comments above it:

Only one of the following comments will be placed above each Index:

- This index is used mainly by the BPEL engine in an unbound equality search and thus should be a NORMAL or GLOBAL HASH index, if its table has been partitioned.
- This index is used mainly by the BPEL engine in a range search and thus should be a NORMAL index, if its table has been partitioned.
- This index is used mainly by the BPEL engine in a range search with the partition key predicate and thus should be a LOCAL index, if its table has been partitioned.
- This index is used mainly by the BPEL console in an unbound equality search and thus should be a NORMAL or GLOBAL HASH index, if its table has been partitioned.
- This index is used mainly by the BPEL console in a range search and thus should be a NORMAL or LOCAL index, if its table has been partitioned.
- This index is used mainly by the BPEL console in a range search with the partition key predicate and thus should be a LOCAL index, if its table has been partitioned.
The number of normal and global indexes affects the performance of partition maintenance operations, like the dropping of a partition. When a partition is dropped all global and normal indexes are marked unusable and need to be rebuilt. The rebuild can be performed online but all SQL queries will produce sub-optimal plans until the rebuild is complete. A better alternative is to use the `UPDATE GLOBAL INDEXES` clause when dropping partitions. This clause will increase the time required to drop a partition but the indexes will remain available to the application.

Primary keys on the tables should usually be normal or global. For monotonically increasing keys, like `CIKEY` in `CUBE_INSTANCE` (and its dependent tables), a global hash partitioned index should show improvements for hot block contention.
9 BPEL Purge Script

The purge script that is currently supplied with the BPEL product can be found here:

```
ORACLE_HOME/bpel/system/database/scripts/purge_instances_oracle.sql
```

It is still independent of the partitioning described in this paper, with one important caveat.

If not all dependent tables are partitioned within a group, then it is likely that these non-partitioned tables will need to be purged. If this is the case then:

**IMPORTANT:** The BPEL purge script MUST run before the table partitions are dropped.

The reason for that is because the purge script works off three collections (placed in temp tables) of completed rows from the tables CUBE_INSTANCE, DLV_MESSAGE, and INVOKE_MESSAGE. These collections are then used to delete rows from the dependent tables before the rows are deleted from the source tables.

Given that, it should be clear that the collections of completed rows cannot be built if the master table partitions have been dropped. If the current purge script is not run before the master partitions are dropped then it could lead to unreferenced data in the dependent tables. (The DBA can of course write a script to remove these unreferenced rows.)

Furthermore, the tables that have been partitioned should have their DELETE statements removed from the purge script. Otherwise the advantage of partitioning these tables will have been lost.
10 CERTIFICATION

Oracle BPEL Process Manager 10.1.3.5 with partitioning has been certified against Oracle RDBMS 10.2.0.4 (+) and 11.1.0.7(+).
11 Purging and Partitioning Methodology

This chapter summarizes the main points from this document into an action plan that you can follow if you wish to do purging and/or partitioning of the Oracle BPEL dehydration store. Note that purging and partitioning is optional. The BPEL engine does not require it; it is only needed if the BPEL data is consuming too much space or you have some other reason to remove the data.

There are three main strategies to reducing the size of the BPEL schema:

- Purge script
- Purging script + Partitioning (or more correctly, dropping table partitions)
- Partitioning all tables

In the first two cases, the same purge script is used – although if you are partitioning, you need to edit the purge script to comment out your partitioned tables.

The purge script uses standard SQL DELETE statements to remove rows from the BPEL tables. For a majority of sites this will be sufficient. But some sites accumulate so much data that the purge script takes too long to run, in which case partitioning becomes the better solution. The trade off is that partitioning involves a lot more database maintenance. Moreover partitioning is an advanced technique and requires a knowledgeable and skilled DBA. By contrast, running the purge script is straightforward and does not require much DBA knowledge.

Before deciding if partitioning is necessary, you should first try the purge script. Only when you don't get the performance you need (e.g., it takes longer to run than the downtime you assigned for maintenance) should you consider partitioning.

If you only use the purge script in your environment, you can skip the remainder of this chapter. Only continue with this chapter if you plan on using partitioning.

11.1 Initial configuration

Once you have decided to implement database partitioning, you need to do some initial configuration just once.

- Using the information throughout this document, decide which groups you would like to partition.
- For each of those groups, decide which tables you would like to partition, remembering that there are some mandatory tables in each group that must be partitioned.
- For each group, decide on the partition interval.
• Create the partition scripts to partition the BPEL schema. No scripts are supplied, each DBA is responsible for creating the partition scripts appropriate for their environment and choices.

• Edit the purge script (purge_instances_oracle.sql) and remove references to any tables that you partitioned.

11.2 Dropping the partitions

Follow the steps outlined in Figure 1 (page 27). Remember that before dropping any partitions, you MUST first run the purge script (which is indicated in

The verification scripts can be run at any time since they are only queries. They should be run in advance of any planned maintenance so that when downtime has arrived, you know which partitions are eligible to be dropped.
FIGURE 1: FLOW FOR PURGING AND PARTITION DROPPING

Start → Are the verify scripts installed? → Run CL_VERIFY.sql, DL_VERIFY.sql, DC_VERIFY.sql (This step only needs to be done once to install the verify packages)

Are there any partitions? Yes → Is Group 1 partitioned? → Yes → Edit and run CL_EXEC_VERIFY.sql in background → Run purge script → Are there any partitions to be dropped? Yes

No → Is Group 2 partitioned? → Yes → Edit and run DL_EXEC_VERIFY.sql in background → Edit purge script to remove references to all partition tables

Is Group 3 partitioned? → Yes → Edit and run DC_EXEC_VERIFY.sql in background

Are there any partitions to be dropped? Yes → Is Group 1 partitioned? → Yes → Is partition eligible to be dropped? → Yes → Drop Group 1 partition

Is Group 2 partitioned? → Yes → Is partition eligible to be dropped? → Yes → Drop Group 2 partition

Is Group 3 partitioned? → Yes → Is partition eligible to be dropped? → Yes → Drop Group 3 partition

Finish
Appendices
A The BPEL Schema

This appendix provides a short description of some of the key BPEL tables. Although the BPEL schema does not include any foreign keys, Figure 2 shows the relationships between the tables as it would be if foreign keys were present.
A.1 Table descriptions

The following list is a brief description of some of the key BPEL tables.

ATTACHMENT
Attachment persistence table. Attachment parts of BPEL Message variables are stored in this table.

ATTACHEMENT_REF
Attachment reference table. An attachment can be referenced by multiple BPEL instances. The references to an attachment are saved in this table.

AUDIT_DETAILS
Stores details for AUDIT_TRAIL events that are large in size. Details that are smaller than a specific size are stored inline with the events in the AUDIT_TRAIL table.

AUDIT_TRAIL
Stores record of actions taken on an instance (application, system, administrative and errors) and stores the audit trail for BPEL instances. The audit trail viewed from the console is modeled from an XML document. As the instance is worked, each activity writes out events to the audit trail as XML that is compresses and stored in a raw column.

CUBE_INSTANCE
For each BPEL process instance an entry is created in the table. This table contains information like creation date, current state, title, process identifier and the last update date. The table also contains the relationship between parent and child instances in fields: cikey, parent_id, and root_id.

CUBE_SCOPE
Stores the scope data for an instance (for example, all variables declared in the BPEL flow and some internal objects that helps route logic throughout the flow).

DOCUMENT_CI_REF
Stores cube instance references to data stored in the XML DOCUMENT table.

DLV_MESSAGE
Delivery service message table. Callback messages are stored here. Stores callback messages upon receipt. The delivery layer will then attempt to correlate the message with the receiving instance. This table only stores the metadata for the messages (for example, current state, process identifier and receive date).

DLV_SUBSCRIPTION
Stores delivery subscriptions for an instance. Whenever an instance expects a message from a partner (for example, the <receive> or <onMessage> activity) a subscription is written out for that specific receive activity.
DOCUMENT_DLV_MSG_REF
INVOKE_MESSAGE and DLV_MESSAGE has references to XML_DOCUMENT. The relationships between these tables are stored here.

INVOKE_MESSAGES
All asynchronous invocation messages are stored in this table before being dispatched to the engine.

WORK_ITEM
Stores generated work items for the process instances.

XML_DOCUMENT
Stores all large objects in the system (for example, INVOKE_MESSAGE documents, DLV_MESSAGE documents). This table stores the data as binary large objects (BLOB). Separating the document storage from the metadata enables the metadata to change frequently without being impacted by the size of the document.

B Instance & Message States

This appendix provides some information on the various state values and what they mean. Refer to Metalink Note 393170.1 for full details.

B.1 Table CUBE_INSTANCE

• public static final int STATE_INITIATED = 0;

  State value for an instance that has just been created. The instance will only have this value as its state after it has been created by the process domain.

• public static final int STATE_OPEN_RUNNING = 1;

  State value for an instance that has been created and has active activities executing. The instance is not in an exception or error condition.

• public static final int STATE_OPEN_SUSPENDED = 2;

  State value for an instance that is unavailable. Performers of any of the activities that belong to this instance cannot take any action until the instance has returned to the running state.

• public static final int STATE_OPEN_FAULTED = 3;

  State value for an instance that has an activity that has thrown an exception. When an activity throws an exception, the instance is flagged as being in an exception state until the exception is bubbled up, caught and handled.

• public static final int STATE_CLOSED_PENDING_CANCEL = 4;
State value for an instance that has started its cancellation procedure. Since cancelling an instance may involve a great deal of business logic, the amount of time the entire cancellation process may take may be anywhere from seconds to days. During this time, the instance is said to be pending cancellation; an instance may not be acted upon during this time.

- `public static final int STATE_CLOSED_COMPLETED = 5;`
  State value for an instance that has been completed. All activities belonging to this instance have also been completed.

- `public static final int STATE_CLOSED_FAULTED = 6;`
  State value for an instance that has an activity that has thrown an exception while the instance is being cancelled. This state is equivalent to `<code>STATE_OPEN_FAULTED</code>` except that when the exception is resolved, the state transitions back to `CLOSED_PENDING_CANCEL` rather than `STATE_OPEN_RUNNING`.

- `public static final int STATE_CLOSED_CANCELLED = 7;`
  State value for an instance that has been cancelled. All activities belonging to this instance have also been cancelled.

- `public static final int STATE_CLOSED_ABORTED = 8;`
  State value for an instance that has been aborted due to administrative control. All activities belonging to this instance are also moved to the aborted state.

- `public static final int STATE_CLOSED_STALE = 9;`
  State value for an instance who's process has been changed since the process was last accessed. No actions may be performed on the instance. All activities that belong to this instance are also moved to the stale state.

B.2 `DLV_MESSAGE, INVOKE_MESSAGE`

- `public static final int STATE_UNRESOLVED = 0;`
  State value for a message/subscriber inserted into system but has not been correlated with its corresponding subscriber/message.

- `public static final int STATE_RESOLVED = 1;`
  State value for a message/subscriber has been correlated with its corresponding subscriber/message but has not been processed by the BPEL domain (i.e., callback).

- `public static final int STATE_HANDLED = 2;`
State value for a message/subscriber has been processed by the BPEL domain (i.e., callback).

- `public static final int STATE_CANCELLED = 3;`

State value for a message/subscriber has been cancelled. It will not be processed by the BPEL domain.
C Relationships between instances

Each instance of a BPEL process results in many rows being written to many tables. By default, the purge script and the verification scripts operate on the assumption that each BPEL instance is independent. Thus when it is determined the state of an instance is complete, for example, all associated rows (which are spread over many tables) for that instance are removed. From the point of view of a single BPEL instance, this is correct behavior. However from a business or application point of view, a single "instance" may incorporate various individual BPEL instances.

For example, Figure 3 shows BPEL Process A calling a second BPEL process, Process B. At runtime, the instance id (CIKEY) for Process A is 1001, and the CIKEY for Process B is 1002 – they are two independent instances.

**FIGURE 3: A BPEL PROCESS CALLING A SUB-BPEL PROCESS**
Let's say the first BPEL process is doing some validation and it calls the second BPEL process to take some action. Process B may be long running, say several days. However Process A completes quickly and its state is set to 5 (completed), while the state of Process B is still open.

If you were to run the verification scripts and drop your partitions, it's entirely possible that the instance rows for CIKEY=1001 will be dropped, while the instance rows for CIKEY=1002 will not be dropped because its state is still open. (This could happen if Process B runs long enough that Process A is one partition interval while Process B is in another partition interval.)

This does no harm from the point of view of the BPEL engine, but for business reasons, you may require that the instances for Process A are not removed until Process B is complete, as you consider them a single business or application "instance". Perhaps you want to be able to look at the audit trail of Process A while Process B is running, for example.

Even though they are separate BPEL instances, the BPEL engine tracks these parent-child relationships using ROOT_ID which spans parent processes, their child process, and their child process, and so on.

If you do not want to purge complete instances who have child instances still open, set the chk_tree flag to true when executing the CI_EXEC_VERIFY.sql script. Refer to section 4.2 on page 9 for details.