

Table of Contents

TENTANG BUKU.....	4
PL/SQL.....	4
DATABASE ARCHITECTURE	5
BACKUP & RECOVERY.....	6
Overview.....	6
BASICS OF BACKUP AND RECOVERY.....	6
BACKUP AND RECOVERY OPERATIONS.....	6
ELEMENTS OF A BACKUP AND RECOVERY STRATEGY	7
KEY DATA STRUCTURES FOR BACKUP AND RECOVERY.....	7
DATAFILES.....	7
CONTROL FILES.....	8
ONLINE REDO LOG FILES.....	8
Circular Use of Redo Log Files.....	9
ARCHIVED REDO LOG FILES.....	9
Automatic Managed Undo.....	9
Understanding Basic Backup.....	10
WHAT TYPES OF FAILURES CAN OCCUR?.....	10
WHAT INFORMATION SHOULD BE BACKED UP?	11
Online Database Backup.....	11
Offline Database Backup.....	11
Whole Database Backup.....	12
Tablespace Backups.....	12
Datafile Backups.....	12
Control File Backups.....	12
Archived Redo Log Backups.....	12
Configuration Files.....	13
WHICH BACKUP METHOD SHOULD BE USED?.....	13
Backup Using Import and Export	13
Making Recovery Manager Backups.....	13
Automatic Disk-Based Backup and Recovery.....	14
SECURITY.....	15
Overview	15
Creating and Managing User Account.....	15
Assigning Default and Temporary Tablespace.....	15
Assigning a Profile to Users.....	16
Granting Object Privileges.....	16
Table object privileges	17
Sequence object privileges	17
Stored functions, procedures, packages, and Java object privileges	17
TRANSACTION MANAGEMENT	19
Concept.....	19
Properties of transaction	19
Transaction State	19
Schedule & Concurrent Transaction	20
Transaction Management Oracle.....	21
Transaction Control	21

Distributed Transaction.....	21
Autonomous Transaction.....	22

Table of Figures

graphics1.....4

DATABASE ARCHITECTURE

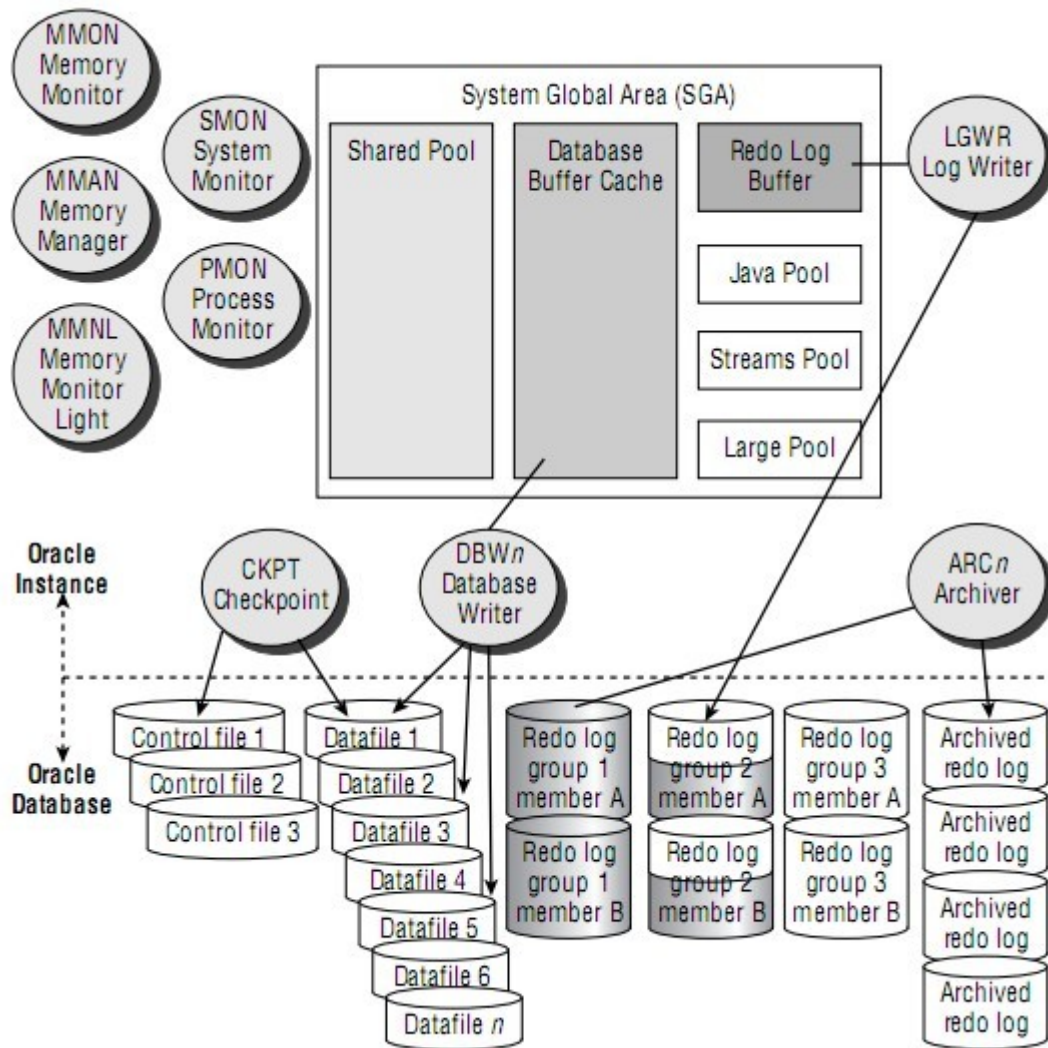


Figure 1: Oracle 10G Architecture

BACKUP & RECOVERY

Overview

Backup and recovery is one of the most important aspects of database administration. If a database crashed and there was no way to recover it, the devastating results to a business could include lost data, lost revenue and customer dissatisfaction. Whether companies operate a single database or multiple databases storing hundreds of gigabytes or even terabytes of data, they share one common factor - the need to back up important data and protect themselves from disaster by developing a backup and recovery plan.

BASICS OF BACKUP AND RECOVERY

A backup is a representative copy of data. This copy can include important parts of a database such as the control file, redo logs, and datafiles. A backup protects data from application error and acts as a safeguard against unexpected data loss, by providing a way to restore original data. Backups are divided into physical backups and logical backups. Physical backups are copies of physical database files. The phrase "backup and recovery" usually refers to the transfer of copied files from one location to another, along with the various operations performed on these files.

In contrast, logical backups contain data that is exported using SQL commands and stored in a binary file. Oracle records both committed and uncommitted changes in redo log buffers. Logical backups are used to supplement physical backups. Restoring a physical backup means reconstructing it and making it available to the Oracle server. To recover a restored backup, data is updated using redo records from the transaction log. The transaction log records changes made to the database after the backup was taken.

Oracle performs crash recovery and instance recovery automatically after an instance failure. In the case of media failure, a database administrator (DBA) must initiate a recovery operation. Recovering a backup involves two distinct operations: rolling the backup forward to a more recent time by applying redo data, and rolling back all changes made in uncommitted transactions to their original state. In general, recovery refers to the various operations involved in restoring, rolling forward, and rolling back a backup. Backup and recovery refers to the various strategies and operations involved in protecting the database against data loss and reconstructing the database should a loss occur.

BACKUP AND RECOVERY OPERATIONS

A backup is a snapshot of a datafile, tablespace, or database at a certain time. If periodic backups of the database have been made and data is lost, users can apply the stored redo information to their latest backup to make the database current again. Oracle enables users to restore an older backup and apply only some redo data, thereby recovering the database to an earlier point in time. This type of recovery is called incomplete media recovery. If the backup was consistent, then users are not required to apply any redo data, at all.

A simple example of media recovery illustrates the concept. Suppose a user makes a backup of the database at noon. Starting at noon, one change to the database is made every minute. At 1 p.m. one

of the disk drives fails, causing the loss of all data on that disk. Fortunately, Oracle records all changes in the redo log. The user can then restore the noon backup onto a functional disk drive and use redo data to recover the database to 1 p.m., reconstructing the lost changes.

ELEMENTS OF A BACKUP AND RECOVERY STRATEGY

Although backup and recovery operations can be intricate and vary from one business to another, the basic principles follow these four simple steps:

1. Multiplex the online redo logs
2. Run the database in ARCHIVELOG mode and archive redo logs to multiple locations
3. Maintain multiple concurrent backups of the control file
4. Take frequent backups of physical datafiles and store them in a safe place, making multiple copies if possible

As long as users have backups of the database and archive redo logs in safe storage, the original database can be recreated.

KEY DATA STRUCTURES FOR BACKUP AND RECOVERY

Before users begin to think seriously about backup and recovery strategy, the physical data structures relevant for backup and recovery operations must be identified. This section discusses the following physical data structures:

- Datafiles
- Control Files
- Online Redo Log Files
- Archived Redo Log Files
- Automatic Managed Undo

DATAFILES

Every Oracle database has one or more physical datafiles that belong to logical structures called tablespaces. The datafile is divided into smaller units called data blocks. The data of logical database structures, such as tables and indexes, is physically located in the blocks of the datafiles allocated for a database. Datafiles hold the following characteristics:

- User-defined characteristics allow datafiles to automatically extend when the database runs out of space.
- One or more physical datafiles form a logical database storage unit called a tablespace.

The first block of every datafile is the header. The header includes important information such as file size, block size, tablespace, and creation timestamp. Whenever the database is opened, Oracle checks to see that the datafile header information matches the information stored in the control file. If it does not, then recovery is necessary. Oracle reads the data in a datafile during normal operation and stores it in the buffer cache. For example, assume that a user wants to access some data in a

table. If the requested information is not already in the buffer cache, Oracle reads it from the appropriate datafiles and stores it in memory.

CONTROL FILES

Every Oracle database has a control file containing the operating system filenames of all other files that constitute the database. This important file also contains consistency information that is used during recovery, such as the:

- Database name
- Timestamp of database creation
- Names of the database's datafiles and online and archived redo log files
- Checkpoint, a record indicating the point in the redo log where all database changes prior to this point have been saved in the datafiles
- Recovery Manager(RMAN) backup meta-data

Users can multiplex the control file, allowing Oracle to write multiple copies of the control file to protect it against disaster. If the operating system supports disk mirroring, the control file can also be mirrored, allowing the O/S to write a copy of the control file to multiple disks. Every time a user mounts an Oracle database, its control file is used to identify the datafiles and online redo log files that must be opened for database operation. If the physical makeup of the database changes, such as a new datafile or redo log file is created, Oracle then modifies the database's control file to reflect the change. The control file should be backed up whenever the structure of the database changes. Structural changes can include adding, dropping, or altering datafiles or tablespaces and adding or dropping online redo logs.

ONLINE REDO LOG FILES

Redo logs are absolutely crucial for recovery. For example, imagine that a power outage prevents Oracle from permanently writing modified data to the datafiles. In this situation, an old version of the data in the datafiles can be combined with the recent changes recorded in the online redo log to reconstruct what was lost. Every Oracle database contains a set of two or more online redo log files. Oracle assigns every redo log file a log sequence number to uniquely identify it. The set of redo log files for a database is collectively known as the database's redo log.

Oracle uses the redo log to record all changes made to the database. Oracle records every change in a redo record, an entry in the redo buffer describing what has changed. For example, assume a user updates a column value in a payroll table from 5 to 7. Oracle records the old value in undo and the new value in a redo record. Since the redo log stores every change to the database, the redo record for this transaction actually contains three parts:

- The change to the transaction table of the undo
- The change to the undo data block
- The change to the payroll table data block

If the user then commits the update to the payroll table - to make permanent the changes executed by SQL statements - Oracle generates another redo record. In this way, the system maintains a careful watch over everything that occurs in the database.

Circular Use of Redo Log Files

Log Writer (LGWR) writes redo log entries to disk. Redo log data is generated in the redo log buffer of the system global area. As transactions commit and the log buffer fills, LGWR writes redo log entries into an online redo log file. LGWR writes to online redo log files in a circular fashion: when it fills the current online redo log file, called the active file, LGWR writes to the next available inactive redo log file. LGWR cycles through the online redo log files in the database, writing over old redo data. Filled redo log files are available for reuse depending on whether archiving is enabled:

- If archiving is disabled, a filled online redo log is available once the changes recorded in the log have been saved to the datafiles.
- If archiving is enabled, a filled online redo log is available once the changes have been saved to the datafiles and the file has been archived.

ARCHIVED REDO LOG FILES

Archived log files are redo logs that Oracle has filled with redo entries, rendered inactive, and copied to one or more log archive destinations. Oracle can be run in either of two modes:

- ARCHIVELOG - Oracle archives the filled online redo log files before reusing them in the cycle.
- NOARCHIVELOG - Oracle does not archive the filled online redo log files before reusing them in the cycle.

Running the database in ARCHIVELOG mode has the following benefits:

- The database can be completely recovered from both instance and media failure.
- The user can perform online backups, i.e., back up tablespaces while the database is open and available for use.
- Archived redo logs can be transmitted and applied to the standby database
- Oracle supports multiplexed archive logs to avoid any possible single point of failure on the archive logs.
- The user has more recovery options, such as the ability to perform tablespace-point-in-time recovery (TSPITR)

Running the database in NOARCHIVELOG mode has the following consequences:

- The user can only back up the database while it is completely closed after a clean shutdown.
- Typically, the only media recovery option is to restore the whole database, which causes the loss of all transactions issued since the last backup.

Automatic Managed Undo

Every Oracle database must have a method of maintaining information that is used to roll back, or undo, changes to the database. Such information consists of records of the actions of transactions, primarily before they are committed. Oracle refers to these records collectively as undo. Historically, Oracle has used rollback segments to store undo. Space management for these rollback

segments has proven to be quite complex. Oracle now offers another method of storing undo that eliminates the complexities of managing rollback segment space, and allows DBAs to exert control over how long undo is retained before being overwritten. This method uses an undo tablespace. Undo records are used to:

- Roll back transactions when a ROLLBACK statement is issued
- Recover the database
- Provide read consistency

When a rollback statement is issued, undo records are used to undo changes that were made to the database by the uncommitted transaction. During database recovery, undo records are used to undo any uncommitted changes applied from the redo log to the datafiles. Undo records provide read consistency by maintaining the before image of the data for users who are accessing the data at the same time that another user is changing it.

Understanding Basic Backup

A backup strategy provides a safeguard against data loss. Answering the following questions can help database administrators develop a strong backup strategy:

- What types of failures can occur?
- What information should be backed up?
- Which backup method should be used?
- Should backups be made online or offline?
- How often should backups be made?
- How can dangerous backup techniques be avoided?

WHAT TYPES OF FAILURES CAN OCCUR?

Data loss can occur for various reasons. Here are some of the most common types of failures that can lead to data loss.

A *statement failure* is a logical failure in the handling statement in an Oracle program. For example, a user issues a statement that is not a valid SQL construction. When statement failure occurs, Oracle automatically undoes any effects of the statement and returns control to the user.

A *process failure* is a failure in a user process accessing Oracle, i.e., an abnormal disconnection or process termination. The failed user process cannot continue work, although Oracle and other user processes can. If the user process fails while modifying the database, Oracle background processes undo the effects of uncommitted transactions.

An *instance failure* is a problem that prevents an Oracle instance, i.e., the SGA and background processes, from continuing to function. Instance failure can result from a hardware problem such as a power outage, or a software problem such as an operating system crash. When an instance fails, Oracle does not write the data in the buffers of the SGA to the datafiles.

A *user or application error* is a user mistake that results in the loss of data. For example, a user can accidentally delete data from a payroll table. Such user errors can require the database or object to be recovered to a point in time before the error occurred. To allow recovery from user error and accommodate other unique recovery requirements, Oracle provides Flashback Technology.

A *media failure* is a physical problem that arises when Oracle tries to write or read a file that is required to operate the database. A common example is a disk head crash that causes the loss of all data on a disk drive. Disk failure can affect a variety of files, including datafiles, redo log files, and control files. Because the database instance cannot continue to function properly, it cannot write the data in the database buffers of the SGA to the datafiles.

WHAT INFORMATION SHOULD BE BACKED UP?

A database contains a wide variety of types of data. When developing backup strategy, DBAs must decide what information they want to copy. The basic backup types include:

- Online Database Backup
- Offline Database Backup
- Whole Database
- Tablespace
- Datafile
- Control File
- Archived Redo Log
- Configuration Files

In deciding what to back up, the basic principle is to prioritize data depending on its importance and the degree to which it changes. Archive logs do not change, for example, but they are crucial for recovering the database, so multiple copies should be maintained, if possible. Expense account tables, however, are constantly updated by users. Therefore, this tablespace should be backed up frequently to prevent having to apply as much redo data during recovery.

Backups can be combined in a variety of ways. For example, a DBA can decide to take weekly whole database backups, to ensure a relatively current copy of original database information, but take daily backups of the most accessed tablespaces. The DBA can also multiplex the all important control file and archived redo log as an additional safeguard.

Online Database Backup

An online backup or also known as an open backup, is a backup in which all read-write datafiles and control files have not been checkpointed with respect to the same SCN. For example, one read-write datafile header may contain an SCN of 100 while other read-write datafile headers contain an SCN of 95 or 90. Oracle cannot open the database until all of these header SCNs are consistent, that is, until all changes recorded in the online redo logs have been saved to the datafiles on disk. If the database must be up and running 24 hours a day, 7 days a week, then you have no choice but to perform online backups of a whole database which is in ARCHIVELOG mode.

Offline Database Backup

In this backup, all datafiles and control files are consistent to the same point in time - consistent with respect to the same SCN, for example. The only tablespaces in a consistent backup that are allowed to have older SCNs are read-only and offline-normal tablespaces, which are consistent with

the other datafiles in the backup. This type of backup allows the user to open the set of files created by the backup without applying redo logs, since the data is already consistent. The only way to perform this type of backup is to shut down the database cleanly and make the backup while the database is closed. A consistent whole database backup is the only valid backup option for databases running in NOARCHIVELOG mode.

Whole Database Backup

The most common type of backup, a whole database backup contains the control file along with all database files that belong to a database. If operating in ARCHIVELOG mode, the DBA also has the option of backing up different parts of the database over a period of time, thereby constructing a whole database backup piece by piece.

Tablespace Backups

A tablespace backup is a subset of the database. Tablespace backups are only valid if the database is operating in ARCHIVELOG mode. The only time a tablespace backup is valid for a database running in NOARCHIVELOG mode is when that tablespace is read-only or offline-normal.

Datafile Backups

A datafile backup is a backup of a single datafile. Datafile backups, which are not as common as tablespace backups and are only valid if the database is run in ARCHIVELOG mode. The only time a datafile backup is valid for a database running in NOARCHIVELOG mode is if that datafile is the only file in a tablespace. For example, the backup is a tablespace backup, but the tablespace only contains one file and is read-only or offline-normal.

Control File Backups

A control file backup is a backup of a database's control file. If a database is open, the user can create a valid backup by issuing the following SQL statement: ALTER DATABASE BACKUP CONTROLFILE to 'location'; or use Recovery Manager (RMAN).

Archived Redo Log Backups

Archived redo logs are the key to successful media recovery. Depending on the disk space available and the number of transactions executed on the database, you want to keep as many days of archive logs on disk and you want to back them up regularly to ensure a more complete recovery.

Configuration Files

Configuration files may consist of spfile or init.ora, password file, tnsnames.ora, and sqlnet.ora. Since these files do not change often, then they require a less frequent backup schedule. If you lost a configuration file it can be easily recreated manually. When restore time is a premium, it will be faster to restore a backup of the configuration file than manually creating a file with a specific format.

WHICH BACKUP METHOD SHOULD BE USED?

Oracle provides users a choice of several basic methods for making backups. The methods include:

- Import Export
- Recovery Manager (RMAN) - A component that establishes a connection with a server process and automates the movement of data for backup and recovery operations.
- Oracle Enterprise Manager - A GUI interface that invokes Recovery Manager.
- Oracle Data Pump - The utility makes logical backups by writing data from an Oracle database to operating system files in a proprietary format. This data can later be imported into a database.
- User Managed - The database is backed up manually by executing commands specific to the user's operating system.

Making Recovery Manager Backups

Recovery Manager (RMAN) is a powerful and versatile program that allows users to make an RMAN backup or image copy of their data. When the user specifies files or archived logs using the RMAN BACKUP command, By default RMAN creates a backup set as output. A backup set is a file or files in a proprietary-specific format that requires the use of the RMAN RESTORE command for recovery operations. In contrast, when the BACKUP AS COPY command is used to create an image copy of a file, it is in an instance-usable format - the user does not need to invoke Recovery Manager to restore or recover it.

When a RMAN command is issued, such as backup or restore, Recovery Manager establishes a connection to an Oracle server process. The server process then back up the specified datafile, control file, or archived log from the target database. The recovery catalog is a central repository containing a variety of information useful for backup and recovery. RMAN automatically establishes the names and locations of all the files needed to back up. Recovery Manager also supports incremental backups - backups of only those blocks that have changed since a previous backup. In traditional backup methods, all the datablocks ever used in a datafile must be backed up.

Automatic Disk-Based Backup and Recovery

The components that creates different backup and recovery-related files have no knowledge of each other or of the size of the file systems where they store their data. With Automatic Disk-Based Backup and Recovery, you can create a flash recovery area, which automates management of backup-related files. Choose a location on disk and an upper bound for storage space, and set a

retention policy that governs how long backup files are needed for recovery, and the database manages the storage used for backups, archived redo logs, and other recovery-related files for your database within that space. Files no longer needed are eligible for deletion when RMAN needs to reclaim space for new files. If you do not use a flash recovery area, you must manually manage disk space for your backup-related files and balance the use of space among the different types of files. Oracle Corporation recommends that you enable a flash recovery area to simplify your backup management.

Workshop

```
exp scott/tiger file=emp.dmp log=emp.log tables=emp rows=yes \ indexes=no
exp scott/tiger file=emp.dmp tables=(emp,dept)
```

```
imp scott/tiger file=emp.dmp full=yes
imp scott/tiger file=emp.dmp fromuser=scott touser=scott tables=dept
```

```
exp SCOTT/TIGER FILE=D:F1.dmp,E:F2.dmp FILESIZE=10m LOG=scott.log
```

Start using datapump export

```
$ expdp scott/tiger DIRECTORY=dmpdir DUMPFILE=scott.dmp
Export: Release 10.2.0.1.0 - 64bit Production on Friday, 31 March, 2006
11:36:07

Copyright (c) 2003, 2005, Oracle. All rights reserved.

Connected to: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 -
64bit Production

With the Partitioning, OLAP and Data Mining options

ORA-39002: invalid operation
ORA-39070: Unable to open the log file.
ORA-39087: directory name DMPDIR is invalid
```

Oops, we need to create a directory first!

```
SQL> CREATE DIRECTORY dmpdir AS '/opt/oracle';
Directory created.

SQL> SELECT directory_path FROM dba_directories WHERE directory_name =
'DATA_PUMP_DIR';
DIRECTORY_PATH
-----
/app/oracle/product/10.2.0/rdbms/log/

$ expdp scott/tiger DIRECTORY=dmpdir DUMPFILE=scott.dmp
Export: Release 10.2.0.1.0 - 64bit Production on Friday, 31 March, 2009
11:41:02
```

```

Copyright (c) 2003, 2005, Oracle. All rights reserved.

Connected to: Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 -
64bit Production

With the Partitioning, OLAP and Data Mining options

Starting "SCOTT"."SYS_EXPORT_SCHEMA_01":  scott/***** DIRECTORY=dmpdir
DUMPFIL=scott.dmp

Estimate in progress using BLOCKS method...
Processing object type SCHEMA_EXPORT/TABLE/TABLE_DATA
Total estimation using BLOCKS method: 175.2 MB
Processing object type SCHEMA_EXPORT/PRE_SCHEMA/PROCACT_SCHEMA
Processing object type SCHEMA_EXPORT/TYPE/TYPE_SPEC
Processing object type SCHEMA_EXPORT/TABLE/TABLE
Processing object type SCHEMA_EXPORT/TABLE/INDEX/INDEX
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Processing object type SCHEMA_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
Processing object type SCHEMA_EXPORT/TABLE/COMMENT
Processing object type SCHEMA_EXPORT/PACKAGE/PACKAGE_SPEC
Processing object type SCHEMA_EXPORT/PROCEDURE/PROCEDURE
Processing object type
SCHEMA_EXPORT/PACKAGE/COMPILE_PACKAGE/PACKAGE_SPEC/ALTER_PACKAGE_SPEC
Processing object type SCHEMA_EXPORT/PROCEDURE/ALTER_PROCEDURE
Processing object type SCHEMA_EXPORT/PACKAGE/PACKAGE_BODY
Processing object type SCHEMA_EXPORT/TABLE/CONSTRAINT/REF_CONSTRAINT
Processing object type SCHEMA_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
Processing object type SCHEMA_EXPORT/JOB
. . exported "SCOTT"."BIGEMP"                145.2 MB 3670016 rows
. . exported "SCOTT"."DEPT"                  5.656 KB      4 rows
. . exported "SCOTT"."EMP"                   7.820 KB     14 rows
. . exported "SCOTT"."ORD_CHARGE_TAB"        5.296 KB      2 rows
. . exported "SCOTT"."SALGRADE"             5.585 KB      5 rows
. . exported "SCOTT"."BONUS"                 0 KB         0 rows
. . exported "SCOTT"."NEWOBJECT1_T"         0 KB         0 rows
. . exported "SCOTT"."T1"                   0 KB         0 rows
Master table "SCOTT"."SYS_EXPORT_SCHEMA_01" successfully loaded/unloaded
*****
Dump file set for SCOTT.SYS_EXPORT_SCHEMA_01 is:
    /app/oracle/scott.dmp
Job "SCOTT"."SYS_EXPORT_SCHEMA_01" successfully completed at 11:44:50

```

Invoking from PL/SQL

```
DECLARE
```

```

hand NUMBER;
BEGIN
  hand := Dbms_DataPump.Open(operation => 'EXPORT',
                             job_mode => 'FULL',
                             ob_name => 'FULLEXPJOB',
                             version  => 'COMPATIBLE');

  Dbms_DataPump.Add_File(handle => hand,
                         filename => 'expdp_plsql.log',
                         directory => 'DMPDIR',
                         filetype => 3);

  Dbms_DataPump.Add_File(handle => hand,
                         filename => 'expdp_plsql.dmp',
                         directory => 'DMPDIR',
                         filetype => 1);

  -- Dbms_DataPump.Set_Parameter(handle => hand,
  --                             name => 'ESTIMATE',
  --                             value => 'STATISTICS');

  Dbms_DataPump.Start_Job(hand);

END;
/

```

Example RMAN

```

% ORACLE_SID=TARGDB
% export ORACLE_SID
% rman target backup_admin/backup_admin

Recovery Manager: Release 10.2.0. - Production
Copyright (c) 1995, 2002, Oracle Corporation. All rights reserved.
connected to target database: TARGDB (DBID=2528050866)
RMAN>

```

```

RMAN> run {
  CONFIGURE RETENTION POLICY TO REDUNDANCY 2;
  CONFIGURE DEFAULT DEVICE TYPE TO DISK;
  CONFIGURE CONTROLFILE AUTOBACKUP ON;
  CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO
'/orabackup1/rman/TARGDB/%F' ;
  CONFIGURE DEVICE TYPE DISK PARALLELISM 2;
  CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT
'/orabackup1/rman/TARGDB/backup_db_%d_S_%s_P_%p_T_%t' MAXPIECESIZE 1024m;
  CONFIGURE CHANNEL 2 DEVICE TYPE DISK FORMAT

```

```
 '/orabackup2/rman/TARGDB/backup_db_%d_S_%s_P_%p_T_%t' MAXPIECESIZE 1024m;
```

Backup database

```
RMAN> run {
# -----
# The following RMAN commands are run each day.
# The steps are:
# - Re-start the database to perform crash recovery, in
#   case the database is not currently open, and was not
#   shut down consistently. The database is started in DBA
#   mode so that normal users cannot connect.
# - Shut down with the IMMEDIATE option to close the
#   database consistently.
# - Startup and mount the database.
# - Backup database.
# - Open database for normal operation.
# -----
startup force dba;
shutdown immediate;
startup mount;
backup database;
alter database open;
}
exit
```

Restore Database

```
set dbid 2528050866;

connect target backup_admin/backup_admin;
startup nomount;

run {
# -----
# Uncomment the SET UNTIL command to restore database to the
# incremental backup taken two days ago.
# SET UNTIL TIME 'SYSDATE-2';
# -----
set controlfile autobackup format for device \
type disk to '/orabackup1/rman/TARGDB/%F';
restore controlfile from autobackup;
alter database mount;
restore database;
recover database noredo;
alter database open resetlogs;
sql "alter tablespace temp add tempfile '/u06/app/oradata/TARGDB/temp01.dbf'
size 500m autoextend on next 500m maxsize 1500m";
}

exit
```

SECURITY

Overview

Creating and Managing User Account

Configuring User Authentication

1. Password authentication

```
CREATE USER username IDENTIFIED BY passwd.
```

The syntax create user with name *username* and password *passwd*.

2. Externally Authentication

Externally authenticated user account do not store or validate password in the database.

```
CREATE USER ops$oracle IDENTIFIED EXTERNALLY.
```

Keyword IDENTIFIED EXTERNALLY tell to database that this user account externally authenticated user.

3. Globally Authentication

When a globally identified user attempts to connect to the database, the database verifies that the username is valid and passes the connection information to the advanced security option for authentication. The advanced security option supports several mechanisms for authentication, including biometrics, X.509 certificates, Kerberos, and RADIUS.

```
CREATE USER spy_master IDENTIFIED GLOBALLY AS 'CN=spy_master, OU=tier2, O=security, C=US';
```

Assigning Default and Temporary Tablespace

Every user is assigned to default tablespace

Assigning default tablespace

```
CREATE USER username IDENTIFIED by passwd DEFAULT TABLESPACE tablespace_name;
```

```
ALTER USER username DEFAULT TABLESPACE tablespace_name;
```

to change database default tablespace (default tablespace assigning to user if default tablespace not provided)

```
ALTER DATABASE DEFAULT TABLESPACE tablespace_name;
```

Every user is assigned a temporary tablespace in which the database stores temporary segments. Temporary segments are created during large sorting operations, such as ORDER BY, GROUP BY, SELECT DISTINCT, MERGE JOIN, or CREATE INDEX.

Temporary segments are also used when a temporary table is used. The database creates and drops temporary segments transparently to the user. Because of the transitory nature of temporary segments, you must use a dedicated tablespace of type TEMPORARY for your user's temporary tablespace setting.

```
CREATE USER username IDENTIFIED BY passwd  
DEFAULT TABLESPACE default_tablespace_name  
TEMPORARY TABLESPACE temp_tablespace_name;
```

```
ALTER USER username  
TEMPORARY TABLESPACE temp_tablespace_name;
```

To change the database default temporary tablespace, use the ALTER DATABASE statement, like this:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp_tablespace_name;
```

Assigning a Profile to Users

In addition to default and temporary tablespaces, every user is assigned a profile. A profile serves two purposes: first, it can limit the resource usage of some resources, and second, it can enforce password-management rules.

The default profile is appropriately named default. To explicitly assign a profile to a user, include the keywords PROFILE profile_name in the CREATE USER or ALTER USER statement. For example, to assign the profile named resource_profile to the new user jiang as well as to the existing user hamish, execute the following SQL:

```
CREATE USER hr IDENTIFIED BY hr  
DEFAULT TABLESPACE users  
TEMPORARY TABLESPACE temp  
PROFILE resource_profile;
```

```
ALTER USER hr  
PROFILE resource_profile;
```

Granting Object Privileges

Object privileges bestow upon the grantee the permission to use a schema object owned by another user in a particular way. There are several types of object privileges. Some privileges apply only to certain schema objects. For example, the INDEX privilege applies only to tables, and the SELECT privilege applies to tables, views, and sequences.

The following object privileges can be granted individually, grouped in a list, or with the keyword ALL to implicitly grant all available object privileges for a particular schema object.

Table object privileges

Oracle 10g provides several object privileges for tables. These privileges give the table owner considerable flexibility in controlling how schema objects are used and by whom. The following privileges are commonly granted, and you should know them well.

- SELECT** : The most commonly used privilege for tables. With this privilege, the table owner permits the grantee to query the specified table with a SELECT statement.
- INSERT** : Permits the grantee to create new rows in the specified table with an INSERT statement.
- UPDATE** : Permits the grantee to modify existing rows in the specified table with an UPDATE statement.
- DELETE** : Permits the grantee to remove rows from the specified table with a DELETE statement.

The following are powerful administrative privileges on tables; grant them cautiously.

- ALTER** : Permits the grantee to execute an ALTER TABLE statement on the specified table. This privilege can be used to add, modify, or rename columns in the table, to move the table to another tablespace, or even to rename the specified table.
- DEBUG** : Permits the grantee to access, via a debugger, the PL/SQL code in any triggers on the specified table.
- INDEX** : Permits the grantee to create new indexes on the table. These new indexes will be owned by a different user than the table, which is an unusual practice. In most cases, the indexes on a table are owned by the same user who owns the table itself.
- REFERENCES**: Permits the grantee to create foreign key constraints that reference the specified table.

Sequence object privileges

Oracle 10g provides only two object privileges for sequences.

- SELECT** : Permits the grantee to access the current and next values (CURRVAL and NEXTVAL) of the specified sequence.
- ALTER** : Permits the grantee to change the attributes of the specified sequence with an ALTER statement.

Stored functions, procedures, packages, and Java object privileges

Oracle 10g provides only two object privileges for stored PL/SQL programs.

DEBUG : Permits the grantee to access, via a debugger, all the public and private variables and types declared in the specified program. If the specified object is a package, both the specification and the body are accessible to the grantee. The grantee can also use a debugger to place breakpoints in the specified program.

EXECUTE : Permits the grantee to execute the specified program. If the specified object is a package, any program, variable, type, cursor, or record declared in the package specification is accessible to the grantee.

```
GRANT SELECT,INSERT,UPDATE,DELETE ON customers TO sales_manager;
```

```
GRANT SELECT ON customers TO PUBLIC;
```

```
GRANT EXECUTE ON package_view_customer TO hr;
```

TRANSACTION MANAGEMENT

Concept

Transaction : Unit of program execution that access and possibly updates various data items.

Properties of transaction

To ensure integrity of the data , transaction must have properties ACID :

Atomicity : A transaction either happens completely, or none of it happens.

Consistency : A transaction take the database from consistent state to the next.

Isolation : The effect of the transaction my not be visible to others transaction until transaction committed

Durability : One transaction committed, it is permanent.

Example T1 is transaction that transfer \$50 from account A to account B the transaction can be defined as :

```
T1 :  
(1) read(A);  
(2) A := A - 50;  
(3) write(A);  
(4) read(B);  
(5) B := B + 50;  
(6) write(B);
```

Atomicity : All sequence must be executed.

Consistency : the consistency requirement here is the sum of A and B be unchanged by the execution of the transaction.

Isolation : During transaction T1 happens, the database state is temporary inconsistent, Transaction T2 cannot read data that changes in transaction T1.

Durability : Once the execution of the transaction completes successfully and the user who initiated the transaction has been notified that the transfer of funds has taken place, it must be the case that no system failure will result in a loss of data corresponding to this transfer funds.

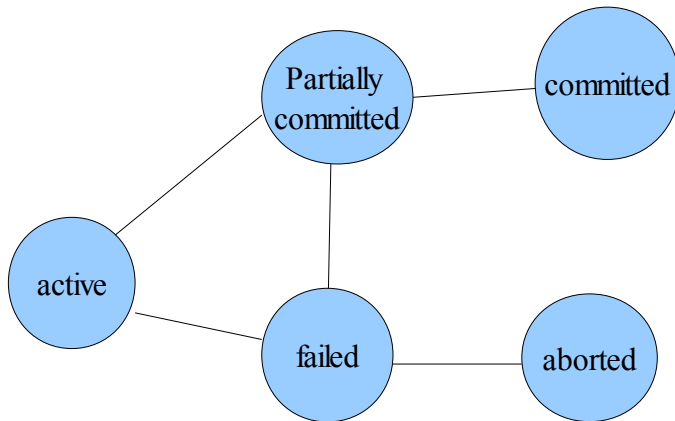
Transaction State

A transaction must be in one of the following states :

Active : The initial state , the transaction stays in this state while it is executing.

Partially committed : after the final statement has been executed

Failed : after the discovery that normal execution can no longer proceed
Aborted : after the transaction has been rolled back and the database has been restored to its state prior the start of the transaction.
Committed : after successful completion.



Schedule & Concurrent Transaction

A Scheduler is list of action (reading, writing, aborting, or committing) from a set of transaction , and the order in which two action of a transaction T appear in a scheduler must be the same as the order in which they appear in T.

Example scheduler of two Transaction.

T1	T2
R(A)	
W(A)	
	R(B)
	W(B)
R(C)	
W(C)	

Concurrent execution is difficult but necessary for performance reason.

1. While one transaction is waiting for a page to be read from a disk, the CPU can process another transaction.
2. Interleaved a short transaction with a long transaction usually allows the short transaction to complete quickly.

Anomalies Due to Interleaved Execution

1. Reading Uncommitted Data (WR Conflicts)
2. Unrepeatable Reads (RW Conflicts)
3. Overwriting Uncommitted Data (WW Conflicts)

Lock Based Concurrency Control

To ensure that only serializable, recoverable schedules are allowed and that no actions of committed transaction are lost while undoing aborted transaction, DBMS uses locking protocol.

Strict Two-Phase Locking (Strict 2PL) is the most widely used that has two rules :

1. If transaction T wants to read an object, it first request a shared lock on object.
2. All locks held by transaction are released when the transaction is completed.

Transaction Management Oracle

Transaction Control

The various statement control in oracle are :

COMMIT : Commit is end of transaction and make changed permanent (durable), Statement used **COMMIT** or **COMMIT WORK**.

ROLLBACK : Rollback is end of transaction and undoes any uncommitted changed you have outstanding. Statement **ROLLBACK** or **ROLLBACK WORK**.

SAVEPOINT : A SAVEPOINT allows you to create a “marked point” within a transaction. You may have multiple Save Points within a single transaction.

ROLLBACK TO <SAVEPOINT> : This is used with the SAVEPOINT command above. You may roll back your transaction to that marked point without rolling back any of the work that preceded it

SET TRANSACTION : This statement allows you to set various transaction attributes, such as its isolation level and whether it is read-only or read write. You can also use this statement to instruct the transaction to use a specific rollback segment.

Distributed Transaction

Distributed transaction is one of feature in oracle, key that is in distributed transaction is DATABASE LINK. Oracle use 2PC protocol in distributed transaction. Limitation is distributed transaction.

- Cannot issue COMMIT over database link
- Cannot issue DDL over database link
- Cannot issue SAVEPOINT over database link.

Example cannot issue commit over database link.

Database A :

```
Procedure InsertA
is
begin
    insert into table_a values (1,2);
    commit;
end InsertA;
```

Database B :

From database B we call procedure InsertA in database A

```
Begin
update tableB set b=123 where id=1;
InsertA@DBLINKA;
End;
```

this will return error because in procedure InsertA we issue commit. Statement commit in procedure InsertA must be removed.

Autonomous Transaction.

Autonomous transactions allow you to create a new transaction within a transaction that may commit or roll back changes, independently of its parent transaction.

Note the use of the PRAGMA AUTONOMOUS_TRANSACTION. This directive tells the database that this procedure, when executed, is to be executed as a new autonomous transaction, independently of its parent transaction.

Database Application Using Java and PHP

Connection Java to Oracle

Example of using jdbc

```
import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Properties;
```

```

public class ConnectionExample {

    final String driverClass      = "oracle.jdbc.driver.OracleDriver";
    final String connectionURLThin = "jdbc:oracle:thin:@server:1521:MYDB";
    final String connectionURLOCI  = "jdbc:oracle:oci8:@MYDB";
    final String userID            = "scott";
    final String userPassword      = "tiger";
    final String queryString       = "SELECT" +
        "      user " +
        "      , TO_CHAR(sysdate, 'DD-MON-YYYY
HH24:MI:SS') " +
        "FROM dual";

    /**
     * The following method provides an example of how to connect to a
     database
     * by registering the JDBC driver using the DriverManager class. This
     method
     * requires you to hardcode the loading of a Driver implementation in
     * your application. this alternative is the least desirable since it
     * requires a rewrite and recompile if your database or database driver
     * changes.
     */
    public void driverManager() {

        Connection con = null;
        Statement stmt = null;
        ResultSet rset = null;

        try {

            System.out.print("\n");
            System.out.print("+-----+\n");
            System.out.print("| USING DriverManager CLASS      |\n");
            System.out.print("+-----+\n");
            System.out.print("\n");

            System.out.print("  Loading JDBC Driver  -> " + driverClass +
"\n");

            DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());

```

```

        System.out.print("  Connecting to          -> " + connectionURLThin
+ "\n");
        con = DriverManager.getConnection(connectionURLThin, userID,
userPassword);
        System.out.print("  Connected as          -> " + userID + "\n");

        System.out.print("  Creating Statement...\n");
        stmt = con.createStatement ();

        System.out.print("  Opening ResultSet...\n");
        rset = stmt.executeQuery(queryString);

        while (rset.next()) {
            System.out.println("  Results...");
            System.out.println("      User          -> " +
rset.getString(1));
            System.out.println("      Sysdate       -> " +
rset.getString(2));
        }

        System.out.print("  Closing ResultSet...\n");
        rset.close();

        System.out.print("  Closing Statement...\n");
        stmt.close();

    } catch (SQLException e) {

        e.printStackTrace();

        if (con != null) {
            try {
                con.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
    }

} finally {

    if (con != null) {
        try {
            System.out.print("  Closing down all connections...\n\n");

```

```

        con.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

}

}

/**
 * The following method provides an example of how to connect to a
database
 * by registering the JDBC driver using the jdbc.drivers property. The
 * DriverManager will load all classes listed in this property
 * automatically. This alternative works well for applications with a
 * command-line interface, but might not be so useful in GUI applications
 * and applets. This is because you can specify properties at the command
 * line.
 */
public void jdbcDriversProperty() {

    Connection con = null;
    Statement stmt = null;
    ResultSet rset = null;

    try {

        System.out.print("\n");
        System.out.print("+-----+\n");
        System.out.print("| USING jdbc.drivers PROPERTY  |\n");
        System.out.print("+-----+\n");
        System.out.print("\n");

        System.out.print("  Loading JDBC Driver  -> " + driverClass +
"\n");
        System.setProperty("jdbc.drivers", driverClass);

        System.out.print("  Connecting to          -> " + connectionURLThin
+ "\n");
        con = DriverManager.getConnection(connectionURLThin, userID,
userPassword);

```

```

        System.out.print("  Connected as          -> " + userID + "\n");

        System.out.print("  Creating Statement...\n");
        stmt = con.createStatement ();

        System.out.print("  Opening ResultSet...\n");
        rset = stmt.executeQuery(queryString);

        while (rset.next()) {
            System.out.println("  Results...");
            System.out.println("      User          -> " +
rset.getString(1));
            System.out.println("      Sysdate      -> " +
rset.getString(2));
        }

        System.out.print("  Closing ResultSet...\n");
        rset.close();

        System.out.print("  Closing Statement...\n");
        stmt.close();

    } catch (SQLException e) {

        e.printStackTrace();

        if (con != null) {
            try {
                con.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
    }

} finally {

    if (con != null) {
        try {
            System.out.print("  Closing down all connections...\n\n");
            con.close();
        } catch (SQLException e) {

```

```

        e.printStackTrace();
    }
}

}

}

/**
 * The following method provides an example of how to connect to a
database
 * by registering the JDBC driver using the Class.forName() method. This
 * complex expression is a tool for dynamically creating an instance of
 * a class when you have some variable representing the class name.
Because
 * a JDBC driver is required to register itself whenever its static
 * initializer is called, this expression has the net effect of
registering
 * your driver for you.
 *
 * NOTE: When using Class.forName("classname"), the JVM is supposed
to
 * be sufficient. Unfortunately, some Java virtual machines do
 * not actually call the static initializer until an instance
of
 * a class is created. As a result, newInstance() should be
 * called to guarantee that the static initializer is run for
 * all virtual machines.
 *
 * This method is by far the BEST in that it does not require hardcoded
 * class names and it runs well in all Java environments. In real-world
 * applications, you should use this method along with a properties file
 * from which you load the name of the driver.
 *
 */
public void classForName() {

    Connection con = null;
    Statement stmt = null;
    ResultSet rset = null;

    try {

```

```

System.out.print("\n");
System.out.print("+-----+\n");
System.out.print("| USING Class.forName()           |\n");
System.out.print("+-----+\n");
System.out.print("\n");

System.out.print("  Loading JDBC Driver  -> " + driverClass +
"\n");

Class.forName(driverClass).newInstance();

System.out.print("  Connecting to           -> " + connectionURLThin
+ "\n");
con = DriverManager.getConnection(connectionURLThin, userID,
userPassword);
System.out.print("  Connected as           -> " + userID + "\n");

System.out.print("  Creating Statement...\n");
stmt = con.createStatement ();

System.out.print("  Opening ResultSet...\n");
rset = stmt.executeQuery(queryString);

while (rset.next()) {
    System.out.println("  Results...");
    System.out.println("      User           -> " +
rset.getString(1));
    System.out.println("      Sysdate        -> " +
rset.getString(2));
}

System.out.print("  Closing ResultSet...\n");
rset.close();

System.out.print("  Closing Statement...\n");
stmt.close();

} catch (ClassNotFoundException e) {

    e.printStackTrace();

} catch (InstantiationException e) {

```

```

        e.printStackTrace();

    } catch (IllegalAccessException e) {

        e.printStackTrace();

    } catch (SQLException e) {

        e.printStackTrace();

        if (con != null) {
            try {
                con.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }

    } finally {

        if (con != null) {
            try {
                System.out.print(" Closing down all connections...\n\n");
                con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

    }

}

/**
 * The following method provides an example of how to connect to a
database
 * using the OCI JDBC Driver.
 *
 */
public void jdbcOCIDriver() {

```

```

Connection con = null;
Statement stmt = null;
ResultSet rset = null;

try {

    System.out.print("\n");
    System.out.print("+-----+\n");
    System.out.print("| USING OCI Driver          |\n");
    System.out.print("+-----+\n");
    System.out.print("\n");

    System.out.print(" Loading JDBC Driver -> " + driverClass +
"\n");
    Class.forName(driverClass).newInstance();

    System.out.print(" Connecting to          -> " + connectionURLOCI +
"\n");
    con = DriverManager.getConnection(connectionURLOCI, userID,
userPassword);
    System.out.print(" Connected as          -> " + userID + "\n");

    System.out.print(" Creating Statement...\n");
    stmt = con.createStatement ();

    System.out.print(" Opening ResultsSet...\n");
    rset = stmt.executeQuery(queryString);

    while (rset.next()) {
        System.out.println(" Results...");
        System.out.println("      User          -> " +
rset.getString(1));
        System.out.println("      Sysdate       -> " +
rset.getString(2));
    }

    System.out.print(" Closing ResultSet...\n");
    rset.close();

    System.out.print(" Closing Statement...\n");
    stmt.close();

```

```

    } catch (ClassNotFoundException e) {

        e.printStackTrace();

    } catch (InstantiationException e) {

        e.printStackTrace();

    } catch (IllegalAccessException e) {

        e.printStackTrace();

    } catch (SQLException e) {

        e.printStackTrace();

        if (con != null) {
            try {
                con.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
    } finally {
        if (con != null) {
            try {
                System.out.print(" Closing down all connections...\n\n");
                con.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}

/**
 * Sole entry point to the class and application.
 * @param args Array of String arguments.
 * @exception java.lang.InterruptedException
 *             Thrown from the Thread class.
 */
public static void main(String[] args)

```

```

        throws java.lang.InterruptedException {

        ConnectionExample conExample = new ConnectionExample();

        conExample.className();

        Thread.sleep(5000);

        conExample.jdbcDriversProperty();

        Thread.sleep(5000);

        conExample.driverManager();

        Thread.sleep(5000);

        conExample.jdbcOCIDriver();

    }

}

```

Calling Procedure from JDBC

```

import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.Types;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import oracle.jdbc.driver.OracleCallableStatement;
import oracle.jdbc.driver.OracleTypes;
import oracle.jdbc.driver.OracleResultSet;

/**
 *
 *-----
 * This class provides an example on the use of REF Cursors to execute SQL
from
 * a JDBC program, simulating dynamic SQL.
 *

```

```

* =====
* DYNAMIC SQL IN JAVA
* =====
*
* JDBC provides APIs for executing Dynamic SQL using PreparedStatement.
* For example:
*
*     PreparedStatement pstmt;
*     pstmt=conn.prepareStatement("SELECT name FROM dept WHERE deptno > ?");
*     pstmt.setInt(1,104);
*     ResultSet c1;
*     c1=pstmt.executeQuery();
*     pstmt.setInt(1,10)
*     while (c1.next ()) {System.out.println (c1.getInt(1));}
*
*
* =====
* REF CURSOR
* =====
*
* Another option of executing dynamic SQL from JDBC is provided in this
* example. Keep in mind that this example will only work with Oracle8i and
* higher. In this case, the procedure uses a PL/SQL procedure which returns
* a REF CURSOR.
*
* A REF CURSOR is similar a pointer in the C programming language. It points
* to rows retrieved from the database using a PL/SQL cursor. The example I
* provide in this class uses a REF CURSOR to point to the result set
* returned by a SELECT statement that retrieves rows from the DEPT table
* using a PL/SQL cursor.
*
* In this example, I call a PL/SQL procedure named "get_dept_ref_cursor"
which
* returns a variable of type "t_ref_cursor".
*
* Stored procedures can return user-defined types, or cursor variables, of
the
* REF CURSOR category. This output is equivalent to a database cursor or a
* JDBC result set. A REF CURSOR essentially encapsulates the results of a
* query.
*
* Advantages of using a REF CURSOR are:

```

```

*
*      1.) Code Reusability
*
*      The same package procedure could be used for other Java and non-
Java
*      applications.
*
*      2.) Load Balancing.
*
*
* =====
* OracleCallableStatement CLASS
* =====
*
* You will notice that in this example, I use an OracleCallableStatement
class
* in place of our typical CallableStatement class. This class defines a
method
* named getCursor() that enables you to read Oracle cursors.
*
*
* =====
* OracleTypes CLASS
* =====
*
* You will also notice the oracle.jdbc.driver.OracleTypes is also used
* when registering the OutParameter. This class defines those special TYPEs
* offered by the Oracle database. This class is similar to java.sql.Types.
*
*
* =====
* NOT USING OracleCallableStatement and OracleResultSet CLASS
* =====
*
* Note that you are not required to use the OracleCallableStatement and
* OracleResultSet classes; you could use the regular CallableStatement
* and ResultSet classes found in java.sql. However, you will need to
* use the getObject() method to read the Oracle cursor. An example of this is
* provided in this example with the performRefCursor2() method.
*
*
*
*
-----

```

```

*
* NOTE: Opening a REF CURSOR for a statement present in a variable is only
*       supported with Oracle8i and higher.
*
* NOTE: In order to successfully use this class, you will need to run the
*       create_all_ddl.sql file included in the same section this example
class
*       is located.
*
*
-----

*/

public class RefCursorExample {

    final static String driverClass    = "oracle.jdbc.driver.OracleDriver";
    final static String connectionURL =
"jdbc:oracle:thin:@localhost:1521:MYDB";
    final static String userID        = "scott";
    final static String userPassword  = "tiger";
    Connection    con                = null;

    /**
    * Construct a RefCursorExample object. This constructor will create an
Oracle
    * database connection.
    */
    public RefCursorExample() {

        try {

            System.out.print("  Loading JDBC Driver  -> " + driverClass +
"\n");
            Class.forName(driverClass).newInstance();

            System.out.print("  Connecting to          -> " + connectionURL +
"\n");
            this.con = DriverManager.getConnection(connectionURL, userID,
userPassword);
            System.out.print("  Connected as        -> " + userID + "\n\n");

        } catch (ClassNotFoundException e) {
            e.printStackTrace();

```

```

        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * This method is used to return a REF CURSOR that will be used to
    retrieve
     * data from a result set. This REF CURSOR is retrieved by the JDBC
    program
     * into a ResultSet.
     *
     * This method Uses the OracleCallableStatement and OracleResultSet
    classes.
     */
    public void performRefCursor() {

        OracleCallableStatement oraCallStmt = null;
        OracleResultSet deptResultSet = null;

        System.out.println("Using OracleCallableStatement / OracleResultSet");
        System.out.println("-----");

        try {

            oraCallStmt = (OracleCallableStatement) con.prepareCall(
                "{? = call ref_cursor_package.get_dept_ref_cursor(?)}"
            );
            oraCallStmt.registerOutParameter(1, OracleTypes.CURSOR);
            oraCallStmt.setInt(2, 104);
            oraCallStmt.execute();

            deptResultSet = (OracleResultSet) oraCallStmt.getCursor(1);

            while (deptResultSet.next()) {
                System.out.println(
                    " - " +

```

```

        deptResultSet.getString(2) + " (" +
deptResultSet.getInt(1) + "), " +
        deptResultSet.getString(3)
    );
}
System.out.println();

oraCallStmt.close();

} catch (SQLException e) {

    e.printStackTrace();

}

}

/**
 * This method is used to return a REF CURSOR that will be used to
retrieve
 * data from a result set. This REF CURSOR is retrieved by the JDBC
program
 * into a ResultSet.
 *
 * This method Uses the the regular CallableStatement and ResultSet
classes.
 */
public void performRefCursor2() {

    CallableStatement cstmt = null;
    ResultSet          rset = null;

    System.out.println("Using CallableStatement / ResultSet");
    System.out.println("-----");

    try {

        cstmt = con.prepareCall(
            "{? = call ref_cursor_package.get_dept_ref_cursor(?)}"
        );
        cstmt.registerOutParameter(1, OracleTypes.CURSOR);
        cstmt.setInt(2, 104);

```

```
        pstmt.execute();

        rset = (ResultSet) pstmt.getObject(1);

        while (rset.next()) {
            System.out.println(
                " - " +
                rset.getString(2) + " (" + rset.getInt(1) + "), " +
                rset.getString(3)
            );
        }
        System.out.println();

        pstmt.close();

    } catch (SQLException e) {

        e.printStackTrace();

    }

}

/**
 * Close down Oracle connection.
 */
public void closeConnection() {

    try {
        System.out.print(" Closing Connection...\n");
        con.close();

    } catch (SQLException e) {

        e.printStackTrace();

    }

}
```

```
/**
 * Sole entry point to the class and application.
 * @param args Array of String arguments.
 * @exception java.lang.InterruptedException
 *           Thrown from the Thread class.
 */
public static void main(String[] args)
    throws java.lang.InterruptedException {

    RefCursorExample mainPrg = new RefCursorExample();
    mainPrg.performRefCursor();
    mainPrg.performRefCursor2();
    mainPrg.closeConnection();

}

}
```