Oracle 11g New Features for Administrators

Summary Sheets

Version 2.1

Editor: Ahmed Baraka

Version History

Version	Date	Updates
1.0	23-Mar-2008	Initial document.
1.1	29-Apr-2008	- Mapping exam objectives to the document topics - Fixing caught bugs in the code examples. - Fixing caught errata and mistakes in some statements. - Adding new topics.
1.15	12-Sep-2008	- Further explanation on using the new PIVOT operator. - Adding minor details in some sections.
1.2	10-Oct-2008	- Adding the section "Adaptive Cursor Sharing" - Adding minor details in some sections. - Fixing caught errata and mistakes in some statements.
1.3	01-Mar-2010	- Adding minor details in some sections.
2.0	23-Dec-2010	- Adding Oracle 11g Release 2 New Features - Removing the statement "that virtual column cannot be a part of partitioning key column". Thanks to Arshad Taqvi.
2.1	22-Jan-2010	- Removing the information about FIXED_DATE parameter as it is not a new feature. Thanks to Bhalla Ravinder.

Document Purpose

This document aims at briefly explaining Oracle Database 11g New Features with concentration on the practical code. It discusses new features related to database administration and application development. The document is edited so that it concentrates on the following:

- Brief description of the concepts. This includes:
  - New terms and definitions.
  - *Significant* advantages and/or disadvantages
  - Concept limitations and precautions
- Code examples

On the other hand, the document avoids the following:

- Detailed explanation of the concepts
- Details of using Oracle Grid Control in implementing the concepts.

What is not covered?

The document discussed new features in Oracle 11g in only the topics as stated in the contents section. New features introduced in other areas are not covered. To mention some of them, this document does not cover the new features in the following areas:

- Oracle Streams
- Data Guard
- Oracle RAC
- Oracle XML DB

Prerequisite

To be able to get advantage of reading this document, you must have solid knowledge on Oracle database 10g administration.

Terms and Acronyms Used

Following are list of the terms and acronyms used in this document:

Term / Acronym	Meaning
Oracle 11g	Whenever Oracle 11g is mentioned in this document, it is meant to be Oracle Database 11g.
ADR	Automatic Diagnostic Repository
Oracle Grid Control	Oracle Grid Control, Database Control or Oracle Enterprise Manager (OEM) are used interchangeably in this document.

Conventions in the Document

When you are asked to click on several links in the Enterprise Manager, the term "*follow the links*" is used. For example, consider the following statement:

To manage created SQL patches, follow the links **Database Home page> Server tab> Query Optimizer section> SQL Plan Control> SQL Patch tab**

This statement means, in the **Database Home** page you click **Server** tab where you will see **Query Optimizer** section. Then you click on **SQL Plan Control** and then on the **SQL Patch tab**.

Resources

S/N	Resource Type	Resource Name
1	Book	Oracle Database 11g New Features for DBAs and Developers, by Sam R. Alapati and Charles Kim, Apress, ISBN: 978-1-59059-910-5
2	Book	Oracle Database 11g New Features by Rober G. Freeman, Oracle Press
3	Oracle Official Documentation	• Oracle Database New Features Guide 11g Release 2 (11.2) E17128-03 • Oracle Database New Features Guide 11g Release 1 (11.1) B28279-02 • Oracle Database Administrator's Guide 11g Release 1 (11.1) B28310-03 • Oracle Database Performance Tuning Guide 11g Release 1 (11.1) B28274-01 • Oracle Database Backup and Recovery User's Guide 11g Release 1 (11.1) B28270-02 • Oracle Database PL/SQL Packages and Types Reference 11g Release 1 (11.1.) B28419-02 • Oracle Database High Availability Overview 11g Release 1 (11.1) B28281-01 • Oracle Database Storage Administrator's Guide 11g Release 1 (11.1) B31107-03 • Oracle Database PL/SQL Language Reference 11g Release 1 (11.1) B28370-02 • Oracle Database VLDB and Partitioning Guide 11g Release 1 (11.1) B32024-01 • Oracle Database SecureFiles and Large Objects Developer's Guide 11g Release 1

		(11.1) B28393-02
4	Oracle Metalink	The following documents were referenced: • Note:453487.1 Title: 11g New Features : Top 5 Features In 11g • Note:454631.1 Title: 11g DBCA New features / Enhancements • Note:445116.1 Title: Using the workload capture and replay in 11G • Note:470199.1 Title: 11g feature: Flashback Data Archive Guide • Note:444149.1 Title: New Background Processes In 11g • Note:432776.1 11g New Feature : Transparent Data Encryption at Tablespace Level
5	Web Site	www.oracle-base.com
6	Web Site	www.psoug.org
7	Web Site	www.orafaq.com
8	Web Site	http://technology.amis.nl
9	Web Site	http://decipherinfosys.wordpress.com/2007/12/03/oracle-11g-cross-tab-report-using-pivot-and-unpivot-operator
10	Web Site	http://www.fadalti.com/oracle/database/oracle_database_11g_new_features.htm
11	Articles by Arup Nanda	http://www.oracle.com/technology/pub/articles/oracle-database-11g-top-features/index.html
12	Blog Site	http://bar-solutions.com/wordpress/
13	Blog Site	http://virag.sharma.googlepages.com/11g
14	Blog Site	http://viveklsharma.blogspot.com
15	Blog Site	http://antognini.ch/blog

Contents

Installation, Database Upgrades and Change Management 9

Installation New Features Support ____ 9
Role and Privilege Changes ____ 9
Deprecated Components ____ 10
New Initialization Parameters Affecting Database Creation ____ 10
DBCA Enhancements ____ 11
Upgrading to Oracle Database 11g ____ 11
Database Replay ____ 13
The SQL Performance Analyzer ____ 17
Patching in Oracle Database Control ____ 21

Database Diagnosis and Repair ____ 22

Introducing Automatic Diagnostic Repository (ADR) ____ 22
Configuring the ADR ____ 22
Using adrci Tool ____ 22
Using The Support Workbench in the OEM ____ 24
Database Health Monitor ____ 25
Data Recovery Advisor ____ 26
SQL Test Case Builder ____ 27
Data Block Corruption Parameters ____ 28

Database Administration ____ 29

Automatic Memory Management ____ 29
Automatic Maintenance Tasks ____ 30
Oracle Flashback-Related New Features ____ 32
LogMiner Interface in Oracle Enterprise Manager ____ 32
Oracle Flashback Transaction Backout ____ 32
Flashback Data Archive ____ 33
Virtual Columns ____ 35
New Data Partitioning Schemes ____ 36
DDL Lock Timeout ____ 36
Explicit Locking of Tables ____ 36
Invisible Indexes ____ 36
Read-Only Tables ____ 37
Deferred Segment Creation ____ 37
Shrinking Temporary Tablespaces and Tempfiles ____ 37
Creating an Initialization Parameter File from Memory ____ 38
Restore Point Enhancements ____ 38
Database Resident Connection Pooling ____ 38
Comparing and Synchronizing Database Objects ____ 40

SQL*Plus New Features ____ 41
Online Application Maintenance ____ 42
Oracle Advanced Compression Option ____ 42
Oracle Scheduler New Features ____ 43
Lightweight Jobs ____ 44
Remote External Jobs ____ 44
Monitoring Job State with Email Notifications ____ 45
File Watcher ____ 47
Finer-grained Dependency Management ____ 50
Enhancements in Oracle Database Resource Manager ____ 50
Enhanced TRUNCATE Statement ____ 51
Dropping Unused Object Storage ____ 52

Performance Tuning ____ 53

PL/SQL Native Compilation ____ 53
Server Result Cache ____ 54
Client Side Result Cache ____ 56
Enhanced Oracle Process Monitoring ____ 57
Subprogram Inlining ____ 58
SQL Tuning Automation ____ 58
SQL Access Advisor Enhancements ____ 59
Changing Statistics Preferences ____ 63
Enhanced Statistics Maintenance ____ 63
SQL Plan Management ____ 66
ADDM New Features ____ 69
AWR New Features ____ 72
Setting Metric Thresholds for Baselines ____ 74
Performance-Related Changes in Database Control ____ 74
Miscellaneous New Performance Tuning Features ____ 74
Real-Time SQL Monitoring ____ 75
Adaptive Cursor Sharing ____ 76

Database Security ____ 77

Stronger Password Hash Algorithm ____ 77
Security Out of the Box ____ 77
Anti Network Attacks Parameters ____ 78
Tablespace Encryption ____ 79
Fine-Grained Access Control for UTL_* Packages ____ 80
Further Security New Features ____ 81

Backup and Recovery New Features ____ 83

Enhanced Block Media Recovery ____ 83
RMAN Substitution Variables ____ 83

New RMAN Configuration Parameters ____ 84
The Multisection Backups ____ 84
Creating Archival Backups ____ 85
VALIDATE Command ____ 85
Configuring an Archived Redo Log Deletion Policy ____ 86
Active Database Duplication ____ 86
Importing and Moving Recovery Catalogs ____ 88
Virtual Private Catalogs ____ 89
Miscellaneous New Features in RMAN ____ 90

Data Pump Utilities ____ 91

Compression Enhancement ____ 91
Encryption Enhancements ____ 91
Reusing a Dump File ____ 92
Remapping Data ____ 92
Renaming Tables During Export or Import ____ 92
Data Pump and Partitioned Tables ____ 92
Ignoring Nondeferred Constraints ____ 93
External Tables Based on Data Pump Driver ____ 93
Enhancement in the Transportable Parameter ____ 93

Automatic Storage Management (ASM) ____ 94

SYSASM Privilege and OSASM Group ____ 94
Upgrading ASM using DBUA ____ 94
Upgrading ASM Manually ____ 94
ASM Restricted Mode ____ 95
Diskgroup Attributes ____ 95
Checking Diskgroup ____ 97
asmcmd Utility Commands ____ 98
Fast Rebalance ____ 99
The FORCE option with Drop Diskgroup Command ____ 99
Miscellaneous ASM New Features ____ 99

PL/SQL New Features ____ 101

PL/SQL New Features ____ 101

Data Warehousing ____ 107

SecureFiles ____ 107
Accessing a LOB Using SQL and PL/SQL ____ 108
Online Redefinition ____ 109
Partition Change Tracking (PCT) ____ 110
Generating SQL Crosstab Report using PIVOT Operator ____ 110
Partitioning Improvements ____ 112

Appendix I Mapping Exam 1Z0-050 Objectives to Document Topics 117

Appendix II Displaying CLOB Contents in SQL Plus 120

Installation, Database Upgrades and Change Management

Installation New Features Support

Following are the changes you will face when installing Oracle Database 11g:

- **Choosing Oracle Base location**

  Oracle Base (set by the environment variable `ORACLE_BASE`) is the top-level directory for installing Oracle software. Oracle Universal Installer now allows you to specify and edit Oracle base location. Oracle recommends you to specify the same Oracle base for multiple Oracle homes.

  If you install Oracle database 11g software with the option of creating a database and you do not specify a value to `ORACLE_BASE`, the installation proceeds with the default value but a message will be logged in the alert log file.

- **Datafile and Flash Recovery Area Locations**

  By default, Datafiles and flash recovery area are located one level below the Oracle base location. In Oracle 10g, it is used to be saved under Oracle home directory.

- **Automatic Diagnostic Repository (ADR)**

  ADR is a single directory location for all error and trace data in the database. It replaces the traditional diagnostic directories such as `bdump`, `cdump`, and `udump`. ADR location is controlled by the new initialization parameter `DIAGNOSTIC_DEST`. Oracle recommends you choose the same ADR base for all Oracle products.

- **New Components**

  Following are new components which are available when installing Oracle 11g:

  - **Oracle Application Express (APEX)**: APEX is a rapid application development tool for developing database centric web-based applications. In Oracle 11g, it is highly enhanced and available in the Oracle database CD.
  - **Oracle SQL Developer**: Oracle SQL Developer is a graphical tool for examining database objects and issuing SQL commands. It is automatically installed, if you create a template-based database.
  - **Oracle Real Application Testing**: Oracle Real Application Testing option is automatically installed with the Enterprise Edition installation. This option includes two solutions to test the effect of the system changes on the real-world applications: Database Reply and SQL Performance Analyzer. Both of those solutions will be discussed in later sections.
  - **Oracle Configuration Manager (OCM)**: OCM is an optional component and it collects information about software configuration in the Oracle home directories and uploads it to the Oracle configuration repository.
  - **Oracle Warehouse Builder:** it is a business intelligence (BI) design tool and is automatically installed as part of the Oracle database software.
  - **Oracle Database Vault**: Oracle Database Vault component enables you to secure business data even from DBAs. In Oracle 11g, it is a component available during installation and to install it, you must select the Custom installation option.
  - **Oracle Shadow Copy Service**: When you install Oracle 11g on Windows 2003 servers, a service named as Volume Shadow Copy Service (VSS) is installed. This service is an infrastructure that enables the users to create snapshots called shadow copies. For details on using VSS, refer to the documentation "Oracle Database Backup and Recovery User's Guide" and Chapter 8 in "Oracle Database Platform Guide for Microsoft Windows"

- **Other Install Options**
  - Oracle Data Mining option is selected by default with the Enterprise Edition installation and is automatically installed when you create a database.
  - Oracle XML DB is now a mandatory component and thus its option is removed from the installation.

Role and Privilege Changes

In Oracle 11g, there is a new system privilege called SYSASM. This privilege should be granted to users who need to perform ASM administrative tasks. Also, in Oracle 11g for Unix/Linux operating systems, you can create `osasm` OS group. Oracle recommends you grant ASM access only to members of the `osasm` group. For further details about using this privilege, refer to the chapter "Automatic Storage Management".

`CONNECT` role has now only `CREATE SESSION` privilege. Be aware that if you upgrade existing database to version 11g, `CONNECT` role will be modified by the upgrade script to have only `CREATE SESSION` privilege.

Deprecated Components

Following are the components deprecated in Oracle 11g:

- iSQL*Plus
- Oracle Workflow
- Oracle Enterprise Manager Java Console
- Oracle Data Mining Scoring Engine
- Raw storage support (installer only)

New Initialization Parameters Affecting Database Creation

As with any new Oracle database version, in Oracle 11g some new initialization parameters are introduced and some parameters are deprecated. In this section we will introduce those new parameters you may set while creating an Oracle 11g database. Notice that this section is not meant to list all the new initialization parameters.

- **Memory Parameters**

In Oracle 11g, the automatic memory management feature is developed. Both the system global area (SGA) and the program global area (PGA) will expand and shrink based on the instance demands. To enable this feature, you set the following new parameters:

`MEMORY_TARGET`	this parameter sets the system-wide usable memory that will be used by the instance for SGA and PGA.
`MEMORY_MAX_TARGET`	this parameter sets the maximum value you can set for `MEMORY_TARGET` parameter.

Further details about using those parameters will be discussed in Automatic Memory Management section.

- **Automatic Diagnostic Repository (ADR)**

ADR is a directory that points to all error data raised in the database. You set it by the new parameter `DIAGNOSTIC_DEST`. It replaces `USER_DUMP_DEST`, `BACKGROUND_DUMP_DEST` and `CORE_DUMP_DEST` parameters.

`DIAGNOSTIC_DEST` defaults to the following value: $ORACLE_BASE/diag/rdbms/$INSTANCE_NAME/$ORACLE_SID

If you haven't set the ORACLE_BASE variable, the value of the DIAGNOSTIC_DEST parameter defaults to $ORACLE_HOME/log.

Further details about using this parameter will be discussed in ADR section.

- **Result Cache Parameters**

In Oracle 11g, a new memory component, named as *result cache*, is introduced. This memory area stores results of frequently run queries. It also saves results of PL/SQL function results. Parameters used to control result cache are: `RESULT_CACHE_MODE`, `RESULT_CACHE_MAX_RESULT`, `RESULT_CACHE_MAX_SIZE`, `RESULT_CACHE_REMOTE_EXPIRATION`, `CLIENT_RESULT_CACHE_SIZE` and `CLIENT_RESULT_CACHE_LAG`. Using those parameters will be discussed in Server Result Cache and Client Side Result Cache sections.

- **DDL Lock Timeout**

The new parameter `DDL_LOCK_TIMEOUT` controls length of time a DDL statement waits for a DML lock. Using this parameter will be discussed in DDL Lock Timeout section.

- **The `DB_ULTRA_SAFE` Parameter**

This parameter is used to set the effective values of the parameters: `DB_BLOCK_CHECKING`, `DB_LOST_WRITE_PROTECT`, `DB_BLOCK_CHECKSUM`. This parameter takes one of the following values:

`off` — this value means any values you set for any of the three parameters will not be overridden.

`data only` — The effective value of the parameters will be as follows:

Parameter	Active Value
`DB_BLOCK_CHECKING`	`medium`
`DB_LOST_WRITE_PROTECT`	`typical`
`DB_BLOCK_CHECKSUM`.	`full`

`data and index` — The effective value of the parameters will be as follows:

Parameter	Active Value
`DB_BLOCK_CHECKING`	`full`
`DB_LOST_WRITE_PROTECT`	`typical`
`DB_BLOCK_CHECKSUM`.	`full`

- **Security Parameters**

Oracle 11g introduces two important security parameters. Following table illustrated those parameters and their usages:

Parameter	Description	Default Value
SEC_CASE_SENSITIVE_LOGON	to enable or disable password case-sensitivity.	true
SEC_MAX_FAILED_LOGIN_ATTEMPTS	Oracle drops the connection after the specified number of login attempts fail for any user.	10

DBCA Enhancements

In Oracle 11g, DBCA go through steps similar to the one in Oracle 10g. Following is a list of new DBCA pages in Oracle 11g:

- **Security Settings**

In this page you can set the created database to use the new enhanced default security settings in Oracle 11g. Those settings include audit settings, password profiles and revoking grants from the public role. Those settings are part of a database option named as Secure Configuration.

If you choose to disable those security settings, the database will be created with the default security options as for Oracle Database 10g Release 2. You can still configure the Secure Configuration option later by invoking the DBCA.

- **Creating a Listener**

If you choose to configure the Enterprise Manager, DBCA will search for a listener configured in the Oracle home. If no listener was found, DBCA asks you to create one with the Net Configuration Assistant tool.

- **Network Configuration**

In Oracle 11g, DBCA allows you to select the listener(s) for which to register the new database.

- **Configuring New Database Options**

In Oracle Database 11g, you can configure the following options when using DBCA:

a) Oracle Application Express

b) Oracle Database Vault

c) Oracle Warehouse Builder

Also, you no longer see the Data Mining in the DBCA as an option to configure. The data mining schema are by default created when the catproc.sql script is run.

- **New Memory Management**

You can enable using the new automatic memory management feature by specifying amount of memory to be used by both SGA and PGA.

- **Switching a Database from Database Control to Grid Control Configuration**

With Oracle Database 11g, DBCA provides the Enterprise Manager Configuration plug-in, which automates the process to switch configuration of a database from Database Control to Grid Control.

Upgrading to Oracle Database 11g

Upgrade path

You can directly upgrade to Oracle 11g, if you current database is 9.2.04 or newer. In other words, it supports direct upgrade to versions 9.2.0.4, 10.1 and 10.2. Otherwise, you should follow one of the upgrade paths:

```
7.3.3  ->      7.3.4   ->      9.2.0.8  -> 11.1
8.0.5  ->      8.0.6   ->      9.2.0.8  -> 11.1
8.1.7  ->      8.1.7.4 ->      9.2.0.8  -> 11.1
9.0.1.3->      9.0.1.4 ->      9.2.0.8  -> 11.1
9.2.0.3 (or lower)     ->      9.2.0.8  -> 11.1
```

Oracle 11g client can access Oracle databases of versions 8i, 9i and 10g.

Upgrade process and COMPATIBLE parameter

The default compatibility value for Oracle 11g is 11.1. You can, however, upgrade to Oracle 11g with a minimum value of the COMPATIBLE parameter of 10.0.0. However, if you upgrade to 11g and keep the COMPATIBLE parameter to 10.0.0, only a small portion of the new features will be available.

Manual Upgrade Procedure

To manually upgrade a database from 10g to Oracle 11g, perform the following steps:

1. Invoke the Pre-Upgrade Information Tool in the database to upgrade.

   This tool is simply the script `$ORACLE_HOME/rdbms/admin/utlu111i.sql`. So, you have to copy this script altogether with the scripts: `utlu111s.sql` and `utlu111x.sql` to a staging directory in the database to upgrade.

   As with previous versions, this tool will examine the target database and display the warnings and recommendations that you should consider before you start the upgrade process such as removing obsolete initialization parameters, setting proper values to some parameters and adding space to key tablespaces.

   Spool the output of running the script for later review.

```
SQL>spool upgrade11g.log
SQL>@utlu111i.sql
                                                                                ...
SQL>spool off
```

2. Backup the database.

3. Set the `COMPATIBLE` parameter to 11.1.0. You can do so by issuing the following command:

```
ALTER SYSTEM SET COMPATIBLE='11.1.0' SCOPE=SPFILE;
```

4. Modify the values of the initialization parameters and remove the obsolete parameters as recommended by the Pre-upgrade tool in the current initialization parameter file.

5. Copy the initialization parameter file to the new Oracle 11g home.

6. Shutdown cleanly the database.

7. If the database is installed in a Windows system, perform the following steps:

   a) Stop the Oracle database service. Usually its name has the format OracleService<SID>.

   b) Delete the service. You can use the `oradim` utility for this purpose.

   `oradim -delete –SID <sidname>`

   c) Use oradim utility in the Oracle 11g home to create a new Oracle Database 11g release instance. Of course, it should use the same SID.

   `oradim -NEW –SID <sidname>`

8. If the database to upgrade is using a password file, move it to the new Oracle 11g home.

9. Change the environment variables `ORACLE_HOME`, `PATH`, and `LD_LIBRARY_PATH` so that they point to the new Oracle Database 11g directories.

10. In the Oracle 11g home, change to directory `$ORACLE_HOME/rdbms/admin` and then start the SQL*Plus

11. Start the database in upgrade mode and then run the upgrade script (it takes long time). When working in upgrade mode, Oracle allows only SYSDBA connections and sets some initialization parameters to specific values that are required to run the upgrade script.

```
SQL>login sys/password as sysdba
SQL>startup upgrade pfile=$ORACLE_HOME/dbs/initorcl.ora
SQL>spool upgrade11g.log
SQL>@ catupgrd.sql
...
SQL>spool off
```

12. After the upgrade script finishes, make sure no error occurred during the upgrade. Usually errors are raised because of lack of shared memory or tablespace size. If there is any error, fix its cause and restart the upgrade script.

13. When the upgrade script successfully finishes, restart the database in OPEN mode.

14. Rum `utlu111s.sql` script (referred to as Post-Upgrade Status tool) to view the results of the upgrade. This tool will view the installed components and their status. If you see a component with invalid status, usually running the script in the next step will set it to valid.

15. Execute the script `utlrp.sql` to in parallel recompile any invalid PL/SQL program unit and Java code.

```
SQL>@utlrp.sql
SQL>SELECT COUNT(*) FROM DBA_INVALID_OBJECTS;
```

16. Because the upgraded database disables all tablespace alerts by setting the threshold to null, set the thresholds for tablespace alert to the values you desire.

If the upgrade process fails, restore the database from the backup.

Upgrading with the DBUA

Database Upgrade Assistance (DBUA) is a GUI utility that facilitates upgrade process. DBUA works in 11g in a similar way to it in 10g. It just now has a screen asking you to define the diagnostic directory.

Beside its simplicity, DBUA has an advantage of being able to upgrade both the database instance and the ASM instance simultaneously. In manual method, you have to upgrade them separately.

Note The database upgrade process moves SQL profiles to `SYSAUX` tablespace. Thus, taking that tablespace offline may lead to degrading the database performance.

Database Replay

Database Replay (sometimes named as Workload Replay) feature in Oracle 11g allows you to reproduce the production database conditions in a testing environment. In other words, with this feature you can capture the actual workload on a production system and replay it in a test system. This way, you can analyze the condition of the production database without working on the actual production database.

This feature enables you to test the impact of applying changes on a production database. These changes could be database upgrades, switching to RAC, application upgrades, operating system upgrades or storage system changes.

Using Database Replay consists of four phases:

1. **Workload Capture**: this is when you record the production database workload.
2. **Workload Preprocessing**: this is to convert the captured workload into replay files.
3. Typically, after the step above, you apply the changes on the test system. Then you start the **Workload Replay** process where you replay the production workload in the test database.
4. **Analysis and Reporting**: when the step above successfully finishes, you generate report on the errors and performance influence.

In the following sections, practical step to implement those phases will be discussed.

To interface with Workload Replay feature in the database, you use a utility named as Workload Replay Client. To invoke the utility, type `wrc` in the command line.

Data Captured by Database Reply

Following is a list of data changes captured by Database Reply:

- DML statements
- DDL statements
- Session control calls (such as `ALTER SESSION`)
- System control calls (such as `ALTER SYSTEM`)

Data Not Captured by Database Reply

The following data changes are not captured by Database Reply:

- Flashback queries
- Scheduled jobs
- Shared Server requests
- Non PL/SQL AQ
- Direct path load from external files using SQL*Loader, database links, external tables, Oracle streams, non-SQL-based object access and distributed transactions.
- Distributed transactions

Preparing for Capturing Workload

Before you capture the workload, perform the following steps:

1. Backup database data that you want to test. Use either RMAN, user-managed online backup, Data Pump utilities or a snapshot standby. This backup will be used later to create a testing environment in a state similar to the production database for the replay process.
2. Any transaction that is underway when you start capturing the workload may not be captured. If you want to capture all transactions, you should restart the database in restricted mode, start the capture process, then open the database for users.
3. Create directory object for storing captured workload.

```
CREATE DIRECTORY workload_dir AS 'C:\Oracle\admin\ora11g\workload';
```

4. Decide whether some of the user sessions should not be captured. You may not need to capture DBA sessions, Oracle Enterprise Manager sessions or any sessions created by third party clients. To achieve this task, use DBMS_WORKLOAD_CAPTURE package as shown in the following steps:
   a. Use ADD_FILTER procedure to add any eliminating sessions based on USER, MODULE, ACTION, PROGRAM, SERVICE or INSTANCE_NUMBER:

```
begin
 DBMS_WORKLOAD_CAPTURE.ADD_FILTER(
  FNAME      => 'FILTER_DBA1',
  FATTRIBUTE => 'USER',
  FVALUE     => 'SYSTEM,DBSNMP' );
end;
```

   b. The DBA_WORKLOAD_FILTERS view provides you information about existing filters. For a filter with the STATUS of USED, it means the filter is deleted. :

```
SELECT NAME, ATTRIBUTE, VALUE FROM DBA_WORKLOAD_FILTERS;
```

   c. Use DELETE_FILTER procedure to delete any existing filter:

```
EXEC DBMS_WORKLOAD_CAPTURE.DELETE_FILTER( FNAME  => 'FILTER_DBA1');
```

Capturing Workload

Use START_CAPTURE procedure in DBMS_WORKLOAD_CAPTURE package to start capturing the workload. If you set the DURATION parameter to NULL, it means that workload capture will continue until the user executes DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE. It is recommended to record the time in the system just before issuing the command.

```
begin
 DBMS_WORKLOAD_CAPTURE.START_CAPTURE(
  NAME     => '1JAN_WORKLOAD',
  DIR      => 'WORKLOAD_DIR',
  DURATION => 2400); -- duration in seconds
end;
```

To stop the capture process before ending of duration period, issue the following command:

```
exec DBMS_WORKLOAD_CAPTURE.FINISH_CAPTURE;
```

After capture process finishes, you can issue query about workload captures using the following command:

```
SELECT ID, NAME, STATUS, ERROR_MESSAGE FROM DBA_WORKLOAD_CAPTURES;
```

You can generate a report about the workload capture you have made:

```
declare
 v_capture_id  number;
 v_capture_rpt clob;
begin
 v_capture_id  := DBMS_WORKLOAD_CAPTURE.GET_CAPTURE_INFO(DIR => 'WORKLOAD_DIR');

 v_capture_rpt := DBMS_WORKLOAD_CAPTURE.REPORT( CAPTURE_ID => v_capture_id , FORMAT =>
DBMS_WORKLOAD_CAPTURE.TYPE_TEXT); -- format could also be TYPE_HTML

-- display contents of v_capture_rpt
end;
```

Alternatively, you can use the following statements:

```
SELECT id, name, status FROM DBA_WORKLOAD_CAPTURES;
SELECT DBMS_WORKLOAD_CAPTURE.REPORT(1, 'HTML') FROM DUAL;
```

If you want to delete from its data dictionary views, used the procedure DELETE_CAPTURE_INFO. However, this procedure does not delete the workload capture files in its directory. If you want to take a new workload capture with the same name, you should manually get rid of its files otherwise an error will be returned when you execute START_CAPTURE procedure.

Preprocessing Workload Capture Data

To be able to replay a Workload Capture, you must preprocess its data. Although it is possible to preprocess in the production database, practically it is done in a test database. Preprocessing includes the following steps:

1. Restore the test database from the backup you made in the production database. The target is to make the same application become in the same state as it has been in the production database.
2. Create a directory object in the test database to hold the workload capture data files.

   ```
   CREATE DIRECTORY replay_dir AS 'c:\oracle\admin\ora11g\replay';
   ```

3. Move the workload data files from the production database to the created directory in the test system.
4. Use the PROCESS_CAPTURE procedure to process the workload data:

   ```
   exec DBMS_WORKLOAD_REPLAY.PROCESS_CAPTURE('REPLAY_DIR');
   ```

Replaying the Workload

Typically, at this stage, you perform the changes you want to undertake on the system. Then you start the replay process. Replaying Workload is done by performing of the following steps:

1. It is recommended to set the time of the test system to the time when the workload was captured on the production system. This is to avoid any invalid time-based data or job-scheduling issues.
2. Take steps to resolve, if any, external references including: database links, external tables, directory objects, and URLs.
3. Initialize the Replay Data: this process means metadata will be read from Workload Capture files and loaded into tables. Workload replay process will read from those tables when it operates. Initializing the replay data is done by invoking the procedure INITIALIZE_REPLAY

   ```
   begin
    DBMS_WORKLOAD_REPLAY.INITIALIZE_REPLAY(
       REPLAY_NAME =>'1JAN_WORKLOAD',
       REPLAY_DIR  =>'REPLAY_DIR'); -- directory name should always be in upper case.
   end;
   ```

4. Remapping Connections: if any session in the production database during the workload capturing used a *connection* to access an external database, this connection should be remapped in the test database so that it connects to the desired database.

   To display the connection mapping information for a workload replay, query the view DBA_WORKLOAD_CONNECTION_MAP.

   ```
   SELECT REPLAY_ID, CONN_ID, CAPTURE_CONN, REPLAY_CONN
   FROM   DBA_WORKLOAD_CONNECTION_MAP
   ```

   To remap connection string in the test database to the required connection strings, you use REMAP_CONNECTION procedure.

   ```
   begin
    DBMS_WORKLOAD_REPLAY.REMAP_CONNECTION (
        CONNECTION_ID =>1, REPLAY_CONNECTION => 'myprod:1521/mydb' );
   end;
   ```

5. Preparing the Workload Replay: this is done with the procedure PREPARE_REPLAY. Following are the three options (set by parameters) that can be configured with this procedure:
   a. SYNCHRONIZATION: (default TRUE) Ensure the replay observes the commit sequence of the capture; i.e. any work is run only after dependent commits in the replay are completed. If you know that transactions in your workload capture are independent, you can set this parameter to FALSE.

b. CONNECT_TIME_SCALE: (default 100) this parameter uses the elapsed time between the time when the workload capture began and when sessions connect. You can use this option to manipulate the session connect time during replay with a given percentage value. The default value is 100, which will attempt to connect all sessions as captured. Setting this parameter to 0 will attempt to connect all sessions immediately.

c. THINK_TIME_SCALE: (default 100) think time is the elapsed time while the user waits between issuing calls. To control replay speed, use the THINK_TIME_SCALE parameter to scale user think time during replay.
If user calls are being executed slower during replay than during capture, you can make the database replay attempt to catch up by setting the THINK_TIME_AUTO_CORRECT parameter to TRUE (the default). This will make the replay client shorten the think time between calls, so that the overall elapsed time of the replay will more closely match the captured elapsed time.

The PREPARE_REPLAY procedure puts the database server in PREPARE mode. Below is a code example:

```
exec DBMS_WORKLOAD_REPLAY.PREPARE_REPLAY ( SYNCHRONIZATION => FALSE );
```

6. Starting Replay Client(s): replay client (represented by the executable wrc) controls the replay of the workload data. You may need to run more wrc from more than one host. This depends on the maximum number of sessions that a single wrc thread can handle and the total number of sessions captured by the workload capture.

   For example, if the workload capture has data of 400 sessions and a single host can handle only 150 sessions, in this case you need three hosts with wrc installed and run on each.

   To know how many hosts and wrc clients you need to operate for your workload capture, run the wrc in the *calibrate* mode as shown below:

   ```
   wrc system/<password> mode=calibrate replaydir=C:\Oracle\admin\ora11g\replay
   ```

   Then, run wrc in *replay* mode (the default) on the client host(s):

   ```
   wrc system/<password> mode=replay replaydir=C:\Oracle\admin\ora11g\replay
   ```

7. Start the replay process using START_REPLAY procedure (notice that wrc client(s) were previously started):

   ```
   exec DBMS_WORKLOAD_REPLAY.START_REPLAY();
   ```

   If, for any reason, you want to cancel the replay process before it finishes, use CANCEL_REPLAY procedure.

For the workload replay, notice the following:

- While the workload is replaying, you can query V$WORKLOAD_REPLAY_THREAD view to list information about all sessions from the replay clients.
- You can obtain information about the workload replays, after they finish, by querying the view DBA_WORKLOAD_REPLAYS.
- After workload replay finishes, all AWR snapshots related to the replay time period is automatically exported. This can also be done manually using EXPORT_AWR procedure.
- Exported snapshots can be imported into the AWR schema owned by SYS user using IMPORT_AWR procedure.

Analyzing Workload Replay Report

After the workload finishes, you start generating a report about it. The report is the desired output of the whole process and will assist you on measuring the difference between the production and test systems or catching any errors during the workload replay. To generate the report, issuing the following code:

```
declare
 v_cap_id NUMBER;
 v_rep_id NUMBER;
 v_rpt    CLOB;
begin
 v_cap_id := DBMS_WORKLOAD_REPLAY.GET_REPLAY_INFO(DIR=>'REPLAY_DIR');

 /* Get the latest replay */
 SELECT MAX(ID) INTO v_rep_id FROM DBA_WORKLOAD_REPLAYS
 WHERE CAPTURE_ID=v_cap_id;

 v_rpt := DBMS_WORKLOAD_REPLAY.REPORT(
    REPLAY_ID => v_rep_id,
    FORMAT => DBMS_WORKLOAD_REPLAY.TYPE_TEXT); -- or XML, HTML
end;
```

Replaying a Database Workload Using Enterprise Manager

To replay a database workload using Enterprise Manager, follow the links: **Software and Support page> Real Application Testing> Database Replay> in the Go to Task column, click the Replay Workload task icon**.

Using OEM, you can also monitor an active workload replay and view a completed workload replay.

The SQL Performance Analyzer

The SQL Performance Analyzer (SPA) aims at measuring the impact of applying any change on the database on the performance of the SQL statements execution. If it finds out performance degradation in one or more SQL statements, it provides you recommendations on how to improve their performance.

This is very useful for a DBA to analyze how a change on the database (including database upgrade) may affect the execution efficiency of SQL statements. Using this tool is explained here because you may consider using it to study the effect of upgrading an Oracle database 10g release 2 to 11g.

SPA can be invoked from OEM or using `DBMS_SQLPA` package.

To use SPA, you perform the following phases:

1. Capture the SQL workload.
2. Measure the performance of the workload before the change.
3. Make a change.
4. Measure the performance of the workload after the change.
5. Compare performance.

Steps that should be followed to perform those phases are explained in the following sections.

It is recommended to make an identical copy of the production database, apply the change on the test database then use SPA on the test database to measure the influence of the applied changes.

Note If you plan to use SPA on a test database, it is highly recommended to make the test database resemble the production database as closely as possible. You can use the RMAN `duplicate` command for this purpose.

Capturing the Production System SQL Workload

You capture the SQL Workload that you intend to analyze and store its statements into a SQL tuning set (STS). Following are the steps to do so:

1. Create STS using the following code:

```
begin
 DBMS_SQLTUNE.CREATE_SQLSET(
  SQLSET_NAME => 'upgrade_sts',
  DESCRIPTION =>'To test upgrade to 11g',
  SQLSET_OWNER =>'SYS'); -- current user is the default
end;
```

2. You can load SQL statements into a STS from different sources, including the cursor cache, Automatic Workload Repository (AWR), and existing SQL Tuning Sets. STS contains information about the captured statements such as execution context, SQL text and execution frequency.

   The following example illustrates how to load `UPGRADE_STS` from an AWR baseline called peak baseline. The data has been filtered to select only the top 30 SQL statements ordered by elapsed time.

```
declare
 baseline_cur DBMS_SQLTUNE.SQLSET_CURSOR;
begin
 --a ref cursor is opened to select from the specified baseline
 OPEN baseline_cur FOR
   SELECT VALUE(p)
   FROM TABLE (DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(
'peak baseline',NULL, NULL, 'elapsed_time', NULL, NULL, NULL,30 )) p;
 --Next the statements and their statistics are loaded into the STS
 DBMS_SQLTUNE.LOAD_SQLSET( SQLSET_NAME=>'UPGRADE_STS', POPULATE_CURSOR=>baseline_cur);
end;
```

The following example loads UPGRADE_STS with SQL statements that are not owned by SYS and their elapsed time is greater than 20,000 seconds.

```
declare
 sql_cur DBMS_SQLTUNE.SQLSET_CURSOR;
begin
 --a ref cursor is opened to select the required SQL statments
 OPEN sql_cur FOR
   SELECT VALUE(p)
   FROM TABLE (DBMS_SQLTUNE.LOAD_SQLSET(
'parsing_schema_name <> ''SYS'' and elapsed_time > 2000000',NULL, NULL, NULL, NULL,1,
NULL, 'ALL')) p;
 --the statements are loaded into the STS
 DBMS_SQLTUNE.LOAD_SQLSET( SQLSET_NAME=>'UPGRADE_STS', POPULATE_CURSOR=>sql_cur);
end;
```

3. If you are using a test database, you should transport the STS created in the previous step to the test system. This can be done by performing the following steps:

   3.1. Create a staging table to hold the STS from the production database. Following is an example code to do that:

   ```
   exec DBMS_SQLTUNE.CREATE_STGTAB_SQLSET( TABLE_NAME =>'sts_table');
   ```

   3.2. Export STS data into the staging table as shown in the following code:

   ```
   begin
    DBMS_SQLTUNE.PACK_STGTAB_SQLSET( SQLSET_NAME => 'UPGRADE_STS',
     STAGING_TABLE_NAME =>'sts_table');
   end;
   ```

   3.3. Export the staging table from the production database and import it to the test database using Data Pump export and import utilities.

   3.4. Import the STS data from the staging table to the test database as shown in the example below:

   ```
   begin
   DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET( SQLSET_NAME => '%',
     REPLACE => TRUE,
     STAGING_TABLE_NAME =>'sts_table');
   end;
   ```

4. Create a tuning task for the created STS as show in the example code below:

   ```
   declare
    v_task varchar2(100);
   begin
   v_task := DBMS_SQLPA.CREATE_ANALYSIS_TASK( SQLSET_NAME => 'UPGRADE_STS',
     TASK_NAME =>'spa_upgrade_task');
   end;
   ```

Note We are using SPA in this scenario to analyze SQL involved in a STS. Actually, you can also use it to analyze a given single SQL statement, a statement in the cursor cache or SQL statements in a workload repository in a given a range of snapshot identifiers

5. If you are using SPA to examine the influence of upgrading a 10g database to 11g, set the parameter OPTIMIZER_FEATURES_ENABLE in the test database to 10.2.0. This enables you to make SPA generate statistics for 10g database.

   ```
   ALTER SYSTEM SET OPTIMIZER_FEATURES_ENABLE ='10.2.0'
   ```

Measuring the Performance Before the Change

Start analyzing the SQL workload before applying the change. This is done by using the procedure EXECUTE_ANALYSIS_TASK. This procedure has the parameter EXECUTION_TYPE which should take one of three possible values: TEST EXECUTE, EXPLAIN PLAN or COMPARE PERFORMANCE. The first value will lead to execute all the SQL Statements in the STS. The second value will generate explain plans for the SQL Statements without executing the statements. The third value is only used to compare two versions of SQL Performance data.

```
begin
DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
  TASK_NAME => 'spa_upgrade_task',
  EXECUTION_TYPE=> 'TEST EXECUTE', -- or EXPLAIN PLAN
  EXECUTION_NAME =>'before_change');
end;
```

After the code above is successfully executed, apply the change on the database.

Measuring the Performance After the Change

If the change was upgrading the database to 11g, do not forget to set the parameter OPTIMIZER_FEATURES_ENABLE to the 11g value after the upgrade finishes:

```
ALTER SYSTEM SET OPTIMIZER_FEATURES_ENABLE ='11.1.0.6'
```

After applying the change, you collect SQL performance data by running EXECUTE_ANALYSIS_TASK again.

```
begin
DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
  TASK_NAME => 'spa_upgrade_task',
  EXECUTION_TYPE=> 'TEST EXECUTE', -- or EXPLAIN PLAN
  EXECUTION_NAME =>'after_change');
end;
```

Comparing the SQL Performance

To compare the SQL performance data before and after the change, you still use the procedure EXECUTE_ANALYSIS_TASK but this time you set the parameter EXECUTION_TYPE to COMPARE PERFORMANCE.

The following code compares the SQL performance data analyzed by the SPA before and after the change for the task spa_upgrade_task. By setting the parameter COMPARISION_METRIC to DISK_READS, it will measure the performance impact of the change on the disk reads. Other possible values include: ELAPSED_TIME, OPTIMIZER_COST, DIRECT_WRITE, PARSE_TIME, and BUFFER_GETS.

```
begin
 DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
   TASK_NAME        => 'spa_upgrade_task',
   EXECUTION_TYPE   => 'COMPARE PERFORMANCE',
   EXECUTION_PARAMS => 'DBMS_ADVISOR.ARGLIST('COMPARISION_METRIC','DISK_READS'));
end;
```

By default, the procedure EXECUTE_ANALYSIS_TASK compares the last two task executions. You can set the names of the executions before and after the system change was made as follows:

```
begin
 DBMS_SQLPA.EXECUTE_ANALYSIS_TASK (
   TASK_NAME        => 'spa_upgrade_task',
   EXECUTION_TYPE   => 'COMPARE PERFORMANCE',
   EXECUTION_PARAMS => DBMS_ADVISOR.ARGLIST('EXECUTION_NAME1', 'before_change',
    'EXECUTION_NAME2', 'after_change',
    'COMPARISON_METRIC','DISK_READS'));
end;
```

Then you can generate a report to show the result of performance comparison. This can be done using REPORT_ANALYSIS_TASK function as shown in the code below:

```
declare
 report clob;
begin
report := DBMS_SQLPA.REPORT_ANALYSIS_TASK (
  TASK_NAME => 'spa_upgrade_task',
  TYPE      => 'TEXT',     -- or HTML, XML
  LEVEL     =>'TYPICAL',  -- BASIC,ALL,IMPROVED,REGRESSED,CHANGED,
                           -- UNCHANGED,CHANGED_PLANS,UNCHANGED_PLANS,ERRORS
  SECTION   =>'SUMMARY'); -- or SECTION_ALL
end;
```

SQL Performance Analyzer Report

The report is divided into the sections: General Information, Result Summary and Result Details. Following table illustrates description of the report's section and subsections.

Section / Subsection	Description
General Information	It contains basic information and metadata about the SQL Performance Analyzer task, the SQL Tuning Set used, and the pre-change and post-change executions.
Result Summary	It summarizes the results of the SQL Performance Analyzer task.
Overall Performance Statistics	It displays statistics about the overall performance of the entire SQL workload. Use this subsection to determine whether the workload performance will improve or degrade after making the system change.
Performance Statistics of SQL Statements	It highlights the SQL statements that are the most impacted by the system change.
Errors	It reports all errors that occurred during an execution.
Result Details	It represents a drill-down into the performance of SQL statements that appears in the Result Summary section of the report.
SQL Statement Details	It summarizes the SQL statement, listing its information and execution details.
Single Execution Statistics	It compares execution statistics of the SQL statement from the pre-change and post-change executions and then summarizes the findings.
Execution Plans	It displays the pre-change and post-change execution plans for the SQL statement.

SQL Performance Analyzer Views

You can use the following queries to monitor SQL Performance Analyzer and view its analysis results:

```
-- to display descriptive information about the created SQL Performance Analyzer tasks
SELECT TASK_ID, TASK_NAME, STATUS, PCT_COMPLETION_TIME, PROGRESS_METRIC,
RECOMMENDATION_COUNT
FROM DBA_ADVISOR_TASKS
WHERE ADVISOR_NAME='SQL Performance Analyzer';

-- to display information about task executions
SELECT TASK_NAME, EXECUTION_NAME, EXECUTION_TYPE, STATUS
FROM   DBA_ADVISOR_EXECUTIONS
WHERE  ADVISOR_NAME='SQL Performance Analyzer';

-- to display the SQL Performance Analyzer findings
-- TYPE possible values: PROBLEM, SYMPTOM, ERROR, and INFORMATION
SELECT TASK_NAME, EXECUTION_NAME, FINDING_NAME, TYPE,
       IMPACT_TYPE, IMPACT "Impact Value", MESSAGE, MORE_INFO
FROM   DBA_ADVISOR_FINDINGS
WHERE  upper(TASK_NAME)= upper ('spa_test');
```

Patching in Oracle Database Control

New Features in Database Control for Patching

Patching through Database Control is enhanced in Oracle 11g. On the Database Control home page, click **Software and Support** to reach the **Database Software Patching** page. The page has the following links:

Link	*Description*
Patch Advisor	The advisor will connect to Metalink and display the available patches applicable to your software installation. It has two sections: o Critical Security Patches o Patch Recommendations by Feature
View Patch Cache	This page displays all the patches downloaded from Metalink. Even if you did not download any patch, OEM by default automatically downloads necessary patches from Metalink when the patch job runs. Cached patches can be then applied to multiple destinations.
Patch Prerequisites	With this page, you can stage the software updates from Metalink or Software Library to a staging location and run prerequisite checks on those updates.
Stage Patch	This page enables you to search the patches on Metalink based on your criteria and then select them.
Apply Patch	This page lets you select an update from Metalink or Software Library to apply.

Online Patching

With Oracle 11g online patching (or called hot patching), you can apply or roll back a database patch while the instance is running. Also it can detect conflicts between two online patches. On the other hand, online patching consumes more memory than the conventional method.

In UNIX systems, you use the script `$ORACLE_HOME/OPatch/opatch` to invoke the online patching.

Database Diagnosis and Repair

Introducing Automatic Diagnostic Repository (ADR)

The Automatic Diagnostic Repository (ADR) is a file system repository to store diagnostic data source such as alter log, trace files, user and background dump files, and also new types of troubleshooting files such as Health Monitor reports, Incident packages, SQL test cases and Data repair records.

In Oracle 11g, there is a new framework (named as fault diagnosability infrastructure) consisting of many tools for diagnosing and repairing the errors in the database. All those tools refer to the ADR in their operation.

ADR is developed to provide the following advantages:

- Diagnosis data, because it is stored in file system, is available even when the database is down.
- It is easier to provide Oracle support with diagnosis data when a problem occurs in the database.
- ADR has diagnosis data not only for the database instance. It has troubleshooting data for other Oracle components such as ASM and CRS.

Note For each database instance two alert log files are generated: one as text file and one with xml format. Contents of the xml-formatted file can be examined using `adrci` tool, which will be discussed in a later section.

Also the xml-formatted alert log is saved in the ADR and specifically in the directory `$ORACLE_BASE/diag/rdbms/$INSTANCE_NAME/$ORACLE_SID/alert`

Problems and Incidents

Problem is a critical error occurred in the database. Each problem has a *problem key*. The problem key consists of the *Oracle error number* and *error argument*. Here is an example: ORA 600 [4899].

The first time a problem occurs, an *incident* is also created. When the same problem takes place again, another incident will be generated for the same problem. Thus, you may have multiple incidents for a one problem.

When an incident occurs, Oracle performs the following:

- An alert will be issued.
- An entry will be added in the alert log file.
- Related data will be stored in the ADR directory.

By default, an incident metadata is purged from the ADR after one year and its files are retained for one month.

Configuring the ADR

To configure the ADR, you only have to define its root directory by setting the value of the initialization parameter `DIAGNOSTIC_DEST`. This root directory is named as ADR base.

```
SELECT VALUE FROM V$PARAMETER WHERE NAME ='diagnostic_dest';

ALTER SYSTEM SET DIAGNOSTIC_DEST ='C:\ORACLE\diag';
```

Query the view `V$DIAG_INFO` to display all the ADR-related locations (also number of active problems and incidents):

```
SELECT NAME, VALUE FROM V$DIAG_INFO;
```

Using adrci Tool

Oracle 11g introduces `adrci` tool. This tool is used to examine contents of ADR repository and also to package information related to a specific problem into a compressed (zip) file to send it to Oracle support.

To invoke the tool, log in as the Oracle software owner and type `adrci` in the command line. To list the tool command line commands, type the following:

```
adrci –help
```

To display the tool commands, type the following command in the `adrci` command line:

```
adrci>help
```

To obtain guiding information about specific command, type the following:

```
adrci>help show incident
```

adrci commands will run on the ADR root (known when dealing with this tool as ADR base). To display the ADR base directory the tool is handling, type the following in the adrci command:

```
adrci>show base
```

adrci tool can deal with all the Oracle homes under the ADR base. If you want to specify which home should be handled by the tool, you must specify the *current homepath*. If you do not specify the current homepath, all the homes will be handled by the tool.

To display homes under the ADR root, issue the following command:

```
adrci>show homes
```

To display the current ADR homepath, issue the following command:

```
adrci>show homepath
```

To set the current ADR home, issue the following command:

```
adrci>set homepath diag\rdbms\ora11g\ora11g
```

Note You can specify multiple homes as current homepaths. In this case, adrci tool will deal with all the specified current homepaths. However, not all adrci commands can work with multiple current homepaths.

Any text output from adrci can be captured and saved in an external text file using spool command:

```
adrci>spool /u01/myfiles/myadrci.txt
adrci> ...
adrci>spool off
```

Using adrci to View the Alert Log

By default, adrci displays the alert log in your default editor. You can use the SET EDITOR command to change your default editor.

```
adrci>set editor notepad.exe
```

To display contents of the alert log file (xml tags will be excluded), issue the following command:

```
adrci>show alert
```

To see only the last 30 messages in the alert, modify the command as follows:

```
adrci>show alert –tail 30
```

To display messages containing ORA-600 errors, issue the following command:

```
adrci>show alert –p "MESSAGE TEXT LIKE '%ORA-600%'*
```

Using adrci to List Trace Files

Trace files can be listed in adrci using the following command:

```
adrci>show tracefile
```

Using adrci to View Incidents

Use the following command to obtain a report about all the incidents in the current homepath(s):

```
adrci>show incident
```

If you want to obtain further details about an incident, issue the command with –p (*predicate string*) option:

```
adrci>show incident –mode detail –p "incident_id=112564"
```

You can use many fields in the predicate options. To list all available fields, issue the command describe incident.

Using adrci to Package Incidents

With adrci tool, you can package all the diagnostic files related to specific problems into a ZIP file to submit it to Oracle support. To do so, you use special commands called IPS as shown in the following steps:

1. Create a logical package: use ips create package command to create an empty logical package as shown in the example below. The package will be given a serially generated number.

```
adrci>ips create package
```

2. Add diagnostic data to the logical package: this is done by ips add incident command as shown below:

```
adrci>ips add incident 112564 package 1
```

   Actually, there are formats of the ips create package command which enables you to perform the steps 1 and 2 in one command. Following are those command formats:
   - ips create package problem
   - ips create package problem key
   - ips create package incident
   - ips create package time

3. Generate the physical package. The files related to the incident will be collected in a ZIP file. The following example shows the command to perform this task:

```
adrci>ips generate package 1 in /u01/myfiles/incidents
```

   If you decide to add or change any diagnostic data later, you can do so by generating an *incremental* ZIP file. Modify the command as follows to achieve that:

```
adrci>ips generate package 1 in /u01/myfiles/incidents incremental
```

   You will notice that the generated file has the phase INC in its name indicating that it is an incremental ZIP file.

   ips commands behavior is controlled by various configuration options. To display those configuration options, use the command ips show configuration.

Using The Support Workbench in the OEM

The Support Workbench is a new OEM facility that has similar task to the command line utility adrci. It aims at examining and reporting critical errors occurring in the database. In order to use the option of uploading incident packages to Oracle, you must install Oracle Configuration Manager.

The facility is accessed by the link **Database Home Page>Software and Support>Support Workbench**.

Resolving a Critical Error

A critical error can be resolved with the Support Workbench by performing the following steps:

1. In OEM home page, examine the critical error alerts in the **Alerts** section table.
2. Click the **error link** provided in the **Message** column.
3. Click on **Go to Metalink** to create a service request. Of course, you will be asked to provide your Oracle Metalink account credentials.
4. In the **Investigate and Resolve section** of the Problem Details page, click on **Quick Package**.
   You have the option to create a Quick Package or Custom Package. With custom package, you can edit diagnostic data before uploading to data. Quick package does not give you this level of control.
5. **Follow the instructions** till you finish with submitting the package.
6. You can then revise the Problem Details page to perform the following:
   - Adding an Oracle bug number to the problem information.
   - Adding comments to the problem activity log.
   - Running the related Oracle advisors.
7. Once the problem is resolved, you close the incident. Follow the link **Support Workbench home>View Problem Details.** Click the incident you want to close and click **Close**.

   By default, open incidents are closed after 30 days. You can disable this automatic behavior in the **Incident Details** page.

Using SQL Repair Advisor

If a SQL statements fails because of a problem in its execution plan (caused for example by a bug), you may want the optimizer to make it running using another successful execution plan. SQL Repair Advisor analyzes failed SQL statements with critical errors and provides recommendations to apply a patch. The patch causes the query optimizer to choose an alternate execution plan for future executions.

To run the advisor, follow the link **Support Workbench home> Problem Details of the failed statement> Investigate and Resolve section> Self Service tab> Resolve heading> SQL Repair Advisor.**

To manage created SQL patches, follow the links **Database Home page> Server tab> Query Optimizer section> SQL Plan Control> SQL Patch tab**

Database Health Monitor

Oracle 11g introduced a new monitoring framework named as the Health Monitor. With Health Monitor, Oracle automatically runs a diagnostic operation (called *check*) to examine any database corruption or error. Checks run in two ways:

- **Reactive**: If the fault diagnosability infrastructure catches a critical error, it automatically runs health checks to search for any further failures related to the caught failure.
- **Manual (or proactive)**: A DBA may invoke the checks manually using DBMS_HM or the OEM.

Note Oracle recommends to frequently invoke the Health checks on the database when it is under low usage to catch any corruption or error that may cause damage in the future.

Health Monitor framework provides several types of database health checks. Following are the checks undertaken by the Health Monitor:

- DB Structure Check
- Data Block Integrity Check
- Redo Integrity Check
- Undo Segment Integrity Check
- Transaction Integrity Check
- Dictionary Integrity Check

To display list of the check that can be performed, issue the following query:

```
SELECT NAME, DESCRIPTION, OFFLINE_CAPABLE FROM V$HM_CHECK;
```

The OFFLINE_CAPABLE column defines whether you can perform the check when the database is offline or not.

Running Health Checks Using the DBMS_HM

A DBA can use DBMS_HM to manually invoke the database check. To retrieve the list of checks that can be run manually by users, issue the following query:

```
SELECT NAME FROM V$HM_CHECK WHERE INTERNAL_CHECK = 'N';
```

Use the procedure RUN_CHECK to perform a database health check. Its first parameter CHECKNAME is mandatory and it takes one of the returned names by the query above.

```
exec DBMS_HM.RUN_CHECK(CHECK_NAME=>'DB Structure Integrity Check', RUN_NAME=>'HM01');
```

Most health checks accept input parameters. You can view parameter names and descriptions with the V$HM_CHECK_PARAM view. Some parameters are mandatory while others are optional. The following query displays parameter information for all health checks:

```
select C.NAME CHECK_NAME, P.NAME PARAMETER_NAME, P.TYPE,
       P.DEFAULT_VALUE, P.DESCRIPTION
from   V$HM_CHECK_PARAM P, V$HM_CHECK C
where  P.CHECK_ID = C.ID and C.INTERNAL_CHECK = 'N'
order by C.NAME;
```

Input parameters are passed to the INPUT_PARAMS argument of the RUN_CHECK procedure as name/value pairs separated by semicolons (;). The following example illustrates how to pass the transaction ID as a parameter to the Transaction Integrity Check:

```
begin
 DBMS_HM.RUN_CHECK (
  CHECK_NAME => ' Transaction Integrity Check',      -- passed value is case sensitive
  RUN_NAME => 'MY_RUN', INPUT_PARAMS => 'TXN_ID=7.33.2');
end;
```

Database Health checks executions are stored in ADR and can be viewed by either querying the V$HM_RUN:

```
SELECT * FROM V$HM_RUN;
```

Another option is to run the adrci command show hm_run:

```
adrci>show hm_run
```

You can view a report of a particular Health check by using the following adrci command:

```
adrci>show report hm_run HM01
```

Alternatively, you can DBMS_HM package as shown in the following code example:

```
declare
 v_rpt clob;
begin
 v_rpt := DBMS_HM.GET_RUN_REPORT('HM01');
end;
```

Findings, if any, detected by the checks can be obtained from V$HM_FINDING and recommendations from V$HM_RECOMMENDATION.

Running Health Checks Using the Enterprise Manager

After connecting as SYSDBA, under the **Advisor Central** page, you will see **Checkers** link which can be used to manually invoke any Health check.

Data Recovery Advisor

Data Recovery Advisor is an Oracle Database 11g tool that automatically diagnoses data failures, determines and presents appropriate repair options, and executes repairs at the user's request. Data Recovery Advisor can diagnose failures such as the following:

- Inaccessible components like datafiles and control files.
- Physical corruptions such as block checksum failures and invalid block header
- Field values
- Inconsistent datafiles (online and offline)
- I/O failures

The advisor however doe not recover from failures on standby databases or RAC environment. This advisor can be used through RMAN or the Enterprise Manager.

Using Data Recovery Advisor with RMAN

Following are the RMAN commands to use Data Recovery Advisor:

1. List failures by running the LIST FAILURE command. Following are variations of using the command:

```
RMAN>LIST FAILURE;
RMAN>LIST OPEN;
RMAN>LIST CLOSED;
```

2. Optionally, execute LIST FAILURE ... DETAIL to list details of an individual failure.

```
RMAN>LIST FAILURE 105 DETAIL;
```

3. If you suspect that failures exist that have not been automatically diagnosed by the database, then run VALIDATE DATABASE to check for corrupt blocks and missing files. If a failure is detected, then RMAN logs it into the ADR, where it can be accessed by the Data Recovery Advisor.

4. Determine repair options by running the ADVISE FAILURE command.

```
RMAN>ADVISE FAILURE;
```

5. Choose a repair option. You can repair the failures manually or run the REPAIR FAILURE command to fix them automatically. By default, the REPAIR FAILURE command prompts the user to confirm the repair, but this can be prevented using the NOPROMPT keyword. Be aware that the previous command must be issued before using REPAIR FAILURE command.

   The following form of the command informs you how RMAN plans to repair the failure:

```
RMAN>REPAIR FAILURE PREVIEW
```

6. You may wish to change the priority of a failure (to HIGH or LOW), if it does not represent a problem to you, or even manually close it. This can be done by the CHANGE FAILURE command:

```
RMAN> CHANGE FAILURE 202 PRIORITY LOW;
```

Note Data Recovery Advisor may detect or handle some logical corruptions. But in general, corruptions of this type require help from Oracle Support Services.

Using Data Recovery Advisor with the Enterprise Manager

Access the Data Recovery Advisor in the Enterprise Manager by following the links: **Availability> Manage> Perform Recovery> Perform Automated Repair**.

SQL Test Case Builder

The SQL Test Case Builder aims at capturing the information pertaining to a SQL-related problem, along with the exact environment under which the problem occurred, so that the problem can be reproduced and tested on a separate Oracle database instance. Once the test case is ready, you can upload the problem to Oracle Support to enable support personnel to reproduce and troubleshoot the problem.

The information gathered by SQL Test Case Builder includes the query being executed, table and index definitions (but not the actual data), PL/SQL functions, procedures, and packages, optimizer statistics, and initialization parameter settings.

The output of the SQL Test Case Builder is a SQL script that contains the commands required to recreate all the necessary objects and the environment. The SQL Test Case Builder can be accessed using the interface package DBMS_SQLDIAG or the Enterprise Manager.

Accessing SQL Test Case Builder Using DBMS_SQLDIAG

The DBMS_SQLDIAG has a procedure named EXPORT_SQL_TESTCASE which is used to generate a SQL test case for a given SQL statement, SQL Id (taken from V$SQL) or an incident id. Following steps should be followed:

1. Create directory to hold the SQL test case files.

```
CREATE DIRECTORY sql_tes_dir AS 'C:\Oracle\TestCase';
```

2. Execute the proper form of EXPORT_SQL_TESTCASE procedure. Following is an example using a passed SQL statement.

```
DECLARE
 V_SQL CLOB := 'SELECT * FROM HR.NAMES WHERE ID BETWEEN 100 AND 1000';
 V_TESTCASE CLOB;
BEGIN
 DBMS_SQLDIAG.EXPORT_SQL_TESTCASE (
  DIRECTORY      =>'SQL_TES_DIR',
  SQL_TEXT       =>V_SQL,
  USER_NAME      =>'HR',
  BIND_LIST      =>NULL,
  EXPORTENVIRONMENT   =>TRUE,
  EXPORTMETADATA      =>TRUE,
  EXPORTDATA          =>TRUE,
  SAMPLINGPERCENT     =>100,
  CTRLOPTIONS         =>NULL,
  TIMELIMIT           =>0,
  TESTCASE_NAME       =>'RETURN_NAMES', -- generated scripts prefix
  TESTCASE            =>V_TESTCASE);
END;
```

Accessing SQL Test Case Builder Using the Enterprise Manager

From Enterprise Manager, the SQL Test Case Builder is accessible only when a SQL incident occurs. SQL-related problem is referred to as a SQL incident.

To access the SQL Test Case Builder, follow the links **the Support Workbench page> Click on an incident ID> Investigate and Resolve section> Oracle Support> Generate Additional Dumps and Test Cases> click on the icon in the Go To Task**.

Data Block Corruption Parameters

The new initialization parameter DB_ULTRA_SAFE sets the default values for other parameters that control protection levels. Also this parameter controls other data protection behavior in the Oracle Database, such as requiring ASM to perform sequential mirror writes. Using this parameter is discussed in the section New Initialization Parameters Affecting Database Creation.

Database Administration

Automatic Memory Management

In Oracle 11g, a new parameter named as MEMORY_TARGET is added to automate memory allocation for both the SGA and PGA. When this parameter is set, the SGA and the PGA memory sizes are automatically determined by the instance based on the database workload.

This parameter is dynamic and can be alter using the ALTER SYSTEM command as shown below:

```
ALTER SYSTEM SET MEMORY_TARGET = 1024M ;
```

However, if the database is not configured to use this parameter and you want to use it, you must restart the database after setting the parameter.

When you configure the database to use MEMORY_TARGET, you should take into consideration the following:

- The parameter STATISTICS_LEVEL must be set to TYPICAL
- The parameter MEMORY_MAX_SIZE controls the maximum value you can set for MEMORY_TARGET. If you do not set a value for this parameter, it defaults to MEMORY_TARGET.
- If you set the parameters SGA_TARGET and PGA_TARGET, Oracle will consider the values as the minimum values for SGA and PGA.
- If you do not set the parameters SGA_TARGET and PGA_TARGET (or set them to zero), no minimum value is considered by Oracle for the SGA and PGA. When the instance starts, it assigns 60 percent to SGA and 40 percent to the PGA.
- When MEMORY_TARGET is configured, the following components are auto tuned: DB BUFFER CACHE, SHARED POOL, JAVA POOL, LARGE POOL and STREAMS POOL.

To set a proper value to the parameter MEMORY_TARGET, query the view V$MEMORY_TARGET_ADVICE.

```
SELECT * FROM V$MEMORY_TARGET_ADVICE;
```

In the Enterprise Manager, follow the links **Database home page> Server tab> Database configuration section> Memory Advisors**

To display current status of the memory components, use the following query:

```
SELECT COMPONENT, ROUND(CURRENT_SIZE/1024/1024) CURRENT_SIZE ,
ROUND(MIN_SIZE/1024/1024) MIN, ROUND(MAX_SIZE/1024/1024) MAX
FROM V$MEMORY_DYNAMIC_COMPONENTS;
```

To know how Oracle has modified the memory area sizes by time, issue the following query:

```
select START_TIME, END_TIME, STATUS, COMPONENT, OPER_TYPE, OPER_MODE,
 PARAMETER, INITIAL_SIZE/1024/1024 INITIAL_SIZE_MB,
 TARGET_SIZE/1024/1024 TARGET_SIZE_MB, FINAL_SIZE/1024/1024 FINAL_SIZE_MB
from V$MEMORY_RESIZE_OPS
order by START_TIME, END_TIME
```

Note On Linux systems, if you receive the following error after setting the MEMORY_TARGET parameter, most likely the reason is that the /dev/shm is allocated a size (can be known by issuing the command df -k) less than SGA_MAX_SIZE:

```
ORA-00845: MEMORY_TARGET not supported on this system.
```

Resolving the issue can be done by the following OS commands:

```
#umount  /dev/shm
#mount -t tmpfs shmfs -o *size=><xx>m* /dev/shm
```

Automatic Maintenance Tasks

New Automatic Maintenance Task

When you install Oracle database 11g, the following predefined automatic maintenance tasks will be created in the scheduler:

- The Automatic Optimizer Statistics Collection task (predefined in Oracle 10g as well)
- The Automatic Segment Advisor task (predefined in Oracle 10g as well)
- The Automatic SQL Tuning Advisor (introduced in Oracle 11g). The SQL Tuning Automation section has a brief description about using this tool.

The later task issues the Automatic SQL Tuning Advisor which examines SQL statements performance and makes SQL profile recommendations to improve them. Detail about using this advisor will be discussed in Performance Management New Features chapter.

The Automated Maintenance Tasks are managed by the AutoTask Background Process (ABP). The process is spawned by the MMON background process at the start of the maintenance window. Following are the process functionalities:

- It converts automatic tasks into Scheduler jobs. It does not execute the maintenance tasks.
- It determines the jobs that need to be created for each maintenance task window.
- It stores task execution history in the SYSAUX tablespace.

To display list of the automatic tasks in the database, issue the following query:

```
SELECT TASK_NAME, OPERATION_NAME, STATUS
FROM DBA_AUTOTASK_TASK;
```

New Maintenance Windows

In Oracle database 11g, there are seven predefined maintenance windows. Their names derived from MONDAY_WINDOW to SUNDAY_WINDOW. Weekdays windows open at 10:00 p.m. and close at 2:00 a.m (4 hours). The weekend windows (Saturday and Sunday) open at 6:00 a.m. and close in 20 hours.

All those windows are assigned to a resource plan, DEFAULT_MAINTENANCE_PLAN, which is enabled automatically when the maintenance windows are opened. The DEFAULT_MAINTENANCE_PLAN resource plan has a number of consumer groups assigned to it named as ORA$AUTOTASK_*_GROUP. This plan prioritizes SYS_GROUP operations and allocates the remaining 5% to diagnostic operations and 25% to automated maintenance operations.

To display list schedule of the predefined maintenance windows in the next 32 days, issue the following query:

```
SELECT WINDOW_NAME, to_char(START_TIME,'DD-Mon-RR hh24:mi') START_TIME, DURATION
FROM DBA_AUTOTASK_SCHEDULE
ORDER BY WINDOW_NAME, START_TIME DESC;
```

To display the predefined consumer groups assigned to the automatic maintenance windows:

```
SELECT * FROM DBA_RSRC_CONSUMER_GROUPS G
WHERE G.CONSUMER_GROUP LIKE 'ORA$AUTO%'
```

Enabling and Disabling Maintenance Tasks

The DBMS_AUTO_TASK_ADMIN package can be to manage the automatic tasks.

To disable all automatic tasks in all maintenance windows, issue the following command:

```
exec DBMS_AUTO_TASK_ADMIN.DISABLE;
```

To disable a particular automated maintenance tasks for all maintenance windows, issue the following command:

```
Begin
 DBMS_AUTO_TASK_ADMIN.DISABLE(
  CLIENT_NAME => 'sql tuning advisor', -- as found in DBA_AUTOTASK_CLIENT
  OPERATION => NULL,
  WINDOW_NAME => NULL); -- all windows will be affected
end;
```

By passing a value to the WINDOW_NAME parameter, you specify to which maintenance window the task is to be disabled.

Similarly, ENABLE procedure is used to enable the automatic tasks in the maintenance windows. For example, to enable all automatic tasks in all the maintenance windows, issue the following command:

```
exec DBMS_AUTO_TASK_ADMIN.ENABLE;
```

Modifying a Maintenance Window

The DBMS_SCHEDULER package includes a SET_ATTRIBUTE procedure for modifying the attributes of a window. Note that you must disable the window before making changes to it. If you change a window when it is currently open, the change does not take effect until the next time the window opens.

For example, the following script changes the duration of the maintenance window SATURDAY_WINDOW to 4 hours:

```
begin
 DBMS_SCHEDULER.DISABLE( NAME => 'SATURDAY_WINDOW');
 DBMS_SCHEDULER.SET_ATTRIBUTE(
   NAME => 'SATURDAY_WINDOW',
   ATTRIBUTE => 'DURATION',
   VALUE => numtodsinterval(4, 'hour'));
 DBMS_SCHEDULER.ENABLE( NAME => 'SATURDAY_WINDOW');
end;
```

Creating a New Maintenance Window

To create a maintenance window, you must create a Scheduler window and then add it to the window group MAINTENANCE_WINDOW_GROUP as in the following example:

```
begin
 DBMS_SCHEDULER.CREATE_WINDOW(
  WINDOW_NAME    => 'EARLY_MORNING_WINDOW',
  DURATION       =>  numtodsinterval(1, 'hour'),
  RESOURCE_PLAN  => 'DEFAULT_MAINTENANCE_PLAN',
  REPEAT_INTERVAL=> 'FREQ=DAILY;BYHOUR=5;BYMINUTE=0;BYSECOND=0');
 DBMS_SCHEDULER.ADD_WINDOW_GROUP_MEMBER(
  GROUP_NAME  => 'MAINTENANCE_WINDOW_GROUP',
  WINDOW_LIST => 'EARLY_MORNING_WINDOW');
end;
```

Removing a Maintenance Window

To remove an existing maintenance window, remove it from the MAINTENANCE_WINDOW_GROUP window group. The window continues to exist but no longer runs automated maintenance tasks. Any other Scheduler jobs assigned to this window continue to run as usual.

```
begin
 DBMS_SCHEDULER.REMOVE_WINDOW_GROUP_MEMBER(
  GROUP_NAME => 'MAINTENANCE_WINDOW_GROUP',
  WINDOW_LIST => 'EARLY_MORNING_WINDOW');
end;
```

Managing Automatic Maintenance Tasks in the Enterprise Manager

Follow the link **Database Home page> Scheduler Central** at the bottom of the page.

From the scheduler home page you can see both automated maintenance tasks running via AutoTask, regular scheduler jobs, or Enterprise Manager jobs.

Automatic Maintenance Task Dictionary Views

Oracle Database 11g has removed the old jobs from the DBA_SCHEDULER_* views, and moved them to DBA_AUTOTASK_* views. Below is a list of queries from some of them.

The DBA_AUTOTASK_CLIENT view displays statistical data for each automated maintenance task (client) over 7-day and 30-day periods.

```
SELECT * FROM DBA_AUTOTASK_CLIENT ORDER BY CLIENT_NAME;
```

The DBA_AUTOTASK_CLIENT_HISTORY view displays per-window history of job execution counts for each automated maintenance task.

```
SELECT CLIENT_NAME, WINDOW_NAME, JOBS_CREATED, JOBS_STARTED, JOBS_COMPLETED
FROM DBA_AUTOTASK_CLIENT_HISTORY
WHERE CLIENT_NAME = 'auto optimizer stats collection';
```

The DBA_AUTOTASK_CLIENT_JOB view displays information about currently running scheduler jobs created for automated maintenance tasks.

```
select CLIENT_NAME, JOB_NAME, JOB_SCHEDULER_STATUS "Job status",
```

```
         TASK_NAME, TASK_TARGET_TYPE, TASK_TARGET_NAME "Name of the target",
         TASK_PRIORITY, TASK_OPERATION
from     DBA_AUTOTASK_CLIENT_JOB
order by CLIENT_NAME;
```

The DBA_AUTOTASK_JOB_HISTORY view displays the history of automated maintenance task job runs.

```
SELECT CLIENT_NAME, JOB_STATUS, JOB_START_TIME, JOB_DURATION
FROM DBA_AUTOTASK_JOB_HISTORY
WHERE CLIENT_NAME = 'auto optimizer stats collection';
```

Oracle Flashback-Related New Features

Oracle Database 11g introduces the following new Flashback-related features:

- LogMiner Interface in Oracle Enterprise Manager
- Oracle Flashback Transaction Backout
- Oracle Flashback Data Archives

Those features will be discussed in the following sections

LogMiner Interface in Oracle Enterprise Manager

Before Oracle Database 11g, in order to use the LogMiner to examine and rollback transactions, you needed to use the DBMS_LOGMNR package to perform command-line-driven log mining. In Oracle 11g, the Enterprise Manager has a graphical interface to extract transaction from the redo logs using LogMiner, which is much easier to use.

To invoke the LogMiner from OEM, follow the links **Database Homepage-> Availability-> View and Manage Transactions**

You can then retrieve the required transactions based on an entered range of time periods or SCNs.

Also, you can enter additional filtering conditions in the **Advanced Query** field under **Query Filter**. For example, to find transactions applied on the record where location = 'BOSTON' in the DEPT table in SCOTT schema, enter SCOTT.% in the TABLE field and "DBMS_LOGMNR.COLMN_PRESENT(UNDO_VALUE,'SCOTT.DEPT.LOC')= 'BOSTON'" in the Additional LogMiner Columns field.

Oracle Flashback Transaction Backout

This feature allows a DBA to back-out a committed transaction and all dependent transactions while the database is still online.

Setting Up for Flashback Transaction Backout

1. Enable supplemental logging with primary key logging as seen in this example:

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

2. Grant EXECUTE on DBMS_FLASHBACK to the user who will be performing the Flashback Transaction Backout.
3. Finally, grant SELECT ANY TRANSACTION to the user who will be performing the Flashback Transaction Backout.

Executing a Flashback Transaction Backout in OEM

Follow the links **Database home page> Schema tab in OEM> Database Objects section> Tables link> select a table> select the Flashback Transaction action** for the table you have selected.

Oracle will use the LogMiner to mine all the transactions on the table over the given period of time or SCN.

Executing a Flashback Transaction Backout Manually

Following are the steps to manually perform the backout:

1. You need to know the transaction ID. The V$TRANSACTION view provides this for you with the XID column if you are in the middle of a transaction. Optionally you could use Log Miner to find the candidate transaction IDs.

```
select A.SQL_TEXT, B.XID
from   V$OPEN_CURSOR A, V$TRANSACTION B, V$SESSION C
```

```
where  A.SID=C.SID AND C.TADDR=B.ADDR
  and  A.SQL_TEXT LIKE '%delete%';
```

2. Use the DBMS_FLASHBACK.TRANSACTION_BACKOUT procedure to back out the transaction.

```
declare
 V_XID SYS.XID_ARRAY;
begin
 V_XID := SYS.XID_ARRAY('03001800BC0D0000');
 DBMS_FLASHBACK.TRANSACTION_BACKOUT( NUMTXNS => 1,
    XIDS=>V_XID, OPTIONS=>DBMS_FLASHBACK.CASCADE);
end;
```

The OPTIONS parameter can also accept the value DBMS_FLASHBACK.NOCASCADE_FORCE. With this option, the user forcibly backs out the given transactions without considering the dependent transactions. The RDBMS executes the UNDO SQL for the given transactions in reverse order of their commit times. If no constraints break, and the result is satisfactory, the user can either COMMIT the changes or else ROLL BACK

Obtaining Information about Flashback Transaction Backouts

Information about transaction Backouts can be obtained from the following views:

- DBA_FLASHBACK_TXN_STATE any transaction shown in this view is backed out.
- DBA_FLASHBACK_TXN_REPORT provides information about the compensating status of all transactions in the database.

Flashback Data Archive

A flashback data archive (FDA) is a new object that holds historical data for one or several tables. FDA mechanism operates as follows:

- Once the FDA is enabled for a table, an internal history table is created for that table.
- A new background process named as FBDA wakes up at system-determined intervals (default five minutes) and copies the undo data for the marked transactions to the history table(s). Thus, undo data marked for archiving are not re-used by the database until it is stored by the FBDA into the history table.

Note You can find the internally assigned names of the history tables by querying the *_FLASHBACK_ARCHIVE_TABLES view. History tables are compressed and internally partitioned.

FDA is part of the "Oracle Total Recall" option in Oracle database 11g. Historical data is stored in compressed form to minimize storage requirements and it is completely transparent to applications.

Data Flashback Archive Requirements

To set up the DFA in a database, the following conditions must apply:

- Automatic undo management must be enabled in the database.
- The tablespace in which the FDA is created must be managed with Automatic Segment Space Management (ASSM).

Setting up the Data Flashback Archive

Following are the steps you should carry out to set up the DFA:

1. To define a system-level DFA, DBA role or the system privilege FLASHBACK ARCHIVE ADMINISTER must be granted. To display users or roles which granted this privilege, issue the following query:

```
SELECT * FROM DBA_SYS_PRIVS WHERE PRIVILEGE LIKE '%FLASHBACK ARC%'
```

2. Create FDA object. You can define its tablespace, retention period and quota size.

```
CREATE FLASHBACK ARCHIVE hr_hist
TABLESPACE fda_archives   -- mandatory (and it must be with ASSM)
QUOTA 5G                  -- optional in M,G,T,P
RETENTION 24 MONTH;       -- mandatory (in YEAR, MONTH, DAY)
```

For the command above, note the following:

- If QUOTA keyword is omitted, the flashback archive will take up all available space in the tablespace.
- Data exceeding the retention period will be automatically purged after one day of the retention expiry period.
- You can use the DEFAULT keyword (CREATE FLASHBACK ARCHIVE DEFAULT) to designate this flashback data archive as the default flashback data archive for the database. Using this keyword requires SYSDBA privilege.

Note At time of writing this document, the statement in the example above may return ORA-55603: Invalid Flashback Archive command error, if you try to create a flashback archive in a non-empty tablespace. I figured out a workaround which is to put the tablespace name between double quotations.

3. Enable Flashback Data Archiving for existing or new tables. A user who wants to do that should be granted either the SYSDBA privilege, the system privilege FLASHBACK ARCHIVE ADMINISTER, or the object privilege FLASHBACK ARCHIVE on the created flashback archive. Following is an example of granting the object privilege:

```
GRANT FLASHBACK ARCHIVE ON hr_hist TO scott;
```

Commands used to enable FDA on tables are illustrated in the examples blow:

```
-- Create the table, using the default archive location.
CREATE TABLE my_table(..) FLASHBACK ARCHIVE;

-- Modify a table to use the default archive location
--  Note: if there is not default flashback archive, an error will be raised
ALTER TABLE my_table FLASHBACK ARCHIVE;

-- Create a table to use a non-default archivelocation
CREATE TABLE my_table (..) FLASHBACK ARCHIVE hr_arc;

-- Modify a table to use a non-default archive location.
ALTER TABLE my_table FLASHBACK ARCHIVE hr_arc;

-- Modify a table to stop (disable) archiving.
ALTER TABLE my_table NO FLASHBACK ARCHIVE;
```

Note Disabling flashback archiving for a table or dropping its flashback archive object will result in all the historical data for that table being lost. It also requires SYSDBA or FLASHBACK ARCHIVE ADMINISTER privilege.

Altering Flashback Archives

Use ALTER FLASBACK command to alter a flashback archive object. Below are examples of using this command:

```
-- make myflash the default flashback archive (as SYSDBA)
ALTER FLASHBACK ARCHIVE myflash SET DEFAULT;

-- add space to the flashback archive
ALTER FLASHBACK ARCHIVE myflash ADD TABLESPACE mytbs;

-- remove the tablespace from use by the flashback archive
-- (assign it to another tablespace first)
ALTER FLASHBACK ARCHIVE myflash REMOVE TABLESPACE mytbs;

-- change the quota for the archive
ALTER FLASHBACK ARCHIVE myflash MODIFY TABLESPACE mytbs QUOTA 10G;

-- undefined quota (make the space unlimited)
ALTER FLASHBACK ARCHIVE myflash MODIFY TABLESPACE mytbs;

-- change the archive retention time
ALTER FLASHBACK ARCHIVE myflash MODIFY RETENTION 2 YEAR;

-- purge all archived data
ALTER FLASHBACK ARCHIVE myflash PURGE ALL;

-- purge data older than 2 days
ALTER FLASHBACK ARCHIVE MYFLASH
PURGE BEFORE TIMESTAMP( SYSTIMESTAMP – INTERVAL '2' DAY);
```

Using Oracle Flashback Data Archives

The normal flashback query and Flashback Versions Query can now use the Flash Archive data to retrieve old data. Following are illustrating examples:

```
SELECT LAST_NAME, SALARY FROM HR.EMPLOYEES
AS OF TIMESTAMP TO_TIMESTAMP ('2008-01-01 00:00:00','YYYY-MM-DD HH24:MI:SS');

SELECT LAST_NAME, SALARY FROM HR.EMPLOYEES
AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '6' MONTH);

SELECT LAST_NAME, SALARY FROM HR.EMPLOYEES
VERSIONS BETWEEN TIMESTAMP
  TO_TIMESTAMP ('2008-01-01 00:00:00','YYYY-MM-DD HH24:MI:SS') -- or MINVALUE
  AND
  TO_TIMESTAMP ('2008-01-01 15:00:00','YYYY-MM-DD HH24:MI:SS') -- or MAXVALUE
WHERE EMPLOYEE_ID = 200;
```

Furthermore, flashback table takes advantage of the Flashback Archive. Following is an example:

```
FLASHBACK TABLE employees TO TIMESTAMP (SYSTIMESTAMP – INTERVAL '6' MONTH);
```

Flashback Data Archive Limitations

For a table with Flashback Archiving enabled, you cannot issue the following DDL commands: `ALTER TABLE` (except adding a column), `DROP TABLE`, `RENAME TABLE` and `TRUNCATE TABLE`.

Obtaining Information about Flashback Data Archive

`DBA_FLASHBACK_ARCHIVE` provides information on all flashback archives contained in the database.

```
SELECT * FROM DBA_FLASHBACK_ARCHIVE;
```

`DBA_FLASHBACK_ARCHIVE_TS` provides information on all tablespaces containing flashback archives.

```
SELECT * FROM DBA_FLASHBACK_ARCHIVE_TS;
```

`DBA_FLASHBACK_ARCHIVE_TABLES` indicates which flashback archive a given table is assigned to.

```
SELECT TABLE_NAME, OWNER_NAME, FLASHBACK_ARCHIVE_NAME FROM DBA_FLASHBACK_ARCHIVE_TABLES;
```

Virtual Columns

In Oracle 11g, you can define a column in a table that contains derived data. You can use virtual columns in you queries, create indexes on them, and even collect statistics on them. There are a few restrictions including:

- You cannot write to a virtual column.
- There is no support for index-organized, external, object, cluster, or temporary tables.
- There is no support for Oracle-supplied datatypes, user-defined types, LOBs, or LONG RAWs.

Creating Tables with Virtual Columns

Following is the syntax you use to create a virtual column as part of the `CREATE TABLE` or `ALTER TABLE` statements:

```
column [datatype] [GENERATED ALWAYS] AS ( <column_expression> ) [VIRTUAL] [( inline_constraint [,...] )]
```

Note the following:

- `GENERATED ALWAYS` and `VIRTUAL` are optional and are just used to clarify that the data is not stored on disk.
- `COLUMN_EXPRESSION` defines the content of the virtual column. It has the following restrictions:
  - The expression cannot reference another virtual column.
  - All columns referenced in the expression must exist in the same table.
  - The output of the column expression must be a scalar value.

Following is a code example for creating a virtual column:

```
CREATE TABLE EMPLOYEES
( empno number PRIMARY KEY,
  sal   NUMBER (8,2) NOT NULL,
  annual_sal AS (sal*12),
  CONSTRAINT MaxAnSal CHECK (annual_sal BETWEEN 0 AND 2000000) );
```

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_DEFAULT
FROM DBA_TAB_COLUMNS
WHERE TABLE_NAME='EMPLOYEES' AND COLUMN_NAME='ANNUAL_SAL';
```

New Data Partitioning Schemes

New partitioning features introduced in Oracle Database 11g including the following:

- Interval partitioning
- Extended composite partitioning
- Reference partitioning
- System partitioning
- System-managed domain indexes

Refer to the Partitioning Improvements section in the Data Warehousing chapter for more details.

DDL Lock Timeout

In Oracle 11g, the new initialization parameter DDL_LOCK_TIMEOUT controls the duration (in seconds) for which a DDL statement will wait for a DML lock. The default value of zero indicates a status of NOWAIT. The maximum value of 1,000,000 seconds will result in the DDL statement waiting forever to acquire a DML lock.

```
ALTER SYSTEM SET DDL_LOCK_TIMEOUT = 60 ;
ALTER SESSION SET DDL_LOCK_TIMEOUT = 60 ;
```

Explicit Locking of Tables

The LOCK TABLE command has a new keyword named as WAIT which lets you specify the maximum time a statement should wait to obtain a DML lock on table.

```
LOCK TABLE .. IN <lockmode> MODE [ NOWAIT | WAIT n ]
```

For the provided syntax, note the following:

- n is an integer value in seconds
- If you do not specify either WIAT nor NOWAIT, Oracle will wait until the lock is available and acquired.

```
LOCK TABLE mytab IN EXCLUSIVE MODE WAIT 60;
LOCK TABLE mytab IN SHARE MODE NOWAIT;
```

Invisible Indexes

Invisible index is an index that is not considered by the optimizer when creating the execution plans. This can be used to test the effect of adding an index to a table on a query (using index hint) without actually being used by the other queries.

Following are the commands to create an invisible index, change visibility of an existing index and obtaining information about invisible indexes:

```
CREATE INDEX name_indx ON employees(emp_name) INVISIBLE;
SELECT /*+ index (EMP_NAME NAME_INDEX) */ ...

ALTER INDEX name_indx VISIBLE;
ALTER INDEX name_indx INVISIBLE;

SELECT INDEX_NAME, VISIBILITY FROM DBA_INDEXES WHERE INDEX_NAME='NAME_INDX';
```

When using invisible indexes, consider the following:

- If you rebuild an invisible index, the resulting operation will make the index visible.
- If you want the optimizer to consider the invisible indexes in its operation, you can set the new initialization parameter OPTIMIZER_USE_INVISIBLE_INDEXES to TRUE (the default is FALSE). You can set the parameter in the system and session levels.

Read-Only Tables

In Oracle 11g, you can set a table to be read only, i.e. users can only query from the table but no DML statement is allowed on the table. Following are the commands to achieve this:

```
ALTER TABLE employees READ ONLY;
ALTER TABLE employees READ WRITE;
SELECT TABLE_NAME, READ_ONLY FROM USER_TABLES WHERE TABLE_NAME='EMPLOYEES';
```

Deferred Segment Creation

Beginning with Oracle Database 11g Release 2, when you create heap or partitioned tables in a locally managed tablespace, the database defers table segment creation until the first row is inserted.

This feature saves disk space in applications that create hundreds or thousands of tables upon installation, many of which might never be populated.

Deferred segment table do not initially appear in *_segments

```
-- db level:
show parameter DEFERRED_SEGMENT_CREATION
alter system set deferred_segment_creation=true scope=both ;


-- table level
create table test ( .. ) SEGMENT CREATION DEFERRED partition by .. ;
create table test ( .. ) SEGMENT CREATION IMMEDIATE ;

-- (11.2.0.2) If you want to create the segments for objects where
--  SEGMENT CREATION DEFERRED is set without waiting for
--  inserting any rows:

-- all the objects in a schema:
conn / as sysdba
begin
 DBMS_SPACE_ADMIN.MATERIALIZE_DEFERRED_SEGMENTS (
 schema_name =>'SA');
end;

-- specific table:
begin
 DBMS_SPACE_ADMIN.MATERIALIZE_DEFERRED_SEGMENTS (
 schema_name =>'SA', table_name=>'EMP');
end;

-- specific partition
begin
 DBMS_SPACE_ADMIN.MATERIALIZE_DEFERRED_SEGMENTS (
  schema_name =>'SA', table_name=>'EMP',
  partition_name=>'PAR01');
end;
```

Shrinking Temporary Tablespaces and Tempfiles

In Oracle 11g, you can shrink temporary tablespaces and tempfiles. Following are the commands to achieve that:

```
ALTER TABLESPACE temp SHRINK SPACE KEEP 100M; -- the KEEP keyword is optional.
ALTER TABLESPACE temp SHRINK TEMPFILE '/u01/app/oracle/oradata/ora11g/temp01.dbf';
-- to obtain information about temporary space usage
SELECT * FROM DBA_TEMP_FREE_SPACE
```

Creating an Initialization Parameter File from Memory

In Oracle 11g, you can create a pfile or spfile from the current values of the initialization parameters (active in the instance). Following is the code example to do so:

```
CREATE PFILE FROM MEMORY;
CREATE SPFILE FROM MEMORY;
```

The generated file will contain all the initialization parameters used by the instance whether exactly set by the DBA or taken as default values.

Restore Point Enhancements

Creating Restore Point "as of" an SCN or a Timestamp

With Oracle 11g, you can create a restore point for a specific SCN in the past or a past point in time.

```
CREATE RESTORE POINT res_jun08 AS OF SCN 2340009;
CREATE RESTORE POINT res_jun08 AS OF TIMESTAMP to_date('01-04-2008 07:30','DD-MM-YYYY
HH24:MI');
```

The database must be able to flashback to the time point you select to create the restore point.

Preserving Restore Points

In Oracle database 10g, Oracle may reach to situations where it deletes restore points (but not the guaranteed restore points). In Oracle 11g, you can create a restore point with PRESERVE option to prevent Oracle from deleting it.

```
CREATE RESTORE POINT myrp PRESERVE;
```

Using Restore Points with Creating Archival Backups

Restore points can be used during the implementation of archival backups. Details of this type of backup are explained in Archival Backups section.

Database Resident Connection Pooling

Database Resident Connection Pooling (DRCP) is a pool of dedicated servers in the database for typical Web application usage scenarios. DRCP can be used by clients that use OCI driver including C, C++ and PHP.

How it works

1. When a client process or thread connects to the database, the connection is established between the client and the connection broker (it is a new background process named as connection monitor CMON) in Oracle database.
2. When the client sends a request, the connection broker will assign the client connection to one of the pooled servers, if available. If no pooled server is free, the connection broker will create a new pooled server. If number of the existing pooled servers reaches to the maximum, the client connection will go into a wait queue till one of the pooled servers becomes free.
3. The client connection releases the pooled server back to the connection pool once its request is serviced by the database.

DRCP Advantages

DRCP provides the following advantages:

- DRCP is especially useful for architectures with multi-process, single-threaded application servers, such as PHP and Apache servers, that cannot do middle-tier connection pooling.
- DRCP enables a significant reduction (compared to the dedicated server mechanism) in key database resources (specially the memory) that are required to support a large number of client connections.
- DRCP boosts the scalability of both the database server and the middle-tier.
- The pool of readily available servers also reduces the cost of re-creating client connections.

Configuring DRCP

DRCP is automatically created with any Oracle 11g database but it is disabled by default. The following sub-section illustrates how to enable it. DRCP is controlled by the following configuration parameters:

INACTIVITY_TIMEOUT	*maximum idle time for a pooled server before it is terminated.*
MAX_LIFETIME_SESSION	time to live TTL duration for a pooled session.
MAX_USE_SESSION	maximum number of times a connection can be taken and released to the pool.
MAX_SIZE and MIN_SZIE	the maximum and minimum number of pooled servers in the connections pool.
INCRSIZE	pool would increment by this number of pooled server when pooled server are unavailable at application request time.
MAX_THINK_TIME	maximum time of inactivity by the client after getting a server from the pool. If the client does not issue a database call after grabbing a server from the pool, the client will be forced to relinquish control of the pooled server and will get an error. The freed up server may or may not be returned to the pool.
SESSION_CACHED_CURSORS	turn on SESSION_CACHED_CURSORS for all connections in the pool. This is an existing initialization parameter

To modify value of any of the parameters above, use ALTER_PARAM procedure in DBMS_CONNECTION_POOL package as show in the example below:

```
begin
 DBMS_CONNECTION_POOL.ALTER_PARAM( PARAM_NAME =>'INACTIVITY_TIMEOUT',
                                   PARAM_VALUE=>'3600');
end;
```

To obtain information about the DRCP configuration, you can query the view DB_CPOOL_INFO.

To restore all the connection pool configuration parameters to their default values, use the code below:

```
exec DBMS_CONNECTION_POOL.RESTORE_DEFAULTS();
```

Enabling DRCP

To enable the DRCP, after connecting as SYSDBA use START_POOL procedure in the DBMS_CONNECTION_POOL package as show in the example below:

```
exec DBMS_CONNECTION_POOL.START_POOL();
```

Note If you start the DRCP and then the instance is restarted, the pool is automatically started.

Disabling DRCP can be done with the code below:

```
exec DBMS_CONNECTION_POOL.STOP_POOL();
```

Client Connection Configuration

If the client is using easy Connect string, DRCP is specified in the following way:

```
hostname.company.com:1521/mydb.company.com:POOLED
```

If the client is using tnsnames.ora file, the configuration should be as follows:

```
MYDB = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp) (HOST=hostname.company.com)
       (SERVER=POOLED)))
```

Also, Oracle extended the OCISessionPool APIs so that they work with DRCP. For further details on that, see the documentation *Oracle Call Interface Programmer's Guide*.

Monitoring DRCP

To monitor DRCP in a database, use the following data dictionary views:

V$_CPOOL_INFO	Shows the status of the connection pool and its configuration information.
V$CPOOL_STATS	Shows statistics about the pool.
V$CPOOL_CC_STATS	Shows details about the connection class-level statistics.

Following are some queries on the views above:

```
SELECT STATUS,MINSIZE,MAXSIZE,INCRSIZE,SESSION_CACHED_CURSORS,INACTIVITY_TIMEOUT
FROM DBA_CPOOL_INFO;

SELECT NUM_OPEN_SERVERS, NUM_BUSY_SERVERS, NUM_REQUESTS, NUM_HITS
       NUM_MISSES, NUM_WAITS, NUM_PURGED, HISTORIC_MAX
FROM V$CPOOL_STATS;
```

Comparing and Synchronizing Database Objects

In Oracle 11g, the DBMS_COMPARISON package is developed to use it when you want to compare two database objects in replicated databases and, if you wish, synchronize data between them. This is practically useful when you have two objects in replicated databases (*shared database objects*) out of synchronization. The package allows you to compare tables, simple views or materialized views (or of course synonyms for them).

To use DBMS_COMPARISON package, the two objects must have a column with an index of primary key or non-null unique constraint. This column is used to uniquely identify the rows in both tables and known as *index column*.

Following are example of using the DBMS_COMPARISON package in comparing two tables in different databases:

1. Run the CREATE_COMPARE procedure in this package to create a comparison.

```
begin
DBMS_COMPARISON.CREATE_COMPARISON(
 COMPARISON_NAME   =>'Compare_EMP',
 SCHEMA_NAME       =>'HR',
 OBJECT_NAME       =>'EMPLOYEES',
 DBLINK_NAME       =>'REMOTE_DB',
 REMOTE_SCHEMA_NAME=>'HR',
 REMOTE_OBJECT_NAME=>'EMP2' );
end;
```

Note If you want to compare two objects in the same database, as a workaround, you can create a database link connecting to the same local database.

2. Run the COMPARE function. This function populates data dictionary views with comparison results. You can invoke the function later at any time to re-compare the objects.

```
declare
 V_CONSISTEN  BOOLEAN;
 V_SCAN_INFO DBMS_COMPARISON.COMPARISON_TYPE;
begin
 V_CONSISTEN  := DBMS_COMPARISON.COMPARE(
   COMPARISON_NAME => 'Compare_Emp',
   SCAN_INFO       => V_SCAN_INFO,
   PERFORM_ROW_DIF =>TRUE );
 DBMS_OUTPUT.PUT_LINE('Scan ID: ' || V_SCAN_INFO.SCAN_ID);
 IF V_CONSISTEN  THEN
  DBMS_OUTPUT.PUT_LINE('Data are synchronized.');
 ELSE
  DBMS_OUTPUT.PUT_LINE('Data are out of synchronization.');
 END IF;
end;
```

3. If you want to examine the comparison results, query the following data dictionary views:

DBA_COMPARISON_SCAN	USER_COMPARISON_SCAN
DBA_COMPARISON_SCAN_SUMMARY	USER_COMPARISON_SCAN_SUMMARY
DBA_COMPARISON_SCAN_VALUES	USER_COMPARISON_SCAN_VALUES
DBA_COMPARISON_ROW_DIF	USER_COMPARISON_ROW_DIF

Following is an example code of using those views to view number of rows that were found different in the two tables:

```
SELECT C.OWNER, C.COMPARISON_NAME, C.SCHEMA_NAME, C.OBJECT_NAME,
 S.CURRENT_DIFF_COUNT
FROM DBA_COMPARISON C, DBA_COMPARISON_SCAN_SUMMARY S
WHERE C.COMPARISON_NAME =  S.COMPARISON_NAME AND C.OWNER = S.OWNER AND SCAN_ID = 45;
```

Following is a query to find the rowid or index values of the rows that either exist only in one of the two data sets or have differences in their data values.

```
SELECT c.COLUMN_NAME, r.INDEX_VALUE
, case when r.LOCAL_ROWID is null
   then 'No'
   else 'Yes'
  end LOCAL_ROWID
, case when r.REMOTE_ROWID is null
   then 'No'
   else 'Yes'
  end REMOTE_ROWID
FROM USER_COMPARISON_COLUMNS c, USER_COMPARISON_ROW_DIF r, USER_COMPARISON_SCAN s
WHERE c.COMPARISON_NAME = 'COMPARE_EMP_AND_CLONE'
 AND r.SCAN_ID = s.SCAN_ID
 AND s.last_update_time > systimestamp – (1/24/15)
 AND r.STATUS = 'DIF'
 AND c.INDEX_COLUMN = 'Y'
 AND c.COMPARISON_NAME = r.COMPARISON_NAME
ORDER BY r.INDEX_VALUE
```

4. If there are differences, and you want to synchronize the database objects at the two databases, then run the CONVERGE procedure in this package as shown in the following code example:

```
SET SERVEROUTPUT ON;
declare
 V_SCAN_INFO DBMS_COMPARISON.COMPARISON_TYPE;
begin
 DBMS_COMPARISON.CONVERGE(
   COMPARISON_NAME => 'Compare_Emp',
   SCAN_ID         =>45,
   SCAN_INFO       => V_SCAN_INFO,
   CONVERGE_OPTIONS=>DBMS_COMPARISON.CMP_CONVERGE_LOCAL_WINS);
               -- alternatively use CMP_CONVERGE_REMOTE_WINS
 DBMS_OUTPUT.PUT_LINE('Local Rows Merged:' || V_SCAN_INFO.LOC_ROWS_MERGED);
 DBMS_OUTPUT.PUT_LINE('Remote Rows Merged:'|| V_SCAN_INFO.RMT_ROWS_MERGED);
 DBMS_OUTPUT.PUT_LINE('Local Rows Deleted:' || V_SCAN_INFO.LOC_ROWS_DELETED);
 DBMS_OUTPUT.PUT_LINE('Remote Rows Deleted:' || V_SCAN_INFO.RMT_ROWS_DELETED);
end;
```

Merged rows in this context mean they were replaced. The option CMP_CONVERGE_LOCAL_WINS indicates that the column values at the local database replace the column values at the remote database when these column values are different. This constant can be specified as 'LOCAL'.

Note To run the COMPARE function or CONVERGE procedure, the following users must have SELECT privilege on each copy of the shared database object:

- The comparison owner at the local database.
- When a database link is used, the user at the remote database to which the comparison owner connects through a database link.

The CONVERGE procedure also requires additional privileges for one of these users at the database where it makes changes to the shared database object. The user must have INSERT, UPDATE, and DELETE privileges on the shared database object at this database.

SQL*Plus New Features

SQL*Plus Error Logging

In Oracle 11g SQL*Plus, the new command SET ERRORLOGGIN can be used to store all errors resulting from executed SQL, PL/SQL and SQL*Plus commands in special error logging table (by default it is SPERRORLOG).

Following are the commands to enable and disable error logging in SQL*Plus:

```
SQL>SHOW ERRORLOGGING
SQL>SET ERRORLOGGIN ON
SQL>SET ERRORLOGGIN OFF
```

The **SET ERRORLOGGIN ON** command creates the error log table. To view the error stored in the table, use a query like the following:

```
SELECT USERNAME, STATEMENT, MESSAGE
FROM SPERRORLOG;
```

New SQL*Plus Connection Syntax

The CONNECT command in Oracle 11g SQL*Plus is enhanced to allow connecting as ASM administrator.

```
CONN[ECT] [{logon | / } [AS {SYSOPER | SYSDBA | SYSASM}]]
```

Online Application Maintenance

Enhanced Online Index Creation and Rebuild

Before Oracle 11g, rebuilding an index online on a table with extensive DML leads to unpredictable results because table exclusive lock (X) is required by rebuilding process. In Oracle 11g, rebuilding an index acquire shared exclusive lock (SX) on the table allowing DML to go on uninterrupted. This new enhancement applies in the following statements:

- Create index online
- Rebuild index online
- Create materialized view log

Enhanced ADD COLUMN Functionality

Before Oracle 11g, adding new columns with DEFAULT values and NOT NULL constraint requires an exclusive lock on the table and the default value to be stored in all existing records. In Oracle 11g, the default value will be stored in the data dictionary instead of the table itself. This will save space (for large tables) and significantly reduce time to execute the modification statement.

Finer Grained Dependencies

Invalidation of dependent objects resulted from modifying structure of the independent object is defused in Oracle 11g. For example, if a view or a PL/SQL procedure references a table and you added a column to the table, the view and the procedure are *not* invalidated. However, if you modify a column in the table and that column is referenced by the view or the procedure, they will be invalidated.

Oracle Advanced Compression Option

Oracle Database 11g has new option named as Oracle Advanced Table Compression option which aims at reducing space occupied by data for both OLTP and warehouse databases. This option provides the following types of compression:

- Compression of data tables even for OLTP environment. (Previous versions had compression option for tables that are mostly read only).
- Compression of unstructured data in SecureFiles.
- Compression of RMAN backups.
- Compression in Data Pump Export files.
- Compression of redo data transmitted to a standby database during redo gap resolution (when data guard is configured).

Note Using Advanced Compression Option in Oracle 11g requires a separate license from Oracle.

Table Compression

Table compression has the advantages of saving storage space, increased I/O performance and reduction in buffer cache memory. On the other hand, it incurs a CPU overhead.

The compression can be specified at the tablespace, table or partition level with the following options:

- `NOCOMPRESS` - The table or partition is not compressed. This is the default.
- `COMPRESS` - This option is considered suitable for data warehouse systems. Compression is enabled on the table or partition during direct-path inserts only.
- `COMPRESS FOR DIRECT_LOAD OPERATIONS` - This option has the same affect as the simple `COMPRESS` keyword (as with Oracle 10g).
- `COMPRESS FOR ALL OPERATIONS` - This option is considered suitable for OLTP systems. As the name implies, this option enables compression for all operations, including regular DML statements. This option requires the `COMPATIBLE` initialization parameter to be set to 11.1.0 or higher.

Following are examples to use the `COMPRESS` clauses:

```
CREATE TABLE ... COMPRESS FOR ALL OPERATIONS;

CREATE TABLE mytab (  ..   created_date  DATE NOT NULL  )
PARTITION BY RANGE (created_date) (
  PARTITION mytab_q1 VALUES LESS THAN (to_date('01/01/2008', 'DD/MM/YYYY')) COMPRESS,
  PARTITION mytab_q2 VALUES LESS THAN (to_date('01/04/2008', 'DD/MM/YYYY')) COMPRESS FOR
DIRECT_LOAD OPERATIONS,
  PARTITION mytab_q3 VALUES LESS THAN (to_date('01/07/2008', 'DD/MM/YYYY')) COMPRESS FOR
ALL


CREATE TABLESPACE mytbs ..
  DEFAULT COMPRESS FOR ALL OPERATIONS;
```

The table compression has the following restrictions:

- You can add or drop columns to a compressed table, only if the `COMPRESS FOR ALL OPERATIONS` option was used.
- Compressed tables cannot have more than 255 columns.
- Compression is not applied to LOB segments.
- The compression clause cannot be applied to hash or hash-list partitions. Instead, they must inherit their compression settings from the tablespace, table or partition settings.
- Table compression cannot be specified for external, clustered or index organized tables.

Compression in SecureFiles

The Oracle SecureFiles feature is an alternative solution to LOB to store unstructured data such as XML files, datasheets and word processor documents. With this feature, compression is implemented by eliminating redundant copies of the same document. In this case, all the copies will point to the same document image.

For further details about SecureFiles, see the section SecureFiles.

Compression in RMAN Backups

The Oracle Advanced Compression option reduces the compression ratio in RMAN backup (by 20%) and increases the backup performance (by 40%) than it does in Oracle 10g.

Compression in Data Pump Export Files

In Oracle 11g, you can compress the export data files in a Data Pump export job (in Oracle 10g, only metadata can be compressed). For further details about using the compression in Data Pump export files, see the section Compression Enhancement in Data Pump.

Compression of Transmitted Redo Data

With the Oracle Advanced Compression option, data transmitted during *redo gab resolution* when Oracle guard is implemented is compressed. This results in increasing the throughput of the process and finishing it in a shorter time (about two times faster).

Oracle Scheduler New Features

Lightweight Jobs and Remote External Jobs are two enhancements included in the scheduler. Those enhancements are discussed in the following sections.

Lightweight Jobs

In Oracle 11g, Schedule jobs are divided into two types: *regular* and *lightweight* jobs. Regular jobs are created and dropped after each execution. They do not cause re-creating overhead and are faster to create. Use lightweight jobs when you need to create and drop hundreds or thousands of jobs per second.

Lightweight jobs must be based on job templates. A job template is a database object that provides the necessary metadata needed for running a job (other than a schedule) and provides a privilege infrastructure that can be inherited by any lightweight job. Typically, a job template is a *program* object with an object type of 'PLSQL_BLOCK' or 'STORED_PROCEDURE'.

The following example creates a lightweight job using a program as a template:

```
DBMS_SCHEDULER.CREATE_JOB (
 JOB_NAME   =>'Light_Job1',
 PROGRAM_NAME    =>'My_Prog',  -- an existing scheduler program object
 REPEAT_INTERVAL =>'freq=daily;by_hour=9',
 ENABLED    =>FALSE, -- default
 AUTO_DROP  =>TRUE,  -- default
 COMMENTS   =>'Lightweight Job',
 JOB_STYLE  => 'LIGHTWEIGHT'); -- or REGULAR
```

The following example creates a set of lightweight jobs in one transaction:

```
declare
  NEWJOB SYS.JOB;
  NEWJOBARR SYS.JOB_ARRAY;
begin
  NEWJOBARR := SYS.JOB_ARRAY();
  NEWJOBARR.EXTEND(5);
 FOR I IN 1..5 LOOP
   NEWJOB := SYS.JOB(JOB_NAME => 'LWJOB_' || TO_CHAR(I),
                     JOB_STYLE => 'LIGHTWEIGHT',
                     JOB_TEMPLATE => 'My_Prog',  -- not PROGRAM_NAME
                     REPEAT_INTERVAL => 'FREQ=MINUTELY;INTERVAL=15',
                     START_DATE => SYSTIMESTAMP + INTERVAL '10' SECOND,
                     ENABLED => TRUE );
                     NEWJOBARR(I) := NEWJOB;
 END LOOP;
 -- create multiple jobs in a single call
 DBMS_SCHEDULER.CREATE_JOBS(NEWJOBARR, 'TRANSACTIONAL');
end;
```

Remote External Jobs

Configuring Remote External Jobs Functionality

In Oracle 11g, you can create in scheduler an external job that runs on a remote host. This type of job is called *remote external* job.

To be able to create a remote external job, the following conditions apply:

- Oracle Scheduler agent (not the database) should be installed on the remote host.
- Register the scheduler agent with the database that needs to run external jobs in the remote host.

To install the scheduler agent, perform the following steps:

1. Run the Oracle Universal Installer from the Oracle Database Gateway software.
2. Select the Oracle Scheduler Agent as the software to install.
3. Specify the hostname and port number.
4. After installation is finished, run `root.sh` as a root user.

To register a database in the scheduler agent, perform the following steps:

1. In the database you want to register, issue the following command as a SYS user:

```
SQL>@ ORACLE_HOME/rdbms/admin/prvtrsch.plb
```

2. Set a registration password for the scheduler agent using the following command:

```
exec DBMS_SCHEDULER.SET_AGENT_REGISTRATION_PASS('mypassword');
```

3. Register the scheduler agent with the database using the following command:

```
$schagent –registerdatabase database_host database_xmldb_http_port
```

To find out the value of the http port, issue the following command:

```
SQL>select dbms_xdb.gethttpport from dual;
```

4. Start the scheduler agent with the following command:

```
$schagent –start
```

5. The scheduler agent can be stopped with the following command:

```
$schagent –stop
```

Creating a Remote External Job

To create a remote external job, the following steps should be carried out:

1. Create a credential for running the remote external job using the following command:

```
exec DBMS_SCHEDULER.CREATE_CREDENTIAL('HostNameCredential', 'OSusername','mypassword');
```

Information about the credentials in the database can be obtained by querying the view DBA_SCHEDULER_CREDENTIALS.

2. Grant the object privilege EXECUTE on the created credential to the required database user:

```
GRANT EXECUTE ON HostNameCredential TO scott;
```

3. Use the following code to create the external job:

```
begin
-- create the job
DBMS_SCHEDULER.CREATE_JOB(
 JOB_NAME    => 'EXT_JOB1',
 JOB_TYPE    => 'EXECUTABLE',
 JOB_ACTION  => '/u01/app/oracle/sendfiles',
 REPEAT_INTERVAL =>'freq=daily;by_hour=9',
 ENABLED     =>FALSE);
-- define its credential
 DBMS_SCHEDULER.SET_ATTRIBUTE (
  NAME      =>'EXT_JOB1',
  ATTRIBUTE =>'CREDENTIAL_NAME',
  VALUE     =>'HostNameCredential' );
-- define its destination
DBMS_SCHEDULER.SET_ATTRIBUTE (
  NAME      =>'EXT_JOB1',
  ATTRIBUTE =>'DESTINATION',
  VALUE     =>'rhostname:12345' ); -- hostname and port number listened to by the agent
-- enable the job
DBMS_SCHEDULER.ENABLE ('EXT_JOB1');
end;
```

To know the port number the scheduler agent is listening to, view the file schagent.conf in the scheduler agent home directory.

Monitoring Job State with Email Notifications

You can configure a job to send e-mail notifications when it changes state.

```
/* Configuration */
CONN / AS SYSDBA
BEGIN
  -- define the SMTP server
  DBMS_SCHEDULER.set_scheduler_attribute('email_server', 'smtp.mydomain.com:25');
```

```
  -- optionally define default sender address, which
  --  is used if the sender parameter is not specified
  DBMS_SCHEDULER.set_scheduler_attribute('email_sender', 'info@mydomain.com');
END;
/


-- to enable/disable encryption is for the SMTP server connection
--  only (11.2.0.2)
-- possible values: NONE, SSL_TLS, STARTTLS
exec DBMS_SCHEDULER.set_scheduler_attribute('email_server_encryption','SSL_TLS')


-- Authentication
-- If the SMTP server requires authentication, then the Scheduler uses the
-- user name and password stored in the specified credential object
-- default NULL
exec dbms_scheduler.create_credential('hrcredential','hr','hrpassword');
exec DBMS_SCHEDULER.set_scheduler_attribute('email_server_credential','hrcredential')



/* Using Email Notification */
-- You call ADD_JOB_EMAIL_NOTIFICATION once for each different set of notifications
-- that you want to configure for a job.

-- associate an email notification with the job
-- using the default subject and body
BEGIN
 DBMS_SCHEDULER.add_job_email_notification (
  job_name   =>  'email_notification_job',
  recipients =>  'info@ahmedbaraka.com',
  events     =>  'job_started, job_succeeded');
END;

-- subject and body specified:
BEGIN
DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
 job_name => 'email_notification_job',
 recipients => 'info@ahmedbaraka.com, alissa@mydomain.com',
 sender => 'do_not_reply@example.com',
 subject => 'Scheduler Job Notification-%job_owner%.%job_name%-%event_type%',
 body => '%event_type% occurred at %event_timestamp%. %error_message%',
 events => 'JOB_FAILED, JOB_BROKEN, JOB_DISABLED, JOB_SCH_LIM_REACHED');
END;


-- configures an additional e-mail notification for the same job
-- for a different event
BEGIN
DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
 job_name => 'email_notification_job',
 recipients => 'info@ahmedbaraka.com',
 events => 'JOB_OVER_MAX_DUR');
END;


-- The notification fires only if a job fails with "600" "700" error codes
BEGIN
 DBMS_SCHEDULER.add_job_email_notification (
   job_name         =>  'email_notification_job',
   recipients       =>  'info@ahmedbaraka.com',
   events           =>  'job_failed',
   filter_condition => ':event.error_code=600 or :event.error_code=700');
END;



/* Removing Email Notification */
```

```
-- remove the notification from specified recipient/event
BEGIN
 DBMS_SCHEDULER.remove_job_email_notification (
  job_name   =>  'email_notification_job',
  recipients =>  'info@ahmedbaraka.com',
  events     =>  'job_succeeded');
END;

-- remove the notification from all recipients and events
BEGIN
 DBMS_SCHEDULER.remove_job_email_notification (
  job_name   =>  'email_notification_job');
END;



/* Obtain Info about Email Notifications */
SELECT job_name, recipient, event, filter_condition, subject, body
FROM user_scheduler_notifications;
```

File Watcher

A file watcher is a new scheduler object that enables a new type of even-based job that is triggered by the arrival of a file in a specified location. File watchers can be defined to monitor locations on the local server and remote servers, provided they have an agent installed on them.

```
/* Obtain Info about FW */
SELECT file_watcher_name, destination, directory_path, file_name, credential_name
FROM   user_scheduler_file_watchers;



/* Configuration */
-- by default, a destination is checked every 10 mins. To change this:
CONN / AS SYSDBA
set serveroutput on
declare
 v varchar2(1000);
begin
 DBMS_SCHEDULER.GET_ATTRIBUTE ( 'FILE_WATCHER_SCHEDULE','REPEAT_INTERVAL', v);
 DBMS_OUTPUT.PUT_LINE(V);
end;
/

BEGIN
  DBMS_SCHEDULER.set_attribute(
    'file_watcher_schedule',
    'repeat_interval',
    'freq=minutely; interval=5');
END;
/




/* Creating File Watcher */
-- create OS credential:
BEGIN
  DBMS_SCHEDULER.create_credential(
    credential_name => 'fw_credential',
    username        => 'oracle',
    password        => 'oracle');
END;
/


-- Grant EXECUTE on the credential to the schema that owns the
--  event-based job that the file watcher will start:
```

```
GRANT EXECUTE ON fw_credential to DSSUSER;


-- create file watcher:
BEGIN
  DBMS_SCHEDULER.create_file_watcher(
    file_watcher_name => 'data_fw',
    directory_path    => '/tmp/test', -- if '?' = ORACLE_HOME
    file_name         => '*.dat', -- wildcard supported
    credential_name   => 'fw_credential',
    destination       => NULL, -- NULL=local server
    enabled           => FALSE);
END;
/


-- Grant EXECUTE on the file watcher to any schema that owns an event-based job
--  that references the file watcher.
GRANT EXECUTE ON data_fw to DSSUSER;


-- create a program raised by the file watcher
BEGIN
  DBMS_SCHEDULER.create_program(
    program_name        => 'import_data_prog',
    program_type        => 'stored_procedure',
    program_action      => 'import_data_proc',
    number_of_arguments => 1,
    enabled             => FALSE);
END;
/


-- define the metadata argument using the event_message attribute
--  the metadata contains info about the file, such as its name:
BEGIN
  DBMS_SCHEDULER.define_metadata_argument(
    program_name       => 'import_data_prog',
    metadata_attribute => 'event_message',
    argument_position  => 1);
END;
/


-- create the defined procedure:
-- It must accept an argument of the SCHEDULER_FILEWATCHER_RESULT type
CREATE TABLE received_files ( fileinfo VARCHAR2(4000), rdate date );

CREATE OR REPLACE PROCEDURE import_data_proc
 (p_sfwr SYS.SCHEDULER_FILEWATCHER_RESULT) AS
 v_message received_files.fileinfo%type;
BEGIN
  v_message  :=  p_sfwr.directory_path  ||  '/'  ||  p_sfwr.actual_file_name  ||  '  ('  ||
p_sfwr.file_size || ')';

  INSERT INTO received_files
  VALUES (v_message, sysdate);
  COMMIT;
END;
/



-- create the job:
BEGIN
  DBMS_SCHEDULER.create_job(
    job_name        => 'import_data_job',
    program_name    => 'import_data_prog',
    event_condition => NULL,  -- 'tab.user_data.file_size < 1024'
    queue_spec      => 'data_fw', -- file watcher name
```

```
    auto_drop       => FALSE,
    enabled         => FALSE);
END;
/



-- By default, the arrival of new files will be ignored if the job is already running.
--  If you need the job to fire for each new arrival, regardless of whether the job is
already
--  running or not, set the PARALLEL_INSTANCES attribute for the job to true. The job
--  will then be run as a lightweight job:
BEGIN
  DBMS_SCHEDULER.set_attribute('import_data_job','parallel_instances',TRUE);
END;
/



-- Enable all the objects:
EXEC DBMS_SCHEDULER.enable('data_fw,import_data_prog,import_data_job');


-- to test:
echo "This is a test" > /tmp/test/f1.dat
echo "This is a test too" > /tmp/test/f2.dat
echo "Yes another test" > /tmp/test/f3.dat

select * from received_files order by rdate desc;



/* Managing File Watchers */
-- enable/disable
EXEC DBMS_SCHEDULER.enable('data_fw') ;
EXEC DBMS_SCHEDULER.disable('data_fw') ;


-- change an attribute:
begin
 DBMS_SCHEDULER.SET_ATTRIBUTE (
  name => 'data_fw',
  attribute =>'directory_path',
  value =>'/home/oracle/receivedfiles' ) ;
end;
/

begin
 DBMS_SCHEDULER.SET_ATTRIBUTE (
  name => 'data_fw',
  attribute =>'file_name',
  value =>'*.txt' ) ;
end;
/

begin
 DBMS_SCHEDULER.SET_ATTRIBUTE (
  name => 'data_fw',
  attribute =>'credential_name',
  value =>'fw_credential2' ) ;
end;
/


-- to drop a file watchers:
DBMS_SCHEDULER.DROP_FILE_WATCHER('data_fw');
```

Finer-grained Dependency Management

Invalidation of dependent schema objects in response to changes in the objects they depend upon is greatly reduced in Oracle Database 11g.

If a single-table view selects only a subset of columns in a table, only those columns are involved in the dependency. For each dependent of an object, if a change is made to the definition of any element involved in the dependency (including dropping the element), the dependent object is invalidated. Conversely, if changes are made only to definitions of elements that are not involved in the dependency, the dependent object remains valid.

However, for triggers, they will be invalidated for a redefined table, even if it is not affected by the table modification. For materialized views that depend on the modified table, you should perform a complete refresh on them.

Enhancements in Oracle Database Resource Manager

Per-Session I/O Limits

In Oracle 11g, you can configure the Database Resource Manager so that sessions that exceed I/O resource consumption limits can be automatically switched to another consumer group.

The following example creates a resource plan directive for the OLTP group that temporarily switches any session in that group to the LOW_GROUP consumer group, if the session exceeds 10,000 I/O requests or exceeds 2,500 Megabytes of data transferred.

```
begin
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
 PLAN => 'DAYTIME',
 GROUP_OR_SUBPLAN => 'OLTP',
 COMMENT => 'Auto Conditional Switch from OLTP group to Low_Group',
 MGMT_P1 => 75, -- resource allocation value for level 1 (replaces CPU_P1)
 SWITCH_GROUP => 'LOW_GROUP',
 SWITCH_IO_REQS => 10000, SWITCH_IO_MEGABYTES => 2500,
 SWITCH_FOR_CALL => TRUE);
end;
```

I/O Calibration

The DBMS_RESOURCE_MANAGER package has a new procedure named as CALIBRATE_IO which assess the I/O performance of the database server storage system by performing an I/O intensive read-only workload. This should only be run during off-peak times when there is no workload on the database.

To successfully run the procedure, it requires the I/O asynchronous disabled; otherwise ORA-56708 is raised. Asynchronous I/O is enabled by setting the FILESYSTEMIO_OPTIONS parameter to ASYNC or SETALL.

```
show parameter FILESYSTEMIO_OPTIONS
alter system set FILESYSTEMIO_OPTIONS=SETALL SCOPE=SPFILE;
shutdown immediate
startup
```

The CALIBRATE_IO procedure accepts two parameters to specify the number of physical disks (default 1) and the maximum tolerable latency (default 20ms). On completion, it returns the maximum I/O requests per second, the maximum MB per second and the actual latency. To calculate the latency time, the procedure requires TIMED_STATISTICS parameter is set to TRUE.

```
SET SERVEROUTPUT ON
declare
  l_max_iops  PLS_INTEGER;
  l_max_mbps  PLS_INTEGER;
  l_actual_latency  PLS_INTEGER;
begin
  DBMS_RESOURCE_MANAGER.calibrate_io ( num_physical_disks => 1,
    max_latency        => 20,
    max_iops           => l_max_iops,
    max_mbps           => l_max_mbps,
    actual_latency     => l_actual_latency);
  DBMS_OUTPUT.put_line ('Max IO ps       = ' || l_max_iops);
  DBMS_OUTPUT.put_line ('Max MB ps       = ' || l_max_mbps);
  DBMS_OUTPUT.put_line ('Actual Latency = ' || l_actual_latency);
end;
```

Information about calibration runs can be obtained from the DBA_RSRC_IO_CALIBRATE view:

```
select * from DBA_RSRC_IO_CALIBRATE;
```

Out-Of-The-Box Mixed Workload Resource Plan

Oracle Database includes a predefined resource plan, MIXED_WORKLOAD_PLAN, that prioritizes interactive operations over batch operations, and includes the required subplans and consumer groups recommended by Oracle. You can use this predefined plan, if it is appropriate for your environment.

The plan is defined as follows:

	CPU Resource Allocation				
Group or Subplan	**Level 1**	**Level 2**	**Level 3**	**Automatic Consumer Group Switching**	**Max Degree of Parallelism**
BATCH_GROUP			100%		
INTERACTIVE_GROUP		85%		switch to group: BATCH_GROUP switch time: 60 seconds switch for call: TRUE	1
ORA$AUTOTASK_SUB_PLAN		5%			
ORA$DIAGNOSTICS		5%			
OTHER_GROUPS		5%			
SYS_GROUP	100%				

In this plan, because INTERACTIVE_GROUP is intended for short transactions, any call that lasts longer than 60 seconds is automatically switched to BATCH_GROUP, which is intended for longer batch operations.

New Resource Manager Performance views

Following are the new dynamic performance views in Oracle 11g to monitor the results of the Resource Manager settings:

V$RSRC_PLAN	displays the names of all currently active resource plans.
V$RSRC_CONSUMER_GROUP	displays data related to currently active resource consumer groups such as the cumulative amount of CPU time consumed, cumulative amount of time waiting for CPU, and cumulative number of CPU waits by all sessions in each consumer group.
V$RSRC_SESSION_INFO	displays Resource Manager statistics per session.
V$RSRC_PLAN_HISTORY	displays a history of when a resource plan was enabled, disabled, or modified on the instance.
V$RSRC_CONS_GROUP_HISTORY	displays a history of consumer group statistics for each entry in V$RSRC_PLAN_HISTORY that has a non-NULL plan.

Maximum CPU Utilization Limit

Use the MAX_UTILIZATION_LIMIT attribute to impose an absolute upper limit on CPU utilization for a resource consumer group. This absolute limit overrides any redistribution of CPU within a plan.

You can also use the MAX_UTILIZATION_LIMIT attribute as the sole means of limiting CPU utilization for consumer groups, without specifying level limits.

```
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
 PLAN => 'APPS_PLAN',
 GROUP_OR_SUBPLAN => 'APP1_GROUP',
 COMMENT => 'Limit CPU for application #1 to 40%',
 MAX_UTILIZATION_LIMIT => 40);
```

Enhanced TRUNCATE Statement

DROP ALL STORAGE is a new option added to TRUNCATE statement in Oracle 11g R2. It drops the whole segment. In addition to the TRUNCATE TABLE statement, DROP ALL STORAGE also applies to the ALTER TABLE TRUNCATE (SUB)PARTITION statement.

```
TRUNCATE TABLE emp DROP ALL STORAGE;
```

Dropping Unused Object Storage

In Oracle Database 11g release 2 (11.2.0.2), the DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS procedure enables you to drop segments for empty tables and partitions that have been migrated from previous releases.

```
-- to drop empty segments from every table in the database:
BEGIN
 DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS();
END;

-- to drop empty segments from the HR.EMPLOYEES table
--  including dependent objects
BEGIN
 DBMS_SPACE_ADMIN.DROP_EMPTY_SEGMENTS(
  schema_name => 'HR',
  table_name => 'EMPLOYEES');
END;
```

Performance Tuning

PL/SQL Native Compilation

PL/SQL Native Compilation was introduced in Oracle 9i. In Oracle 11g, The PL/SQL compiler may now generate processor-specific native code directly from the PL/SQL source code without needing to use a third-party C compiler and, as in Oracle 10g, stores the code in the database catalog. When a unit is needed, the Oracle executable loads the code directly from the catalog into memory, without first staging the unit through a .DLL or .so file. This leads to the following advantages:

- No third-party C compiler.
- Faster in compilation (Oracle claims by two-and-a-half times).
- Faster in execution (by 20 times).

Note Note that this feature may not be available on all platforms in the first release of Oracle 11g.

Configuring the Database or Session to Automatic PL/SQL Native Compilation

The new dynamic parameter `PLSQL_CODE_TYPE` is used to make any newly compiled PL/SQL program unit use native PL/SQL compilation. The parameter takes on of two values:

`INTERPRETED`	PL/SQL program units will be compiled into PL/SQL bytecode format and the PL/SQL interpreter engine will execute them.
`COMPILED`	PL/SQL program units will be compiled to machine code and executed natively.

The parameter can be modified in the system or session level. It will affect the newly compiled program units.

Using PL/SQL Native Compilation in the Program Unit Level

You can recompile a program unit in the native mode using the `ALTER .. COMPILE` command as illustrated in the following example:

```
select PLSQL_CODE_TYPE from ALL_PLSQL_OBJECT_SETTINGS where name='MY_PROC';
ALTER PROCEDURE MY_PROC COMPILE PLSQL_CODE_TYPE=NATIVE;
```

Recompiling Database PL/SQL Program Units Using PL/SQL Native Compilation

To recompile all program units in a database using the native PL/SQL compilation and make the native compilation as the default, perform the following steps:

1. Set the parameter `PLSQL_CODE_TYPE` as follows:

   ```
   ALTER SYSTEM SET PLSQL_CODE_TYPE=NATIVE SCOPE=SPFILE;
   ```

2. Make sure the `PLSQL_OPTIMIZER_LEVEL` parameter value is at least 2.

   ```
   SHOW PARAMETER PLSQL_OPTIMIZER_LEVEL
   ALTER SYSTEM SET PLSQL_OPTIMIZER_LEVEL=2 SCOPE=SPFILE;
   ```

3. Cleanly shutdown the database
4. Start the database in upgrade mode.
5. Execute the following script will recompile all existing PL/SQL units using native compilation:

   ```
   @ORACLE_HOME/rdbms/admin/dbmsupgnv.sql
   ```

6. Restart the database.

Note If you want to recompile the PL/SQL program units back using the interpreted compilation, perform the same steps above except setting the `PLSQL_CODE_TYPE` parameter to `INTERPRETED` and replacing the scrip `dbmsupgnv.sql` with `dbmsupin.sql`.

Server Result Cache

In Oracle 11g, there is a new SGA component called *result cache*, which is used cache SQL query and PL/SQL function results. The database serves the results for the executed SQL queries and PL/SQL functions from the cache instead of re-executing the actual query. Of course, the target is to obtain high response time. The cached results stored become invalid when data in the dependent database objects is modified.

As clear from its concept, result cache is mostly useful in for frequently executed queries with rare changes on the retrieved data.

Result Cache Restrictions

Following are some restrictions with regards to the SQL result cache:

- Queries against data dictionary objects and temporary tables are not supported.
- Queries that use the following SQL functions: `CURRENT_DATE`, `CURRENT_TIMESTAMP`, `LOCAL_TIMESTAMP`, `USERENV/SYS_CONTEXT`, `SYS_GUID`, `SYSDATE` and `SYS_TIMESTAMP` are not supported.
- Queries with bind variables can reuse a cached result only for identical variable values.
- Results of the queries retrieving non current version of data are not cached in the result cache.
- Results of the flashback queries are not cached.

Restrictions on PL/SQL Function Result Cache include:

- The function cannot be defined in a module using invoker's rights.
- The function cannot be used in an anonymous block.
- The function cannot have any OUT or IN OUT parameters.
- The function cannot have IN parameters that are BLOB, CLOB, NCLOB, REF CURSOR, collections, objects, or records.
- The function cannot return a BLOB, CLOB, NCLOB, REF CURSOR, OBJECTS, or records. It can return a collection as long as the collection does not contain one of these types.

Configuring Result Cache

You can enable and disable result cache in the database server using the parameter `RESULT_CACHE_MAX_SIZE`. This parameter specifies the maximum amount of SGA memory (in bytes) that can be used by the Result Cache. If the value of this parameter is 0, then the feature is disabled.

Memory allocated for the result cache is taken from the shared pool. The default value of `RESULT_CACHE_MAX_SIZE` parameter is derived from values of other parameters and as shown in the following table:

Parameter	Default Percentage of Shared Pool to Result Cache
`MEMORY_TARGET`	0.25%
`SGA_TARGET`	0.5%
`SHARED_POOL_SIZE`	1%

Of course, you can increase value of the `RESULT_CACHE_MAX_SIZE` parameter but in all cases Oracle does not allocate more than 75 percent of shared pool memory to result cache.

```
ALTER SYSTEM SET RESULT_CACHE_MAX_SIZE =128M;
```

Note `RESULT_CACHE_MAX_SIZE` cannot be dynamically changed if its value is set to 0 during database startup.

Controlling Result Cache Behavior

The `RESULT_CACHE_MODE` initialization parameter determines the SQL query result cache mode. The parameter specifies when a `ResultCache` operator is spliced into a query's execution plan. The parameter accepts the following values:

`MANUAL` The `ResultCache` operator is added, only if you use the `RESULT_CACHE` hint in the SQL query.

`FORCE` The `ResultCache` operator is added to the root of all `SELECT` statements, if that is possible. However, if the statement contains a `NO_RESULT_CACHE` hint, then the hint takes precedence over the parameter setting.

The parameter can be modified in the system or session level.

```
ALTER SYSTEM SET RESULT_CACHE_MODE =FORCE;
ALTER SESSION SET RESULT_CACHE_MODE =FORCE;
```

Following is an example of using the RESULT_CACHE hint.

```
SELECT /*+ result_cache */
AVG(SALARY), E.DEPARTMENT_ID
FROM HR.EMPLOYEES E, HR.DEPARTMENTS D
WHERE E.DEPARTMENT_ID = D.DEPARTMENT_ID GROUP BY E.DEPARTMENT_ID;
```

The parameter RESULT_CACHE_MAX_RESULT specifies the percentage of RESULT_CACHE_MAX_SIZE that any single result can use. Its default value is five.

```
ALTER SYSTEM SET RESULT_CACHE_MAX_RESULT =25;
```

The parameter RESULT_CACHE_REMOTE_EXPIRATION specifies the number of minutes that a result using a remote object is allowed to remain valid. Setting this parameter to 0 (the default) implies that results using remote objects should not be cached. Setting this parameter to a non-zero value can produce stale answers.

PL/SQL Function Result Cache

When PL/SQL function result cache is enabled, Oracle will check the result cache to see if a previous call to the function exists (using the same parameter passed values) and if so it will return the cached result instead of executing the function.

A PL/SQL function can take advantage of the PL/SQL Function Result Cache by adding the RESULT_CACHE clause to the function definition. In the optional RELIES_ON clause, specify any tables or views on which the function results depend. Following is an example:

```
CREATE OR REPLACE FUNCTION get_name (id NUMBER) RETURN VARCHAR2
 RESULT_CACHE RELIES_ON(emp) IS ...

-- Package specification
CREATE OR REPLACE PACKAGE department_pks IS
 -- Function declaration
 FUNCTION get_dept_info (dept_id NUMBER) RETURN dept_info_record RESULT_CACHE;
END department_pks;

CREATE OR REPLACE PACKAGE BODY department_pks AS
 -- Function definition
 FUNCTION get_dept_info (dept_id NUMBER) RETURN dept_info_record
  RESULT_CACHE RELIES_ON (EMPLOYEES)
 IS
 BEGIN ...
```

If there is a dependent table in a function and that table was modified, then the result cache will be invalidated (cache miss). In this case, the function will be re-executed when called. The same will also occur if you re-compile the function.

Caution When a session reads from a PL/SQL function result cache, the function body is not executed. This means, if the function includes any IO or auditing code, this code will not actually be executed.

PL/SQL Cached functions works across sessions. This means, if the function is cached by a session, its result cache will also be used when executing the same function and arguments by other sessions.

If you need to apply a hot patch PL/SQL code in a running system to a PL/SQL module on which a result cached function directly or transitively depends, then the cached results associated with the result cache function are not automatically flushed. In this case, the following steps must be undertaken:

1. Place the result cache in bypass mode, and flush existing result. When bypass mode is turned on, it implies that cached results are no longer used and that no new results are saved in the cache.

```
begin
 DBMS_RESULT_CACHE.BYPASS(TRUE);
 DBMS_RESULT_CACHE.FLUSH;
end;
```

2. Apply the PL/SQL code patches.
3. Resume use of the result cache, by turning off the cache bypass mode.

```
exec DBMS_RESULT_CACHE.BYPASS(FALSE);
```

Monitoring Result Cache

If you display the explain plan for any query with a RESULT_CACHE hint, you will notice the ResultCache operator. You can use the CACHE_ID value provided in the explain plan to find details about the cached query results using the V$RESULT_CACHE_OBJECTS view, as shown in the following example:

```
select NAME, STATUS, ROW_COUNT, BLOCK_COUNT, NAMESPACE,
to_char(CREATION_TIMESTAMP,'HH12:MI AM') CREATE_TIME
from   V$RESULT_CACHE_OBJECTS
where  CACHE_ID='ctpgzz1qb222tfqw61j203h01b';
```

Following are the possible values for the STATUS column and their descriptions:

NEW	Result is still under construction
PUBLISHED	Result is available for use
BYPASS	Result will be bypassed from use
EXPIRED	Result has exceeded expiration time
INVALID	Result is no longer available for use

V$RESULT_CACHE_STATISTICS view provides information and statistics on cache settings and memory usage.

```
select ID, NAME, VALUE from V$RESULT_CACHE_STATISTICS
```

The NAME column possible values are as follows:

Block Size	(Bytes) size of each memory block
Block Count	Maximum number of memory blocks allowed
Block Count	Current Number of memory blocks currently allocated
Result Size	Maximum (Blocks) Maximum number of blocks allowed for a single result
Create Count	Success Number of cache results successfully created
Create Count	Failure Number of cache results that failed to create
Find Count	Number of cached results that were successfully found
Invalidation	Count Total number of invalidations
Delete Count	Invalid Number of invalid cached results deleted
Delete Count	Valid Number of valid cached results deleted

The V$RESULT_CACHE_MEMORY view displays all the memory blocks and their status. Of course, number of rows in this view increases as the result cache is enlarged by its usage.

The V$RESULT_CACHE_DEPENDENCY view displays the depends-on relationship between the cached results objects.

Monitoring and Managing Result Cache with DBMS_RESULT_CACHE

Beside the dictionary views, the package DBMS_RESULT_CACHE can also be used to monitor and manage result cache usage. Below are some examples of using it:

```
-- check the status of the Result Cache
--  Note: this is the reliable method to know whether result cache is enabled or not
SQL>select DBMS_RESULT_CACHE.STATUS from dual;

-- display report on result cache memory
SQL>set serveroutput on
SQL>exec DBMS_RESULT_CACHE.MEMORY_REPORT

-- turn bypass mode on and off
SQL>exec DBMS_RESULT_CACHE.BYPASS (TRUE);

-- to flush the result cache
SQL>exec DBMS_RESULT_CACHE.FLUSH
```

Client Side Result Cache

In Oracle 11g, you can configure the client process to cache its query results in the client memory (or in the server).

When dealing with the client cache, consider the following:

- To take advantage of client cache, you must use Oracle 11g client libraries and connect to an Oracle 11g database.
- Client cache can be configured in a host that runs any application that uses Oracle Database 11g OCI client like: ODBC, ODB.NET, PHP, JDBC-OCI and Oracle precompilers.
- Client cache is suitable for frequently executed queries to static data.

Server Side Configuration

To configure client cache in the database server, set the following parameters:

CLIENT_RESULT_CACHE_SIZE	This static parameter specifies the maximum size of the client per-process result set cache (in bytes). It represents the combined size of caches in all clients. If it is set to zero (default), the client cache is disabled.
CLIENT_RESULT_CACHE_LAG	This static parameter specifies the maximum time (in milliseconds) since the last round trip to the server, before which the OCI client query execute makes a round trip to get any database changes related to the queries cached on the client.

Client Side Configuration

You can use an optional client configuration file (or make the settings part of the sqlnet.ora file on the client). Client settings will override client cache settings in the server. The following parameters can be set in the client:

OCI_RESULT_CACHE_MAX_SIZE	This parameter specifies the maximum size of the client per-process result set cache (in bytes).
OCI_RESULT_CAHCE_MAX_RSET_SIZE	This parameter specifies the maximum size of a single query result in the client cache for a process (in bytes).
OCI_RESULT_CAHCE_MAX_RSET_ROWS	This parameter specifies the maximum number of rows a single query result in the client cache for a process.

After implementing this configuration, simply the SELECT statement passed by the application to Oracle server should only use the hint /*+ result_cache */. As a result, retrieved rows are cached on the client side saving network round trips and the server CPU overhead.

To see the clients that used client cache, issue the following queries:

```
select * from CLIENT_RESULT_CACHE_STAT$;
select * from V$CLIENT_RESULT_CACHE_STATS;
```

See *Oracle Call Interface Programmer's Guide* documentation for more information about the client query cache.

Enhanced Oracle Process Monitoring

Monitoring background processes is enhanced in Oracle 11g so that more detailed statistics can be obtained about their operations. The following columns are added in the V$SYSTEM_EVENT view:

TOTAL_WAITS_FG	Total number of waits that a foreground/background session waited for the event.
TOTAL_TIMEOUTS_FG	Total number of timeouts for an event that a foreground/background session waited.
TIME_WAITED_FG	Amount of time that a foreground/background session waited for the event in centaseconds.
AVERAGE_WAIT_FG	Average wait time that a foreground/background process waited for the event in centaseconds.
TIME_WAITED_MICRO_FG	Total time in microseconds that a foreground session spent waiting on an event.

The following query is to find all cumulative waits of more than one second that occurred on foreground/background processes:

```
SELECT EVENT, TOTAL_WAITS_FG TWG, TOTAL_TIMEOUTS_FG TTF,
TIME_WAITED_FG TWF, AVERAGE_WAIT_FG AWF, TIME_WAITED_MICRO_FG TWMF
FROM V$SYSTEM_EVENT
WHERE TIME_WAITED_MICRO_FG > 1000000
AND WAIT_CLASS !='Idle';
```

The AWR-related view DBA_HIST_SYSTEM_EVENT provides the following similar new columns:

TOTAL_WAITS_FG	Total number of waits for an event, from a foreground session.
TOTAL_TIMEOUTS_FG	Total number of timeouts for an event, from a foreground session.
TIME_WAITED_MICRO_FG	Total time in microseconds that a foreground session spent waiting on an event.

Subprogram Inlining

Subprogram inlining, or Intra Unit Inlining, is a new feature in Oracle 11g that when used appropriately which result in higher PL/SQL execution performance. Details about this feature are discussed in Subprogram Inlining section.

SQL Tuning Automation

The SQL Tuning Advisor is run by default every night during the automated maintenance window. Basically, the advisor catches the SQL statements from AWR that are candidate for tuning (they are called *buckets*) during four different time periods. It then automatically creates SQL profile for any poor SQL statement, if that helps. Tuned plans are automatically added to the SQL plan baselines by the automatic SQL tuning task.

If you want the Automatic SQL Tuning process to stop accepting and implementing the recommended SQL profiles automatically, issue the following code:

```
exec   DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER(TASK_NAME   =>   'SYS_AUTO_SQL_TUNING_TASK',
PARAMETER=>'ACCEPT_SQL_PROFILES', VALUE=>' FALSE');
```

The advisor also may recommend actions like creating new indexes, refreshing statistics or re-writing the statement. These actions, however, are not automatically implemented by the advisor.

The best method to Manage Automatic SQL Tuning is by using OEM. Follow the links **Home Page> Server page> Oracle Scheduler section> Automatic Maintenance Tasks link> Automatic SQL Tuning link> Task status section> Configure button.**

DBMS_SQLTUNE package is the PL/SQL interface to invoke SQL Tuning Advisor. Examples of using this package can be found in the section "The SQL Performance Analyzer". Further details on using it can be obtained from the documentation "Oracle Database Performance Tuning Guide 11g Release 1 (11.1)" page 17-10.

On Oracle 11g Release 2 (11.2.0.2), a new package named as DBMS_AUTO_SQLTUNE should be used instead of the DBMS_SQLTUNE package. The new package provides more restrictive access to the Automatic SQL Tuning feature.

To use the DBMS_AUTO_SQLTUNE package, you must have the DBA role, or have EXECUTE privileges granted by an administrator. The only exception is the EXECUTE_AUTO_TUNING_TASK procedure, which can only be run by SYS.

```
exec DBMS_AUTO_SQLTUNE.SET_AUTO_TUNING_TASK_PARAMETER( parameter =>
'ACCEPT_SQL_PROFILES', value => 'TRUE');


-- To set the number of days until the task is deleted:
exec dbms_auto_sqltune.set_auto_tuning_task_parameter('DAYS_TO_EXPIRE', 90);
```

To manually execute the automatic execution task :

```
Manually execute the automatic execution task :
-- Only SYS can call this API.
set serveroutput on

-- function format
set serveroutput on
DECLARE
 retVal VARCHAR2(1000);
BEGIN
  retVal := dbms_auto_sqltune.EXECUTE_AUTO_TUNING_TASK;
  dbms_output.put_line(retVal);
END;
/

-- procedure format:
exec dbms_auto_sqltune.EXECUTE_AUTO_TUNING_TASK;
```

To view the automatic SQL tuning report:

```
VARIABLE my_rept CLOB;
BEGIN
:my_rept :=DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK(
 begin_exec => NULL, end_exec => NULL,
 type => 'TEXT',
 level => 'TYPICAL',
```

```
 section => 'ALL',
 object_id => NULL,
 result_limit => NULL);
END;
/
PRINT :my_rept
```

Note There are some typing errors in the documentation about the function DBMS_AUTO_SQLTUNE.REPORT_AUTO_TUNING_TASK. The true possible values for the parameter SECTION are: SUMMARY, FINDINGS, PLANS, INFORMATION, ERRORS and ALL (default).

SQL Access Advisor Enhancements

SQL Access Advisor is enhanced in Oracle 11g with the following improvements:

- The advisor can now recommend creating partitions on tables, provided that the statement has some predicates and joins on the columns of the NUMBER or DATE data types and the tables have at least 10,000 rows..
- Expected gain can be estimated by the advisor.
- A task can be broken up into parts called *publish points* or *intermediate results*. Those intermediate results can be checked while the advisor is executing.

To access the tool in OEM, follow the links **Database Home page> Advisor Central> SQL Advisors> SQL Access Advisor**

The following examples show three methods for invoking the tool in PL/SQL using DBMS_ADVISOR package:

1. Creating a task linked to a workload. When creating a workload, its input could be SQL cache or a STS.
2. Creating a task linked to a STS.
3. Quick tune for a single SQL statement.

```
-- Method 1: to invoke SQL Access Advisor task linked to a workload
declare
  l_taskname     VARCHAR2(30)   := 'sql_access_test_task';
  l_task_desc    VARCHAR2(128)  := 'Test SQL Access';
  l_wkld_name    VARCHAR2(30)   := 'test_work_load';
  l_saved_rows   NUMBER         := 0;
  l_failed_rows  NUMBER         := 0;
  l_num_found    NUMBER;
BEGIN
  -- create an SQL Access Advisor task.
  select COUNT(*)
  into   l_num_found
  from DBA_ADVISOR_TASKS
  where TASK_NAME =  l_taskname ;
  IF l_num_found = 0 THEN
   DBMS_ADVISOR.CREATE_TASK (
    ADVISOR_NAME => DBMS_ADVISOR.SQLACCESS_ADVISOR,
    TASK_NAME    => l_taskname,
    TASK_DESC    => l_task_desc);
  END IF;

  -- reset the task ( remove all recommendations, and intermediate data from the task)
  DBMS_ADVISOR.RESET_TASK(TASK_NAME => l_taskname);

  -- create a workload.
  SELECT COUNT(*)
  INTO   l_num_found
  FROM   USER_ADVISOR_SQLW_SUM
  WHERE  WORKLOAD_NAME = l_wkld_name;

  IF l_num_found = 0 THEN
    DBMS_ADVISOR.CREATE_SQLWKLD(WORKLOAD_NAME => l_wkld_name);
  END IF;

  -- link the workload to the task
  SELECT count(*)
```

```
  INTO   l_num_found
  FROM   USER_ADVISOR_SQLA_WK_MAP
  WHERE  TASK_NAME     = l_taskname
    AND  WORKLOAD_NAME = l_wkld_name;

  IF l_num_found = 0 THEN
    DBMS_ADVISOR.ADD_SQLWKLD_REF(
      TASK_NAME     => l_taskname,
      WORKLOAD_NAME => l_wkld_name);
  END IF;

  -- Set workload parameters.
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(l_wkld_name, 'ACTION_LIST',
DBMS_ADVISOR.ADVISOR_UNUSED);
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(l_wkld_name, 'MODULE_LIST',
DBMS_ADVISOR.ADVISOR_UNUSED);
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(l_wkld_name, 'SQL_LIMIT',
DBMS_ADVISOR.ADVISOR_UNLIMITED);
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(l_wkld_name, 'ORDER_LIST',
'PRIORITY,OPTIMIZER_COST');
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(l_wkld_name, 'USERNAME_LIST',
DBMS_ADVISOR.ADVISOR_UNUSED);
  DBMS_ADVISOR.SET_SQLWKLD_PARAMETER(l_wkld_name, 'VALID_TABLE_LIST',
DBMS_ADVISOR.ADVISOR_UNUSED);


 -- unmark the required option
 /*
  -- import the current contents of the server's SQL cache
  DBMS_ADVISOR.IMPORT_SQLWKLD_SQLCACHE(l_wkld_name, 'REPLACE', 2, l_saved_rows,
l_failed_rows);

  -- load a SQL workload from an existing SQL Tuning Set
  DBMS_ADVISOR.IMPORT_SQLWKLD_STS (l_wkld_name, 'SOURCE_STS_Name', 'REPLACE',2,
l_saved_rows, l_failed_rows);
*/
  -- Set task parameters.
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, '_MARK_IMPLEMENTATION', 'FALSE');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'EXECUTION_TYPE', 'INDEX_ONLY');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'MODE', 'COMPREHENSIVE');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'STORAGE_CHANGE',
DBMS_ADVISOR.ADVISOR_UNLIMITED);
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DML_VOLATILITY', 'TRUE');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'ORDER_LIST', 'PRIORITY,OPTIMIZER_COST');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'WORKLOAD_SCOPE', 'PARTIAL');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DEF_INDEX_TABLESPACE',
DBMS_ADVISOR.ADVISOR_UNUSED);
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DEF_INDEX_OWNER',
DBMS_ADVISOR.ADVISOR_UNUSED);
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DEF_MVIEW_TABLESPACE',
DBMS_ADVISOR.ADVISOR_UNUSED);
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DEF_MVIEW_OWNER',
DBMS_ADVISOR.ADVISOR_UNUSED);

  -- Execute the task: control will not return till the execution finishes
  DBMS_ADVISOR.execute_task(task_name => l_taskname);
END;
/


-- Method 2: to invoke SQL Access Advisor linked to a specific STS
declare
  l_taskname     VARCHAR2(30)   := 'sql_access_test_task2';
  l_task_desc    VARCHAR2(128)  := 'Test SQL Access for a STS';
  l_wkld_name    VARCHAR2(30)   := 'test_work_load';
  l_sts_name     VARCHAR2(30)   := 'test_sts';
  l_saved_rows   NUMBER         := 0;
  l_failed_rows  NUMBER         := 0;
  l_num_found    NUMBER;
BEGIN
  -- create an SQL Access Advisor task, if it doesn’t exist
  select COUNT(*)
  into   l_num_found
```

```
  from DBA_ADVISOR_TASKS
  where TASK_NAME =  l_taskname ;
 IF l_num_found = 0 THEN
 DBMS_ADVISOR.CREATE_TASK (
    ADVISOR_NAME => DBMS_ADVISOR.SQLACCESS_ADVISOR,
    TASK_NAME    => l_taskname,
    TASK_DESC    => l_task_desc);
  END IF;

  -- reset the task ( remove all recommendations, and intermediate data from the task)
  DBMS_ADVISOR.RESET_TASK(TASK_NAME => l_taskname);

  -- check if STS already exists
  select count(*)
  into   l_num_found
  from   DBA_SQLSET where upper(NAME) = upper(l_sts_name) ;

  IF l_num_found <> 0 THEN
    DBMS_SQLTUNE.DROP_SQLSET ( sqlset_name => l_sts_name);
  END IF;

  -- create STS
  DBMS_SQLTUNE.CREATE_SQLSET(SQLSET_NAME => l_sts_name, DESCRIPTION =>'To test Access
Advisor');

 /* unmark the required option
 -- (Option 1) Load l_sts_name from an AWR baseline.
 -- The data has been filtered to select only the top 30 SQL statements ordered by
elapsed time.
 declare
  baseline_cur DBMS_SQLTUNE.SQLSET_CURSOR;
 begin
  -- a ref cursor is opened to select from the specified baseline
  OPEN baseline_cur FOR
    SELECT VALUE(p)
    FROM TABLE (DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(
   'peak baseline',NULL, NULL, 'elapsed_time', NULL, NULL, NULL,30 )) p;

   -- Next the statements and their statistics are loaded into the STS
   DBMS_SQLTUNE.LOAD_SQLSET( SQLSET_NAME=>l_sts_name, POPULATE_CURSOR=>baseline_cur);
 end;

 -- (Option 2) Load l_sts_name with SQL statements that are not owned by SYS and
 --  their elapsed time is greater than 20,000 seconds.
 declare
   sql_cur DBMS_SQLTUNE.SQLSET_CURSOR;
 begin
   -- a ref cursor is opened to select the required SQL statments
   OPEN sql_cur FOR
     SELECT VALUE(p)
     FROM TABLE (DBMS_SQLTUNE.SELECT_CURSOR_CACHE('parsing_schema_name <> ''SYS'' and
elapsed_time > 2000000',NULL, NULL, NULL, NULL,1, NULL, 'ALL')) p;
     -- the statements are loaded into the STS
     DBMS_SQLTUNE.LOAD_SQLSET( SQLSET_NAME=>l_sts_name, POPULATE_CURSOR=>sql_cur);
 end;

 -- (Option 3) Copy the contents of a SQL workload object to a SQL Tuning Set
 -- check the example above for creating a workload
  DBMS_ADVISOR.COPY_SQLWKLD_TO_STS ('My_WorkLoad', l_sts_name, 'REPLACE');
 */

  -- link the STS to the task
  DBMS_ADVISOR.ADD_STS_REF (l_taskname, null, l_sts_name);

  -- Set task parameters.
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, '_MARK_IMPLEMENTATION', 'FALSE');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'EXECUTION_TYPE', 'INDEX_ONLY');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'MODE', 'COMPREHENSIVE');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'STORAGE_CHANGE',
DBMS_ADVISOR.ADVISOR_UNLIMITED);
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DML_VOLATILITY', 'TRUE');
```

```
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'ORDER_LIST', 'PRIORITY,OPTIMIZER_COST');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'WORKLOAD_SCOPE', 'PARTIAL');
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DEF_INDEX_TABLESPACE',
DBMS_ADVISOR.ADVISOR_UNUSED);
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DEF_INDEX_OWNER',
DBMS_ADVISOR.ADVISOR_UNUSED);
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DEF_MVIEW_TABLESPACE',
DBMS_ADVISOR.ADVISOR_UNUSED);
  DBMS_ADVISOR.SET_TASK_PARAMETER(l_taskname, 'DEF_MVIEW_OWNER',
DBMS_ADVISOR.ADVISOR_UNUSED);

  -- Execute the task: control will not return till the execution finishes
  DBMS_ADVISOR.EXECUTE_TASK(task_name => l_taskname);
END;
/


-- Method 3: Quick Tune for a single given statement
begin
  -- a task and a workload will be created then the task will be executed
  DBMS_ADVISOR.QUICK_TUNE(
    ADVISOR_NAME => DBMS_ADVISOR.SQLACCESS_ADVISOR,
    TASK_NAME    => 'names_quick_tune',
    ATTR1        => 'SELECT id FROM hr.names n WHERE id  = 100');
end;
/
```

Following are examples of how to manage the task and obtain information about the advisor's output repot.

```
-- while the task is being executed, you can monitor it using the following query:
select TASK_NAME, STATUS, PCT_COMPLETION_TIME, ERROR_MESSAGE
from   DBA_ADVISOR_LOG
where  TASK_NAME ='sql_access_test_task';

-- if you need to terminate the executing task (may be time consuming)
exec DBMS_ADVISOR.CANCEL_TASK(TASK_NAME =>'sql_access_test_task');

-- Display the resulting script ( method 1)
SET LONG 100000
SET PAGESIZE 50000
SELECT DBMS_ADVISOR.GET_TASK_SCRIPT('sql_access_test_task') AS script FROM dual;
SET PAGESIZE 24

-- Display the resulting script ( method 2)
CREATE DIRECTORY ADVISOR_RESULTS AS 'C:\TEMP\';
exec DBMS_ADVISOR.CREATE_FILE(DBMS_ADVISOR.GET_TASK_SCRIPT('sql_access_test_task'),
'ADVISOR_RESULTS','advscript.sql');

-- alternatively, use the following queries
--  benefit is the total improvement in execution cost of all the queries
select REC_ID, RANK, BENEFIT, TYPE "Recommendation Type"
from   DBA_ADVISOR_RECOMMENDATIONS
where  TASK_NAME = 'sql_access_test_task'
order by RANK;

-- display processed statements in the workload
select SQL_ID, REC_ID, PRECOST, POSTCOST,
       (PRECOST-POSTCOST)*100/PRECOST AS PERCENT_BENEFIT
from   USER_ADVISOR_SQLA_WK_STMTS
where  TASK_NAME = 'sql_access_test_task' AND workload_name = 'test_work_load';

-- see the actions for each recommendations
select REC_ID, ACTION_ID, SUBSTR(COMMAND,1,30) AS COMMAND
from   USER_ADVISOR_ACTIONS where TASK_NAME = 'sql_access_test_task'
ORDER BY rec_id, action_id;

-- to delete a given task
exec DBMS_ADVISOR.DELETE_TASK('sql_access_test_task');
```

Changing Statistics Preferences

The function DBMS_STATS.GET_PARAM is used to in Oracle 10g to return the default values of parameters of the DBMS_STATS package. This function is now obsolete in Oracle 11g and replaced with GET_PREFS procedure. Following is an example:

```
SET SERVEROUTPUT ON
declare
 v_value varchar2(100);
begin
   v_value := DBMS_STATS.GET_PREFS (
     PNAME =>'STALE_PERCENT',
     OWNNAME =>'HR',
     TABNAME =>'EMPLOYEES');
 DBMS_OUPTPUT.PUT_LINE(v_value);
end;
```

Regarding the GET_PREFS function, consider the following:

- PNAME parameter indicates the preference name and can take one of the following values: CASCADE, DEGREE, ESTIMATE_PERCENT, METHOD_OPT, NO_INVALIDATE, GRANULARITY, PUBLISH, INCREMENTAL and STALE_PERCENT.
- If the OWNNAME and TABNAME are provided and a preference has been entered for the table, the function returns the preference as specified for the table. In all other cases it returns the global preference if it has been specified, otherwise the default value is returned.

SET_GLOBAL_PREFS, SET_DATABASE_PREFS, SET_SCHEMA_PREFS, SET_TABLE_PREFS procedures are used to set the statistics preferences for the global, database, schema or table levels respectively. Following is an example:

```
begin
 DBMS_STATS.SET_GLOBAL_PREFS ( PNAME =>'ESTIMATE_PERCENT', PVALUE =>'75');
end;
```

Similarly, the procedures DELETE_*_PREFS are used to delete current statistics preferences. EXPORT_*_PREFS and IMPORT_*_PREFS procedures are used to export and import statistics preferences. Following is an example:

```
begin
 DBMS_STATS.EXPORT_DATABASE_PREFS(
  STATTAB =>'mytable',    -- table name to where statistics should be exported
  STATID  =>'prod_prefs', -- identifier to associate with these statistics
  STATOWN =>'HR');        -- Schema containing stattab (if other than ownname)
end;
```

Enhanced Statistics Maintenance

Pending and Published Statistics

Starting with Oracle 11g, when gathering statistics, you have the option to automatically publish the statistics at the end of the gather operation (default behavior), or to have the new statistics saved as pending. Saving the new statistics as pending allows you to validate the new statistics and publish them only if they are satisfactory.

You can check whether or not the statistics will be automatically published checking the value of the PUBLISH attribute using the DBMS_STATS package as in the following example:

```
SELECT DBMS_STATS.GET_PREFS('PUBLISH') PUBLISH FROM DUAL;
```

You can change the PUBLISH setting at either the schema or table level. Following are examples to do so:

```
-- setting PUBLISH at schema level
exec DBMS_STATS.SET_SCHEMA_PREFS('HR', 'PUBLISH', 'FALSE');

-- setting PUBLISH at table level
exec DBMS_STATS.SET_TABLE_PREFS('HR','EMPLOYEES', 'PUBLISH', 'FALSE');
```

Published statistics are stored in data dictionary views, such as DBA_TAB_STATISTICS and USER_IND_STATISTICS. Pending statistics are stored in views such as DBA_TAB_PENDING_STATISTICS and DBA_IND_PENDING_STATISTICS.

```
select NUM_ROWS, BLOCKS, AVG_ROW_LEN, SAMPLE_SIZE, LAST_ANALYZED
from   DBA_TAB_PENDING_STATISTICS where  OWNER='HR' AND TABLE_NAME ='EMPLOYEES';
```

By default, the optimizer uses the published statistics stored in the data dictionary views. If you want the optimizer to use the newly collected pending statistics, set the initialization parameter OPTIMIZER_PENDING_STATISTICS to TRUE (the default value is FALSE), and then run a workload against the table or schema or just gather its statistics:

```
ALTER SESSION SET OPTIMIZER_PENDING_STATISTICS = TRUE;
```

The optimizer will use the pending statistics (if available) instead of the published statistics when compiling SQL statements. If the pending statistics are valid, they can be made public by executing the following statement:

```
-- for the whole database
exec DBMS_STATS.PUBLISH_PENDING_STATS(null, null);

-- publishing specific database object pending statistics
exec DBMS_STATS.PUBLISH_PENDING_STATS('HR','EMPLOYEES');
```

If you do not want to publish the pending statistics, delete them by executing the following statement:

```
exec DBMS_STATS.DELETE_PENDING_STATS('HR','EMPLOYEES');
```

Restoring Previous Versions of Statistics

With Oracle Database 11g, you can restore previous versions of statistics. DBMS_STATS package has the following procedures to do that: RESTORE_DICTIONARY_STATS, RESTORE_FIXED_OBJECTS_STATS, RESTORE_SCHEMA_STATS, RESTORE_SYSTEM_STATS, and RESTORE_TABLE_STATS.

First, query the view DBA_OPTSTAT_OPERATIONS to know when gathering the statistics has been done using DBMS_STATS. Then decide to which point in time you want to restore the statistics.

```
-- list available versions of statistics
SELECT * FROM DBA_OPTSTAT_OPERATIONS;

-- restore to the point you want
begin
 -- restore statistics of a specific schema
 DBMS_STATS.RESTORE_SCHEMA_STATS(
  OWNNAME => 'HR',
  AS_OF_TIMESTAMP => '19-FEB-08 06.00.08.477333 AM -06:00');

 -- restore statistics of a specific table
 DBMS_STATS.RESTORE_SCHEMA_STATS(
  OWNNAME => 'HR',
  TABNAME => 'EMPLOYEES',
  AS_OF_TIMESTAMP => '19-FEB-08 06.00.08.477333 AM -06:00');
end;
```

Oracle will manage the historical statistics repository, purging the statistics on a regular basis, by default every 31 days. To adjust this retention, consider the following examples:

```
-- get the current retention value
select DBMS_STATS.GET_STATS_HISTORY_RETENTION from dual;

-- get the oldest timestamp where statistics history is available
select DBMS_STATS.GET_STATS_HISTORY_AVAILABILITY from dual;

-- set the retention value
exec DBMS_STATS.ALTER_STATS_HISTORY_RETENTION ( 120 ); -- in days
```

MultiColumn Statistics

When multiple columns from a single table are used together in the where clause of a query (multiple single column predicates), Oracle optimizer in previous versions (before 11g) does not understand the relationship between the columns. In Oracle 11g, statistics on these columns as a group (column group) can be gathered and thus resulting in high enhancement in CBO cardinality estimation.

Note The *MultiColumn Statistics* and *Expression Statistics* new features are both constitute what is called by Oracle as *Extended Statistics*. Expression Statistics is discussed in new following sub-section.

You can also create column groups manually by using the CREATE_EXTENDED_STATS function in the DBMS_STATS package. This function creates a column statistics entry in the system for a user specified column group or an expression in a table. This function returns a system-generated name of this newly created entry for the extension. Following is an example:

```
declare
 V_NAME VARCHAR2(30);
begin
 -- stats of the combined columns will be collected next time the stats is gathered
 V_NAME := DBMS_STATS.CREATE_EXTENDED_STATS(
            OWNNAME   => NULL,
            TABNAME   => 'EMPLOYEES',
            EXTENSION => '(STATE_ID,COUNTRY_ID)');

 -- you can then issue the gathering process
 DBMS_STATS.GATHER_TABLE_STATS  (NULL,  'EMPLOYEES',  METHOD_OPT='for  all  columns  size
skewonly' );
end;
```

The DBMS_STATS.GATHER_TABLE_STATS procedure can also be used to create column group and gather its statistics all in one step. The keyword FOR COLUMNS is used in this case as shown in the following example:

```
begin
 DBMS_STATS.GATHER_TABLE_STATS ('HR', 'EMPLOYEES',
  METHOD_OPT=>'for all columns size skewonly for columns (STATE_ID,COUNTRY_ID)' );
end;
```

Note The default value of METHOD_OPT is 'FOR ALL COLUMNS SIZE AUTO' which makes Oracle create column groups for a table, based on the workload analysis, similar to how it is done for histograms.

You can use the methods in the following code examples to retrieve information on column groups that have been created:

```
-- you can query the data dictionary USER_STAT_EXTENSIONS
select EXTENSION_NAME, EXTENSION from USER_STAT_EXTENSIONS where TABLE_NAME='EMPLOYEES';

-- you can query USER_TAB_COL_STATISTICS (extension name appears as COLUMN_NAME)
select COLUMN_NAME, NUM_DISTINCT, HISTOGRAM
from USER_TAB_COL_STATISTICS where TABLE_NAME = 'EMPLOYEES';

-- you can use DBMS_STATS.SHOW_EXTENDED_STATS_NAME function
select DBMS_STATS.SHOW_EXTENDED_STATS_NAME(OWNNAME   => 'HR',
                                           TABNAME   => 'EMPLOYEES',
                                           EXTENSION => 'STATE_ID,COUNTRY_ID') AS E_NAME
from dual;
```

After gathering the multi-column statistics as show in the example, when you check the explain plan for a query of a where condition like "STATE_ID = 'CA' AND COUNTRY_ID = 'US'", you will notice that the optimizer has retrieved the correct number of expected retrieved rows. Practically, this will lead to a significant improvement in the statement execution.

Following is how to drop a column group that you have previously defined:

```
exec DBMS_STATS.DROP_EXTENDED_STATS('HR', 'EMPLOYEES', '(STATE_ID,COUNTRY_ID)');
```

Expression Statistics

In Oracle 11g, you can create statistics on an expression. Following are examples to do that:

```
declare
 V_NAME VARCHAR2(30);
begin
 -- to create expression extended stats (not statistics are yet gathered)
 V_NAME := DBMS_STATS.CREATE_EXTENDED_STATS(
            OWNNAME   => NULL,
```

```
            TABNAME   => 'EMPLOYEES',
            EXTENSION => '(lower(last_name))');
end;

begin
 -- to create expression extended stats and gather the statistics in one step
 DBMS_STATS.GATHER_TABLE_STATS(OWNNAME=>NULL,TABLE_NAME=>'EMPLOYEES',
 METHOD_OPT=>'for all columns size skewonly for columns (lower(last_name))');
end;
```

After executing the code above, if you issue a query from EMPLOYEES table with a condition like LOWER(LAST_NAME)='ABC', the optimizer has statistics about the retrieved rows and thus will be able to estimate the correct number of returned rows. Consequently, the optimizer will most likely create a more efficient plan than if those statistics were not present.

Use the DBA_STAT_EXTENSIONS data dictionary view to retrieve information on expression statistics that have been created in the database.

```
select EXTENSION_NAME, EXTENSION from USER_STAT_EXTENSIONS where TABLE_NAME='EMPLOYEES';
```

Following is an example of the removal of an extended expression statistic:

```
exec DBMS_STATS.DROP_EXTENDED_STATS(null, 'EMPLOYEES', '(lower(lat_name))' );
```

Note that you will not be able to drop an extended expression statistics, if a function-based index is dependent on that statistic (ORA-20000 error will be returned).

Automatically Collecting Statistics on Tables

In Oracle Database 11g, statistics are collected automatically for tables regardless of the MONITORING and NOMONITORING keywords used when creating them. Those keywords are deprecated and ignored, if used.

SQL Plan Management

SQL plan management (SPM), is a new feature in Oracle 11g that prevents performance regressions resulting from sudden changes to the execution plan of a SQL statement by providing components for capturing, selecting, and evolving SQL plan information. Changes to the execution plan may be resulted from database upgrades, system and data changes, application upgrade or bug fixes.

When SPM is enabled, the system maintains a plan history that contains all plans generated by the optimizer and store them in a component called *plan baseline*. Among the plan history in the plan baseline, plans that are verified not to cause performance regression are marked as acceptable. The plan baseline is used by the optimizer to decide on the best plan to use when compiling a SQL statement.

Repository stored in data dictionary of plan baselines and statement log maintained by the optimizer is called *SQL management base* (SMB).

SQL Plan management is implemented by undertaking the following phases:

1. Capturing SQL Plan Baselines: this can be done automatically or manually.
2. Selecting SQL Plan Baselines by the optimizer
3. Evolving SQL Plan Baselines

Note Stored outlines will be de-supported in a future release in favor of SQL plan management.

Automatic Plan Capture

To enable automatic plan capture, set the OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES initialization parameter to TRUE in the system or session level. Its default value is FALSE.

```
alter system set OPTIMIZER_CAPTURE_SQL_PLAN_BASELINES = TRUE;
```

Manual Plan Loading

SQL plan baselines can be manually loaded with existing plans for a set of SQL statements. The plans are not checked for performance but will be loaded with the ACCEPTED status.

You can load the plans from SQL Tuning Sets, AWR Snapshots, or Cursor Cache. Following are code examples to achieve that:

```
-- to load plans from a SQL Tuning Set (STS)
declare
```

```
 my_plans pls_integer;
begin
 my_plans := DBMS_SPM.LOAD_PLANS_FROM_SQLSET( SQLSET_NAME => 'tset1');
end;

-- to load plans from a AWR baseline
declare
 baseline_cursor DBMS_SQLTUNE.SQLSET_CURSOR;
 my_plans pls_integer;
begin
 -- create STS
 DBMS_SQLTUNE.CREATE_SQLSET(
  SQLSET_NAME => 'Top30_STS',
  DESCRIPTION => 'Top 30 SQL Statements in peak workload');

 -- load STS from AWR
 -- select the top 30 SQL statements ordered by elapsed time
 OPEN baseline_cursor FOR
  SELECT VALUE(p)
   FROM TABLE (DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(
    'peak baseline', -- name of the AWR baseline
    NULL, NULL,
    'elapsed_time',
    NULL, NULL, NULL, 30)) p;
 DBMS_SQLTUNE.LOAD_SQLSET(SQLSET_NAME => 'Top30_STS',
   POPULATE_CURSOR => baseline_cursor);
 my_plans := DBMS_SPM.LOAD_PLANS_FROM_SQLSET( SQLSET_NAME => 'Top30_STS');
end;

-- loading Plans from the Cursor Cache
declare
 my_sql_plan pls_integer;
 v_sql varchar2(1000);
begin
 FOR dd in (select sql_id from v$sqlarea
                where lower(sql_text) like 'select * from scott.emp') LOOP
   IF LENGTH (dd.sql_id) > 0 THEN
     my_sql_plan := DBMS_SPM.LOAD_PLANS_FROM_CURSOR_CACHE(sql_id=>dd.sql_id);
   END IF;
END LOOP;
end;
```

Selecting SQL Plan Baselines

Each time a SQL statement is compiled, the optimizer will first use a cost-based search method to build a best-cost plan, it will compare it to the plans in the plan baseline. Eventually, the optimizer will decide to select the lowest-cost plan in the plan baseline.

To enable the use of SQL plan baselines, the `OPTIMIZER_USE_SQL_PLAN_BASELINES` parameter must be set to TRUE.

```
alter system set OPTIMIZER_USE_SQL_PLAN_BASELINES = TRUE ;
```

Note When you use autotrace, you can tell if a baseline is being used. You will see the following note in the autotrace output:

```
SQL plan baseline "SYS_SQL_PLAN_a3185cea611ea913" used for this statement
```

Evolving SQL Plan Baselines

During the SQL plan baseline evolution phase, the optimizer determines if non-accepted plans in the baseline should be accepted. This can be done by the following three ways:

- When the plan is manually loaded to the SQL plan baseline. In this case, all loaded plans are added as accepted plans.
- When a SQL plan baseline plan is manually evolved.
- Automatic SQL Tuning (SQL Tuning Advisor).

A SQL plan baseline plan can be manually evolved using the function `DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE`. This function tries to evolve new plans added by the optimizer to the plan history of existing plan baselines and if

it verifies that a new plan performs better than a plan chosen from the corresponding plan baseline, the new plan is added as an accepted plan. The following is an example:

```
declare
 report clob;
begin
 report := DBMS_SPM.EVOLVE_SQL_PLAN_BASELINE( SQL_HANDLE => 'SYS_SQL_593bc74fca8e6738');
 DBMS_OUTPUT.PUT_LINE(report);
end;
```

When tuning SQL statements with the SQL Tuning Advisor, if the advisor finds a tuned plan and verifies its performance to be better than a plan chosen from the corresponding SQL plan baseline, it makes a recommendation to accept a SQL profile.

Fixed SQL Plan Baselines

A SQL plan baseline is fixed if it contains at least one enabled plan whose FIXED attribute is set to YES. The optimizer will pick a fixed plan with the least cost even though a non-fixed plan may have an even lower cost. The optimizer will not add new plans to a fixed SQL plan baseline.

```
declare
 n PLS_INTEGER;
begin
 n:= DBMS_SPM.ALTER_SQL_PLAN_BASELINE (
   PLAN_NAME  => 'SYS_SQL_PLAN_bbedc741a57b5fc2',  --  or SQL_HANDLE
   ATTRIBUTE_NAME  =>'fixed',
   ATTRIBUTE_VALUE =>'YES');
 DBMS_OUTPUT.PUT_LINE('Number of Altered Plans:' || n);
end;
```

Displaying SQL Plan Baselines

You can view the plans stored in the SQL plan baseline for a given statement or a given plan.

```
-- the FORMAT paramaeter accepts BASIC, TYPICAL (default) or ALL
select * from table( DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE(
  SQL_HANDLE=>'SYS_SQL_209d10fabbedc741',
  FORMAT=>'basic'));

select * from table( DBMS_XPLAN.DISPLAY_SQL_PLAN_BASELINE(
  PLAN_NAME=>'SYS_SQL_PLAN_bbedc741a57b5fc2',
  FORMAT=>'basic'));

select ORIGIN, SQL_HANDLE, PLAN_NAME, ENABLED, ACCEPTED, FIXED from
DBA_SQL_PLAN_BASELINES where CREATOR='HR';
```

Viewing SPM Configuration Parameters

The current configuration settings for the SQL management can be viewed using the following query:

```
select PARAMETER_NAME, PARAMETER_VALUE from DBA_SQL_MANAGEMENT_CONFIG;
```

Disk Space Usage by SMP

By default, the limit for the SPM is no more than 10% of the size of the SYSAUX tablespace. The allowable range for this limit is between 1% and 50%. To change the percentage limit, use the CONFIGURE procedure of the DBMS_SPM package:

```
exec DBMS_SPM.CONFIGURE('space_budget_percent',30);
```

When the space occupied by SQL management base exceeds the defined space budget limit, a weekly database alert is generated.

Purging Policy

As part of the automated task in the maintenance window, any plan that has not been used for more than 53 weeks are purged. To configure the retention period, use the CONFIGURE procedure of the DBMS_SPM package:

```
exec DBMS_SPM.CONFIGURE('plan_retention_weeks',105); -- between 5 and 523
```

Importing and Exporting SQL Plan Baselines

Oracle Database supports the export and import of SQL plan baselines using its import and export utilities or Oracle Data Pump, as follows:

1. On the original system, create a staging table as follows:

```
exec DBMS_SPM.CREATE_STGTAB_BASELINE(TABLE_NAME => 'stage1');
```

2. Pack the SQL plan baselines you want to export from the SQL management base into the staging table as follows:

```
declare
 my_plans number;
begin
 -- you can also specify the PLAN_NAME or SQL_HANDLE
 my_plans := DBMS_SPM.PACK_STGTAB_BASELINE(
  TABLE_NAME => 'stage1', ENABLED => 'yes', CREATOR => 'HR');
 DBMS_OUTPUT.PUT_LINE('Number of SQL plans packed: ' || my_plans);
end;
```

3. Export the staging table `stage1` into a flat file using the export command or Oracle Data Pump.
4. Transfer the flat file to the target system.
5. Import the staging table `stage1` from the flat file using the import command or Oracle Data Pump.
6. Unpack the SQL plan baselines from the staging table into the SPM on the target system as follows:

```
declare
  v_plans number;
begin
  v_plans := DBMS_SPM.UNPACK_STGTAB_BASELINE( TABLE_NAME => 'stage1', FIXED => 'yes');
  DBMS_OUTPUT.PUT_LINE('Number of SQL Plans Unpacked: ' || v_plans);
end;
```

Dropping SQL plan baselines

You can drop SQL plan baselines with the `DBMS_SPM.DROP_SQL_PLAN_BASELINE` function which returns the number of plans that were removed.

```
declare
 v_plans_dropped PLS_INTEGER;
begin
 -- you can pass PLAN_NAME or SQL_HANDLE or both
 v_plans_dropped:=DBMS_SPM.DROP_SQL_PLAN_BASELINE(
   SQL_HANDLE =>'SYS_SQL_353e8c17a551f70c',
   PLAN_NAME  =>'SYS_SQL_PLAN_a551f70c695cc014');
 DBMS_OUTPUT.PUT_LINE('Number of SQL Plans Dropped: ' || v_plans);
end;
```

ADDM New Features

Database ADDM

Oracle Database 11g has added a new layer of analysis to ADDM called *Database* ADDM. The mode ADDM was working in Oracle 10g is now called *instance* ADDM. The main target of database ADDM is to analyze and report on RAC environment.

To enable Database ADDM, you set the parameter `INSTANCES` in `DBMS_ADVISOR`. Following are the available options to set this parameter:

```
-- Disable Database ADDM for all instances
exec DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER('ADDM','INSTANCES','UNUSED');

-- Configure Database ADDM for instances 1 and 3 only
exec DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER('ADDM','INSTANCES','1,3');

-- Configure Database ADDM for all instances
exec DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER('ADDM','INSTANCES','ALL');
```

The results of this additional level of analysis will appear in several places in OEM, such as the Cluster database home page in the Performance Analysis page.

ADDM New Views

Oracle Database 11g has added the following new ADDM-related views:

DBA_ADDM_TASKS	displays information about all ADDM tasks in the database.
DBA_ADDM_INSTANCES	provides instance-level information for ADDM tasks that finished executing.
DBA_ADDM_FINDINGS	displays the ADDM findings discovered by all advisors in the database
DBA_ADDM_FDG_BREAKDOWN	describes the contribution for each finding from the different instances.

Following are some queries from those views:

```
-- STATUS takes one of the following: INITIAL, EXECUTING, INTERRUPTED, COMPLETED, ERROR
select TASK_NAME, ADVISOR_NAME, STATUS, PCT_COMPLETION_TIME, ERROR_MESSAGE
from   DBA_ADDM_TASKS
order by EXECUTION_END desc

-- TYPE takes on of the following: PROBLEM, SYMPTOM, ERROR, INFORMATION
-- can be linked to DBA_ADDM_TASKS using TASK_ID
select TASK_NAME, FINDING_NAME, TYPE, OBJECT_ID, IMPACT_TYPE, IMPACT, MESSAGE
from   DBA_ADDM_FINDINGS
order by IMPACT desc

-- STATUS takes one of the following: ANALYZED, BOUNCED, NO_SNAPS, NO_STATS, NOT_FOUND
select INSTANCE_NAME, STATUS
from   DBA_ADDM_INSTANCES
```

Finding Classifications

A *finding name* has been added to the Advisor framework in Oracle Database 11g. The finding name provides additional information that helps to classify the finding being given. Finding name can be found in the DBA_ADVISOR_FINDING_NAMES view.

```
select to_char(EXECUTION_END, 'hh24') hour , count(*)
from   DBA_ADVISOR_FINDINGS a, DBA_ADVISOR_TASKS b
where  FINDING_NAME like 'CPU Usage%'
  and  a.TASK_ID=b.TASK_ID
group by to_char(EXECUTION_END, 'hh24')
order by 1;
```

Managing ADDM with DBMS_ADDM

Oracle Database 11g introduces the DBMS_ADDM package to assist the DBA in administration of Oracle ADDM. Using the package DBMS_ADVISOR is still active. Below are code examples on using the DBMS_ADDM package:

```
/* to execute a database wide ADDM analysis and report on the results */
-- get the list of valid snapshots within the last 4 hours
select INSTANCE_NUMBER, SNAP_ID
from   WRM$_SNAPSHOT
where  END_INTERVAL_TIME < SYSTIMESTAMP - INTERVAL '4' HOUR
order by 1,2;
INSTANCE_NUMBER SNAP_ID
--------------- ----------
1               24
2               23
2               25
2               26

VAR tname varchar2(60);
begin
 :tname:='ADDM Database Task';
 DBMS_ADDM.ANALYZE_DB(:tname, 25, 26); -- or use ANALYZE_INST ANALYZE_PARTIAL procedures
end;

set long 1000000
spool /tmp/addmrpt.txt
SELECT DBMS_ADDM.GET_REPORT(:tname ) FROM dual;
spool off
-- remove the ADDM analysis
exec DBMS_ADDM.DELETE('ADDM Database Task');
```

Directives

To exclude various ADDM analysis and findings from appearing in the report, you can set directives. These directives can be assigned to a specific ADDM task, or can be set as a system directive. Directives can be set via command line, or from within OEM. See the following examples of the procedures used to set directives:

```
/* To create a new ADDM task to analyze a local instance.
 The result of GET_REPORT will only show an 'Undersized SGA' finding if the finding
 is responsible for at least 2 average active sessions during the analysis period,
 and this constitutes at least 10% of the total database time during that period. */
var tname VARCHAR2(60);
begin
DBMS_ADDM.INSERT_FINDING_DIRECTIVE(
  TASK_NAME => NULL, -- apply the directive to all subsequently created ADDM Tasks
  DIR_NAME  => 'Undersized SGA Directive', -- directives must be given unique names
  FINDING_NAME => 'Undersized SGA', -- derived from NAME in DBA_ADVISOR_FINDING_NAMES
  MIN_ACTIVE_SESSIONS => 2,
  MIN_PERC_IMPACT => 10);
 :tname := 'my_instance_analysis_mode_task';
 DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
end;
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;

/* To create a directive to prevent ADDM from creating actions to alter the
   value of a specific system parameter. */
var tname varchar2(60);
BEGIN
 DBMS_ADDM.INSERT_PARAMETER_DIRECTIVE(
   TASK_NAME => NULL,
   DIR_NAME  => 'System Parameter Directive',
   PARAMETER_NAME=>'sga_target');
 :tname := 'my_instance_analysis_mode_task';
 DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

```
/* To creates a directive that will prevent ADDM from creating actions to "run
Segment Advisor" for specific owner, segment, subsegment, or a specific object number. */
var tname VARCHAR2(60);
BEGIN
 DBMS_ADDM.INSERT_SEGMENT_DIRECTIVE(
   TASK_NAME => NULL,
   DIR_NAME=>'my Segment directive',
   OWNER_NAME=>'SCOTT', -- owner of the segment to be filtered out
   OBJECT_NAME=>null, -- all objects (wildcards are allowed)
   SUB_OBJECT_NAME=>null, -- a partition or sub partition
   OBJECT_NUMBER => null); -- found in views DBA_OBJECTS or DBA_SEGMENTS
 :tname := 'my_instance_analysis_mode_task';
 DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

```
/* to create a directive to limit reporting of actions on the SQL id 'abcd123456789'.
The result of GET_REPORT will only show actions for that SQL (actions to tune the
SQL, or to investigate application using it) if the SQL is responsible for at
least 2 average active sessions during the analysis period, and the average
response time was at least 1 second. */

var tname VARCHAR2(60);
BEGIN
DBMS_ADDM.INSERT_SQL_DIRECTIVE(
  TASK_NAME => NULL,
  DIR_NAME =>'my SQL directive',
  SQL_ID   =>'abcd123456789',
  MIN_ACTIVE_SESSIONS =>2,
  MIN_RESPONSE_TIME=>1000000);
:tname := 'my_instance_analysis_mode_task';
DBMS_ADDM.ANALYZE_INST(:tname, 1, 2);
END;
SELECT DBMS_ADVISOR.GET_TASK_REPORT(:tname, 'TEXT', 'ALL') FROM DUAL;
```

A new column, FILTERED, which can be found in a number of views, indicates if a particular row in the view was filtered out by a directive. Views with the FILTERED column include: DBA_ADVISOR_FINDINGS, DBA_ADVISOR_RECOMMENDATIONS, DBA_ADVISOR_ACTIONS.

Following are examples of the procedures to remove directives:

```
exec DBMS_ADDM.DELETE_FINDING_DIRECTIVE(DIR_NAME =>'my_directive');
exec DBMS_ADDM.DELETE_PARAMETER_DIRECTIVE(DIR_NAME =>'my_directive');
exec DBMS_ADDM.DELETE_SEGMENT_DIRECTIVE(DIR_NAME =>'my_directive');
exec DBMS_ADDM.DELETE_SQL_DIRECTIVE(DIR_NAME =>'my_directive');
```

AWR New Features

Default Retention of AWR Snapshots Changed

By default, Oracle Database 11g will now retain eight days of AWR snapshot information (as opposed to seven in Oracle 10g).

New Types of AWR Baselines

Oracle Database 11g offers the following new types of AWR baseline:

- Moving Window Baselines: currently there is only one named as SYSTEM_MOVING_WINDOW
- Baseline Templates: they enable you to create baselines for a contiguous time period in the future. There are two types of baseline templates:
  - Single baseline
  - Repeating baseline

Moving Window Baselines

A moving window baseline (its name is always SYSTEM_MOVING_WINDOW) corresponds to all AWR data that exists within the AWR retention period. This is useful when using adaptive thresholds because the AWR data in the entire AWR retention period can be used to compute metric threshold values.

You can resize the moving window baseline by changing the number of days in the moving window to a value that is equal to or less than the number of days in the AWR retention period as in the following example:

```
-- first increase AWR retention period (in minutes)
exec DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS(RETENTION=> 43200);

-- then you can increase window size (in days)
exec DBMS_WORKLOAD_REPOSITORY.MODIFY_BASELINE_WINDOW_SIZE ( WINDOW_SIZE => 30);

-- info about the window
select BASELINE_NAME, BASELINE_TYPE, START_SNAP_TIME, END_SNAP_TIME
from  DBA_HIST_BASELINE;
```

Single Baseline Template

A single baseline template can be used to create a baseline during a single, fixed time interval in the future. For example, you can create a single baseline template to generate a baseline that is captured on July 2, 2007 from 5:00 p.m. to 8:00 p.m.

```
begin
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (
 START_TIME => '2007-07-02 17:00:00 PST',
 END_TIME   => '2007-07-02 20:00:00 PST',
 BASELINE_NAME => 'Baseline_070702',
 TEMPLATE_NAME => 'Template_070702',
 EXPIRATION    => 30, -- (in days) if unspecified, it will never expire
 DBID => 3310949047   -- optional: if unspecified, the local db id is used
 );
end;
```

Repeating Baseline Template

A repeating baseline template is used to automatically create baselines that repeat during a particular time interval over a specific period in the future. For example, you can create a repeating baseline template to generate a baseline that repeats every Monday from 5:00 p.m. to 8:00 p.m. for the year 2007.

```
begin
DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (
 DAY_OF_WEEK => 'monday', -- the day of the week on which the baseline will repeat
 HOUR_IN_DAY => 17, -- (0 to 23) the hour in the day when the baseline will start
 DURATION => 3, -- number of hours the baseline will last
 EXPIRATION => 30, -- number of days to retain each created baseline
 START_TIME => '2007-04-02 17:00:00 PST',
 END_TIME => '2007-12-31 20:00:00 PST',
 BASELINE_NAME_PREFIX =>'bas_07_mon', -- will be appended to the baseline names
 TEMPLATE_NAME => 'template_2007_mondays',
 DBID => 3310949047);
end;
```

Rename Baselines

You can use the statement as in the following example to rename existing baselines:

```
begin
DBMS_WORKLOAD_REPOSITORY.RENAME_BASELINE(
 OLD_BASELINE_NAME =>'workload_baseline',
 NEW_BASELINE_NAME =>'workload_baseline0407',
 DBID => 3310949047);
end;
```

Dropping Baseline Templates

Following is an example on how to drop a baseline template:

```
Exec DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE_TEMPLATE( TEMPLATE_NAME => 'MyTemplate' );
```

Obtaining Information About Existing Baselines

Use the code in the following example to obtain information about existing baselines in the database:

```
-- info about Baseline Templates
select TEMPLATE_NAME, TEMPLATE_TYPE,
START_TIME,END_TIME, DAY_OF_WEEK, HOUR_IN_DAY, DURATION, EXPIRATION, REPEAT_INTERVAL
from DBA_HIST_BASELINE_TEMPLATE;

-- info on baselines taken in the system
```

```
-- if BASELINE_TYPE equals GENERATED, it is generated by the system using a template
select BASELINE_NAME, BASELINE_TYPE, START_SNAP_ID, START_SNAP_TIME,
       END_SNAP_ID, END_SNAP_TIME, MOVING_WINDOW_SIZE,
       EXPIRATION, TEMPLATE_NAME
from   DBA_HIST_BASELINE;

-- details about the baseline
--  PCT_TOTAL_TIME Amount of time captured in snapshots, divided by the total possible
--   time for this baseline
select INSTANCE_NUMBER, BASELINE_ID, BASELINE_NAME, BASELINE_TYPE,
       START_SNAP_ID, START_SNAP_TIME, END_SNAP_ID, END_SNAP_TIME,
       SHUTDOWN, ERROR_COUNT, PCT_TOTAL_TIME, LAST_TIME_COMPUTED,
       MOVING_WINDOW_SIZE, EXPIRATION, TEMPLATE_NAME
from DBA_HIST_BASELINE_DETAILS
```

Setting Metric Thresholds for Baselines

Setting Metric Thresholds for the Default Moving Baseline using Workload Profiles

To set metric thresholds for the default moving baseline in OEM:

1. Follow the links **Database Home page> Related Links section> Baseline Metric Thresholds> Quick Configuration> Workload Profile section> select one of the following options:**
   - Primarily OLTP
   - Primarily Data Warehousing
   - Alternating
2. Click **Continue** then click **Finish**.

Setting Metric Thresholds for Selected Baselines

To set a metric threshold for a selected baseline in OEM:

1. Follow the links **Database Home> Related Links> Baseline Metric Thresholds**
2. In the **View** list, select **Basic Metrics**.
3. Specify the metric you want to edit (from the Category/Name column) and the name of a baseline (from the AWR Baseline column) then click **Edit Thresholds**.
4. Modify any threshold setting then click **Apply Thresholds**.

Performance-Related Changes in Database Control

Customized Performance Page

You can in Oracle 11g set up a customized Database Performance page in Database Control.

Click the **Settings link** to go to the **Performance Page Settings** page. The Performance Page Settings page has two sections. The **Detailed Chart Settings section** lets you choose defaults for displaying the instance activity charts. The **Baseline Display Settings section** lets you select if and how the AWR baseline values are displayed in the Performance page charts.

Average Active Sessions

The **average active sessions section** on the **Performance page** offers shows, in graphical form, what proportion of time users are waiting and the time they spend on an activity, such as IO.

ADDM Performance Analysis

On the Database Control Database home page, you can now see an ADDM performance analysis summary table, which provides you a quick overview of current ADDM findings.

Miscellaneous New Performance Tuning Features

Active session history (ASH) enhancements

ASH statistics are enhanced to provide row-level activity information for each SQL statement that is captured. As with Oracle 10g, current ASH is stored in `V$ACTIVE_SESSION_HISTORY` view and its historical data stored in the `DBA_HIST_ACTIVE_SESS_HISTORY` view.

Enhanced I/O statistics

I/O statistics are collected for all I/O calls made by Oracle Database in the following dimensions: consumer group, database file, and database function. Following are the new views providing this information:

V$IOSTAT_CONSUMER_GROUP	captures I/O statistics for consumer groups. If Oracle Database Resource Manager is enabled, I/O statistics for all consumer groups that are part of the currently enabled resource plan are captured.
V$IOSTAT_FILE	This view displays I/O statistics of database files that are or have been accessed. The SMALL_SYNC_READ_LATENCY column displays the latency for single block synchronous reads (in milliseconds), which translates directly to the amount of time that clients need to wait before moving onto the next operation.
V$IOSTAT_FUNCTION	The V$IOSTAT_FUNCTION view captures I/O statistics for database functions (such as the LGWR and DBWR).

Following are examples of queries on those views:

```
-- I/O stats for Datafiles and Tempfiles
-- Single block operations are small I/Os that are less than or equal to 128 kilobytes.
-- SMALL_SYNC_READ_LATENCY is Latency for single block synch reads (ms)
select FILE_NO, FILETYPE_NAME,
SMALL_SYNC_READ_REQS "synch single block read reqs",
SMALL_READ_REQS  "single block read requests",
SMALL_WRITE_REQS  "single block write requests",
round(SMALL_SYNC_READ_LATENCY/1000,2) "Single Block Read Latency (s)",
LARGE_READ_REQS "multiblock read requests",
LARGE_WRITE_REQS "multiblock write requests",
ASYNCH_IO "asynch I/O Availability"
from   V$IOSTAT_FILE
where  FILETYPE_ID IN (2,6); -- data file and temp file

-- I/O stats by functionality
select FUNCTION_NAME,
       SMALL_READ_REQS  "single block read requests",
       SMALL_WRITE_REQS  "single block read requests",
       LARGE_READ_REQS "multiblock read requests",
       LARGE_WRITE_REQS "multiblock write requests",
       NUMBER_OF_WAITS "I/O waits",
       round(WAIT_TIME/1000,2) "Total wait time (ms)"
from   V$IOSTAT_FUNCTION
order by FUNCTION_NAME;
```

Real-Time SQL Monitoring

The real-time SQL monitoring feature of Oracle Database enables you to monitor the performance of SQL statements while they are executing. By default, SQL monitoring is automatically started when a SQL statement runs parallel, or when it has consumed at least 5 seconds of CPU or IO time in a single execution.

The statistics for monitored SQL statement execution can be displayed using the V$SQL_MONITOR and V$SQL_PLAN_MONITOR views.

```
select STATUS,
  to_char(M.FIRST_REFRESH_TIME,'hh24:mi') "First Refresh Time",
  to_char(M.LAST_REFRESH_TIME,'hh24:mi') "Last Refresh Time",
  M.ELAPSED_TIME "Elapsed Time (micro s)",
  M.CPU_TIME "CPU Time (micro s)",
  M.BUFFER_GETS,
  M.DISK_READS,
  M.USER_IO_WAIT_TIME "IO Wait Time (micro s)"
from   V$SQL_MONITOR M, V$SQL S
where  M.SQL_ID = S.SQL_ID AND
       upper(SQL_TEXT) like 'SELECT%FROM%NAMES%'
```

An alternative more convenient method is to generate the SQL monitor report using the DBMS_SQLTUNE.REPORT_SQL_MONITOR function as follows:

```
variable my_rept CLOB;
BEGIN
:my_rept :=DBMS_SQLTUNE.REPORT_SQL_MONITOR();
```

```
END;
/
print :my_rept
```

The SQL monitoring feature is enabled at the database level by default when the STATISTICS_LEVEL initialization parameter is either set to ALL or TYPICAL (the default value). Additionally, the CONTROL_MANAGEMENT_PACK_ACCESS parameter must be set to DIAGNOSTIC+TUNING (the default value).

To force SQL monitoring at the SQL statement level, use the MONITOR hint. To prevent the hinted SQL statement from being monitored, use the NO_MONITOR reverse hint.

```
select /*+MONITOR*/ from ..;
```

Adaptive Cursor Sharing

When bind variables are used in the SELECT statements against columns containing skewed data, they sometimes lead to less than optimum execution plans. The optimizer creates the execution plan during the hard parse when the statement is first presented to the server. This execution plan is then used for all every execution of the statement, regardless of the bind variable values.

Oracle 11g uses Adaptive Cursor Sharing which compares the effectiveness of execution plans between executions with different bind variable values. If it notices suboptimal plans, it allows certain bind variable values, or ranges of values, to use alternate execution plans for the same statement. This functionality requires no additional configuration.

Database Security

Stronger Password Hash Algorithm

In Oracle Database 11g, the SHA-1 standard became the new algorithm for password hashing. SHA-1 is a 160-bit hash employed in several widely used security applications and protocols, including TLS and SSL, PGP, SSH, S/MIME, and IPsec.

Security Out of the Box

Default Database Security Settings

If you create a database using DBCA and you select to enable the security settings, Oracle by default will create the database with new level of security settings as follows:

Database audits of relevant SQL statements and privileges

Modifications to the default profile. The new default profile is set with these settings:

```
PASSWORD_LOCK_TIME=1
PASSWORD_GRACE_TIME=7
PASSWORD_LIFE_TIME=180
FAILED_LOGIN_ATTEMPTS=10
PASSWORD_REUSE_MAX=UNLIMITED
PASSWORD_REUSE_TIME=UNLIMITED
```

DBCA will automatically set the `AUDIT_TRAIL` initialization parameter to `DB`.

To display the privileges that are audited, issue the following query:

```
select PRIVILEGE, SUCCESS, FAILURE
from   DBA_PRIV_AUDIT_OPTS
order by PRIVILEGE;
```

Default Auditing

Whether you create a new Oracle 11g database or you upgrade the database to Oracle 10g, if you accept to the auditing defaults, the following operations will be audited:

Alter any procedure	**Create any job**	**Drop any table**
Alter any table	**Create any library**	**Drop profile**
Alter database	**Create any procedure**	**Drop user**
Alter profile	**Create any table**	**Exempt access policy**
Alter system	**Create external job**	**Grant any object privilege**
Alter user	**Create public database link**	**Grant any privilege**
Audit role by access	**Create session**	**Grant any role**
Audit system	**Create user**	**Audit system by access**
Drop any procedure		

As with previous versions, audit data is stored in AUD$ table which should be manually maintained by you. If you want to disable auditing a specific operation, use the `NOAUDIT` command.

Delayed Failed Logins

If a user tries to connect to the database multiple times using an erroneous password, the database will delay the response to the client after the third attempt. The delays of response back to the client are repeated even if the connections are initiated from different IP addresses or hosts.

Case-Sensitive Passwords

Oracle 11g introduces case-sensitive passwords for databases created with the default Oracle Database 11g enhanced security.

The `SEC_CASE_SENTITIVE_LOGON` parameter must be set to TRUE to enable case-sensitive database passwords.

```
alter system set SEC_CASE_SENSITIVE_LOGON = TRUE;
```

If you import a dump file of users from Oracle Database 9i or 10g, users' passwords will remain case-insensitive until you manually reset them. Same rule apply when you upgrade a database from an earlier version to Oracle 11g. The following query will display the database users whose passwords are not case sensitive:

```
select USERNAME, PASSWORD_VERSIONS
from   DBA_USERS
```

```
where  PASSWORD_VERSIONS NOT LIKE '%11G%'
order by USERNAME;
```

Case-Sensitive Password Files

Passwords created in the password file can be set as case-sensitive by using the new option ignorecase with the utility orapwd. Following is an example:

```
orapwd file=$ORACLE_HOME/dbs/orapw$ORACLE_SID password=ORAcle123 \
entries=25 ignorecase=n
```

Change Default User Passwords

The new DBA_USERS_WITH_DEFPWD view reports those accounts with default passwords for Oracle-supplied database accounts. It is a good practice to change passwords of users displayed by this view.

```
select USERNAME from DBA_USERS_WITH_DEFPWD order by USERNAME;
```

Database Links and Case Sensitivity

If you create a database link in an earlier version of Oracle than 11g, you must alter the password on Oracle Database 11g to the uppercase equivalent of the password designated in the database link's CONNECT TO USERNAME IDENTIFIED BY clause section.

Hiding Password Hash Values in DBA_USERS

To provide further level of security, the DBA_USERS view in Oracle Database 11g has the password column blanked out instead of displaying the hashed value of the password.

```
select USERNAME, PASSWORD from DBA_USERS order by USERNAME;
```

New Password Verification Function

Oracle 11g provides a new password verification function with stronger settings than those in the functions of earlier versions. This function, however, is not enabled by default.

The script $ORACLE_HOME/rdbms/admin/utlpwdmg.sql creates the new function (named as VERIFY_FNCTION_11G), enables it in the default profile and also it creates the Oracle 10g function for legacy compatibility.

```
@$ORACLE_HOME/rdbms/admin/utlpwdmg.sql
```

The function forces restrictions on the database users' passwords like minimum eight characters, cannot be same or similar to the user name or the hostname, and must has at least one letter and one digit.

Anti Network Attacks Parameters

Oracle Database 11g provides some initialization parameters to protect against Internet attacks including the following:

SEC_PROTOCOL_ERROR_FURTHER_ACTION

Specifies the further execution of a server process when receiving bad packets from a possibly malicious client. Following are its possible values:

- CONTINUE — The server process continues execution. The database server may be subject to a Denial of Service (DoS) if bad packets continue to be sent by a malicious client.
- (DELAY, n) — The client experiences a delay of *n* seconds before the server process accepts the next request from the same client connection.
- (DROP, n) — The server forcefully terminates the client connection after *n* bad packets. The server protects itself at the expense of the client (for example, a client transaction may be lost).

SEC_PROTOCOL_ERROR_TRACE_ACTION

Specifies the trace level when bad packets are received from a possibly malicious client.

- NONE — The database server ignores the bad packets and does not generate any trace files or log messages.
- TRACE — A detailed trace file is generated when bad packets are received, which can be used to debug any problems in client/server communication.
- LOG — A minimal log message is printed in the alert logfile and in the server trace file. A minimal

amount of disk space is used.

ALERT An alert message is sent to a DBA or monitoring console.

SEC_MAX_FAILED_LOGIN_ATTEMPTS

Defines the number of authentication attempts that a given client connection can make on the server before the client process is dropped. The default value is 10.

SEC_RETURN_SERVER_RELEASE_BANNER

Determines if the server banner will be returned to a client connection. Not returning the banner will make hacking a database more difficult since the user will not know which version of the database they are trying to hack.

Tablespace Encryption

In Oracle Database 11g, as an extension to Transparent Data Encryption (TDE), you can encrypt an entire tablespace. Tablespace encryption relies on encryption keys in a wallet outside the database. When you use an encrypted tablespace, the entire tables and associated indexes in the tablespace are encrypted. Also, the data remains encrypted when it is stored in the redo logs.

Encrypted Tablespace Limitations

- You cannot encrypt an existing tablespace.
- exp and imp utilities are not supported with objects in the encrypted tablespaces. Whereas expdp and impdp utilities are supported.
- You cannot re-create the tablespace encryption key.
- The NO SALT option is not supported.
- Temporary and undo tablespaces cannot be encrypted.
- You cannot transport an encrypted tablespace to a database that already has an Oracle wallet for TDE. In this case, use Oracle Data Pump to export the objects in the tablespace using the expdp with ENCRYPTION_MODE=password and then import them to the destination database.
- COMPATIBLE parameter must be set to 11.1 or higher.
- BFILES and external tables are not encrypted.
- Logically, encrypted tablespace is less efficient than normal un-encrypted tablespace.

Caution Losing the master key or the wallet file will lead to losing the data in the encrypted tablespace. Always include the wallet file in your backup plan and save the master key password in safe place.

Setting up TDE

As with Oracle 10g, you can perform the following steps to set up TDE:

Add the following to the sqlnet.ora file:

```
ENCRYPTION_WALLET_LOCATION =
 (SOURCE=
   (METHOD=file)
     (METHOD_DATA=
 (DIRECTORY=C:\oracle\OraDb10g\admin\ora10g\wallet)))
```

Set the master key. This is done only ***once***:

```
ALTER SYSTEM SET ENCRYPTION KEY IDENTIFIED BY <password>;
```

Opening and Closing the Wallet

If you restart the instance, the Wallet must be opened using the following command:

```
ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY password>;
ALTER SYSTEM SET ENCRYPTION WALLET CLOSE;
```

To verify that a wallet is open, you can query the V$ENCRYPTION_WALLET view:

```
select WRL_PARAMETER, STATUS from V$ENCRYPTION_WALLET;
```

Encrypting a Tablespace

The tablespace creation statement for an encrypted tablespace has the following syntax:

```
CREATE TABLESPACE <tbsp_name> ...
[ENCRYPTION [USING <ALGORITHM>]] -- specify encryption algorithm
DEFAULT STORAGE(ENCRYPT)  -- encrypt objects in the tablespace
```

The ALGORITHM clause accepts one of the following values:

- AES192 Advanced Encryption Standard (the default).
- 3DES168 Triple Data Encryption Standard 168-bit encryption
- AES128 Advanced Encryption Standard 128-bit encryption
- AES256 Advanced Encryption Standard 256-bit encryption

To know whether an existing tablespace is encrypted or not, issue the following query:

```
select vt.NAME, vet.ENCRYPTIONALG, vet.ENCRYPTEDTS
from   V$ENCRYPTED_TABLESPACES vet, V$TABLESPACE vt
where  vet.TS#=vt.TS#
```

Fine-Grained Access Control for UTL_* Packages

Oracle Database 11g provides a mechanism to refine the level of access to the network access packages UTL_TCP, UTL_SMTP, UTL_MAIL, UTL_HTTP, and UTL_INADDR. You can use the DBMS_NETWORK_ACL_ADMIN package to facilitate management of the UTL_* network access packages as in the following steps:

1) Create an Access Control List (ACL): All ACL definitions are stored in XML DB in the form of XML documents. The ACL XML files reside in the /sys/acls directory of the XML DB repository. Following is an example of using the CREATE_ACL procedure to create an XML file called dba.xml:

```
begin
  DBMS_NETWORK_ACL_ADMIN.CREATE_ACL (
   ACL => 'dba.xml', -- case sensitive
   DESCRIPTION=> 'Network Access Control for the DBAs',
   PRINCIPAL  => 'SCOTT', -- user or role the privilege is granted or denied (upper case)
   IS_GRANT   => TRUE, -- privilege is granted or denied
   PRIVILEGE  => 'connect', -- or 'resolve' (case sensitive)
   START_DATE => null,  -- when the access control entity ACE will be valid
   END_DATE   => null); -- ACE expiration date (TIMESTAMP WITH TIMEZONE format)
end;
```

Regarding the PRIVILEGE parameter, the database user needs the connect privilege to an external network host computer if he or she is connecting using the UTL_TCP, UTL_HTTP, UTL_SMTP, and UTL_MAIL utility packages. To resolve a host name that was given as a host IP address, or the IP address that was given as a host name, with the UTL_INADDR package, grant the database user the resolve privilege.

You can then query the RESOURCE_VIEW view to find the dba.xml ACL in the /sys/acls directory:

```
select ANY_PATH
from   RESOURCE_VIEW
where  ANY_PATH LIKE '/sys/acls/dba%'
```

Too may entries in the ACL may lead to significant XML DB performance drop because ACL are checked for each access to Oracle XML DB repository. As general rule of thumb, ACL check operations perform best when the number of ACEs in the ACL is at 16 entries or less.

2) Add Access Control Entries: Once you create the initial ACL, you can continue to add more privileges to the XML file. The following example will add the user RAMI to the dba.xml file and grant him network access:

```
begin
 DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE (
 ACL => 'dba.xml',
 PRINCIPAL => 'RAMI',
 IS_GRANT => TRUE,
 PRIVILEGE => 'connect',
 START_DATE => null, -- if the time interval is defined,
 END_DATE => null);  --  the ACE will expire after the specified date range
end;
COMMIT;
```

In ACL, the security entries are evaluating in order precedence. If you have two contradicting entries in the list, the first one in the order will take effect. You can control the order number of an added entry as follows:

```
begin
 DBMS_NETWORK_ACL_ADMIN.ADD_PRIVILEGE (
  POSITION => 1, -- on the top
  ACL => 'dba.xml',    PRINCIPAL => 'SAMI',
  IS_GRANT => FALSE,   PRIVILEGE => 'connect',
  START_DATE => null,  END_DATE => null);
end;
```

3) Assign Hosts: The ASSIGN_ACL procedure is used to authorize access to one or more network hosts as follows:

```
begin
DBMS_NETWORK_ACL_ADMIN.ASSIGN_ACL (
 ACL => 'dba.xml',   HOST => 'dbaexpert.com',
 LOWER_PORT => 80,   UPPER_PORT => 443);
end;
COMMIT;
```

The lower port and the upper port define the lower and the upper boundaries of the allowable port range. They should be set for connect privileges not resolve privileges.

4) Validate that the ACL permissions worked accordingly. Following is an example to test the code in the previous step.

```
select UTL_HTTP.REQUEST('http://www.ahmedbaraka.com') from dual;
```

If the sufficient ACL privileges or ACL assignments are not provided, you will receive the ORA-24247 error.

Access Control Lists Maintenance

Use DELETE_PRIVILEGE to remove an access control entry from the XML file.

```
exec DBMS_NETWORK_ACL_ADMIN.DELETE_PRIVILEGE( ACL=>'dba.xml', PRINCIPAL=> 'RAMI');
```

Use the DROP_ACL procedure to remove the XML file from the /sys/acls directory as follows:

```
exec DBMS_NETWORK_ACL_ADMIN.DROP_ACL ( ACL=>'dba.xml' );
```

Query Your Access Control List

To display list of the ACLs created in the database, use the following query:

```
select HOST, LOWER_PORT, UPPER_PORT, ACL from DBA_NETWORK_ACLS
```

You can query the DBA_NETWORK_ACL_PRIVILEGES view to query network privileges granted or denied for the access control list as follows:

```
select PRINCIPAL, PRIVILEGE, IS_GRANT
 from   DBA_NETWORK_ACL_PRIVILEGES
 where  ACL like '%dba.xml'
```

Logged on users can use the following query to see their access entries in the dba.xml file:

```
select HOST, LOWER_PORT, UPPER_PORT, STATUS privilege
from   USER_NETWORK_ACL_PRIVILEGES
where HOST in
(select * from
    table(DBMS_NETWORK_ACL_UTILITY.DOMAINS('dbaexpert.com')))
and PRIVILEGE = 'connect'
order by DBMS_NETWORK_ACL_UTILITY.DOMAIN_LEVEL(host) desc, LOWER_PORT;
```

Further Security New Features

New SYSASM Privilege for ASM Administration

SYSASM is a new privilege introduced in Oracle 11g. Users who are granted this privilege can perform ASM administration tasks. The idea behind this privilege is to separate database management and the storage

management responsibilities. For further details about using this privilege, refer to the chapter "Automatic Storage Management".

Data Pump Encryption

Oracle Database 11g introduces the encryption of Data Pump dump files. For details of all the new enhancements related to Data Pump, refer to the chapter "Data Pump".

RMAN Virtual Private Catalog

In Oracle Database 11g, you can restrict access to the recovery catalog by granting access to only a subset of the metadata in the recovery catalog. For complete details on RMAN virtual private catalogs, refer to the section "Virtual Private Catalogs".

Backup and Recovery New Features

Enhanced Block Media Recovery

In Oracle Database 11g, there is a new command to perform block media recovery, named the `recover ... block` command replacing the old `blockrecover` command. The new command is more efficient since because it searches the flashback logs for older uncorrupted versions of the corrupt blocks. This requires the database to work in archivelog mode and has the Database Flashback enabled.

While the block media recovery is going on, any attempt by users to access data in the corrupt blocks will result in an error message, telling the user that the data block is corrupt.

Following are examples of using the new command:

```
RECOVER DATAFILE 2 BLOCK 24 DATAFILE 4 BLOCK 10;
RECOVER DATAFILE 2 BLOCK 24 DATAFILE 4 BLOCK 10 FROM TAG = sundaynight;
RECOVER DATAFILE 2 BLOCK 24 DATAFILE 4 BLOCK 10 FROM BACKUPSET = 11;

-- to fix all corrupted blocks in the database
-- after validate database, corrupted blocks are reported in V$DATABASE_BLOCK_CORRUPTION
RMAN> VALIDATE DATABASE;
RMAN> RECOVER CORRUPTION LIST;
```

RMAN Substitution Variables

You can now use substitution variables in RMAN command files, which you can then incorporate in shell scripts. RMAN command line now has a new keyword, that is `USING`, which is used to pass substitution variables to the command line. Following are examples on how to do that.

Following are the steps to create dynamic shell script that accepts substitution variables in a Unix platform:

1. Create an RMAN command file that uses substitution variables. Following is an example:

```
# quarterly_backup.cmd
CONNECT TARGET /
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE sbt  PARMS 'ENV=(OB_MEDIA_FAMILY=&1)';
  BACKUP DATABASE TAG &2 FORMAT '/disk2/bck/&1%U.bck' KEEP FOREVER
  RESTORE POINT &3;
}
EXIT;
```

2. Create a shell script that you can use to run the RMAN command file created in the previous step. Following is an example:

```
#!/bin/tcsh
# name: runbackup.sh
# usage: use the tag name and number of copies as arguments
set media_family = $argv[1]
set format = $argv[2]
set restore_point = $argv[3]
rman @'/u01/scripts/quarterly_backup.cmd' USING $media_family $format $restore_point
```

3. Run the shell script and pass the values to the substitution variables. Following is an example:

```
%runbackup.sh archival_backup bck0906 FY06Q3
```

Following is an example of achieving the same concept steps but in Windows platform:

```
-- RMAN command file hr_backup.cmd
CONNECT TARGET /
RUN
{
  ALLOCATE CHANNEL c1 DEVICE TYPE disk;
  BACKUP DATAFILE 'c:\oracle\oradata\ora11g\HR_TBS1.DBF'
    TAG &1 FORMAT 'c:\orabackup\&1%U.bck';
  SQL 'CREATE RESTORE POINT &2';}
EXIT;

-- Batch file hr_backup.bat
rman @'C:\orabackup\scripts\hr_backup.cmd' USING %1 %2

--- run the batch file
hr_backup.bat HR01MAR08 RS_HR01MAR08
```

New RMAN Configuration Parameters

The COMPRESSION ALGORITHM and ARCHIVELOG DELETION POLICY parameters are new RMAN configuration parameters in the Oracle Database 11g.

The OMPRESSION ALGORITHM parameter sets the compression algorithm used by RMAN for compressing a backup. Its possible values are the following:

	consumes more CPU resource than ZLIB, but will usually produce more compact backups.
ZLIB	It is optimized for CPU efficiency. It requires the Oracle Advanced Compression option.

The ARCHIVELOG DELETION POLICY parameter will be discussed in Configuring an Archived Redo Log Deletion Policy section.

The Multisection Backups

Oracle 11g lets you back up and restore a large file in sections. In a *multisection backup*, each RMAN channel backs up a different section of a datafile. You can have up to 256 sections per datafile.

This leads to the following benefits:

- Backup performance of a large file is significantly enhanced by allocating multiple channels to backup the datafile.
- If a backup process failed, only those sections that were not backed up prior to the backup failure are required to backup.
- If you have a datafile of a size larger than the maximum size of your storage medium unit (for example the tape), you can backup the datafile into sections with each section of a size equals to the size of your storage unit.

Use the new backup command clause SECTION SIZE to perform multisection backups as follows:

```
run
{ allocate channel c1 device type disk ;
  allocate channel c2 device type disk ;
  allocate channel c3 device type disk ;
  backup SECTION SIZE 500m tablespace example; } -- three channels operate in parallel
```

The example above shows how to use the multisection backup in a block. If the backup command was executed in command-line, the output backupset will have backup pieces of size 500 MB each. No parallelism, however, will take place, if it was not configured in RMAN.

The V$BACKUP_DATAFILE and RC_BACKUP_DATAFILE views have the new column SECTION_SIZE. This column specifies the number of blocks in each section of a multisection backup and its value is 0 for whole file backups.

```
select PIECE#, SECTION_SIZE from V$BACKUP_DATAFILE;
```

Creating Archival Backups

The BACKUP ... KEEP command can be used to create a backup that is both *all-inclusive* and exempt from the backup retention policy. The backup is all-inclusive because every file needed (including archived redo logs) to restore and recover the database is backed up to a single disk or tape location for long-term storage. The general name for a backup created with BACKUP ... KEEP is an *archival backup*.

In Oracle 11g, some modifications made on the RMAN BACKUP ... KEEP command. In the new version of the command, the KEEP, NOKEEP, FOREVER, and UNTIL TIME options are retained. However, the LOGS and NOLOGS options are not there any longer. Instead, you have a new option, RESTORE POINT. The RESTORE POINT option lets RMAN automatically create a normal restore point.

The following examples illustrate how to use the command:

```
RUN
{
 ALLOCATE CHANNEL c1 DEVICE TYPE sbt PARMS 'ENV=(OB_MEDIA_FAMILY=archival_backup)';

 -- with forever option (recovery catalog is required)
 BACKUP DATABASE TAG BAKQ108 KEEP FOREVER RESTORE POINT FY08Q1;

 -- backup will be kept for 365 days (long-term)
 BACKUP DATABASE TAG BAKQ108 KEEP UNTIL TIME 'SYSDATE+365' RESTORE POINT FY08Q1;

 -- After one day, the backup becomes obsolete,
 --  regardless the configured retention policy
 BACKUP DATABASE FORMAT '/u01/oraclebck/%U.bck'
   TAG TESTDB KEEP UNTIL 'SYSDATE+1' RESTORE POINT TESTDB08;
}
```

Note BZIP2 You can't use the KEEP clause for backup files in the flash recovery area. Also, you cannot use the CHANGE ... KEEP command for backup files stored in the flash recovery area.

If you want to change the status of a regular backup to an archival backup, use the CHANGE command as follows:

```
CHANGE BACKUP TAG 'weekly_bkp' KEEP FOREVER;

-- make it follow back the retention policy
CHANGE BACKUP TAG 'weekly_bkp' NOKEEP;
```

Restoring an Archival Backup

The procedure for restoring the archival backup is the same as for duplicating a database except in the DUPLICATE command you must specify the restore point that was created with the archival backup. Details about how to achieve that is explained in the documentation "Oracle Database Backup and Recovery User's Guide 11g Release 1 (11.1) B28270-02", section "Using DUPLICATE to Restore an Archival Backup", page 23-21.

VALIDATE Command

You can use the new command VALIDATE to manually check for physical and logical corruptions in datafiles, backup sets, and even individual data blocks. The command by default checks for physical corruption. You can optionally specify CHECK LOGICAL. Corrupted blocks are reported in V$DATABASE_BLOCK_CORRUPTION.

Following are examples of some VALIDATE command options:

```
validate [CHECK LOGICAL] database;
validate SKIP INACCESSIBLE database;
validate copy of database;
validate tablespace hr_tbs;
validate copy of tablespace hr_tbs;
validate backupset 17,18;
validate datafile 3 block 24;

-- validates recovery files created in the current and all previous flash recovery area
--  destinations
validate recovery area;
```

Note The VALIDATE command checks only for intrablock corruption both physical and logical in nature. It doesn't check for interblock corruption.

Validating large datafiles can be speeded up by splitting the checked files into sections and those sections are checked in parallel. Following is an example on how to do that:

```
run
{ALLOCATE CHANNEL CH1 DEVICE TYPE DISK;
 ALLOCATE CHANNEL CH2 DEVICE TYPE DISK;
 VALIDATE DATAFILE 3 SECTION SIZE = 500M;}
```

Configuring an Archived Redo Log Deletion Policy

You can use RMAN to create a persistent configuration that controls when archived redo logs are eligible for deletion from disk or tape. This deletion policy applies to all archiving destinations, including the flash recovery area. When the policy is configured, it applies on the automatic deletion of the logs in the flash recovery area and the manual deletion by the `BACKUP ... DELETE` and `DELETE ... ARCHIVELOG` commands.

To enable an archived redo log deletion policy, run the `CONFIGURE ARCHIVELOG DELETION POLICY BACKED UP` *n* `TIMES` command with the desired options. Following are examples:

```
-- archived redo logs are eligible for deletion when there are at least two backups of
--  them on the tape
CONFIGURE ARCHIVELOG DELETION POLICY TO BACKED UP 2 TIMES TO SBT;

-- disable the policy
CONFIGURE ARCHIVELOG DELETION POLICY TO NONE;
```

Caution The `BACKUP ARCHIVELOG` command is affected by the archived log deletion policy. If the deletion policy is configured with the `BACKED UP` *n* `TIMES` clause, then a `BACKUP ARCHIVELOG` command copies the logs unless *n* backups already exist on the specified device type. If *n* backups of the logs exist, then the `BACKUP ARCHIVELOG` command skips the logs.

You can, however, override the archived redo log deletion policy you configured by specifying the `FORCE` clause in the `BACKUP ARCHIVELOG` command.

Note `DELETE OBSOLETE` considers only the backup retention policy and does not use the configured archived log deletion policy to determine which logs are obsolete. In contrast, `DELETE ARCHIVELOG ALL` considers only the configured archived log deletion policy.

Active Database Duplication

In Oracle Database 11g, you can directly duplicate a database over the network without having to back up and provide the source database files. This direct database duplication is called *active database duplication*. It can be done either with Database Control or through RMAN. Instance that runs the duplicated database is called *auxiliary instance*.

Prerequisites

- Both the target and destination databases must be on an identical operating system platform.
- Oracle Net must be aware of both the target and duplicate instances.
- The password file must exist for the source database and both the target and destination databases must have the same SYS password.
- The target database must be open or in mount state.
- If the target database is open, it must be in ARCHIVELOG mode.

Implementing active database duplication includes the following steps:

1. Decide on Duplicate File Names

If you are duplicating to a different host that uses the same directory structure as the source host, and if you want to name the duplicate files the same as the source database files, then skip to the next step.

1.1 Decide on what are the names of the duplicate files on the destination server. Duplicate files include: control files, datafiles, online redo logs, and tempfiles. When you issue the `DUPLICATE` command later, you will use its options to implement the new names.

2. Prepare the Auxiliary Instance

2.1 Create a password file in the destination server with the same SYSDBA password as the source database. You can create the password file manually or by specifying the PASSWORD FILE option on the DUPLICATE command.

```
-- use PASSWORD FILE option
RMAN>DUPLICATE TARGET DATABASE ... PASSWORD FILE ...

-- manually
#orapwd FILE=PWDora11g2.ora PASSWORD=mypassword ENTRIES=10 ignorecase=n
```

2.2 Establish Oracle Net connectivity to the auxiliary instance in both the source and destination servers. Also add the auxiliary database service to the listener configuration file in the source server. Following are examples of a configuration in tnsnames.ora and listener.ora files:

```
dup1 =
  (DESCRIPTION =
    (ADDRESS_LIST =
        (ADDRESS =
          (COMMUNITY = tcp.world)
          (PROTOCOL = TCP)
          (Host = 192.168.255.32)
          (Port = 1521)
        )
    )
    (CONNECT_DATA = (SID =dup1) (SERVER = DEDICATED)
    )
  )

SID_LIST_LISTENER =
(SID_DESC =
  (GLOBAL_DBNAME = prod1)
  (ORACLE_HOME = /u01/app/oracle/product/10g/)
  (SID_NAME =prod1)
)
(SID_DESC =
 (GLOBAL_DBNAME = dup1)
 (ORACLE_HOME = /u01/app/oracle/product/10g/)
 (SID_NAME =dup1)
)
)
```

2.3 Including SPFILE option in the DUPLICATE command will make RMAN copy the SPFILE from the source server to the destination. In this case, you need to create a text-based initialization parameter file for the auxiliary instance that contains only one parameter: DB_NAME which can be set to an arbitrary value.

```
#contents of the init.ora in the destination
DB_NAME=dup1
```

2.4 Use SQL*Plus to connect to the auxiliary database instance with SYSOPER privileges. Start the instance in NOMOUNT mode, specifying the PFILE parameter:

```
SQL>conn sys/mypassword@dup1 as sysoper
SQL>STARTUP NOMOUNT pfile=c:\..\pfile.init
```

3. Start and Configure RMAN Before Duplication

3.1. Start RMAN and connect to the source database as TARGET, the duplicate database instance as AUXILIARY, and, if applicable, the recovery catalog database.

```
RMAN>CONNECT TARGET SYS@prod # source database
RMAN>CONNECT AUXILIARY SYS@dupdb # duplicate database instance
RMAN>CONNECT CATALOG rman@catdb # recovery catalog database
```

3.2. You may want to increase the parallelism setting of your source database disk channels so that RMAN copies files over the network in parallel.

```
CONFIGURE DEVICE TYPE DISK PARALLELISM 3 BACKUP TYPE TO BACKUPSET;
```

4. Run the DUPLICATE command

Following are the options to issue the DUPLICATE command in the RMAN session. After the duplication is finished, RMAN will restart the instance with RESETLOGS.

```
-- duplicating a database to a remote host with the same directory structure
DUPLICATE TARGET DATABASE
TO dupdb
PASSWORD FILE -- RMAN should duplicate the password file from the source
FROM ACTIVE DATABASE
SPFILE
NOFILENAMECHECK;

-- if you want to recover the duplicate database to one week ago
DUPLICATE TARGET DATABASE
TO dupdb
PASSWORD FILE
SPFILE
NOFILENAMECHECK
UNTIL TIME 'SYSDATE-7';

-- duplicating a database to a remote host with a different directory structure
DUPLICATE TARGET DATABASE TO dupdb
FROM ACTIVE DATABASE
DB_FILE_NAME_CONVERT '/oracle/oradata/prod/','/scratch/oracle/oradata/dupdb/'
SPFILE
PARAMETER_VALUE_CONVERT '/oracle/oradata/prod/', '/scratch/oracle/oradata/dupdb/'
SET SGA_MAX_SIZE '300M'
SET SGA_TARGET '250M'
SET LOG_FILE_NAME_CONVERT '/oracle/oradata/prod/redo/',
'/scratch/oracle/oradata/dupdb/redo/';
```

Importing and Moving Recovery Catalogs

You can use the IMPORT CATALOG command in RMAN to merge one recovery catalog schema into another.

Prerequisites for Importing a Recovery Catalog

If a database is currently registered in both the source and destination catalog schemas, then unregister the database from source catalog schema before performing the import.

```
connect target sys/mypassword@db1 catalog rman/rman@dest
UNREGISTER DATABASE;
```

When using IMPORT CATALOG, the version of the source recovery catalog schema must be equal to the current version of the RMAN executable with which you run the command. If the source catalog schema is a lower version, then upgrade it to the current version before importing the schema.

```
# to determine the schema version of the recovery catalog
# start SQL*Plus and connect to the recovery catalog database as the catalog owner
SQL>conn rman/ramn@mydb

#the highest version in the RCVER table is the current catalog schema version
SQL>SELECT * FROM rcver;

# to upgrade a recovery catalog schema to the current RMAN version
# make sure the CREATE TYPE privilege is granted to  the RECOVERY_CATALOG_OWNER role
SQL>SELECT PRIVILEGE
FROM ROLE_SYS_PRIVS
WHERE ROLE='RECOVERY_CATALOG_OWNER' AND PRIVILEGE = 'CREATE TYPE';
GRANT CREATE TYPE TO rman;

# start RMAN and connect to the recovery catalog database
RMAN catalog rman/rman@mydb

# upgrade the catalog (run the command twice to confirm)
RMAN>UPGRADE CATALOG;
RMAN>UPGRADE CATALOG;
```

Importing a Recovery Catalog

Following are examples of using the IMPORT CATALOG command:

```
#connect to the destination recovery catalog
RMAN>connect catalog rman/rman@dest

#Issue the import catalog command
# all RMAN repository metadata is imported from src to dest
RMAN>IMPORT CATALOG rman1/rman1@src;
# in the source, you will notice all databases are unregistered from RMAN repository
RMAN>LIST INCARNATION;
RMAM>connect target rman1/rman1@src;
RMAN>LIST INCARNATION;

# to import from source without the automatice databases unregistration
RMAN>IMPORT CATALOG rman1/rman1@src NO UNREGISTER;

# to import metadata of specific databases
RMAN>IMPORT CATALOG rman1/rman1@src DBID = 123456, 123457;
RMAN>IMPORT CATALOG rman1/rman1@src DB_NAME = mydb1, mydb2;
```

Note If you have global stored scripts in the both recovery catalogs with identical names, RMAN automatically renames the scripts from the source recovery catalog (the format for the renamed files is COPY OF Script_Name).

Moving a Recovery Catalog to Another Database

By following the same steps of using the IMPORT CATALOGE to import a recovery catalog schema, you can also move a catalog schema from one database (source) to another (destination). Only make sure the destination has a new recovery catalog without any database registered in it.

Virtual Private Catalogs

In Oracle Database 11g, you can restrict access to the recovery catalog by granting access to only a subset of the metadata in the recovery catalog. The subset that a user has read/write access to is termed as *virtual private catalog*, or just *virtual catalog*. The central or source recovery catalog is now called the *base recovery catalog*.

Following are the steps to create a new private catalog for the database user SCOTT:

```
# grant the role RECOVERY_CATALOG_OWNER to the user
SQL>GRANT RECOVERY_CATALOG_OWNER TO scott;

# in RMAN session, connect as the base catalog owner
RMAN>CONNECT CATALOG rman/rman@mydb
RMAN>GRANT CATALOG FOR DATABASE db1, db2 TO SCOTT;

# connect as the granted user (virtual catalog onwer) and create the virtual catalog
RMAN>CONNECT CATALOG scott/lion@mydb
RMAN>CREATE VIRTUAL CATALOG;

# make sure only granted dbs are seen
RMAN>LIST INCARNATION;
```

If the catalog is to be used for releases pre-Oracle 11g clients, in the SQL*Plus log on as the virtual private catalog owner and run the following procedure, where "rman" is the name of the base catalog owner:

```
SQL> CONN scott/lion@mydb
SQL> EXEC rman.DBMS_RCVCAT.CREATE_VIRTUAL_CATALOG;
```

Note A virtual private catalog owner can create a local stored script, but has only read-only access to global scripts.

The CATALOG FOR DATABASE privileges include the privilege to register and unregister those databases for which the catalog for database privilege was granted.

The set of views and synonyms that makes up the virtual private catalog is stored in the schema of the virtual catalog owner.

Managing Virtual Private Catalogs

The base recovery catalog owner can optionally grant a virtual catalog owner the right to register new target databases in the recovery catalog by specifying the REGISTER database clause with the GRANT command. Following is an example:

```
RMAN> grant register database to scott;
```

The virtual catalog owner must have the SYSDBA and SYSOPER privileges on the traget database, to perform most of the RMAN operations on it.

Following are examples of removing the privileges from a virtual catalog owner:

```
# To remove recovery catalog access to a database from a user:
RMAN>CONNECT CATALOG RMAN/RMAN@MYDB;
RMAN>REVOKE CATALOG FOR DATABASE db1 FROM scott;

# To revoke the ability to register new databases from a virtual private catalog owner:
RMAN>REVOKE REGISTER DATABASE FROM scott;

# To revoke both the catalog and register privileges from a user:
RMAN>REVOKE ALL PRIVILEGES FROM scott;
```

Dropping a Virtual Private Catalog

Virtual private catalog owners can drop the private recovery catalog they own by issuing the DROP CATALOG command. Following is an example:

```
# Log in as the virtual catalog owner:
RMAN>CONNECT CATALOG scott/<password>@mydb;

# Issue the drop catalog command
RMAN>DROP CATALOG;
```

Caution When the DROP CATALOG command is issued by the virtual catalog owner, all the metadata pertaining to it is deleted from the base recovery catalog.

Miscellaneous New Features in RMAN

Archived Redo Log Failover

The archived redo log failover feature enables RMAN to complete a backup even when some archiving destinations have missing logs or contain logs with corrupt blocks. The RMAN will search and use an alternative destination.

Optimized Backing Up of Undo Data

In Oracle Database 11g, during a backup, the committed data is not backed up, thus leading to a saving of storage space as well as faster backups for large OLTP-type databases. Since the new optimized undo backup is automatically enabled, you do not have to configure anything special to take advantage of this feature.

Data Pump Utilities

Compression Enhancement

In Oracle Database 11g, Oracle provides the mechanism to compress both data and metadata during the extract operation. The available options for the COMPRESSION parameter are as follows:

```
compression={all | data_only | metadata_only | none}
```

Following is an example:

```
$expdp full=yes userid="'/ as sysdba'"  dumpfile=dbadir:full.compress.dmp compression=all
```

Encryption Enhancements

To secure the exported dump file, the following new parameters are presented in Oracle 11g Data pump: ENCRYPTION, ENCRYPTION_PASSWORD and ENCRYPTION_ALGORITHM. To enable encryption, you must specify either the ENCRYPTION or ENCRYPTION_PASSWORD parameter, or both. Those parameters are valid only in the Enterprise Edition of Oracle Database 11g.

ENCRYPTION Parameter

This parameter specifies whether or not to encrypt data before writing it to the dump file set.

The default value depends upon the combination of encryption-related parameters that are used. If only the ENCRYPTION_PASSWORD parameter is specified, then the ENCRYPTION parameter defaults to ALL. If neither ENCRYPTION nor ENCRYPTION_PASSWORD is specified, then ENCRYPTION defaults to NONE.

The ENCRYPTION parameter has the following options:

```
ENCRYPTION = {all | data_only | encrypted_columns_only | metadata_only | none}
```

Following is an example:

```
expdp hr DUMPFILE=dp_dir.hr_enc.dmp JOB_NAME=enc ENCRYPTION=data_only
ENCRYPTION_PASSWORD=mypassword
```

ENCRYPTION_ALGORITHM Parameter

This parameter specifies which cryptographic algorithm should be used to perform the encryption. Following is its syntax:

```
ENCRYPTION_ALGORITHM = { AES128 | AES192 | AES256 }
```

The ENCRYPTION_ALGORITHM parameter requires that you also specify either the ENCRYPTION or ENCRYPTION_PASSWORD parameter.

Following is an example:

```
expdp hr DIRECTORY=dp_dir DUMPFILE=hr_enc.dmp /
 ENCRYPTION_PASSWORD=mypassword ENCRYPTION_ALGORITHM=AES128
```

ENCRYPTION_MODE Parameter

This parameter works the same way the encryption mode was operating in RMAN in Oracle 10g. It specifies the type of security to use when encryption and decryption are performed. It syntax is as follows

```
ENCRYPTION_MODE = { DUAL | PASSWORD | TRANSPARENT }
```

PASSWORD mode requires that you provide a password when creating encrypted dump file sets.

TRANSPARENT mode allows an encrypted dump file set to be created without any intervention from a database administrator (DBA), provided the required Oracle Encryption Wallet is available.

DUAL mode creates a dump file set that can later be imported either transparently or by specifying a password that was used when the dual-mode encrypted dump file set was created.

Following is an example:

```
expdp hr DIRECTORY=dp_dir DUMPFILE=hr_enc.dmp
 ENCRYPTION=all ENCRYPTION_PASSWORD=mypassword
 ENCRYPTION_ALGORITHM=AES256 ENCRYPTION_MODE=dual
```

Reusing a Dump File

In Oracle 11g data pump export utility, the new parameter REUSE_DUMPFILES enables you to overwrite a preexisting dump file.

Following is an example:

```
expdp hr DIRECTORY=dp_dir DUMPFILE=hr.dmp TABLES=employees REUSE_DUMPFILES=y
```

Remapping Data

There is a new parameter that allows you during export or import to modify the input or output data based on your remapping scheme. The REMAP_DATA parameter specifies a remap function that takes as a source the original value of the designated column and returns a remapped value that will replace the original value in the dump file. The syntax of the using the parameter is as follows:

```
REMAP_DATA=[schema.]tablename.column_name:[schema.]pkg.function
```

Following is an example of how to use it.

```
-- function(s) used by the remap
Create or replace package remap_pckg
as
 function modify_char ( p_in_data varchar2) return varchar2;
end;
/

Create or replace package body remap_pckg
as
function modify_char (p_in_data varchar2) return varchar2
as
 v_return varchar2(30);
begin
v_return:=translate(p_in_data,'abcdefghijklmnopqrstuvwxyz','bcdefghijklmnopqrstuvwxyza');
 return v_return;
end;
end;
/

expdp hr/passwd DIRECTORY=dp_dir DUMPFILE=remap.dmp
TABLES=hr.employees REMAP_DATA=hr.employees.last_name:hr.remap_pckg.modifychar
```

Remap function should not issue any COMMIT or ROLLBACK statements.

Renaming Tables During Export or Import

In Oracle 11g, the Data Pump allows you to rename a table during the import process with the REMAP_TABLE parameter. Syntax of this parameter is as follows:

```
REMAP_TABLE=[schema.]old_tablename[.partition]:new_tablename
```

Following are examples of using this parameter

```
impdp dumpfile=dp_dir:docs.dmp REMAP_TABLE=hr.docs:docs2 userid=hr/password
impdp dumpfile=dp_dir:docs.dmp REMAP_TABLE=hr.docs.part1:docs3 userid=hr/password
```

Note Tables will not be remapped if they already exist even if the TABLE_EXISTS_ACTION is set to TRUNCATE or APPEND.

Data Pump and Partitioned Tables

The new parameter PARTITION_OPTIONS specifies how table partitions should be created during an import operation. This parameter takes one of the following values:

NONE	creates tables as they existed on the system from which the export operation was performed.

DEPARTITION promotes each partition or subpartition to a new individual table. The default name of the new table will be the concatenation of the table and partition name or the table and subpartition name, as appropriate.

MERGE combines all partitions and subpartitions into one table.

The default is DEPARTITION when partition names are specified on the TABLES parameter and TRANPORTABLE=ALWAYS is set (whether on the import operation or the export). Otherwise, the default is NONE.

Restrictions

- If the export operation that created the dump file was performed with the TRANSPORTABLE method and if a partition or subpartition was specified, then the import operation must use the DEPARTITION option.
- If the export operation that created the dump file was performed with the transportable method, then the import operation cannot use the MERGE option.
- If there are any grants on objects being departitioned, an error message is generated and the objects are not loaded.

Following are examples of using this new parameter:

```
-- merge all the partitions in sh.sales into one non-partitioned table in scott schema.
impdp system/mypassword TABLES=sh.sales PARTITION_OPTIONS=merge DIRECTORY=dp_dir
DUMPFILE=sales.dmp REMAP_SCHEMA=sh:scott

-- unload the P_Q1Y08 partition of the sh.sales table with the TRANSPORTABLE=ALWAYS
expdp TABLES=sh.sales:p_q1y08 USERID=sh/sh DIRECTORY=dp_dir DUMPFILE=p_q1y08.dmp \
LOGFILE=logq1y08.log REUSE_DUMPFILES=Y TRANSPORTABLE=always

-- import the P_Q1Y08 partition of the sh.sales table
impdp USERID="'/ as sysdba'" PARTITION_OPTIONS=departition DUMPFILE=dp_dir:p_q1y08.dmp \
LOGFILE=logdir:logq1y08.log TRANSPORT_DATAFILES='+FRA/dba11g1/kb2.dbf'
```

Ignoring Nondeferred Constraints

In Oracle 11g, setting the DATA_OPTIONS parameter to SKIP_CONSTRAINT_ERRORS will cause the import program to skip errors generated by the nondeferred database constraints. In the case of deferred constraints, imports will always be rolled back.

```
impdp Robert/robert DIRECTORY=data_pump_dir DUMPFILE=remap.dmp tables=ROBERT.NAMES
data_options=SKIP_CONSTRAINT_ERRORS
```

External Tables Based on Data Pump Driver

In Oracle 11g, creating external tables using ORACLE_DATAPUMP driver is enhanced.

You can take advantage of the new COMPRESSION and ENCRYPTION (required TDE enabled) options.

```
create table docs organization external
 ( type ORACLE_DATAPUMP
   default directory dbadir
   access parameters
   ( logfile logdir:docs COMPRESSION ENABLED) -- you can add ENCRYPTION ENABLED
    location ('docs.dmp') )
 as select * from documents
```

Another improvement, a row error will not cause a table load to abort. In Oracle Database 10g, you needed the REJECT LIMIT clause to do so.

Enhancement in the Transportable Parameter

With Oracle 11g, when you use the TRANSPORTABLE parameter in the data pump import and export utilities, only the metadata associated with specific tables, partitions, or subpartitions will be extracted, rather than all metadata. Other than that, the behavior of the parameter remains the same as in the previous version.

Automatic Storage Management (ASM)

SYSASM Privilege and OSASM Group

This feature introduces a new SYSASM privilege that is specifically intended for performing ASM administration tasks. Using the SYSASM privilege instead of the SYSDBA privilege provides a clearer division of responsibility between ASM administration and database administration.

Following are code examples illustrating how to use this privilege:

```
-- grant the privilege
GRANT SYSASM TO firas;

-- check the granted privilege
SELECT * FROM V$PWFILE_USERS;

-- ASM management commands are available to Adam
CONNECT firas/his_password
ALTER DISKGROUP dg1 DISMOUNT;
ALTER DISKGROUP dg2 MOUNT;
.. and so on.
```

Be aware that users with SYSOPER privilege have some ASM privileges. Following table shows available and restricted ASM privilege for users with SYSOPER privilege:

Avaiable ASM Privilege	*Restricted ASM Privilege*
STARTUP AND SHUTDOWN	CREATE DISKGROUP / DISK
ALTER DISKGROUP MOUNT	DROP DISKGROUPS / DISKS
ALTER DISKGROUP DISMOUNT	ALTER DISKGROUP / DISK RESIZE
ALTER DISKGROUP ONLINE DISK	
ALTER DISKGROUP OFFLINE DISK	
ALTER DISKGROUP REBALANCE	
ALTER DISKGROUP CHECK	

OSASM is a new operating system group that is used exclusively for ASM. Members of the OSASM group can connect as SYSASM using operating system authentication and have full access to ASM.

Upgrading ASM using DBUA

Database Update Assistant (DBUA) can be used to upgrade the ASM instance from Oracle Database 10g to Oracle Database 11g. To do that, perform the following steps:

1. Change the directory to the new $ORACLE_HOME/bin and launch DBUA.

```
cd $ORACLE_HOME/bin
$ ./dbua
```

2. In the Upgrades Operations page, click the **Upgrade Automatic Storage Management Instance** radio button, and click the **Next** button.
3. In the summary page, confirm the source and target information, then click on **Finish** button.
4. When the operation finishes, a successful message should be displayed. The DBUA utility logs its operation in the $ORACLE_BASE/cfgtoollogs/dbua/logs/ASMUpgrade.log.

Upgrading ASM Manually

Following are the steps you follow to upgrade an existing Oracle 10g ASM to 11g:

1. Install the Oracle Database 11g software to a new ORACLE_HOME directory.
2. Update the /etc/oratab or /var/opt/oracle/oratab file with the new ASM ORACLE_HOME location.
3. Copy the ASM initialization file from the old ORACLE_HOME to the new one.
4. Edit any directory-based parameters (such as diag and dump) in the ASM initialization file as required.

5. If you are upgrading a non-RAC ASM instance, you should reconfigure the Oracle CSS using the new ORACLE_HOME. You can do this by executing the `localconfig` command from the new home. Once the CSS configuration is complete, you need to change your ORACLE_HOME to the new Oracle version 11.1 ORACLE_HOME and start the ASM instance.

```
cd $ORACLE_HOME/bin
 # ./localconfig reset
```

6. If you are upgrading a ASM instance in a RAC environments, you can modify the new ASM home within the OCR using the `srvctl` utility as follows:

```
srvctl modify asm -n racnode1 -i +ASM1 -o /apps/oracle/product/11.1.0/asm -p
init+ASM1.ora
```

7. Grant the `SYSASM` role to the SYS

```
GRANT SYSASM to sys;
```

8. If you have obsolete initialization parameters, you can address them now. To get a listing of all the obsolete initialization parameters, refer to the ASM alert log file.

ASM Restricted Mode

In Oracle 11g, you can start the ASM instance in restricted mode. When in restricted mode, databases will not be permitted to access the ASM instance. Also, individual diskgroup can be set in restricted mode.

```
-- in the ASM instance level
SQL>STARTUP RESTRICT;

-- in the diskgroup level
SQL>ALTER DISKGROUP DATA MOUNT RESTRICTED;

-- check status of diskgroups
SQL>SELECT NAME,STATE FROM V$ASM_DISKGROUP;
```

`ORA-15236` error will be returned, if the database tries to access an ASM instance started in restricted mode.

Diskgroup Attributes

Oracle Database 11g introduces a new concept called ASM attributes at the diskgroup level. The attributes for the diskgroup can be established at create diskgroup time or can be modified using the `ALTER DISKGROUP` command later.

Following are the attributes you can set:

- Allocation unit (AU) sizes.
- The `compatible.rdbms` attribute.
- The `compatible.asm` attribute.
- `disk_repair_time` in units of minute (M) or hour (H) and is set by the `ALTER DISKGROUP` command
- The redundancy attribute for a specific template.
- The stripping attribute for a specific template.

All of the diskgroup attributes can be queried from the `V$ASM_ATTRIBUTE` view.

Consider the following examples:

```
CREATE DISKGROUP data
 disk '/dev/raw/raw1',
 ...
 attribute 'au_size' = '16M', 'compatible.asm' = '11.1';

ALTER DISKGROUP data SET ATTRIBUTE 'compatible.asm' = '11.1.0.0.0';

select NAME, VALUE from V$ASM_ATTRIBUTE where GROUP_NUMBER=1;
```

Variable AU Sizes

The default size of Allocation Unit (AU) is 1 MB which is sufficient for most regular databases. However, when you have databases with TB sizes, you will have enormous number of AUs. With Oracle 11g, AU size can be specified at diskgroup creation time to 1, 2, 4, 8, 16, 32, or 64MB in size. You can check the AU size through the following query:

```
select NAME, ALLOCATION_UNIT_SIZE from V$ASM_DISKGROUP;
```

Compatibility Settings

Compatibility in ASM is controlled in three ways, as shown below:

COMPATIBLE initialization parameter	The compatible initialization parameter can be set for either ASM or the database instance. It takes one of the following values: 10.1, 10.2, or 11.1. Setting the initialization parameter to a lesser value than the software release will exclude availability of the new features introduced in the new release.
RDBMS Compatibility	This is a diskgroup-level compatibility and is specified by setting the COMPATIBLE.RDBMS attribute. This attribute determines the minimum COMPATIBLE database initialization parameter setting for any database instance that uses the disk group. Its default value is 10.1.
ASM Compatibility	This is a diskgroup-level compatibility and is specified by setting the COMPATIBLE.ASM attribute. It determines the minimum software version for an ASM instance that uses the disk group.

If you assign any of the compatibility setting to a higher value, you cannot later reverse it to a lower value.

Following are some queries to obtain information about the compatibility settings:

```
-- diskgroup compatibility setting
select NAME, BLOCK_SIZE, ALLOCATION_UNIT_SIZE AU_SIZE, STATE,
COMPATIBILITY ASM_COMP, DATABASE_COMPATIBILITY DB_COMP
from V$ASM_DISKGROUP;

-- Compatibility of the database clients that use the ASM
select DB_NAME, STATUS,SOFTWARE_VERSION,COMPATIBLE_VERSION from V$ASM_CLIENT;
```

ASM Fast Mirror Resync

Any problems that make a failure group temporarily unavailable are considered *transient failures* that can be recovered by the ASM *fast mirror resync* feature. Disk path malfunctions; such as cable failures, host bus adapter failures, controller failures, or disk power supply interruptions; can cause transient failures.

ASM fast resync keeps track of pending changes to extents on an OFFLINE disk during an outage. The extents are resynced when the disk is brought back online.

Following are the steps to enable and handle this feature:

```
-- diskgroup compatibility must be set to 11.1
ALTER DISKGROUP dg1 SET ATTRIBUTE 'compatible.asm' = '11.1';
ALTER DISKGROUP dg1 SET ATTRIBUTE 'compatible.rdbms'='11.1';

-- specify the duration of the disk_repair_time (default is 3.6 hour)
ALTER DISKGROUP dg1 SET ATTRIBUTE 'disk_repair_time' = '5H'; -- in hours
ALTER DISKGROUP dg1 SET ATTRIBUTE 'disk_repair_time' = '40M'; -- minutes

-- verify the attribute settings
select NAME, VALUE from V$ASM_ATTRIBUTE;

-- if you get an offline disk because of a transient failure, you can see the
--  remaining time left in SECONDS before ASM drops an offline disk
select NAME, HEADER_STATUS, MOUNT_STATUS, MODE_STATUS, STATE, REPAIR_TIMER/60 from
V$ASM_DISK WHERE GROUP_NUMBER=1;

-- while the fix is in progress, if you want to reset the elapsed time, just take
--  the disk(s) offline
ALTER DISKGROUP dg1 OFFLINE DISK d3_0001;
ALTER DISKGROUP dg1 OFFLINE DISKS IN FAILGROUP f2;

-- you can also make a disk offline with a repair time different from its
--  disk_repair_time attribute
ALTER DISKGROUP dg1 OFFLINE DISK d3_0001 DROP AFTER 50m;

-- disks in a failure group (f2) can also be taken offline
ALTER DISKGROUP dg1 OFFLINE DISKS IN FAILGROUP f2 DROP AFTER 5m;
```

```
-- if the disk needs to be dropped immediately and before the repair time has expired
--  Note: ALTER DISKGROUP DROP DISK will not work
ALTER DISKGROUP dg1 OFFLINE DISK D3_0001 DROP AFTER 0m;

-- after the disk(s) are fixed, you can bring them online
ALTER DISKGROUP dg1 ONLINE ALL;
ALTER DISKGROUP dg1 ONLINE DISK d3_0001;
```

Checking Diskgroup

Starting from Oracle Database 11g, you can validate the internal consistency of ASM diskgroup metadata using the `ALTER DISKGROUP ... CHECK` command. Summary of errors is logged in the ASM alert log file.

```
-- check specific diskgroup with automatic repair
SQL>ALTER DISKGROUP data CHECK;

-- disable automatic repair
SQL>ALTER DISKGROUP data CHECK NOREPAIR;
```

asmcmd Utility Commands

Oracle 11g introduces new commands in the asmcmd utility and it also provides backward compatibility with Oracle Database 10g ASM instances. Following are summary of some of them:

Command Syntax	Description and Examples
`lsct [-gH] [group]`	Lists information about current ASM clients. `>lsct dgroup1`
`lsdg [-gcH] [group]`	lists all diskgroups and their attributes. `>lsdg dgroup2`

`lsdsk [-ksptagcHI] [-d diskg_roup_name] [pattern]`	lists the disks that are visible to ASM by scanning the disk headers of the disks seen by the value of the `ASM_DISKSTRING`
	`>lsdsk -k -d DATA *_001`
	`>lsdsk -s -d DATA *_001`
	`>lsdsk -t -d DATA *_001`
	`>lsdsk -c -t -d DATA *_001`
	`>lsdsk -g -t -d DATA *_001`

`cp [-ifr] [connect_string:]src_fname [connect_string:]tgt_fname`
`cp [-ifr] [connect_string:]src_fnameN, src_fnameN+1 ... [connect_string:]tgt_directory`
 	
The `connect_string` is in the form of:
`user_name@host_name[.port_number].SID`
-i interactive
-f force overwrite (aliases cannot be overwritten)
-r recursive
	Enables you to copy files between ASM disk groups on local instances to and from remote instances.
	`>cp +dg1/vdb.ctf1 /backups/vdb.ctf1`
	`>cp /home/oracle/encrypted.dmp +dg1`
	`>cp vdb.ctf1 /tmp`
	`# the target ASM instance must be registered with the LISTENER`
	`>cp +DATA/DBA11g/DATAFILE/DOCS_D1.289.631914611 sys@rac1.+ASM:+DATA/DBA11g1/datafile/xxx`

`remap`	Repairs a range of physical blocks on disk (only blocks exhibiting read disk I/O errors are repaired) excluding those with corrupted contents.
	Internally, it reads the blocks from a good copy of an ASM mirror and rewrites them to an alternate location on disk, if the blocks on the original location cannot be properly read.
	`remap <disk group name> <disk name> <block range>`
	`> remap DISK_GRP1 DATA_0001 5000-5999`

Backing up and Restoring Diskgroup Metadata

The `md_backup` command captures information about ASM disks, diskgroup and failure group configurations, and template and alias directory structures, and stores them in a user-designated backup text file. Following is the basic syntax of the command:

```
md_backup [-b <backup_file_path> ] [-g diskgroup_name [-g diskgroup_name …]]
```

Following is an example of using the command:

```
md_backup –b /tmp/asm_backup.mdb -g dg1 –g dg2
```

If the backup file already exists, you should remove it before issuing the command.

If you issue the `md_backup` command without any option, it creates a file named as `ambr_backup_intermediate_file` which contains the metadata information of all the mounted diskgroups.

The `md_restore` command reads the backup file and restores a disk group. You can set its options to build a script file that contains the SQL statements required to rebuild the ASM components from the backup file. Following is the syntax of the command and description of its switches:

```
md_restore -b <backup_file> [-li]
  [-t (full)|nodg|newdg] [-f <sql_script_file>]
  [-g '<diskgroup_name>,<diskgroup_name>,...']
  [-o '<old_diskgroup_name>:<new_diskgroup_name>,...']
```

`-t` type of restore.

`full` tag specifies that all the diskgroups should be re-created using the same configuration from the

MDB backup file.

nodg restore metadata only and skip the diskgroup creation.

newdg create disk group with a different name and restore metadata; -o is required. This tag is used to allow the user to change diskgroup name.

`-f` write SQL commands to <sql_script_file> instead of executing them.

`-o` override option is used only with the newdg option to remap the diskgroup name, disk name, paths, and failure groups.

`-i` ignore errors. By default, the command aborts when it encounters an error.

`-l` log all messages to a log file.

Following are examples of using the command:

```
# To perform a restore of the dg1 diskgroup from the MDB backup file, use this:
md_restore –b /tmp/backupfile –t full –g dg1 -i

# To just restore the metadata for the dg1 diskgroup (the diskgroup already exists).
md_restore –b /tmp/backupfile –t nodg –g dg1 –i

# To create a different diskgroup name:
md_restore –b /tmp/backupfile –t newdg -o "DGNAME=dg1:dg3" –i

# To apply the override options as specified in the dg_over.txt file and restore
#  from the backup file:
md_restore –b /tmp/backupfile –t newdg –of /tmp/dg_override.txt –i
```

Note that md_backup is a backup of the metadata of the ASM instance. The data is being backed up by RMAN. After the diskgroup is created, along with all the directories, you can restore the RMAN backup to the diskgroup.

Bad Block Recovery

If ASM cannot read a physical block from a disk, it considers that the block has IO error. In this case, ASM will automatically read a mirrored block and write a relocated copy to produce successful copy. However, you can manually repair blocks that have read disk I/O errors using the remap command. Following is the syntax of the command:

```
remap <diskgroup name> <disk name> <block range>
```

Fast Rebalance

In Oracle 11g, fast rebalance eliminates ASM messaging among the ASM instances in a RAC configuration when a disk is added to a disk group.

This feature is enabled by using the STARTUP RESTRICT or ALTER DISKGROUP … MOUNT RESTRICT commands. When the diskgroup is in restricted mode, databases are not allowed to access the datafiles in it.

The FORCE option with Drop Diskgroup Command

If a disk is destroyed beyond repair, you want to drop it. But because the disk is practically damaged, you cannot mount it and thus you cannot issue the DROP DISKGROUP command against it. In such a condition, Oracle 11g provides the FORCE INCLUDING CONTENTS option to drop the diskgroup even if it is not mounted. Following is an example:

```
SQL>DROP DISKGROUP dg7 FORCE INCLUDING CONTENTS;
```

Miscellaneous ASM New Features

Oracle 11g introduces some other new features in its ASM related to the clustered ASM architecture. Discussing those features is beyond the scope of this document. For details about them, you can refer to the new documentation "*Oracle Database Storage Administrator's Guide 11g*". Following is a brief description about those features:

Preferred Read Failure Groups

You can configure ASM to read from a secondary extent if that extent is closer to the node instead of ASM reading from the primary copy which might be farther from the node. This configuration is described as Preferred *Read Failure Group*. Using preferred read failure groups is most useful in extended clusters.

ASM Rolling Upgrades

ASM rolling upgrades enable you to independently upgrade or patch clustered ASM nodes without affecting database availability, thus providing greater uptime. Rolling upgrade means that all of the features of a clustered ASM environment function when one or more of the nodes in the cluster use different software versions.

Oracle Direct NFS

Oracle Database 11g kernel has built-in support for the network file system (NFS) without relying on OS support for NFS. If Oracle's Direct NFS is not able to open the NFS server, Oracle will utilize the operating system's NFS client as specified in /etc/fstab and post an error message in the alert log. This feature has the following advantages:

- You can achieve better IO performance, more efficient system resource utilization, and lower operating costs in NAS environments.
- You can implement Network Interface Card (NIC) bonding without the need of the expensive advanced Ethernet switches. It is not either necessary to have homogeneous network cards.

Using ASM versus Direct NFS is still controversial subject.

Note In Oracle 11g ASM, you can create an ASM disk group using NFS files.

PL/SQL New Features

CONTINUE Statement

In Oracle 11g PL/SQL, you can exit the current iteration of a loop using the new statements: CONTINUE or CONTINUE-WHEN. When a CONTINUE statement is encountered, the current iteration of the loop completes immediately and control passes to the next iteration of the loop, as in the following example:

```
DECLARE
 x NUMBER := 0;
BEGIN
 LOOP -- After CONTINUE statement, control resumes here
  DBMS_OUTPUT.PUT_LINE ('Inside loop: x = ' || TO_CHAR(x));
  x := x + 1;
  IF x < 3 THEN
   CONTINUE;
  END IF;
  DBMS_OUTPUT.PUT_LINE ('Inside loop, after CONTINUE: x = ' || TO_CHAR(x));
  EXIT WHEN x = 5;
 END LOOP;
 DBMS_OUTPUT.PUT_LINE (' After loop: x = ' || TO_CHAR(x));
END;
```

When a CONTINUE-WHEN statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the current iteration of the loop completes and control passes to the next iteration. The previous example can be altered as in the following code:

```
DECLARE
 x NUMBER := 0;
BEGIN
 LOOP -- After CONTINUE statement, control resumes here
  DBMS_OUTPUT.PUT_LINE ('Inside loop: x = ' || TO_CHAR(x));
  x := x + 1;
  CONTINUE WHEN x < 3;
  DBMS_OUTPUT.PUT_LINE ('Inside loop, after CONTINUE: x = ' || TO_CHAR(x));
  EXIT WHEN x = 5;
 END LOOP;
 DBMS_OUTPUT.PUT_LINE (' After loop: x = ' || TO_CHAR(x));
END;
```

Sequences Without Selecting from Dual

It is no longer necessary to execute a SELECT .. FROM DUAL statement in PL/SQL when you want to assign a NEXTVAL of a sequence to a variable. Following is an example:

```
DECLARE
 new_Val NUMBER;
BEGIN
 new_Val := my_sequence.nextval;
 ...
END;
```

SIMPLE_INTEGER

SIMPLE_INTEGER is a new data type introduced in Oracle 11g PL/SQL. It has the same range as of PLS_INTEGER (–2,147,483,648 to 2,147,483,647) but with the following differences:

PLS_INTEGER	SIMPLE_INTEGER
cannot be null	accepts NULL
less performance in native compiled PL/SQL	higher performance in native compiled PL/SQL
when it takes values out of its range, error returned	when it takes values out of its range, the value wrap from smallest to largest and from largest to smallest

Named and Mixed Notation in PL/SQL Subprogram Invocations

The use of NAME=>value is now supported in PL/SQL function calls that are contained in expressions in SQL statements. So for example, all of the following SELECT statements are now valid:

```
SELECT EMP_PKG.GET_INFO(EMPLOYEE_ID,DEPARTMENT_ID) FROM EMPLOYEES;

SELECT EMP_PKG.GET_INFO(EMPLOYEE_ID,P_DEPT_ID=>DEPARTMENT_ID) FROM EMPLOYEES;

SELECT EMP_PKG.GET_INFO(P_EMP_ID=>EMPLOYEE_ID,P_DEPT_ID=> DEPARTMENT_ID) FROM EMPLOYEES;
```

Subprogram Inlining

Subprogram inlining replaces a subprogram call (to a subprogram in the same program unit) with a copy of the called subprogram. This will lead to better performance in almost all the cases.

The PLSQL_OPTIMIZE_LEVEL initialization parameter specifies the level of optimization used to compile the PL/SQL library unit and is used to switch on and off subprogram inlining in a PL/SQL code. The parameter accepts the following values:

1 No PL/SQL compilation optimizations are done.

2 PL/SQL will rearrange code for performance but will not automatically inline subprograms. It will inline subprogram calls the developer has flagged with the pragma INLINE directive.

3 In addition to the level 2 optimizations, the PL/SQL compiler will automatically inline subprograms where performance gains are predicted, as well as place a high priority on inlining programmer flagged calls.

The pragma INLINE compiler directive specifies that a subprogram call is, or is not, to be inlined. It must appear immediately before the subprogram call. It takes the following two arguments:

identifier — The name of the subprogram.

Mode — Either YES or NO.

For NO, no inlining will occur for the subprogram.

If YES and PLSQL_OPTIMIZE_LEVEL=2, the subprogram will be inlined.

If YES and PLSQL_OPTIMIZE_LEVEL=3, the optimizer will place a high priority on inlining the subprogram. The optimizer may find a better optimization that does not need inlining.

Following are illustrated examples:

```
-- example 1
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=2;
DECLARE
 PROCEDURE PROC1(p_idata IN VARCHAR2) IS
BEGIN
 ....
 dbms_output.put_line(dbms_utility.format_call_stack());
END;
BEGIN
 -- inlining is NOT used
 PROC1('test');
END;

-- example 2
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=3;
DECLARE
 PROCEDURE PROC1(p_idata IN VARCHAR2) IS
BEGIN
 ....
 dbms_output.put_line(dbms_utility.format_call_stack());
END;
BEGIN
 -- inlining is NOT used
 pragma INLINE(PROC1,'YES');
 PROC1('test');
 -- inlining is used
 pragma INLINE(PROC1,'YES');
 PROC1('test');
END;

-- example 3
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=3;
```

```
DECLARE
 PROCEDURE PROC1(p_idata IN VARCHAR2) IS
BEGIN
 ....
 dbms_output.put_line(dbms_utility.format_call_stack());
END;
BEGIN
 -- inlining may occur, if performance gain is predicted
 PROC1('test');

 -- inlining is used
 pragma INLINE(PROC1,'YES');
 PROC1('test');
END;
```

PL/SQL Function Result Cache

Fundamentals of using this new feature are described in the section "PL/SQL Function Result Cache". For further details, refer to the documentation "Oracle Database PL/SQL Language Reference 11g".

Ordering of Triggers

In Oracle 11g, you can control the order on which the triggers on a table are fired using the new clause FOLLOWS. Following is an example of using this new clause.

```
CREATE TRIGGER Trigger3
BEFORE INSERT ON EMPLOYEES
FOLLOWS trigger2,trigger1
WHEN ...
```

The triggers you specify with FOLLOWS clause must already exist, be defined on the same table as the trigger being created, and have been successfully compiled. They need not be enabled.

Creating Triggers As ENABLED or DISABLED

Two new keywords, ENABLED and DISABLED, have been added to the trigger creation statement. They appear immediately before the optional WHEN clause and after the optional FOLLOWS clause.

```
CREATE TRIGGER EMP_CHECK_SAL
BEFORE INSERT ON EMPLOYEES
DISABLED
WHEN ...
```

Compound Triggers Type

Oracle 11g introduces a new type of triggers called compound triggers. A compound trigger implements all of the timing-point logic (before statement, before each row, after each row, after statement) within its body, and all of those sections share the same set of variables declared in the trigger's common declaration section. The compound trigger makes it easier if you want triggers of various timing points to share common data.

The following example illustrates the general syntax of creating a compound trigger:

```
CREATE TRIGGER trigger_name
FOR UPDATE OF salary ON employees
COMPOUND TRIGGER
-- Declaration Section
-- Variables declared here have firing-statement duration.
 threshold CONSTANT SIMPLE_INTEGER := 200;

BEFORE STATEMENT IS
BEGIN
 ...
END BEFORE STATEMENT;

AFTER STATEMENT IS
BEGIN
 ...

END AFTER STATEMENT;

BEFORE EACH ROW IS
BEGIN
 ...
END BEFORE EACH ROW;

AFTER EACH ROW IS
BEGIN
 ...
```

```
END AFTER EACH ROW;
END trigger_name;
```

When using compound triggers, consider the following:

- A compound trigger defined on a view has an `INSTEAD OF EACH ROW` timing-point section, and no other timing-point section.
- Timing-point sections must appear in the order shown in the example above.
- Any section can include the functions `Inserting`, `Updating`, `Deleting`, and `Applying`.

Following are some of the restrictions in the compound triggers:

- A compound trigger must be defined on either a table or a view.
- An exception that occurs in one section must be handled in that section. It cannot transfer control to another section.
- Only the `BEFORE EACH ROW` section can change the value of `:NEW`.

Two common uses of the compound triggers are:

- To accumulate rows inserted into a table by a statement to bulk-insert them into another table. The target here is to have some performance gain.
- To avoid the mutating-table error (ORA-04091).

Examples of implementing those two targets are copied from Oracle documentation and pasted in the following examples:

```
-- Compound Trigger Records Changes to One Table in Another Table
CREATE TABLE employee_salaries (
 employee_id NUMBER NOT NULL,
 change_date DATE NOT NULL,
 salary NUMBER(8,2) NOT NULL,
 CONSTRAINT pk_employee_salaries PRIMARY KEY (employee_id, change_date),
 CONSTRAINT fk_employee_salaries FOREIGN KEY (employee_id)
 REFERENCES employees (employee_id) ON DELETE CASCADE)
/

CREATE OR REPLACE TRIGGER maintain_employee_salaries
FOR UPDATE OF salary ON employees
COMPOUND TRIGGER
-- Declaration Section:
-- Choose small threshhold value to show how example works:
 threshhold CONSTANT SIMPLE_INTEGER := 7;
 TYPE salaries_t IS TABLE OF employee_salaries%ROWTYPE INDEX BY SIMPLE_INTEGER;
 salaries salaries_t;
 idx SIMPLE_INTEGER := 0;

PROCEDURE flush_array IS
 n CONSTANT SIMPLE_INTEGER := salaries.count();
BEGIN
 FORALL j IN 1..n
  INSERT INTO employee_salaries VALUES salaries(j);
  salaries.delete();
  idx := 0;
 DBMS_OUTPUT.PUT_LINE('Flushed ' || n || ' rows');
END flush_array;

-- AFTER EACH ROW Section:
AFTER EACH ROW IS
BEGIN
 idx := idx + 1;
 salaries(idx).employee_id := :NEW.employee_id;
 salaries(idx).change_date := SYSDATE();
 salaries(idx).salary := :NEW.salary;
 IF idx >= threshhold THEN
  flush_array();
 END IF;
END AFTER EACH ROW;

-- AFTER STATEMENT Section:
AFTER STATEMENT IS
BEGIN
```

```
 flush_array();
END AFTER STATEMENT;
END maintain_employee_salaries;
```

The following example implements a business rule that states that an employee's salary increase must not exceed 10% of the average salary for the employee's department.

```
CREATE OR REPLACE TRIGGER check_Employee_Salary_Raise
 FOR UPDATE OF Salary ON Employees
COMPOUND TRIGGER
 Ten_Percent CONSTANT NUMBER := 0.1;
 TYPE Salaries_t IS TABLE OF Employees.Salary%TYPE;
 Avg_Salaries Salaries_t;
 TYPE Department_IDs_t IS TABLE OF Employees.Department_ID%TYPE;
 Department_IDs Department_IDs_t;
 TYPE Department_Salaries_t IS TABLE OF Employees.Salary%TYPE
 INDEX BY VARCHAR2(80);
 Department_Avg_Salaries Department_Salaries_t;

BEFORE STATEMENT IS
BEGIN
 SELECT AVG(e.Salary), NVL(e.Department_ID, -1)
 BULK COLLECT INTO Avg_Salaries, Department_IDs
 FROM Employees e
 GROUP BY e.Department_ID;
 FOR j IN 1..Department_IDs.COUNT() LOOP
  Department_Avg_Salaries(Department_IDs(j)) := Avg_Salaries(j);
 END LOOP;
END BEFORE STATEMENT;

AFTER EACH ROW IS
BEGIN
 IF :NEW.Salary - :Old.Salary >
  Ten_Percent*Department_Avg_Salaries(:NEW.Department_ID) THEN
  Raise_Application_Error(-20000, 'Raise too big');
 END IF;
END AFTER EACH ROW;
END check_Employee_Salary_Raise;
```

Converting between Dynamic Cursor and REF CURSOR

In Oracle Database 11g, the supplied package `DBMS_SQL` has a new function, `TO_REFCURSOR`, which converts the `DBMS_SQL` dynamic cursor to a ref cursor. Here is an example of such a conversion:

```
CREATE OR REPLACE PROCEDURE list_trans_by_store ( p_store_id NUMBER ) IS
     TYPE num_tab IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
     TYPE type_refcur IS REF CURSOR;
     c_ref_trans_cur TYPE_REFCUR;
     c_trans_cur     NUMBER;
     trans_id        NUM_TAB;
     trans_amt       NUM_TAB;
     ret             INTEGER;
     l_stmt          CLOB;
  begin
     c_trans_cur := Dbms_Sql.Open_Cursor;
     l_stmt :=
         'select trans_id, trans_amt from trans where store_id = :store_id';
     DBMS_SQL.PARSE(c_trans_cur, l_stmt, dbms_sql.native);
     DBMS_SQL.BIND_VARIABLE(c_trans_cur, 'store_id', p_store_id);
     ret := DBMS_SQL.EXECUTE(c_trans_cur);
     c_ref_trans_cur := DBMS_SQL.TO_REFCURSOR(c_trans_cur);
     FETCH c_ref_trans_cur BULK COLLECT into trans_id, trans_amt;
     FOR ctr in 1 .. trans_id.COUNT LOOP
         DBMS_OUTPUT.PUT_LINE(trans_id(ctr) || ' ' || trans_amt(ctr));
     END LOOP;
     CLOSE c_ref_trans_cur;
END;
```

Suppose you want to write a generic procedure where you do not know the column list in the select clause at compile time. This is where the native dynamic SQL comes in handy; you can define a ref cursor for that. Now, suppose you do not know the bind variable as well, for which DBMS_SQL is more appropriate. How can you accomplish this complex requirement with minimal code? Simple: Just start with DBMS_SQL for the bind part and then convert it to ref cursor later for the other part.

Similarly, if you want to convert a Native Dynamic SQL to REF CURSOR, you will need to call another function, TO_CURSOR_NUMBER:

```
cur_handle := DBMS_SQL.TO_CURSOR_NUMBER (c_ref_cur);
```

The ref cursor specified by the variable c_ref_cur must be opened prior to this call. After this call, the life of the ref cursor is over; it can be manipulated only as a DBMS_SQL cursor.

Suppose you know the binds at compile time but not the select list; you start with native dynamic sql with a ref cursor and later change it to DBMS_SQL to describe and fetch the columns from the cursor.

Data Warehousing

SecureFiles

Oracle Database 11g completely reengineers LOB storage to what they call *SecureFiles*. The previous BLOB data types are referred to as *BasicFiles*. The primary focus of the SecureFiles feature is to do the following:

- Improve performance
- Reduce space consumption
- Enhance security

Creating SecureFile LOB Columns

Use the clause `STORE AS SECUREFILE` or `STORE AS BASICFILE` to create a SecureFile or BasicFile columns in a table. This is, however, controlled by the initialization parameter `DB_SECUREFILE`. Using the parameter will be discussed in the next sub-section. Following is an example of creating a table with a SecureFile LOB column:

```
CREATE TABLE employees
( employee_id NUMBER NOT NULL PRIMARY KEY,
  name        VARCHAR2(255) NOT NULL,
  ...
  cv_content  BLOB )
TABLESPACE tools
LOB (cv_content) STORE AS SECUREFILE (
 KEEP_DUPLICATES  -- or DEDUPLICATE
 TABLESPACE tools ENABLE STORAGE IN ROW CHUNK 8192 PCTVERSION 10 NOCACHE LOGGING);
```

`DEDUPLICATE` option makes Oracle stores a single copy of a file when storing many identical files. Thus, space is significantly saved. This option can be altered for an existing SecureFile:

```
ALTER TABLE employees MODIFY LOB(cv_content) (DEDUPLICATE);
```

Be aware that the command shown in the example above will make Oracle read the values in the column and remove duplicates.

You can enable encryption for SecureFiles LOBs using the TDE. Following is an example:

```
ALTER TABLE employees MODIFY (cv_content ENCRYPT USING 'AES192');
```

Available encryption algorithms are: AES192 (the default), 3DES168, AES128, and AES256. Encryption for SecureFiles must use SALT. You can query the `DBA_ENCRYPTED_COLUMNS` view to confirm that LOB columns are encrypted.

Encryption can be disabled using the following command:

```
ALTER TABLE employees MODIFY (blob_content DECRYPT)
```

Compression in SecureFiles can be enabled by using the `COMPRESS` clause in the `CREATE TABLE` or `ALTER TABLE` commands as in the following example:

```
.. LOB (cv_content) STORE AS SECUREFILE (COMPRESS HIGH) -- or medium
```

Note that Oracle will not be able to significantly reduce the size of a SecureFile, if you store an already compressed file into it.

Compression can be disabled using the `NOCOPRESS` clause. Remember that compression and decompression will take effect immediately when you enable or disable compression.

Note Beside `ALTER TABLE` command, deduplication, encryption, and compression settings of SecureFile LOBs can be altered using `DBMS_LOB.SETOPTIONS`.

The `DB_SECUREFILE` Initialization Parameter

Creating SecureFiles in the database is controlled by the new dynamic initialization parameter `DB_SECUREFILE`. This parameter specifies whether or not to treat LOB files as SecureFiles. Following are its acceptable values:

`NEVER`	Any LOBs that are specified as SecureFiles are created as BasicFile LOBs. All SecureFile-specific storage options and features (for example, compress, encrypt, deduplicate) will cause an exception. The BasicFile LOB defaults will be used for storage options not specified.
`PERMITTED`	(the default) LOBs are allowed to be created as SecureFiles.

ALWAYS — All LOBs created in the system are created as SecureFile LOBs. Any BasicFile LOB storage options are ignored. The SecureFile defaults will be used for all storage options not specified.

IGNORE — The SECUREFILE keyword and all SecureFile options are ignored.

If the COMPATIBLE initialization parameter is not set to 11.1 or higher, then LOBs are not treated as SecureFiles. Also, SecureFile LOB must be created in a Automatic Segment Space Managed (ASSM) tablespace.

Following is an example of modifying this parameter:

```
ALTER SYSTEM SET DB_SECUREFILE='ALWAYS';
```

Space used by SecureFile

The procedure DBMS_SPACE.SPACE_USAGE can be used to obtain the space usage information of data blocks in a segment. There is a form of this procedure that specifically returns information about SECUREFILE LOB space usage. It displays the space actively used by the LOB column, freed space that has retention expired, and freed space that has retention unexpired.

Following is the syntax of that form of the procedure:

```
DBMS_SPACE.SPACE_USAGE(
segment_owner        IN VARCHAR2,
segment_name         IN VARCHAR2,
segment_type         IN VARCHAR2, possible values are: TABLE, TABLE PARTITION, TABLE SUBPARTITION
                                  INDEX, INDEX PARTITION, INDEX SUBPARTITION, CLUSTER, LOB,
                                  LOB PARTITION, LOB SUBPARTITION
segment_size_blocks  OUT NUMBER,
segment_size_bytes   OUT NUMBER,
used_blocks          OUT NUMBER,
used_bytes           OUT NUMBER,
expired_blocks       OUT NUMBER,
expired_bytes        OUT NUMBER,
unexpired_blocks     OUT NUMBER,
unexpired_bytes      OUT NUMBER,
partition_name       IN VARCHAR2 DEFAULT NULL);
```

Accessing a LOB Using SQL and PL/SQL

In most cases, you can use the same SQL semantics on a LOB column (BasicFile or SecureFile) that you would use on a VARCHAR2 column. Following are the SQL operations that are *not* supported on LOB columns:

```
SELECT DISTINCT clobCol from...
SELECT... ORDER BY clobCol
SELECT... GROUP BY clobCol
SELECT clobCol1 from tab1 UNION SELECT clobCol2 from tab2;
SELECT... WHERE tab1.clobCol = tab2.clobCol
CREATE... ON tab(clobCol)...
```

Accessing LOB columns in Oracle 11g PL/SQL is the same as in the previous version. Following are examples:

```
-- LOBs with INSERT, UPDATE, and SELECT Operations
CREATE TABLE t (id number, clob_col CLOB, blob_col BLOB);
 INSERT INTO t VALUES(1,'row1', 'aaaaa');

declare
 c_buffer VARCHAR2(100);
begin
 INSERT INTO t(id, clob_col, blob_col)
  VALUES(2, 'row2', 'FFFF'); -- blob is passed a hexadecimal number

 UPDATE t SET clob_col= 'ROW2'
  WHERE id=2;

 -- This will get the LOB column if it is up to 100 bytes, (exception otherwise)
 SELECT clob_col INTO c_buffer FROM t WHERE id=2;
end;

-- LOBs in Assignments and Parameter Passing
```

```
declare
 var_buf VARCHAR2(100);
 clob_buf CLOB;
 blob_buf BLOB;
begin
 SELECT clob_col, blob_col INTO clob_buf, blob_buf FROM t WHERE id=1;
  var_buf  := clob_buf; -- coversion from VARCHAR2 to CLOB
  clob_buf := var_buf;  -- CLOB to VARCHAR2 conversion
end;

CREATE OR REPLACE PROCEDURE FOO ( a IN OUT CLOB) IS
begin
 -- Any procedure body
 a := 'abc';
end;

CREATE OR REPLACE PROCEDURE BAR (b IN OUT VARCHAR2) IS
begin
 -- Any procedure body
 b := 'xyz';
end;
```

Online Redefinition

Online redefinition is the only recommended method for migration of BasicFile LOBs to SecureFiles. It can be done at the table or partition level.

Online Redefinition Advantages:

- No need to take the table or partition offline.
- Can be done in parallel.

Online Redefinition Disadvantages

- Additional storage equal to the entire table or partition and all LOB segments must be available.
- Global indexes need to be rebuilt.

Following is an example of how to migrate a table using Online Redefinition:

```
-- Grant privileges required for online redefinition.
GRANT EXECUTE ON DBMS_REDEFINITION TO pm;
GRANT ALTER ANY TABLE TO pm;
GRANT DROP ANY TABLE TO pm;
GRANT LOCK ANY TABLE TO pm;
GRANT CREATE ANY TABLE TO pm;
GRANT SELECT ANY TABLE TO pm;

-- Privileges required to perform cloning of dependent objects.
GRANT CREATE ANY TRIGGER TO pm;
GRANT CREATE ANY INDEX TO pm;

CONNECT pm

CREATE TABLE cust(c_id NUMBER PRIMARY KEY,
                  c_zip NUMBER,
                  c_name VARCHAR(30) DEFAULT NULL,
                  c_lob CLOB);
INSERT INTO cust VALUES(1, 94065, 'hhh', 'ttt');

-- Creating Interim Table
-- no need to specify constraints because they are copied over from the original table
CREATE TABLE cust_int(c_id NUMBER NOT NULL,
                      c_zip NUMBER,
                      c_name VARCHAR(30) DEFAULT NULL,
                      c_lob CLOB)
                      LOB(c) STORE AS SECUREFILE (...);

declare
 col_mapping VARCHAR2(1000);
begin
```

```
 -- map all the columns in the interim table to the original table
 col_mapping :='c_id c_id , '||'c_zip c_zip , '||'c_name c_name, '||'c_lob c_lob';
 DBMS_REDEFINITION.START_REDEF_TABLE('pm', 'cust', 'cust_int', col_mapping);
end;

declare
 error_count pls_integer := 0;
begin
 DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS('pm', 'cust', 'cust_int',
   1, TRUE,TRUE,TRUE,FALSE, error_count);
 DBMS_OUTPUT.PUT_LINE('errors := ' || TO_CHAR(error_count));
end;

exec DBMS_REDEFINITION.FINISH_REDEF_TABLE('pm', 'cust', 'cust_int');

-- Drop the interim table
DROP TABLE cust_int;
DESC cust;

-- to prove that the primary key on the c_id column is preserved after migration.
INSERT INTO cust VALUES(1, 94065, 'hhh', 'ttt');
SELECT * FROM cust;
```

When online redefinition is applied on a table, all triggers are invalidated and are automatically revalidated with the next DML execution on the table.

Partition Change Tracking (PCT)

A group of data dictionary views in Oracle 11g are improved to provide information about refreshing the materialized view partition. Following are the related modifications:

- In the DBA_MVIEWS view, the following columns added: NUM_PCT_TABLES, NUM_FRESH_PCT_REGIONS, NUM_STALE_PCT_REGIONS.
- The view DBA_MVIEW_DETAIL_RELATIONS was added with the following columns: DETAILOBJ_PCT, NUM_FRESH_PCT_PARTITIONS, NUM_STALE_PCT_PARTITIONS.
- New two views are introduced in Oracle 11g named as ALL_MVIEW_DETAIL_PARTITION and ALL_MVIEW_DETAIL_SUBPARTITION. They provides freshness information for each PCT partition and sub-partition.

Generating SQL Crosstab Report using PIVOT Operator

Oracle 11g SQL introduces an excellent new SELECT statement operator: PIVOT. This operator enables you to easily generate crosstab report using SQL language.

The following example produces a report of the maximum quarter sales amount for every quarter where the quarters are displayed as columns:

```
SELECT *
FROM
(  SELECT PRODUCT_DESC, QUARTER_ID, QUARTER_AMT FROM PRODUCT_ROW )
PIVOT
 (MAX(QUARTER_AMT) FOR QUARTER_ID IN (1, 2, 3, 4));


PRODUCT_DESC                  1          2          3          4
-------------------- ---------- ---------- ---------- ----------
Sharpie                     130        215        190        300
Pencils                    2100        918       3280       1315
```

Note: Unfortunately, using the syntax above, you cannot use a sub-query in the list of values provided for the QUARTER_ID in the pivot expression. Therefore, you cannot dynamically provide the possible values of the columns. This is only possible, if you want to generate the report in XML format.

You can use column aliases in the generated report as follows:

```
SELECT *
FROM
(  SELECT PRODUCT_DESC, QUARTER_ID, QUARTER_AMT FROM PRODUCT_ROW )
PIVOT
 (MAX(QUARTER_AMT) FOR QUARTER_ID IN (1 AS QTR1, 2 AS QTR2, 3 AS QTR3, 4 AS QTR4))

PRODUCT_DESC               QTR1       QTR2       QTR3       QTR4
-------------------- ---------- ---------- ---------- ----------
Sharpie                     130        215        190        300
Pencils                    2100        918       3280       1315
```

Following is another example of pivoting on multiple columns:

```
SELECT *
FROM (SELECT video_name, month, quantity_rented, rental_type
      FROM video_mstr_vw)
PIVOT (SUM(quantity_rented)
       FOR (rental_type, month) IN (
                                    (10000, '01') as SR_Jan,
                                    (10001, '01') as IR_Jan,
                                    (10000, '02') as SR_Feb,
                                    (10001, '02') as IR_Feb )
       )
ORDER BY SR_Jan, IR_Jan, SR_Feb, IR_Feb
```

The other new operator UNPIVOT has an opposite effect. Suppose you have a spreadsheet with the following format:

```
SQL> desc cust_matrix
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PUCHASE FREQUENCY                                  NUMBER(3)
 NEW YORK                                           NUMBER
 CONN                                               NUMBER
 NEW JERSEY                                         NUMBER
 FLORIDA                                            NUMBER
 MISSOURI                                           NUMBER

SQL> select * from cust_matrix;

PUCHASE FREQUENCY   NEW YORK       CONN NEW JERSEY    FLORIDA   MISSOURI
----------------- ---------- ---------- ---------- ---------- ----------
                1      33048        165          0          0          0
                2      33151        179          0          0          0
                3      32978        173          0          0          0
                4      33109        173          0          1          0
```

You can convert the columns representing the state into rows as follows:

```
select *
  from CUST_MATRIX
UNPIVOT
(
  state_counts
    for state_code in ("NEW YORK","CONN","NEW JERSEY","FLORIDA","MISSOURI"))
order by "Puchase Frequency", state_code ;

Puchase Frequency STATE_CODE STATE_COUNTS
----------------- ---------- ------------
                1 Conn                165
                1 Florida               0
                1 Missouri              0
                1 New Jersey            0
                1 New York          33048
                2 Conn                179
                2 Florida               0
                2 Missouri              0
```

Partitioning Improvements

The Partition Advisor

Partitioning advice is available within the SQL Access Advisor as part of the Enterprise Manager. SQL Access Advisor can now recommend creating partitions to improve the performance.

If the advisor uses the partitioning as one of its recommendations, you can review the SQL script that converts a heap table into a partitioned one.

Reference Partitions

Reference partitioning allows the partitioning of two tables related to one another by a referential constraint. The partitioning key is resolved and enforced through an existing parent-child relationship. Reference partitions is useful when you want to partition a child table in the same fashion as in the parent table but the child table does not have the same partitioned columns.

When you create the child table, use the clause `PARTITION BY REFERENCE` to create a reference partition as in the following example:

```
CREATE TABLE parent_tab (
  id           NUMBER NOT NULL,
  code         VARCHAR2(10) NOT NULL,
  description  VARCHAR2(50),
  created_date DATE,
  CONSTRAINT parent_tab_pk PRIMARY KEY (id) )
PARTITION BY RANGE (created_date)
(
   PARTITION part_2007 VALUES LESS THAN (TO_DATE('01-JAN-2008','DD-MON-YYYY')),
   PARTITION part_2008 VALUES LESS THAN (TO_DATE('01-JAN-2009','DD-MON-YYYY'))
);

CREATE TABLE child_tab (
  id             NUMBER NOT NULL,
  parent_tab_id  NUMBER NOT NULL, -- it must be NOT NULL for reference partition
  code           VARCHAR2(10),
  description    VARCHAR2(50),
  created_date   DATE,
  CONSTRAINT child_tab_pk PRIMARY KEY (id),
  CONSTRAINT child_parent_tab_fk FOREIGN KEY (parent_tab_id)
                               REFERENCES parent_tab (id) )
PARTITION BY REFERENCE (child_parent_tab_fk);
```

The column on which the parent table is partitioned, `CREATED_DATE`, does not exist in the child table, yet the child table is partitioned on that column.

If you issue the following query, you will notice that the child table will have partitions created with the same name as its parent table. Also, the `HIGH_VALUE` is null, indicating that the boundaries are derived from the parent table.

```
SELECT TABLE_NAME, PARTITION_NAME, HIGH_VALUE, NUM_ROWS
FROM   USER_TAB_PARTITIONS
ORDER BY TABLE_NAME, PARTITION_NAME;
```

Details of the reference partitions can be obtained from the following query:

```
SELECT TABLE_NAME, PARTITIONING_TYPE, REF_PTN_CONSTRAINT_NAME
FROM USER_PART_TABLES;
```

Reference partitions will be physically located in the same tablespace of the parent table if the `TABLESPACE` clause is not specified. The tablespace for the reference partition can be overridden by using the `DEPENDENT TABLES` clause as in the following example:

```
ALTER TABLE parent_tab
ADD PARTITION part_2006 VALUES LESS THAN (TO_DATE('01-JAN-2007','DD-MON-YYYY'))
TABLESPACE parent_tab
DEPENDENT TABLES
(child_tab (PARTITION part_2006 TABLESPACE child_tbs));
```

The following restrictions apply to reference partitioning:

- The foreign key columns referenced in constraint must be NOT NULL.
- The foreign key constraint cannot use the ON DELETE SET NULL clause.
- Reference partitioning cannot be used for index-organized tables, external tables, or domain index storage tables.
- The ROW MOVEMENT setting for both tables must match.

Interval Partitioning

Interval partitioning is an extension of range partitioning which instructs the database to automatically create partitions of a specified interval when data inserted into the table exceeds all of the existing range partitions.

The PARTITION BY RANGE clause is used in the normal way to identify the *transition point* for the partition, then the new INTERVAL clause used to calculate the range for new partitions when the values go beyond the existing transition point. Following is an example:

```
CREATE TABLE interval_tab (
  id           NUMBER,
  code         VARCHAR2(10),
  description  VARCHAR2(50),
  created_date DATE
)
PARTITION BY RANGE (created_date)
INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))
STORE IN (TBS1, TBS2, TBS3, TBS4) -- optional
(   PARTITION part_01 VALUES LESS THAN (TO_DATE('01-NOV-2007','DD-MON-YYYY')) );
```

If you want to create a partition for every quarter, change the interval command to INTERVAL (NUMTOYMINTERVAL(3,'MONTH')) making sure that the first partition was created at the end of a given quarter. The NUMTOYMINTERVAL function can be used to create partitions on a smaller scale like daily, hourly, per minute, or even down to the second.

If you use the STORE IN clause to list a group of tablespaces, Oracle will, in a round-robin fashion, create partitions in the tablespaces listed.

You can convert an existing non-interval range partition table into an interval partition using the ALTER TABLE command as in the following example:

```
ALTER TABLE interval_tab SET INTERVAL (NUMTOYMINTERVAL(1, 'MONTH'))
```

You can obtain information about created partitions by using the following query:

```
EXEC DBMS_STATS.GATHER_TABLE_STATS(USER, 'INTERVAL_TAB');

SELECT TABLE_NAME, PARTITION_NAME, HIGH_VALUE, NUM_ROWS
FROM   USER_TAB_PARTITIONS
ORDER BY TABLE_NAME, PARTITION_NAME;
```

When using interval partitioning, consider the following restrictions:

- You can only specify one partitioning key column, and it must be of NUMBER or DATE type.
- Interval partitioning is not supported for index-organized tables (IOTs).
- Automatically created partitions are named by the system using SYS_P*n* format. You can, however, use the ALTER TABLE .. RENAME PARTITION .. TO command to rename them.
- Interval partitioning does not support subpartitions.
- A MAXVALUE partition cannot be defined for an interval partitioned table.
- NULL values are not allowed in the partition column.

Extended Composite Partitioning

Prior to Oracle Database 11g, the only composite partitioning schemes were range-list and range-hash partitioning. Oracle Database 11g added the following composite partitioning methods:

- **Composite Range-Range**: This partitioning method will partition data using the range method. Then for each partition, subpartitions are created using the range method.
- **Composite List-Range**
- **Composite List-Hash**
- **Composite List-List**

Following are examples of creating tables with extended composite partitions:

```
-- Composite Range-Range Partitioning
CREATE TABLE Alerts_Range_Range
( alert_notification_id  NUMBER,
  subject                VARCHAR2(2000),
  message                VARCHAR2(4000),
  creation_date          DATE DEFAULT SYSDATE,
  closed_date            DATE)
PARTITION    BY RANGE (creation_date)
SUBPARTITION BY RANGE (closed_date)
( PARTITION p_Q1_07 VALUES LESS THAN (to_date('01-APR-2007','dd-MON-yyyy'))
( SUBPARTITION p_JAN07 VALUES LESS THAN (to_date('01-FEB-2007','dd-MON-yyyy')),
  SUBPARTITION p_FEB07 VALUES LESS THAN (to_date('01-MAR-2007','dd-MON-yyyy')),
  SUBPARTITION p_MAR07 VALUES LESS THAN (to_date('01-APR-2007','dd-MON-yyyy')),
  SUBPARTITION p_PRE_Q107 VALUES LESS THAN (MAXVALUE) )
, PARTITION p_Q2_07 VALUES LESS THAN (to_date('01-JUL-2007','dd-MON-yyyy'))
( SUBPARTITION p_APR07 VALUES LESS THAN (to_date('01-MAY-2007','dd-MON-yyyy')),
  SUBPARTITION p_MAY07 VALUES LESS THAN (to_date('01-JUN-2007','dd-MON-yyyy')),
  SUBPARTITION p_JUN2007 VALUES LESS THAN (to_date('01-JUL-2007','dd-MON-yyyy')),
  SUBPARTITION p_PRE_Q207 VALUES LESS THAN (MAXVALUE)));


-- Composite List-Range Partitioning
CREATE TABLE Docs_List_Rist
(document_id       NUMBER,
 document_category VARCHAR2(10),
 organization_id   NUMBER,
 creation_date     DATE)
PARTITION BY LIST (organization_id)
 SUBPARTITION BY RANGE (creation_date)
 ( PARTITION org1 VALUES (1)
   (SUBPARTITION p1_07q1 VALUES LESS THAN (to_date('01-APR-2007','dd-MON-yyyy')),
    SUBPARTITION p1_07q2 VALUES LESS THAN (to_date('01-JUL-2007','dd-MON-yyyy')) ),
   PARTITION org2 VALUES (2)
   (SUBPARTITION p2_07q1 VALUES LESS THAN (to_date('01-APR-2007','dd-MON-yyyy')),
    SUBPARTITION p2_07q2 VALUES LESS THAN (to_date('01-JUL-2007','dd-MON-yyyy')) )
 );


-- Composite List-Hash Partitions
CREATE TABLE list_hash_tab (
  id           NUMBER,
  code         VARCHAR2(10),
  description  VARCHAR2(50),
  created_date DATE
)
PARTITION BY LIST (code)
SUBPARTITION BY HASH (id)
(
   PARTITION part_aa VALUES ('AA')
   (  SUBPARTITION part_aa_01 ,
      SUBPARTITION part_aa_02 ),
   PARTITION part_bb VALUES ('BB')
   (  SUBPARTITION part_bb_01 ,
      SUBPARTITION part_bb_02  )
);

-- Composite List-List Partitions
CREATE TABLE Docs_List_List
( document_id       NUMBER,
  document_category VARCHAR2(10),
  organization_id   NUMBER,
  creation_date     DATE
 )
PARTITION    BY LIST (organization_id)
SUBPARTITION BY LIST (document_category)
( PARTITION org1 VALUES (1)
 ( SUBPARTITION p1_cat_dg    VALUES ('DG'),
   SUBPARTITION p1_cat_asm   VALUES ('ASM'),
```

```
   SUBPARTITION p1_cat_sql   VALUES ('SQL'),
   SUBPARTITION p1_cat_plsql VALUES ('PLSQL'),
   SUBPARTITION p1_cat_rac   VALUES ('RAC'),
   subpartition p1_cat_def   VALUES (default) ),

 PARTITION org2 VALUES (2)
 ( SUBPARTITION p2_cat_dg    VALUES ('DG'),
   SUBPARTITION p2_cat_asm   VALUES ('ASM'),
   SUBPARTITION p2_cat_sql   VALUES ('SQL'),
   SUBPARTITION p2_cat_plsql VALUES ('PLSQL'),
   SUBPARTITION p2_cat_rac   VALUES ('RAC'),
   SUBPARTITION p2_cat_def   VALUES (default) ) );
```

Virtual Column-Based Partitioning

You can create a partition based on a virtual column. Virtual column-based partitioning is supported with all basic partitioning strategies, including interval and interval-* composite partitioning.

For information about virtual columns, refer to the "Virtual Columns" section.

Following is an example of creating virtual column-based partitions:

```
CREATE TABLE hr_employees
( employee_id NUMBER        NOT NULL,
  name        VARCHAR2(55),
  dept_id     NUMBER,
  total_package as (salary + bonus ) VIRTUAL  )

 PARTITION BY RANGE(total_package)
 (partition p_10k values less than (10000),
  partition p_10k_35k values less than (35000),
  partition p_35k_50k values less than (50000),
...
 partition p_1000k_5000k values less than (5000000),
 partition p_other values less than (maxvalue));
```

System Partitioning

With system partitioning, a table is physically divided into partitions but the application decides to which partition the rows should be stored.

System partitions are created using the `PARTITION BY SYSTEM` clause. Inserting into such a table must be partition aware. However, update and delete operations can be performed with or without the partition-aware syntax. Be aware that when you perform updates and deletes without the partition-aware syntax, Oracle scans every partition on the table.

Following is an example:

```
CREATE TABLE docs
( id      NUMBER,
  name    VARCHAR2(255),
  desc    VARCHAR2(1000))
  PARTITION BY SYSTEM
 ( PARTITION docs_p1 TABLESPACE docs_d1,
   PARTITION docs_p2 TABLESPACE docs_d2,
   PARTITION docs_p3 TABLESPACE docs_d3,
   PARTITION docs_p4 TABLESPACE docs_d4  );


-- PARTITION must be stated
INSERT INTO docs PARTITION (docs_p1)
VALUES (1, 'Oracle 11g New Features', 'New features in Oracle 11g Database.');

-- with DELETE command, PARTITION can be stated
DELETE FROM docs PARTITION (docs_p2) WHERE doc_id=1002;
DELETE FROM docs PARTITION (docs_p2);

-- PARTITION can be used in queries to target specific partitions
SELECT COUNT(*) FROM  docs PARTITION (docs_p1)
```

With system partitions, consider the following restrictions:

- If you specify the `PARTITION BY SYSTEM` clause without defining partitions, a single partition is created with the name in the format of "SYS_Pn".
- If you specify `PARTITION BY SYSTEM PARTITIONS` *n* clause, the database creates "n" partitions with the name in the format of "SYS_Pn". The range of allowable values for "n" is from 1 to 1024K-1.
- System partitioning is not available for index-organized tables or a table that is part of a cluster.
- System partitioning can play no part in composite partitioning.
- You cannot split a system partition.
- System partitioning cannot be specified in a `CREATE TABLE ... AS SELECT` statement.
- To insert data into a system-partitioned table using an INSERT INTO ... AS subquery statement, you must use partition-extended syntax to specify the partition into which the values returned by the subquery will be inserted.
- Unique local indexes cannot be created on the partition key.
- Traditional partition pruning and partition-wise joins are not supported on the tables with system partitions.

Appendix I

Mapping Exam 1Z0-050 Objectives to Document Topics

The exam 1Z0-050 (titled as *Oracle Database 11g: New Features*) is the required exam to pass in order to upgrade your Oracle 10g OCP certificate on Oracle DBA to Oracel11g OCP certificate. The table below maps the exam objectives to the document topics. This will help you using this document to review your preparation for the exam.

Be aware that the objectives mentioned here are as of time of writing this document and Oracle might change them. To know the latest objectives, refer to Oracle Education website. Click there on **Certification** link.

Objective	Section / Subsection
Installation and Upgrade Enhancements	
Install Oracle Database 11g	Installation New Features Support Role and Privilege Changes Deprecated Components New Initialization Parameters Affecting Database Creation DBCA Enhancements
Upgrade your database to Oracle Database 11g	Upgrading to Oracle Database 11g
Oracle Direct NFS	Oracle Direct NFS
Use online patching	Patching in Oracle Database Control
Storage Enhancements	
Setup ASM fast mirror resynch	ASM Fast Mirror Resync
Understand scalability and performance enhancements	Fast Rebalance Miscellaneous ASM New Features
Setup ASM disk group attributes	Diskgroup Attributes
Use various new manageability options	N/A
Use the md_backup, md_restore, and ASMCMD extensions	asmcmd Utility Commands
Intelligent Infrastructure Enhancements	
Creating and using AWR baselines	N/A
Setting AWR Baseline Metric Thresholds	Setting Metric Thresholds for Baselines
Control Automated Maintenance Tasks	Automatic Maintenance Tasks
Using Database Resource Manager New Features	Enhancements in Oracle Database Resource Manager
Using new scheduler features	Oracle Scheduler New Features
Performance Enhancements	
ADDM Enhancements	ADDM New Features
Setup Automatic Memory Management	Automatic Memory Management
Enhancements in statistics collection	Changing Statistics Preferences Enhanced Statistics Maintenance
Partitioning and Storage-Related Enhancements	Partitioning Improvements
Implement the new partitioning methods	Partitioning Improvements
Employ Data Compression	Oracle Advanced Compression Option
SQL Access Advisor Overview	SQL Access Advisor Enhancements

Create SQL Access Advisor analysis session using PL/SQL	SQL Access Advisor Enhancements
Using RMAN Enhancements	
Managing Archive logs	Configuring an Archived Redo Log Deletion Policy
Duplicating a Database	Active Database Duplication
Back up large files in multiple sections	The Multisection Backups
Perform Archival Backups	Creating Archival Backups
Use the md_backup, md_restore, and repair ASMCMD extensions	asmcmd Utility Commands
Using Flashback and Logminer	Oracle Flashback-Related New Features
Overview of Flashback Data Archive	
Manage Flashback Data Archive	Flashback Data Archive
Back-out transactions using Flashback Transaction	Oracle_Flashback_Transaction_Backout
Working with Logminer	N/A
Diagnosability Enhancements	
Setup Automatic Diagnostic Repository	Introducing Automatic Diagnostic Repository (ADR) Configuring the ADR
Use Support Workbench	Using The Support Workbench in the OEM
Run health checks	Database Health Monitor
Use SQL Repair Advisor	Using SQL Repair Advisor
Database Replay	
Overview of Workload Capture and Replay	Database Replay
Using Workload capture and replay	Database Replay
Using the Data Recovery Advisor	Data Recovery Advisor
Overview of Data Recovery Advisor	Data Recovery Advisor
Repairing data failures using Data Recovery Advisor	Data Recovery Advisor
Perform proactive health check of the database	Database Health Monitor
Security: New Features	
Configure the password file to use case sensitive passwords	Case-Sensitive Password Files
Encrypt a tablespace	Tablespace Encryption
Configure fined grained access to network services	Fine-Grained Access Control for UTL_* Packages
Oracle SecureFiles	
Use Secure File LOBS to store documents with Compression, Encryption, De-duplication, Caching	SecureFiles
Use SQL and PL/SQL APIs to access Secure File LOBS	Accessing a LOB Using SQL and PL/SQL
Miscellaneous New Features	N/A
Describe and use the enhanced online table redefinition	Online Redefinition
Enhanced finer grained dependency management	Finer-grained Dependency Management

Use Enhanced DDL - Apply the improved table lock mechanism, Create invisible indexes	Explicit Locking of Tables Invisible Indexes
Use Query Result Cache and PL/SQL Result Cache	Server Result Cache
Adaptive Cursor Sharing	Adaptive Cursor Sharing
Temporary Tablespace Enhancements	Shrinking Temporary Tablespaces and Tempfiles
SQL Performance Analyzer	
Overview of SQL Performance Analyzer	The SQL Performance Analyzer
Using SQL Performance Analyzer	The SQL Performance Analyzer
SQL Plan Management	SQL Plan Management
SQL Plan Baseline Architecture	SQL Plan Management
Set up SQL Plan Baseline	SQL Plan Management
Using SQL Plan Baseline	SQL Plan Management
Automatic SQL Tuning	
Setup and modify Automatic SQL Tuning	SQL Tuning Automation
Interpret reports generated by Automatic SQL Tuning	N/A

Appendix II

Displaying CLOB Contents in SQL Plus

Many Oracle supplied packages used in database administration have OUT parameter of CLOB data type. The following examples show how to display CLOB data in SQL Plus session:

```
/*  Method 1   */
VAR report clob
begin
 -- assign value to :report or pass it to the OUT parameter
 ...
end;
/

SET LONG 100000 LONGCHUNKSIZE 10000 LINESIZE 120
PRINT :report



/*  Method 2   */
SET SERVEROUTPUT ON
SET LONG 100000 LONGCHUNKSIZE 10000 LINESIZE 120
declare
 report clob;
begin
 -- assign value to REPORT or pass it to the OUT parameter
 ...
 dbms_output.put_line ('Report : ' || report);
end;
/
```