Oracle Maximum Availability Architecture

Automated Database Upgrades using Oracle Active Data Guard and DBMS_ROLLING

Best Practices

ORACLE WHITE PAPER | DECEMBER 2017

.

Introduction

Oracle Active Data Guard provides several new PL/SQL packages and DDL commands to automate the previous manual steps of performing a database rolling upgrade to a new Oracle patch set, database release, or to perform other planned maintenance. The process starts with a primary and physical standby database at the current version and ends with both primary and physical standby database at the new version. The automation includes handling the switchover of production to the new version. It also performs extensive validation at every step of the process. If problems are encountered users can choose to either correct the error and resume the upgrade or roll back to the original state of the configuration. It is implemented using the new DBMS_ROLLING PL/SQL package, which allows you to upgrade the database software in an Oracle Data Guard configuration in a rolling manner. The Rolling Upgrade Using Oracle Active Data Guard feature requires a license for the Oracle Active Data Guard option.

Using the package to perform rolling database version upgrades is not available until the current version of the database is on the first patchset of Oracle Database 12*c* (12.1.0.2). This means that you must use the Transient Logical Standby upgrade procedure when the current database version is 11g (release 1 or 2) or 12.1.0.1. However, you can use the DBMS_ROLLING packages for other rolling maintenance such as:

» Adding partitioning to non-partitioned tables
» Changing BasicFiles LOBs to SecureFiles LOBs
» Changing XMLType stored as CLOB to XMLtype stored as binary XML
» Altering tables to be OLTP-compressed

The DBMS_ROLLING package is idempotent in that any procedure can be rerun after a failure and picks up from the failing step.

Concepts

The rolling process splits databases into two groups, the LEADING group and the TRAILING group.

The LEADING group is defined as the first group of databases to have the maintenance performed. That is the ‘future primary’ database and those designated to protect the future primary in multiple standby configurations.

The TRAILING group is defined as the original primary and the standby databases designated to protect the original primary while the LEADING group is going through maintenance.

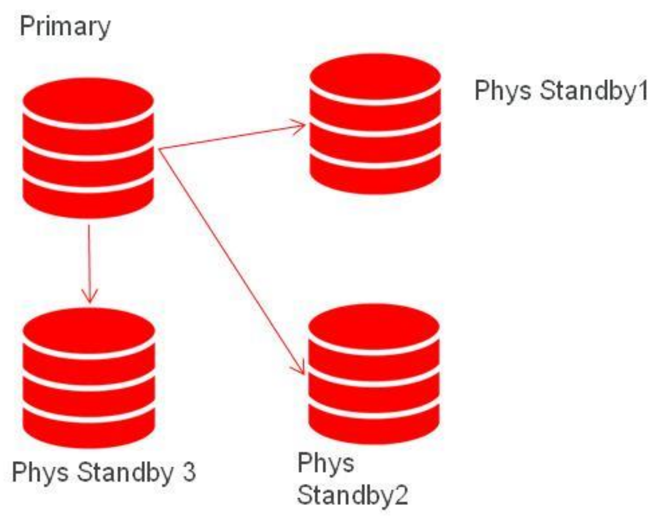

Figure 1: Original configuration

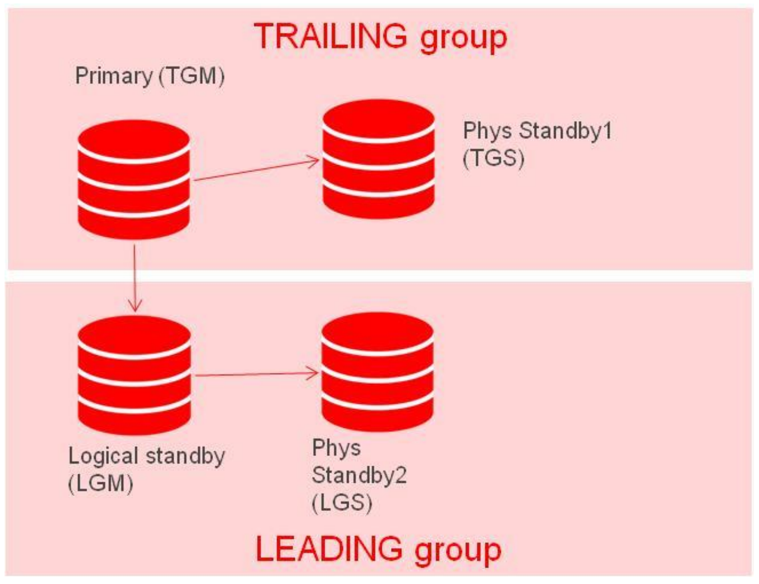

Figure 1: Configuration after START_PLAN execution with LEADING and TRAILING groups

For example, in an upgrade the LEADING group would be upgraded to the new database version while the TRAILING group runs on the old version of the database software.

This concept of groups provides robustness:

» It can handle failures during the rolling upgrade process. The original primary or the Trailing Group Master (TGM) database can fail. You can initiate a regular failover operation to any other physical standby in the trailing group, and then designate the new primary database as the TGM.

» It allows data protection of the Leading Group Master (LGM) that is, designated future primary) during the rolling upgrade process. You can set up physical standbys for the LGM database, and thus protect it during the upgrade process and also achieve Zero Data Loss protection after the upgrade. After the LGM has been successfully upgraded, a failure in the LGM can be accommodated by failing over to any of its physical standby databases in the leading group. You can then designate the failover target database to take over the role of the LGM.

In the simplest case (a single physical standby configuration) there would only be a TGM and LGM.

Choosing a Database Upgrade Method

Database upgrade means taking the database to a later major release, maintenance release, or patch set. The following Oracle features are available to perform database upgrades:

» Upgrading with Database Upgrade Assistant (DBUA)

» Upgrading with Data Guard SQL Apply or Transient Logical Standby Database

» Upgrading with Oracle GoldenGate

The method you choose to perform database upgrades can vary depending on the following considerations:

» Downtime required to complete the upgrade

» Setup time and effort required before the downtime

» Temporary additional resources necessary (for example, disk space or CPU)

» Complexity of the steps allowed to complete the upgrade

The following table lists the methods that you can use for database upgrades, and recommends what method to use for particular cases.

DATABASE UPGRADE METHODS

Upgrade Method	Use This Method When...
Upgrading with Database Upgrade Assistant (DBUA)	Recommended method when the maintenance window is sufficient or when data type constraints prohibit the use of the other methods in this table.
Upgrading with Data Guard SQL Apply or Transient Logical Standby Database	DBUA cannot finish within the maintenance window and the database is not a candidate for Oracle RAC rolling patch upgrade. Use a transient logical standby when the configuration has only a physical standby database.
Upgrading with Oracle GoldenGate	Oracle GoldenGate is already used for complete database replication or when the database version predates Oracle 10*g* (the minimum version for Oracle Data Guard database rolling upgrades), or when additional flexibility for replicating back to the previous version is required (fast fallback option) or where zero downtime upgrades using multi-master replication is required.

Regardless of the upgrade method you use, you should follow the guidelines and recommendations provided in the *Oracle Database Upgrade Guide* and its companion document, "Oracle 11gR2 Upgrade Companion" in My Oracle Support Note 785351.1 at https://support.oracle.com/rs?type=doc&id=785351.1.

Using DBMS_ROLLING

The DBMS_ROLLING package consists of eight procedures that are listed in order of execution below:

INT_PLAN

Sets the target primary (LGM), identifies the databases in the configuration and initializes plan parameters to the defaults. All standby databases are initial assigned the TRAILING group and must be manually changed with SET_PARAMETER. Query DBA_ROLLING_PARAMETERS to view the values set by INIT_PLAN.

Example: exec dbms_rolling.init_plan(‘standby’)

SET_PARAMETER

Used to change the parameters generated by INIT_PLAN. This would be used to change a standby to the LEADING group or set appropriate timeouts for the process. See Oracle Appendix A for a full list of parameters and details.

Example: exec dbms_rolling.set_parameter(‘standby2’,’member’,’LEADING’);

BUILD_PLAN

Validates plan parameters and creates or modifies a rolling plan. Query DBA_ROLLING_PLAN for all steps generated by BUILD_PLAN. BUILD_PLAN must be run before START_PLAN and must be re-run after parameter changes via SET_PARAMETER.

Example: exec dbms_rolling.build_plan;

START_PLAN

Starts the rolling operation and executes all steps of the START phase of the plan as described in DBA_ROLLING_PLAN including the creating of guaranteed restore points on all databases which will be used during ROLLBACK or FINISH_PLAN operations .

Upon successful completion of START_PLAN the future primary database passed into INIT_PLAN will be a fully configured logical standby and LEADING group master (LGM). Standby databases designated as LEADING group will be LEADING group standbys (LGS) receiving redo from the LGM.

Example: exec dbms_rolling.start_plan;

SWITCHOVER

Executed on the TGM after maintenance is completed on the LEADING group. This procedure executes the switchover to the logical standby LGM from the primary TGM. After completion of the switchover the LGM will be a primary database open read-write and the TGM will be a mounted logical standby database.

Example: exec dbms_rolling.switchover;

FINISH_PLAN

Executing FINISH_PLAN converts the LGM to a physical standby database and flashes all TRAILING group databases to the guaranteed restore point created in START_PLAN. Media recovery is then started and all redo from that SCN is applied from the LGM thus synchronizing the databases.

Example: exec dbms_rolling.finish_plan;

ROLLBACK_PLAN

Restores the entire configuration back to the way it was prior to START_PLAN. This can only be called if SWITCHOVER has not been executed since START_PLAN was called.

Example: exec dbms_rolling.rollback_plan;

DESTROY_PLAN

This procedure destroys any existing upgrade plan, its parameters, and all resources associated with a rolling operation.

Example: exec dbms_rolling.destroy_plan;

Process Flow

The following diagram depicts the process flow for the DBMS_ROLLING.

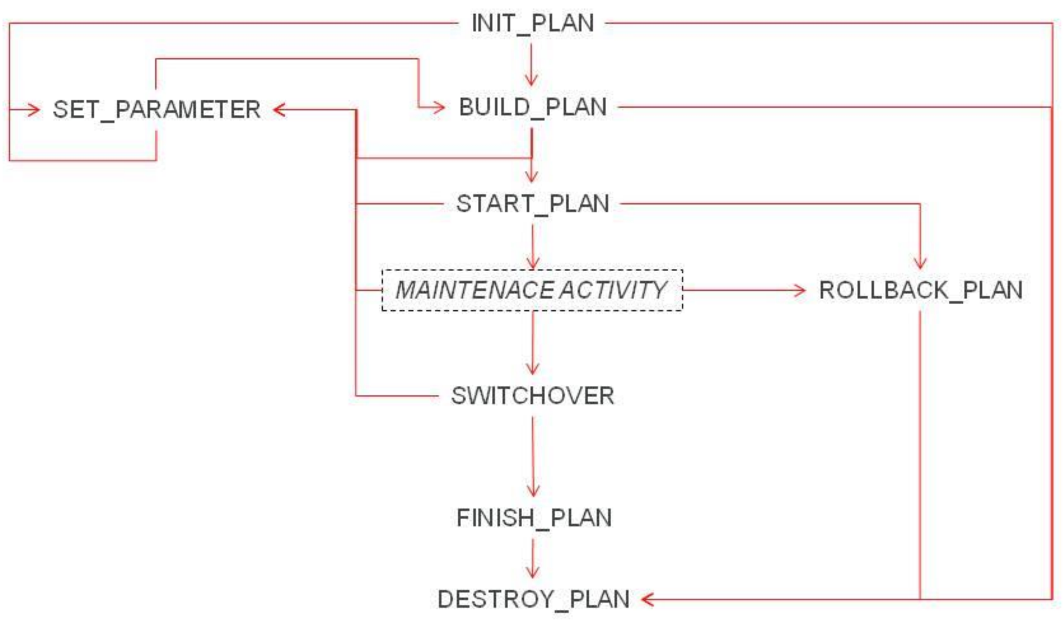

Figure 2. Process Flow

DBMS_ROLLING Phases and Downtime/Fallback Options

The following table describes the various phases of the DBMS_ROLLING process and any Recovery Time Objective (RTO) or Recovery Point Objective (RPO) implications for that phase. In addition, fallback options are listed for each phase.

DBMS_ROLLING PHASES

DBMS_ROLLING Phase	Action Performed	Service Downtime	RPO, RTO Implications	Fallback Options

Prerequisite Phase	When upgrading from 11g or 12.1.0.1, query the DBA_LOGSTDBY_UNSUPPORTED (CDB_LOGSTDBY_UNSUPPORTED if the primary is a container database) view. When upgrading from 12.1.0.2 query the DBA_ROLLING_UNSUPPORTED view. If no rows are returned, proceed. If rows are returned, decide to limit application function or take full downtime.	None	None	None required
dbms_rolling.init_plan	Identifies the databases in the configuration and initializes plan parameters	None	RPO not affected RTO not affected	Execute dbms_rolling.destroy_plan to remove the initialized plan.
dbms_rolling.build_plan	Validates plan parameters and creates or modifies a rolling plan	None	RPO not affected RTO not affected	Execute dbms_rolling.destroy_plan to remove the plan build.
dbms_rolling.start_plan	Starts the rolling operation and executes all steps of the START phase. Upon successful completion of START_PLAN the future primary database will be a fully configured logical standby.	None	RPO increases slightly during bounce when converting physical standby to a logical standby RTO: • Increases to ~ 2 minutes as the physical standby is converted to a logical standby. • Once converted RTO is less than previous if failover to logical standby is acceptable. • RTO increases by ~ 3 minutes if we execute dbms_rolling.rollback_plan which converts logical standby back into a physical standby. • With additional trailing physical standby RTO is unaffected	Execute dbms_rolling.rollback_plan which restores the entire configuration back to the way it was prior to start_plan. This can only be called if switchover has not been executed since start_plan was called.
Upgrade database	Upgrade the database and its installed options from 12.1.0.2 to 12.2.0.1 using Database Upgrade Assistant (recommended) or manually.	None	RPO not affected RTO: • Prior to upgrading the logical standby the RTO is not affected. • RTO increases after or during upgrade as prior to a failover to the logical standby the database upgrade would need to be flashed back. This flashback adds 1 to 2 minutes	Fallback options: • If upgrading with dbua then flashback database to restore point created prior to the upgrade. Testing shows that the flashback of an upgraded database completes in about 30 seconds. • If upgrading manually then the restore point must be created manually prior to the upgrade.

			to the overall RTO • With additional physical standbys, the RTO is unaffected	Flashback of an upgraded database completes in about 30 seconds. • Restore backup of database that was taken just prior to upgrade.
dbms_rolling.switchover	Executed on the primary after the upgrade has been completed on the logical standby. This procedure executes the switchover from the logical standby to the primary.	15 to 20 Seconds	RPO: • Not affected prior to switchover • After the switchover any update to the new primary database results in growing RPO RTO: • Increases by ~30 seconds during the switchover process • If the user decides to leave the new logical standby at the old version after the switchover completes then RTO continues to grow • RTO also increases if user decides to run downgrade on new primary database	Fallback options: • If no updates have been made to the new 12.2.0.1 primary then perform a switchover to return the starting configuration • If updates have occurred on the new 12.2.0.1 primary the fallback option is to run downgrade scripts. This is a zero data loss fallback but comes with increased RTO • Once updates start on the new 12.2.0.1 primary then any fallback options that involve the standby will result in data loss. Activate 12.1.0.2 logical standby and redirect sessions
Startup standby (originally the TGM) in the 12.2.0.1 home	Shutdown the logical standby and bring back to mount on 12.2.0.1 software.	None	RPO increases slightly (15 seconds) during the bounce of the standby RTO increases slightly (15 seconds) during the bounce of the standby	Fallback options: • Restart standby on 12.1.0.2 software
dbms_rolling.finish_plan	Executing FINISH_PLAN converts the logical standby (former primary) to a physical standby database and flashes all trailing group databases to the guaranteed restore point created in START_PLAN. Media recovery is then started and all redo from that scn is applied.	None	RPO increases slightly (15 seconds) during the bounce of the standby RTO: • Increased RTO as the logical standby is flashback to a physical standby. • RTO begins to decrease as the physical standby begins apply redo from the upgrade process. • Once all upgraded redo has been applied and physical standby is in sync with the primary the RTO returns to normal values.	Fallback options: • If conversion of the logical standby into a physical standby fails then fallback option is to reinstate the standby with a new backup of the primary database.

DBMS_ROLLING Process

The following section will provide a detailed example of how to perform an upgrade using DBMS_ROLLING.

1. **DBMS_Rolling Process Pre-requisites**

Before you perform a rolling upgrade, you should determine whether any of the tables involved contain data types that are unsupported on logical standby databases. To do this, you query the DBA_ROLLING_UNSUPPORTED view.

A rolling upgrade performed using DBMS_ROLLING supports more object types than a manual rolling upgrade operation. For example, only upgrades performed with DBMS_ROLLING support queue tables. Additionally, a rolling upgrade performed using DBMS_ROLLING also supports more PL/SQL packages.

Refer to Appendix B for information on replication support in the context of rolling upgrades performed using the DBMS_ROLLING package.

Note that when the dictionary build is executed, supplemental logging is enabled implicitly on the Primary (TGM). Supplemental logging can affect database performance and should be assessed prior to performing a rolling upgrade to determine any impact. Also, supplemental logging must be manually disabled once the upgrade procedure has completed.

From a Data Guard broker configuration the starting configuration looks like:

```
Configuration - db1

Protection Mode: MaxPerformance

Members:

db1 - Primary database
db1stby1 - Physical standby database
db1stby2 - Physical standby database
db1stby3 - Physical standby database
```

As a best practice we want a physical standby protecting the starting existing primary database and then another physical standby protecting the standby that will eventually become the new primary database. To achieve this, the following redo route changes were made to adjust the broker configuration:

```
edit database db1 set property redoroutes='(LOCAL : db1stby1 SYNC, db1stby2 ASYNC)';
edit database db1stby1 set property redoroutes='(LOCAL : db1 SYNC)';
edit database db1stby2 set property redoroutes='(db1 : db1stby3 ASYNC)';
edit database db1stby3 set property redoroutes='(LOCAL : db1stby2 SYNC)';
```

The broker configuration now looks like:

```
Configuration - db1

Protection Mode: MaxPerformance
Members:
db1 - Primary database
 db1stby1 - Physical standby database
 db1stby2 - Physical standby database
 db1stby3 - Physical standby database (receiving current redo)

Fast-Start Failover: DISABLED
```

This concept of groups provides robustness:

» It can handle failures during the rolling upgrade process. The original primary or the TGM database can fail. You can initiate a regular failover operation to any other physical standby in the trailing group, and then designate the new primary database as the TGM.

» It allows data protection of the LGM (that is, designated future primary) during the rolling upgrade process. You can set up physical standbys for the LGM database, and thus protect it during the upgrade process and also achieve Zero Data Loss after the upgrade.

After the LGM has been successfully upgraded, a failure in the LGM can be accommodated by failing over to any of its physical standby databases. You can then designate the failover target database to take over the role of the LGM.

In the simplest case there would only be a TGM and LGM.

2. Performing DBMS_ROLLING Upgrade: Step-by-Step Process

1. Initialize the plan. Sets the target primary (LGM), identifies the databases in the configuration and initializes plan parameters to the defaults. All standby databases are initial assigned the TRAILING group and must be manually changed with SET_PARAMETER. Query DBA_ROLLING_PARAMETERS to view the values set by INIT_PLAN.

```
SQL> show parameter log_archive_config

NAME TYPE VALUE
------------------------------------ ----------- ------------------------------
log_archive_config string dg_config=(db1,db1stby1,db1stb
y2,db1stby3)

SQL> exec dbms_rolling.init_plan('db1stby11');

PL/SQL procedure successfully completed.
```

2. Display the parameters for the current plan build

```
SQL> col name format a30
SQL> col scope format a15
SQL> col curval format a30
SQL> set pages 999|
SQL> select scope, name, curval from dba_rolling_parameters order by scope, name;

SCOPE NAME CURVAL
--------------- ------------------------------ ------------------------------
db1 INVOLVEMENT FULL
db1 MEMBER NONE
db1stby1 INVOLVEMENT FULL
db1stby1 MEMBER TRAILING
db1stby2 INVOLVEMENT FULL
db1stby2 MEMBER TRAILING
db1stby3 INVOLVEMENT FULL
db1stby3 MEMBER TRAILING
ACTIVE_SESSIONS_TIMEOUT 3600
ACTIVE_SESSIONS_WAIT 0
BACKUP_CONTROLFILE rolling_change_backup.f
DICTIONARY_LOAD_TIMEOUT 3600
DICTIONARY_LOAD_WAIT 0
DICTIONARY_PLS_WAIT_INIT 300
DICTIONARY_PLS_WAIT_TIMEOUT 3600
EVENT_RECORDS 10000
FAILOVER 0
GRP_PREFIX DBMSRU_
IGNORE_BUILD_WARNINGS 1
IGNORE_LAST_ERROR 0
LAD_ENABLED_TIMEOUT 600
LOG_LEVEL INFO
READY_LGM_LAG_TIME 600
READY_LGM_LAG_TIMEOUT 60
READY_LGM_LAG_WAIT 0
SWITCH_LGM_LAG_TIME 600
SWITCH_LGM_LAG_TIMEOUT 60
SWITCH_LGM_LAG_WAIT 1
```

```
SWITCH_LGS_LAG_TIME 60
SWITCH_LGS_LAG_TIMEOUT 60
SWITCH_LGS_LAG_WAIT 0
UPDATED_LGS_TIMEOUT 10800
UPDATED_LGS_WAIT 1
UPDATED_TGS_TIMEOUT 10800
UPDATED_TGS_WAIT 1

35 rows selected.
```

3. Build the plan. Validates plan parameters and creates or modifies a rolling plan. Query DBA_ROLLING_PLAN for all steps generated by BUILD_PLAN.BUILD_PLAN must be run before START_PLAN and must be re-run after parameter changes via SET_PARAMETER.

```
SQL> exec dbms_rolling.build_plan;

PL/SQL procedure successfully completed.
```

4. Display the plan. DBA_ROLLING_PLAN displays the instructions which constitute the active upgrade plan. Each row in DBA_ROLLING_PLAN identifies a specific instruction scheduled to execute at a specific database. Instructions are created as a result of successful calls to the DBMS_ROLLING.BUILD_PLAN procedure. During execution, groups of instructions are scheduled in batches to execute at remote databases. Groups of instructions are guaranteed to complete in BATCHID order.

```
SQL> col instid format 999
col target format a10
col phase format a10
col description format a65
set lines 99
set pages 999

SELECT instid, target, phase, description FROM DBA_ROLLING_PLAN;

INSTID TARGET PHASE DESCRIPTION
------ ---------- ---------- ---------------------------------------------------------------
--
1 db1 START Verify database is a primary
2 db1 START Verify MAXIMUM PROTECTION is disabled
3 db1stby1 START Verify database is a physical standby
4 db1stby1 START Verify physical standby is mounted
5 db1stby2 START Verify database is a physical standby
6 db1stby2 START Verify physical standby is mounted
7 db1stby3 START Verify database is a physical standby
8 db1stby3 START Verify physical standby is mounted
9 db1 START Verify server parameter file exists and is modifiable
10 db1stby1 START Verify server parameter file exists and is modifiable
11 db1stby2 START Verify server parameter file exists and is modifiable
12 db1stby3 START Verify server parameter file exists and is modifiable
13 db1 START Verify Data Guard Broker configuration is disabled
14 db1stby1 START Verify Data Guard Broker configuration is disabled
15 db1stby2 START Verify Data Guard Broker configuration is disabled
16 db1stby3 START Verify Data Guard Broker configuration is disabled
17 db1 START Verify flashback database is enabled
18 db1 START Verify available flashback restore points
19 db1stby1 START Verify flashback database is enabled
20 db1stby1 START Verify available flashback restore points
21 db1stby2 START Verify flashback database is enabled
22 db1stby2 START Verify available flashback restore points
23 db1stby3 START Verify flashback database is enabled
24 db1stby3 START Verify available flashback restore points
```

```
25 db1stby1 START Stop media recovery
26 db1stby2 START Stop media recovery
27 db1stby3 START Stop media recovery
28 db1stby1 START Drop guaranteed restore point DBMSRU_INITIAL
29 db1stby1 START Create guaranteed restore point DBMSRU_INITIAL
30 db1stby2 START Drop guaranteed restore point DBMSRU_INITIAL
31 db1stby2 START Create guaranteed restore point DBMSRU_INITIAL
32 db1stby3 START Drop guaranteed restore point DBMSRU_INITIAL
33 db1stby3 START Create guaranteed restore point DBMSRU_INITIAL
34 db1 START Drop guaranteed restore point DBMSRU_INITIAL
35 db1 START Create guaranteed restore point DBMSRU_INITIAL
36 db1stby1 START Start media recovery
37 db1stby1 START Verify media recovery is running
38 db1stby2 START Start media recovery
39 db1stby2 START Verify media recovery is running
40 db1stby3 START Start media recovery
41 db1stby3 START Verify media recovery is running
42 db1 START Verify user_dump_dest has been specified
43 db1 START Backup control file to rolling_change_backup.f
44 db1stby1 START Verify user_dump_dest has been specified
45 db1stby1 START Backup control file to rolling_change_backup.f
46 db1stby2 START Verify user_dump_dest has been specified
47 db1stby2 START Backup control file to rolling_change_backup.f
48 db1stby3 START Verify user_dump_dest has been specified
49 db1stby3 START Backup control file to rolling_change_backup.f
50 db1 START Get current redo branch of the primary database
51 db1stby1 START Wait until recovery is active on the primary's redo branch
52 db1stby1 START Stop media recovery
53 db1 START Execute dbms_logstdby.build
54 db1stby1 START Convert into a transient logical standby
55 db1stby1 START Open database
56 db1stby1 START Get redo branch of transient logical standby
57 db1stby1 START Get reset scn of transient logical redo branch
58 db1stby1 START Configure logical standby parameters
59 db1stby1 START Start logical standby apply
60 db1stby1 START Enable compatibility advance despite presence of GRPs
61 db1 START Log pre-switchover instructions to events table
62 db1stby1 START Record start of user upgrade of db1stby1
63 db1stby1 SWITCH Verify database is in OPENRW mode
64 db1stby1 SWITCH Record completion of user upgrade of db1stby1
65 db1stby1 SWITCH Scan LADs for presence of db1 destination
66 db1stby1 SWITCH Scan LADs for presence of db1stby2 destination
67 db1stby1 SWITCH Scan LADs for presence of db1stby3 destination
68 db1stby1 SWITCH Test if db1 is reachable using configured TNS service
69 db1stby1 SWITCH Test if db1stby2 is reachable using configured TNS service
70 db1stby1 SWITCH Test if db1stby3 is reachable using configured TNS service
71 db1 SWITCH Enable log file archival to db1stby1
72 db1stby1 SWITCH Start logical standby apply
73 db1stby1 SWITCH Archive all current online redo logs
74 db1stby1 SWITCH Wait until apply lag has fallen below 600 seconds
75 db1 SWITCH Log post-switchover instructions to events table
76 db1 SWITCH Switch database to a logical standby
77 db1stby1 SWITCH Wait until end-of-redo has been applied
78 db1stby2 SWITCH Wait until end-of-redo has been applied
79 db1stby3 SWITCH Wait until end-of-redo has been applied
80 db1 SWITCH Disable log file archival to db1stby2
81 db1 SWITCH Disable log file archival to db1stby3
82 db1 SWITCH Archive all current online redo logs
83 db1stby1 SWITCH Switch database to a primary
84 db1 SWITCH Enable compatibility advance despite presence of GRPs
85 db1stby2 SWITCH Enable compatibility advance despite presence of GRPs
86 db1stby3 SWITCH Enable compatibility advance despite presence of GRPs
87 db1stby2 SWITCH Stop media recovery
88 db1stby3 SWITCH Stop media recovery
89 db1 SWITCH Synchronize plan with new primary
90 db1 FINISH Verify only a single instance is active
```

```
91 db1 FINISH Verify database is mounted
92 db1 FINISH Flashback database
93 db1 FINISH Convert into a physical standby
94 db1stby2 FINISH Verify database is mounted
95 db1stby2 FINISH Flashback database
96 db1stby3 FINISH Verify database is mounted
97 db1stby3 FINISH Flashback database
98 db1stby1 FINISH Verify database is open
99 db1stby1 FINISH Save the DBID of the new primary
100 db1stby1 FINISH Save the logminer session start scn
101 db1 FINISH Wait until transient logical redo branch has been registered
102 db1stby2 FINISH Wait until transient logical redo branch has been registered
103 db1stby3 FINISH Wait until transient logical redo branch has been registered
104 db1 FINISH Start media recovery
105 db1stby2 FINISH Start media recovery
106 db1stby3 FINISH Start media recovery
107 db1 FINISH Wait until apply/recovery has started on the transient branch
108 db1stby2 FINISH Wait until apply/recovery has started on the transient branch
109 db1stby3 FINISH Wait until apply/recovery has started on the transient branch
110 db1 FINISH Wait until upgrade redo has been fully recovered
111 db1stby2 FINISH Wait until upgrade redo has been fully recovered
112 db1stby3 FINISH Wait until upgrade redo has been fully recovered
113 db1 FINISH Prevent compatibility advance if GRPs are present
114 db1stby1 FINISH Prevent compatibility advance if GRPs are present
115 db1stby2 FINISH Prevent compatibility advance if GRPs are present
116 db1stby3 FINISH Prevent compatibility advance if GRPs are present
117 db1 FINISH Drop guaranteed restore point DBMSRU_INITIAL
118 db1stby1 FINISH Drop guaranteed restore point DBMSRU_INITIAL
119 db1stby2 FINISH Drop guaranteed restore point DBMSRU_INITIAL
120 db1stby3 FINISH Drop guaranteed restore point DBMSRU_INITIAL

120 rows selected.
```

The columns in this view display the following information:

» INSTID - The Instruction ID, which is the order in which the instruction is to be performed. Instructions are typically performed in groups.

» PHASE - Every instruction in the upgrade plan is associated with a particular phase. A phase is a logical grouping of instructions which is performed by a procedure in the DBMS_ROLLING PL/SQL package. When a DBMS_ROLLING procedure is invoked, all of the associated instructions in the upgrade plan for that phase are executed. Possible phases are as follows:

  - START: Consists of activities related to setup such as taking restore points, instantiation of the transient logical standby database, and configuration of LGS databases. Activities in this phase are initiated when you call the DBMS_ROLLING.START_PLAN procedure.
  - SWITCH: Consists of activities related to the switchover of the transient logical standby into the new primary database. Activities in this phase are initiated when you call the DBMS_ROLLING.SWITCHOVER procedure.
  - FINISH: Consists of activities related to configuring standby databases for recovery of the upgrade redo. Activities in this phase are initiated when you call the DBMS_ROLLING.FINISH_PLAN procedure.
    - EXEC_STATUS - The overall status of the instruction.
    - PROGRESS - The progress of an instruction's execution. A value of REQUESTING indicates an instruction is being transmitted to a target database for execution. A value of EXECUTING indicates the instruction is actively being executed. A value of REPLYING indicates completion information is being returned.
    - DESCRIPTION - The specific operation that is scheduled to be performed.
    - TARGET - The site at which a given instruction will be performed.
    - EXEC_INFO - Additional contextual information related to the instruction.

Upgrade plans need to be revised after any change to the rolling upgrade or database configuration. A configuration change could include any of the following:

- init.ora parameter file changes at any of the databases participating in the rolling upgrade
- database role changes as a result of failover events
- rolling upgrade parameter changes

To revise an active upgrade plan, you simply call the BUILD_PLAN procedure again. In some cases, the BUILD_PLAN procedure may raise an error if a given change cannot be accepted. For example, setting the ACTIVE_SESSIONS_WAIT parameter will have no effect if the switchover has already occurred.

It is recommended that you call the BUILD_PLAN procedure to process a group of parameter changes rather than processing parameters individually.

The following example demonstrates how to configure the plan to wait for the apply lag to fall below 60 seconds before switching over to the future primary:

```
DBMS_ROLLING.SET_PARAMETER('SWITCH_LGM_LAG_WAIT', '1');

DBMS_ROLLING.SET_PARAMETER('SWITCH_LGM_LAG_TIME', '60');
```

The following example demonstrates resetting the LOG_LEVEL global parameter back to its default value.

```
DBMS_ROLLING.SET_PARAMETER (
  name=>'LOG_LEVEL',
  value=>NULL);
```

5. Run start plan. Starts the rolling operation and executes all steps of the START phase of the plan as described in DBA_ROLLING_PLAN including the creating of guaranteed restore points on all databases which will be used during ROLLBACK or FINISH_PLAN operations. Upon successful completion of START_PLAN the future primary database passed into INIT_PLAN will be a fully configured logical standby and LEADING group master (LGM). Standby databases designated as LEADING group will be LEADING group standbys (LGS) receiving
redo from the LGM.

```
SQL> exec dbms_rolling.start_plan

PL/SQL procedure successfully completed.
```

Dictionary build times on EBS R12.2.5 from the EBS bench mark kit, a full install, took approximately 70 seconds. More than the number of objects the biggest driver of the build time is the following:

"DBMS_LOGSTDBY.BUILD waits for all transactions (including distributed transactions) that are active at the time of the procedure invocation to complete before returning."

So long running transactions could hold up the completion. The good news is I have seen no application impact from a dictionary build and it is done online.

In addition, when dictionary build is performed the following command is performed implicitly: alter database add supplemental log data (primary key, unique index) columns. Prior to start_plan being ran the database started with the following status:

```
SQL> select SUPPLEMENTAL_LOG_DATA_MIN,SUPPLEMENTAL_LOG_DATA_PK,SUPPLEMENTAL_LOG_DATA_UI from
v$database;

SUPPLEME SUP SUP
-------- --- ---
NO       NO  NO
```

After start_plan has finished the status changes to:

```
SQL>  select SUPPLEMENTAL_LOG_DATA_MIN,SUPPLEMENTAL_LOG_DATA_PK,SUPPLEMENTAL_LOG_DATA_UI from
v$database;

SUPPLEME SUP SUP
-------- --- ---
IMPLICIT YES YES
```

Note that once the DBMS_ROLLING process completes supplemental logging will still be enabled and will need to be disabled manually.

6. Perform upgrade on the standby (now a logical standby) using the Database Upgrade Assistant (dbua) or your preferred method. Follow the Upgrade Guide for upgrading your database.

```
SQL> exec dbms_rolling.switchover;

PL/SQL procedure successfully completed.
```

7. The former primary is now a logical standby/ former standby is now a primary

```
SQL> select database_role,open_mode from v$database;

DATABASE_ROLE OPEN_MODE
---------------- --------------------
LOGICAL STANDBY READ WRITE

SQL> select database_role,open_mode from v$database;

DATABASE_ROLE OPEN_MODE
---------------- --------------------
PRIMARY READ WRITE
```

8. Start the former primary instance under the new home.

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> exit
```

9. Copy init.ora and password file to the new ORACLE_HOME/dbs and applicable tnsnames.ora descriptors to the proper tnsnames.ora file under the new home. Set the environment to use the new home and mount the standby instance.

```
SQL> startup mount
ORACLE instance started.
Total System Global Area 2.5770E+10 bytes
Fixed Size 6870952 bytes
Variable Size 3422554200 bytes
Database Buffers 2.2012E+10 bytes
Redo Buffers 328671232 bytes
Database mounted.
```

10. On the new primary execute DBMS_ROLLING.FINISH_PLAN in order to convert the new standby to a physical standby, flash it back and roll it forward(see dba_rolling_plan for all steps executed by finish_plan)

```
SQL> exec dbms_rolling.finish_plan;

PL/SQL procedure successfully completed.
```

11. Primary is mounted with apply running

```
SQL> select database_role,open_mode from v$database;

DATABASE_ROLE OPEN_MODE
---------------- --------------------
PHYSICAL STANDBY MOUNTED

SQL> select status from v$managed_standby where process='MRP0';

STATUS
------------
APPLYING_LOG
```

3. Post DBMS_ROLLING Upgrade Tasks :

Note that once the DBMS_ROLLING process completes supplemental logging will still be enabled and will need to be disabled manually. To disable supplemental logging run the following command:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;
```

Rolling Back a Rolling Upgrade

To roll back a rolling upgrade procedure, you can call the DBMS_ROLLING.ROLLBACK_PLAN procedure, as follows:

```
DBMS_ROLLING.ROLLBACK_PLAN;
```

The ROLLBACK_PLAN procedure has the following requirements:

» The ROLLBACK_PLAN procedure can only be called if the DBMS_ROLLING.SWITCHOVER procedure has not been previously called.

» Before you can use the ROLLBACK_PLAN procedure you must set the transient logical standby database back to a mounted state because a flashback database is imminent.

» If the Oracle Database software was already upgraded, then you must restart the resultant physical standbys on the older version, and start media recovery.

Appendix A

VALID VALUES FOR DBMS_ROLLING.SET_PARAMETER PROCEDURE

Parameter Name	Global?	Description	Default
ACTIVE_SESSIONS_TIMEOUT	Yes	The maximum amount of time in seconds to enforce ACTIVE_SESSIONS_WAIT before halting the rolling upgrade. This parameter is only valid if ACTIVE_SESSIONS_WAIT is set to 1.	3600
ACTIVE_SESSIONS_WAIT	Yes	Whether the switchover operation will wait for active sessions to finish. If set to 1, the SWITCHOVER procedure waits for active sessions to compete. If set to 0, the SWITCHOVER procedure kills active sessions to expedite the switchover.	0
BACKUP_CONTROLFILE	Yes	File name of the backup control file that is created during a rolling upgrade.	rolling_change_backup.f
DGBROKER	Yes	Use Data Guard broker for managing apply, recovery, and log archive destinations.	1 if broker is enabled, 0 otherwise.
DICTIONARY_LOAD_TIMEOUT	Yes	The maximum amount of time in seconds to enforce DICTIONARY_LOAD_WAIT before halting the rolling upgrade. This parameter is only valid if DICTIONARY_LOAD_WAIT is set to 1.	3600
DICTIONARY_LOAD_WAIT	Yes	Whether the instantiation of the transient logical standby will include a wait for the complete loading of the data dictionary snapshot in redo. If set to 1, then the START_PLAN procedure will not return until the dictionary has been completely loaded. If set to 0, then the START_PLAN procedure will only verify that the loading of the dictionary has started.	0
DICTIONARY_PLS_WAIT_INIT	Yes	The time in seconds to wait in between attempts to quiesce PL/SQL activity in order to write the data dictionary to redo.	300
DICTIONARY_PLS_WAIT_TIMEOUT	Yes	The maximum amount of time in seconds to attempt to quiesce PL/SQL activity in order to write the data dictionary to redo.	3600
EVENT_RECORDS	Yes	The maximum number of records to permit in DBA_ROLLING_EVENTS	10000
FAILOVER	Yes	Automatically attempt to adjust the upgrade plan as a result of a failover event. This parameter resets its value to 0 upon completion of a subsequent call to BUILD_PLAN.	0
GRP_PREFIX	n/a	Execution of procedures in DBMS_ROLLING results in a number of Guaranteed Restore Points (GRP) taken in various databases participating in the Data Guard configuration. All such GRPs have the same prefix in their names. You can use this parameter to override the default prefix.	DBMSRU
IGNORE_BUILD_WARNINGS	Yes	Ignore warnings which would otherwise raise exceptions during execution of the BUILD_PLAN procedure.	1
IGNORE_LAST_ERROR	Yes	Ignore last encountered error upon startup of next rolling operation. This parameter resets its value to 0 upon invocation of a procedure call which resumes the rolling upgrade.	0
LAD_ENABLED_TIMEOUT	Yes	The maximum time in seconds to wait for a recently enabled log archive destination to reach a VALID state.	600
LOG_LEVEL	Yes	Logging level for the DBS_ROLLING PL/SQL package. A value of INFO results in the logging of errors and relevant non-fatal warnings. A value of FULL results in the logging of all events.	INFO

MEMBER	No	The upgrade group in which the specified database is a member. A value of LEADING indicates that the standby is a member of the leading upgrade group. As such, it is a standby of the Leading Group Master (LGM). The LGM is the database which is converted into the transient logical standby, and which becomes the new primary after the switchover. A value of TRAILING indicates that the standby is a member of the trailing upgrade group. As such, it is a standby of the Trailing Group Master (TGM). The TGM is the original primary database.	LEADING
READY_LGM_LAG_TIME	Yes	The apply lag time in seconds associated with the READY_LGM_LAG_WAIT parameter.	600
READY_LGM_LAG_TIMEOUT	Yes	The maximum amount of time in seconds to enforce READY_LGM_LAG_WAIT before halting the rolling upgrade. This parameter is only valid if READY_LGM_LAG_WAIT is set to 1.	60
READY_LGM_LAG_WAIT	Yes	Whether the START_PLAN procedure will wait for the apply lag on the leading group master to fall below READY_LGM_LAG_TIME seconds before returning control back to the user. If set to 1, the wait is performed. If set to 0, the wait is not performed.	0
SWITCH_LGM_LAG_TIME	Yes	The apply lag time in seconds associated with the SWITCH_LGM_LAG_WAIT parameter.	600
SWITCH_LGM_LAG_TIMEOUT	Yes	The maximum amount of time in seconds to enforce SWITCH_LGM_LAG_WAIT before halting the rolling upgrade. This parameter is only valid if SWITCH_LGM_LAG_WAIT is set to 1.	60
SWITCH_LGM_LAG_WAIT	Yes	Whether the SWITCHOVER procedure will wait for the apply lag on the leading group master to fall below SWITCH_LGM_LAG_TIME seconds before initiating the switchover. If set to 1, the wait is performed. If set to 0, the wait is not performed.	1
SWITCH_LGS_LAG_TIME	Yes	The apply lag time in seconds associated with the SWITCH_LGS_LAG_WAIT parameter.	60
SWITCH_LGS_LAG_TIMEOUT	Yes	The maximum amount of time in seconds to enforce SWITCH_LGS_LAG_WAIT before halting the rolling upgrade. This parameter is only valid if SWITCH_LGS_LAG_WAIT is set to 1.	60
SWITCH_LGS_LAG_WAIT	Yes	Whether the SWITCHOVER procedure will wait for the apply lag on the leading group standbys to fall below SWITCH_LGS_LAG_TIMEseconds before initiating the switchover. If set to 1, the wait is performed. If set to 0, the wait is not performed.	0
UPDATED_LGS_TIMEOUT	Yes	The maximum amount of time in seconds to enforce UPDATED_LGS_WAIT before halting the rolling upgrade. This parameter is only valid if UPDATED_LGS_WAIT is set to 1.	10800
UPDATED_LGS_WAIT	Yes	Whether the SWITCHOVER procedure will wait for the leading group standbys to complete recovery of all upgrade redo before initiating the switchover. If set to 1, the wait is performed. If set to 0, the wait is not performed	1
UPDATED_TGS_TIMEOUT	Yes	The maximum amount of time in seconds to enforce UPDATED_TGS_WAIT before halting the rolling upgrade. This parameter is only valid if UPDATED_TGS_WAIT is set to 1.	10800

`UPDATED_TGS_WAIT`	Yes	Whether the `FINISH_PLAN` procedure will wait for the trailing group standbys to complete recovery of all upgrade redo before returning control to the user. If set to 1, the wait is performed. If set to 0, the wait is not performed.	1

Appendix B

The following is a list of unsupported data types in the context of rolling upgrades performed using the DBMS_ROLLING package.

Unsuported datatypes as of 12.2.0.1:

Tables with columns of the following types are not supported by Logical Standby.

1. Nested Tables
2. Identity Columns
3. ROWID
4. BFILE
5. Temporal Validity columns
6. PKREF
7. PKOID
8. SDO_RDF_TRIPLE_S

Unsupported partitioning/table organizations methods:

Tables that are partitioned with the following methods are not supported by Logical Standby:

1. Reference partitioning
2. System partitioning

DBA_LOGSTDBY_NOT_UNIQUE

The DBA_LOGSTDBY_NOT_UNIQUE view displays all tables that have no primary and no non-null unique indexes. If these tables also have out-of-line columns, e.g. LOB, XML, etc. the table should not be replicated as data divergence will likely occur. Replication of tables without indexes will likely cause table scans/performance issues during replication.

Tables without Scalars:

Tables with only LONG, LONG RAW, LOB (CLOB/BLOB), XML, ADT, VARRAY or BFILE columns (or some mix thereof) are not supported as these types cannot be used to identify a row during replication.

NON-replicated schemas:

The following schemas are automatically skipped by Logical Standby.

ANONYMOUS	APPQOSSYS	AUDSYS
BI	CTXSYS	DBSFWUSER
DBSNMP	DIP	DMSYS
DVF	DVSYS	EXDSYS
EXFSYS	GGSYS	GSMADMIN_INTERNAL
GSMCATUSER	GSMUSER	LBACSYS
MDSYS	MGMT_VIEW	MTSSYS
ODM	ODM_MTR	OJVMSYS
OLAPSYS	ORACLE_OCM	ORDDATA
ORDPLUGINS	ORDSYS	OUTLN
REMOTE_SCHEDULER_AGENT	SI_INFORMTN_SCHEMA	SPATIAL_CSW_ADMIN
SPATIAL_CSW_ADMIN_USR	SPATIAL_WFS_ADMIN	SPATIAL_WFS_ADMIN_USR
SYS	SYS$UMF	SYSBACKUP
SYSDG	SYSKM	SYSMAN
SYSRAC	SYSTEM	TSMSYS
WKPROXY	WKSYS	WK_TEST
WMSYS	XDB	XS$NULL

XTISYS

Specific DDLs not Replicated

Various DDLS are not replicated and are listed below.

This list includes (among other things) DDLs to the following objects:

DATABASE
PLUGGABLE DATABASE
CONTROL FILE
SPFILE/PFILE
DISK GROUP
SNAPSHOT
SUMMARY
DATABASE LINK
RECYCLE BIN
RESTORE POINT
ASSEMBLY
FLASHBACK ARCHIVE

System Generated Names:

System generated names can be a problem as logical replication may result in different names. Thus, DDLs issued for objects with system generated names will typically fail to replicate. (DDLs on system generated indexes are common)

Edition Based Redefinition

Basic support for edition based operations exists, however EBR typically depends upon on-line DDL execution. On-line DDL operations introduced in RDBMS 11.x and 12.x such as ALTER TABLE ADD COLUMN are not supported as on-line operations during rolling upgrade (or any time supplemental logging is enabled). Instead these operations are silently downgraded to a blocking model which prevents on-line execution.

Application Containers

Support for basic replication of application containers exists, but users may not execute any application INSTALL, UPGRADE or PATCH operations during the rolling upgrade.

Procedures Pragma-ed as UNSUPPORTED:

PL/SQL marked with pragma UNSUPPORTED will stop SQL apply at the point of procedure invocation so that manual intervention can be taken.

Schema	**Package**	**Procedure**	**Pragma**
SYS	DBMS_REDEFINITION	ABORT_UPDATE	PRAGMA UNSUPPORTED
SYS	DBMS_REDEFINITION	EXECUTE_UPDATE	PRAGMA UNSUPPORTED
XDB	DBMS_XDBZ	ADD_APPLICATION_PRINCIPAL	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDBZ	CHANGE_APPLICATION_MEMBERSHIP	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDBZ	DELETE_APPLICATION_PRINCIPAL	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDBZ	SET_APPLICATION_PRINCIPAL	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_ADMIN	CREATENONCEKEY	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_ADMIN	INSTALLDEFAULTWALLET	PRAGMA UNSUPPORTED w/ COMMIT

XDB	DBMS_XDB_ADMIN	MOVEXDB_TABLESPACE	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_ADMIN	REBUILDHIERARCHICALINDEX	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	ADDAUTHENTICATIONMAPPING	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	ADDAUTHENTICATIONMETHOD	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	ADDTRUSTMAPPING	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	ADDTRUSTSCHEME	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	CLEARHTTPDIGESTS	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	DELETEAUTHENTICATIONMAPPING	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	DELETEAUTHENTICATIONMETHOD	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	DELETETRUSTMAPPING	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	DELETETRUSTSCHEME	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	ENABLECUSTOMAUTHENTICATION	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	ENABLECUSTOMTRUST	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	ENABLEDIGESTAUTHENTICATION	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	ISGLOBALPORTENABLED	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	SETDYNAMICGROUPSTORE	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	SETGLOBALPORTENABLED	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XDB_CONFIG	SETHTTPCONFIGREALM	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XMLINDEX	DROPPARAMETER	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XMLINDEX	MODIFYPARAMETER	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XMLINDEX	REGISTERPARAMETER	PRAGMA UNSUPPORTED w/ COMMIT
XDB	DBMS_XMLSCHEMA	COPYEVOLVE	PRAGMA UNSUPPORTED w/ COMMIT

Appendix C

The following views are used to determine the status prior to, during, and after a DBMS_ROLLING upgrade process:

DBA_ROLLING_DATABASES

DBA_ROLLING_DATABASES lists all the databases eligible for configuration with rolling operations.

DBA_ROLLING_EVENTS

DBA_ROLLING_EVENTS lists all the events reported from the DBMS_ROLLING PL/SQL package.

DBA_ROLLING_PARAMETERS

DBA_ROLLING_PARAMETERS lists the available parameters of the DBMS_ROLLING PL/SQL package.

DBA_ROLLING_PLAN

DBA_ROLLING_PLAN displays the instructions which constitute the active upgrade plan. Each row in DBA_ROLLING_PLAN identifies a specific instruction scheduled to execute at a specific database. Instructions are created as a result of successful calls to the DBMS_ROLLING.BUILD_PLAN procedure.

During execution, groups of instructions are scheduled in batches to execute at remote databases. Groups of instructions are guaranteed to complete in BATCHID order.

DBA_ROLLING_STATISTICS

DBA_ROLLING_STATISTICS provides a list of rolling operation statistics.

DBA_ROLLING_STATUS

DBA_ROLLING_STATUS displays the overall status of the rolling operation.

DBA_ROLLING_UNSUPPORTED

DBA_ROLLING_UNSUPPORTED displays the schemas, tables, and columns in those tables that contain unsupported data types for a rolling upgrade operation for a logical standby database using the DBMS_ROLLING PL/SQL package. Use this view before you perform a rolling upgrade using DBMS_ROLLING to determine what is unsupported.

APPENDIX D

Cloud Database Upgrade Use Case:

- Specific for CDBs
- Small planned maintenance window (e.g. 1 hour) with service termination after 15 minutes. Recommend planned maintenance window that has least amount of workload and with zero long running transactions and batch.
- Service downtime less than 1 minute for initial switchover (part of any uptime SLA)
- 30 minutes post switchover evaluation phase
- Service downtime less than 1 minute if switchback to original primary required (**not part of any uptime SLA**)
- Simple rollback after 30 minutes validation and switchback. Estimated 5 minutes. (**not part of any uptime SLA since application performance testing and pre-validation were a prerequisite for the customer prior to the upgrade**)
- Any downtime due to database failures resulting in database failover options is still part of the uptime SLA.

Why DBMS_ROLLING over Oracle GoldenGate for database upgrade use case?

- Significantly simpler due to built-in automation and checks
- Data type restrictions are essentially the same as GoldenGate
- Ability to leverage existing standby database which eliminates additional storage or system resource requirements. Standby database will be a very common managed cloud implementations.
- Ability to integrate with additional standby databases.

Advantages of GoldenGate specific for database upgrade use case

- Fallback to previous release without data loss

Considerations for both GG and DBMS_ROLLING solutions

- Data type and API restrictions
- Data Lags for certain workloads may be a concern. Performance tuning for certain transactions (e.g. batch) can be challenging
- Both GG and DBMS_ROLLING solution will benefit from additional standby database

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle

facebook.com/oracle

twitter.com/oracle

oracle.com

Integrated Cloud Applications & Platform Services

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. This document is provided *for* information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0615

Active Data Guard DBMS Rolling MAA Best Practices
December 2017
Author: Michael T Smith
Contributing Authors: