Oracle9*i* New Features for Administrators

Student Guide • Volume 2

D11318GC11
Edition 1.1
July 2001
D33453

ORACLE®

Authors

David Austin
Ric van Dyke
Jean-Francois Verrier
Michael Möller
Lex de Haan

Technical Contributors and Reviewers

Harald van Breederode
Bruce Ernst
Lothar Flatz
Scott Gossett
Arturo Gutierrez
Merrill Holt
Magnus Isaksson
Martin Jensen
Sushil Kumar
Stefan Lindblad
Russ Lowenthal
Patricia McElroy
Sujatha Muthulingam
Peter Sharman
Sabine Teuber

Publisher

May Lonn Villareal

Contents

13

File Management

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- **Understand the concept and benefits behind Oracle-Managed Files (OMF)**
- **Create and manage OMF files**
- **Use SQL syntax to remove associated OS files when removing a non-OMF tablespace from the database**
- **Create and alter default temporary tablespaces**

ORACLE

Oracle-Managed Files Overview

- **Oracle creates and deletes files as needed for tablespaces, temp files, online logs, control files.**
- **You only specify the OS directory to be used for each file type:**
  - **Reduce corruption caused by administrators**
  - **Reduce wasted disk space consumed by obsolete files**
  - **Simplify creation of test and development databases**
  - **Make development of portable applications easier**
- **OMF and non-OMF files can coexist**

ORACLE

Oracle-Managed Files Overview

The Oracle-Managed Files (OMF) goal is to simplify the administration of Oracle databases by eliminating the need for administrators to manage the files composing an Oracle database directly. Administrators specify operations in terms of database objects rather than by file names. Oracle internally uses standard file system interfaces to create and delete files as needed for tablespaces, temp files, online logs, and control files. Administrators only specify the OS file system directory to be used for a particular type of file. The Oracle server ensures that a unique file (an Oracle-managed file) is used and deleted when no longer needed.

OMF has the following advantages:

- Reduces corruption caused by administrators specifying the wrong file
- Reduces wasted disk space consumed by obsolete files
- Simplifies creation of test and development databases
- Makes development of portable third-party tools easier because it eliminates the need to put operating system-specific file names in SQL scripts.

Note: Using Oracle-Managed Files does not eliminate any existing functionality. Existing databases are able to operate as they always have. New files can be created as managed files while old ones are administered in the old way. Thus, a database can have a mixture of Oracle-managed and unmanaged files.

Also, this feature does not affect the creation or naming of administrative files such as trace files, audit files, alert files, and core files.

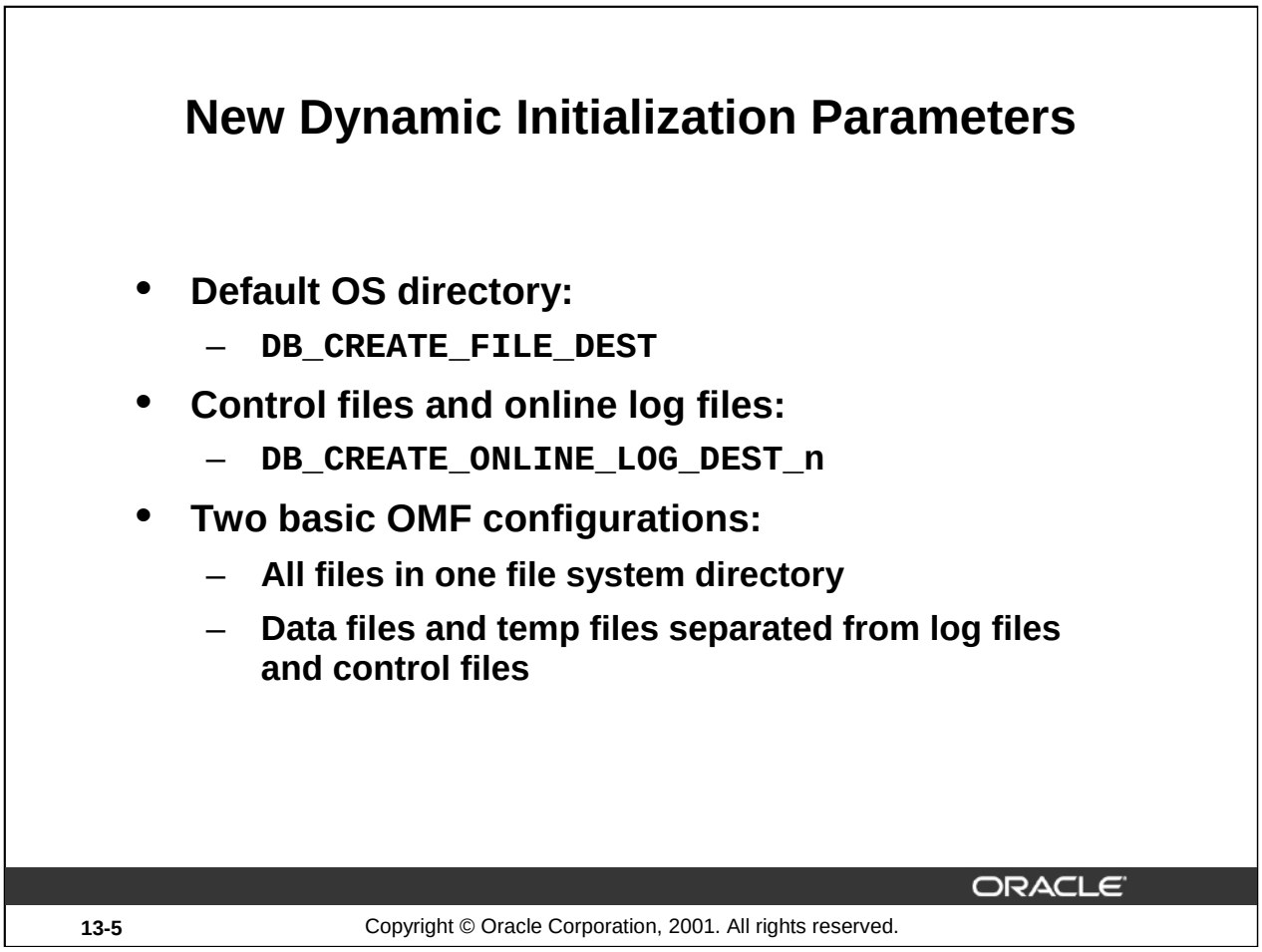
Who Can Use Oracle-Managed Files?

- **Low end databases**
- **Databases that are supported by the following:**
  - **A logical volume manager that supports striping/RAID and dynamically extensible logical volumes**
  - **A file system that provides large, extensible files**

ORACLE

Who Can Use Oracle-Managed Files?

The OMF feature is not intended to ease administration of systems that use raw disks. This feature provides better integration with operating system functionality for disk space allocation. Because there is no operating system support for allocation of raw disks (it is done manually), this feature cannot help. On the other hand, because OMF requires that you use the operating system file system (unlike raw disks), you lose control over how files are laid out on the disks, and thus, you lose some tuning ability.



New Initialization Parameters

The default file system directory is the location where Oracle creates database files (data files, temp files, control files, and redo log files) when no file specification has been given in a corresponding creation operation. The DBA defines this location with the `DB_CREATE_FILE_DEST` initialization parameter.

Two basic OMF configurations are envisioned:

1. A single initialization parameter, `DB_CREATE_FILE_DEST`, is specified. This is probably a low end database. All the data files, temp files, control files and online logs are created in the same file system location.
2. `DB_CREATE_FILE_DEST` is set to give the default location for data files and temp files. `DB_CREATE_ONLINE_LOG_DEST_n` is set to give the default locations for online logs and control files. This configuration provides good separation of data files and online logs, and optional Oracle multiplexing of online logs and control files (`DEST_1`, `DEST_2`, …, `DEST_5`).

Note: Some operations already do not require a specific file creation to be supplied (For example `CREATE DATABASE`). These operations have a port-specific default creation location. The new initialization parameters provide a way to override that default.

Also, it is possible to dynamically modify the above parameters with `ALTER SESSION` or `ALTER SYSTEM` commands.

OMF Example

1. **Setting the initialization parameters**

```
DB_CREATE_FILE_DEST         = '/u01/oradata/'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata/'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata/'
```

2. **Creating a database**

```
CREATE DATABASE;
```

Note: Every created data file and log file is 100 MB by default, with auto extensibility set, with unlimited size for data files

ORACLE

OMF Example

Assume that you want to create a database where the data files and temp files are separated from the online logs and control files. The online logs and control files are Oracle multiplexed.

The DB_CREATE_FILE_DEST parameter sets the default file system directory for the data files, and temp files (/u01/oradata). DB_CREATE_ONLINE_LOG_DEST_1 and DB_CREATE_ONLINE_LOG_DEST_2 set the default file system directories for online log and control file creation. Each online log and control file is multiplexed across the two directories (/u02/oradata and /u03/oradata).

Once the initialization parameters are set, you can create the database with the CREATE DATABASE statement.

Because a DATAFILE clause is not present and the DB_CREATE_FILE_DEST initialization parameter is set, the system tablespace data file is created in the default file system (/u01/oradata). The file is auto extensible with an initial size of 100 MB and an unlimited maximum size. The file is an Oracle managed file.

Similarly, because a LOGFILE clause is not present, two online log groups are created. Each log group has two members, with one member in the DB_CREATE_ONLINE_LOG_DEST_1 location and the other member in the DB_CREATE_ONLINE_LOG_DEST_2 location. The log files are created with a size of 100 MB. The log file members are Oracle-managed files.

OMF File Names Structure

- **OMF files comply with OFA**
  - **Control files** `ora_%u.ctl`
  - **Redo log files** `ora_%g_%u.log`
  - **Data files** `ora_%t_%u.dbf`
  - **Temporary data files** `ora_%t_%u.tmp`
- **You can manipulate existing OMF files like normal files in SQL commands.**
- **In order to determine if a database file is OMF:**
  - **Look at the `alert.log` file**
  - **If the `alert.log` is no longer available, look at the file name**

ORACLE

OMF File Names Structure

OMF file names are accepted in SQL commands wherever a file name is used to identify an existing file. That is why there is no additional flag in the data dictionary to make the distinction between OMF and non-OMF files. Only the name can tell. For many commands there is an alternate method for identifying the file (a file number, for example) so that the name does not have to be typed.

If a statement that creates an Oracle-managed file finds an error or does not complete due to some failure, then any Oracle-managed files created by the statement are automatically deleted as part of the recovery of the error or failure. However, because of the large number of potential errors that can occur with file systems and storage subsystems, there can be situations where you must manually remove the files using operating system commands. When an Oracle-managed file is created, its filename is written to the alert file. This information can be used to find the file if it is necessary to manually remove the file.

On the above slide the meaning of the wildcard characters is the following:

- `%u` is an 8 character string that guarantees uniqueness
- `%t` is the tablespace name, truncated if necessary to fit into the maximum length file name. The tablespace name is placed before the uniqueness string so that all data files for a tablespace will appear next to each other in an alphabetic file listing
- `%g` is the log file group number
- "`ora_`" identifies the file as an Oracle-managed file

OMF File Names Structure (continued)

Note: The above discussion is based on Solaris systems, but on other platforms the names should be similar, subject to the constraints of the platform's naming rules. Undo files do not have a special extension as with temp files; they are considered just like any other data files.

Managing OMF Control Files: Database Creation

- **If no `CONTROL_FILES` parameter is specified, Oracle uses (in order of precedence):**
  - **`DB_CREATE_ONLINE_LOG_DEST_n`**
  - **`DB_CREATE_FILE_DEST`**
- **If none of the above are specified, the default destination is used (non-OMF).**
- **After database creation, create a `CONTROL_FILES` entry in the initialization parameter file (see Note)**
- **If there is a `CONTROL_FILES` parameter specified, then the behavior is unchanged**

ORACLE

Managing OMF Control Files

Here are the two cases:

- If the `CONTROL_FILES` initialization parameter is not set when the database is created, a control file is created in each of the `DB_CREATE_ONLINE_LOG_DEST_n` locations. If those parameters are not specified, then Oracle tries to use the `DB_CREATE_FILE_DEST` parameter. If not specified, then Oracle creates a non-OMF control file in the default directory depending on the operating system used. If Oracle creates an OMF control file, you are then required to add the `CONTROL_FILES` parameter, set to the generated file names, in the initialization parameter file.

  Note: If a persistent parameter file is used (new in Oracle9*i*), the `CONTROL_FILES` parameter is automatically set and saved when the database is created.
- If the `CONTROL_FILES` parameter is set in the initialization parameter file, then depending on the specified names (OMF format or not), the created control files can be OMF or not.

Impact of OMF on CREATE CONTROLFILE Command

- If the DB_CREATE_… parameters are specified then the created control file will be OMF.
- You must create a CONTROL_FILES entry in the INIT.ORA file unless a server parameter file is used.
- You must supply filenames in the DATAFILE clause even for existing OMFs.
- Depending on the [NO]RESETLOGS clause you must:
  - Supply log file name if using NORESETLOGS
  - Let Oracle create OMF redo logs if using RESETLOGS

ORACLE

Impact of OMF on CREATE CONTROLFILE Command

When you issue the CREATE CONTROLFILE statement, a control file is created (or reused, if REUSE is specified) in the files specified by the CONTROL_FILES initialization parameter. If the CONTROL_FILES parameter is not set, then the control file is created in the default control file destinations. In order of precedence, the default destination is:

- DB_CREATE_ONLINE_LOG_DEST_n
- DB_CREATE_FILE_DEST, if no DB_CREATE_ONLINE_LOG_DEST_n specified
- If neither of the above parameters are specified, then the default location is used as in previous versions (this is system dependent), and the control file is non-OMF.

If Oracle creates an OMF control file, and there is a server parameter file, then Oracle creates a CONTROL_FILES initialization parameter for the server parameter file. If there is no server parameter file, then you must create a CONTROL_FILES entry in the INIT.ORA file. If the data files in the database are Oracle-managed files, then the Oracle generated filenames for the files must be supplied in the DATAFILE clause of the statement.

Impact of OMF on CREATE CONTROLFILE Command (continued)

If the online redo log files are Oracle-managed files, then the [NO]RESETLOGS keyword determines what can be supplied in the LOGFILE clause:

- NORESETLOGS: The Oracle-generated filenames for the Oracle-managed online redo log files must be supplied in the LOGFILE clause.
- RESETLOGS: The online redo log file names can be supplied as with the CREATE DATABASE statement.

Managing OMF Redo Log Files

- **You can add a complete group with the `ALTER DATABASE ADD LOGFILE` command (no file specification, `init.ora` parameters are used).**

```
ALTER DATABASE ADD LOGFILE;
```

```
ALTER DATABASE ADD LOGFILE SIZE 10M;
```

- **You continue to add or remove individual members by specifying full file names.**
- **If you drop a group, all the corresponding OMF files are deleted at the OS level as well.**
- **Archived logs cannot be OMF files.**

ORACLE

Managing OMF Redo Log Files

You can use the `ALTER DATABASE ADD LOGFILE` statement to add a new group to your current online redo log. The filename in the `ADD LOGFILE` clause is optional if you are using Oracle-managed files. If a filename is not provided, then a redo log file is created in the default log file destination. In order of precedence, the default destination is defined as follows:

- If `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then an Oracle-managed log file member is created in each directory specified in the parameters (up to `MAXLOGMEMBERS` for the database)
- If the `DB_CREATE_FILE_DEST` initialization parameters specified, and no `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters are specified, then an Oracle-managed log file member is created in the directory specified in the parameter.

If a filename is not provided and you have not provided one of the initialization parameters required for creating Oracle-managed files, then the statement returns an error. The default size for an Oracle-managed log file is 100 MB but as the second example above shows, you can override this default.

Online redo log file members continue to be added and dropped by specifying complete filenames (they are considered as OMF or not depending on the files names).

Archiving of log files works as it does today using `LOG_ARCHIVE_DEST_n` and `LOG_ARCHIVE_FORMAT` initialization parameters. The archived logs are *not* OMF.

Managing OMF Tablespaces

- The `CREATE TABLESPACE` command has been modified: `DATAFILE` clause no longer mandatory.

```
CREATE TABLESPACE TBS1 [DATAFILE SIZE 200M];
```

- By default, OMF files are 100 MB in size and set to auto extend with unlimited size.
- When you drop a tablespace, all OMF files are also deleted at the OS level.
- You can also add OMF files to a tablespace.

```
ALTER TABLESPACE TBS1 ADD DATAFILE;
```

- The default file system directory can be dynamically changed.

```
ALTER SYSTEM SET DB_CREATE_FILE_DEST='/oradata/';
```

ORACLE

Managing OMF Tablespaces

The default storage for all future tablespace is the location specified by the `DB_CREATE_FILE_DEST` initialization parameter. If not specified, an error is raised during OMF tablespace creation.

The `CREATE TABLESPACE` command has been modified so that the `DATAFILE` clause is no longer required. The data file is created in the file system specified by the initialization parameter `DB_CREATE_FILE_DEST`.

By default, OMF data files are created with an initial size of 100 megabytes and they are auto extensible with an unlimited maximum size. As shown in the first example above, you can change those default values. The file names are internally generated and can be seen by selecting the usual views. When the tablespace is dropped, the Oracle-managed files for the tablespace are automatically removed.

Oracle does not automatically create a new data file. More space can be added to the tablespace by adding another Oracle-managed data file. The `ADD DATAFILE` command has been modified so that a file specification is no longer required. The default file system can be changed dynamically by changing the `DB_CREATE_FILE_DEST` initialization parameter using `ALTER SYSTEM` or `ALTER SESSION` commands. This does not change any existing data files. It only affects future creations.

Note: Everything true for normal tablespaces is also true for `[DEFAULT]TEMPORARY` and `UNDO` tablespaces. So `CREATE TEMPORARY|UNDO TABLESPACE`, and `ALTER TABLESPACE ADD TEMPFILE` commands no more need a file specification.

OMF Examples

```
SQL> create tablespace tbs02
  2  datafile size 300m, size 300m;
```

```
SQL> alter database add logfile size 400m;
```

```
SQL> alter database recover datafile
  2  'ora_tbs1_2ixfh90q.dbf';
```

```
SQL> alter table emp allocate extent
  2  (datafile 'ora_tbs1_2ixfh90q.dbf');
```

```
SQL> alter database create datafile
  2  '/u03/oradata/payroll/ora_tbs1_sd84oqy9.dbf'
  3  as new;
```

ORACLE

OMF Examples

Here are some examples of OMF commands:

- The first example creates two OMF data files in the default directory but with specified size.
- The second example adds an OMF log file to the default directory with a 400 MB size.
- The third example recovers the specified OMF data file.
- The fourth example adds one extent to the table in the specified file.
- The last example creates a new OMF data file in the default data file destination. The old OMF file, if it exists, will be deleted.

Note: The `AS NEW` clause can be given to create an Oracle-managed data file in the default data file location. If the `AS NEW` clause is specified and the old file is an Oracle-managed file then the old file is deleted.

It should not be necessary for an administrator to use this command because RMAN uses this as a means of restoring a file when no backup exists. This would normally happen when the file has been created since the last scheduled backup of the tablespace.

OMF and Standby Databases

- New STANDBY_FILE_MANAGEMENT parameter
  - AUTO: File adds on the primary are recreated on the standby using the same file names
  - MANUAL: Any OMF file adds on the primary are reproduced on the standby with different names
- Path can be different depending on DB_FILE_NAME_CONVERT
- DB_CREATE_FILE_DEST can be specified but only relevant after Standby activation (future files) and possibly when adding OMF data file on the primary
- Redo of tablespaces drop causes all corresponding standby OMF files to be purged

ORACLE

OMF and Standby Databases

A new standby initialization parameter, STANDBY_FILE_MANAGEMENT, is added. Its default value is FALSE and can have the following values:

- AUTO: A redo of a file add (OMF and non-OMF) will re-create the file with the same name if it does not already exist. The file name conversion parameters are applied first (if any). Also, other operations that add, remove or rename files will not be allowed.
- MANUAL: Redo for a non-OMF file is handled as it is today: an entry is created in the control file, but a file is not created. Redo for an OMF file add causes an OMF file with a new name to be created in the standby's default destination (DB_CREATE_FILE_DEST). File name conversion parameters are not applied to the file name.

Redo of a tablespace drop (from the primary site), causes the tablespace's OMF files to be purged. In this case, non-OMF files (if any) are not purged for compatibility reasons.

ACTIVATE STANDBY DATABASE, CLEAR LOGFILE, and OPEN RESETLOGS create new OMF log files in the default destinations if there are any OMF log files in the control file, and the control file is a standby.

Except for the above case, if the DB_CREATE_FILE_DEST parameter is specified in the Standby INIT.ORA file file, Oracle uses it whenever you activate the standby. Oracle uses this parameter to know where to create the files now and in the future.

Automatically Drop Non-OMF Data Files

- **A new option is added to the `DROP TABLESPACE` command to delete the OS files associated with the tablespace:**

```
DROP TABLESPACE TBS1 INCLUDING CONTENTS
AND DATAFILES [CASCADE CONSTRAINTS];
```

- **A new option is added to the `ALTER DATABASE TEMPFILE DROP` command to delete the corresponding OS files:**

```
ALTER DATABASE TEMPFILE '…'
DROP INCLUDING DATAFILES;
```

- **No specific redo is generated for OS files removal.**

ORACLE

Automatically Drop Non-OMF Data Files

The goal of this new feature is to simplify the administration of Oracle databases by providing administrators with an option to automatically remove a tablespace's operating system files when removing the tablespace from the database. This is working even if the tablespace does not contain OMF files.

Changes have been made to the `DROP TABLESPACE` and `ALTER DATABASE TEMPFILE` commands in order to support this feature:

`INCLUDING CONTENTS AND DATAFILES`: Deletes the contents and operating system files of the tablespace. A message is written to the alert log for each file deleted. The command succeeds even if an operating system error prevents the deletion of a file. A message describing the error is written to the alert log.

`DROP INCLUDING DATAFILES`: Drops the temp file from the database and deletes the operating system file. The tablespace remains. A message iswritten to the alert log for each file deleted. The command succeeds even if an operating system error prevents the deletion of the file. A message describing the error is written to the alert log.

In addition, rollback of some operations that create operating system files will remove the operating system files. This includes: `ALTER TABLESPACE ADD DATAFILE`, `CREATE TABLESPACE`, and `CREATE TEMPORARY TABLESPACE`.

Note: No specific redo is generated for OS file removal. So, the removal of the operating system file is not repeated in a standby database.

Default Temporary Tablespace

- **The Oracle9*i* default temporary tablespace feature allows you to specify a database-wide default temporary tablespace.**
- **The default temporary tablespace can be created using the `CREATE DATABASE` or `ALTER DATABASE` commands.**
- **When specified with the `CREATE DATABASE` command, the default temporary tablespace is locally managed and cannot be specified as `SYSTEM`.**

ORACLE

Default Temporary Tablespace

The default temporary tablespace feature specifies that a temporary tablespace is to be created at database creation time. This tablespace is used as the default temporary tablespace for users who are not otherwise assigned a temporary tablespace.

Users can be explicitly assigned a default temporary tablespace in the `CREATE USER` statement. But if no temporary tablespace is specified, they default to using the `SYSTEM` tablespace. It is not good practice to allow users to store even temporary data in the `SYSTEM` tablespace. To avoid this problem, and to avoid the need to assign every user a default temporary tablespace at `CREATE USER` time, you can use this feature.

Note: A default temporary tablespace can be locally or dictionary managed and can be defined with the `CREATE TEMPORARY TABLESPACE` command or with the `CREATE TABLESPACE … TEMPORARY` command.

Default Temporary Tablespace

- If a default temporary tablespace is not defined at database creation, the SYSTEM tablespace is the default temporary tablespace and a warning in ALERT.LOG is made.
- Users not explicitly assigned to a temporary tablespace are assigned to the default temporary tablespace.
- You can query the DATABASE_PROPERTIES view to retrieve the current default temporary tablespace:

```
SQL> SELECT property_value
  2  FROM   database_properties
  3  WHERE  property_name =
  4         'DEFAULT_TEMP_TABLESPACE';
```

ORACLE

Default Temporary Tablespace

Oracle strongly encourages the creation of a default temporary tablespace when creating the database in order to move away from using the SYSTEM tablespace for temporary data.

Users can obtain the name of the current default temporary tablespace using the DATABASE_PROPERTIES view. The PROPERTY_NAME column contains the value "DEFAULT_TEMP_TABLESPACE" and the PROPERTY_VALUE column contains the default temporary tablespace name.

Create Default Temporary Tablespace at Database Creation Time

```
CREATE DATABASE db1 CONTROLFILE REUSE
  LOGFILE  'log1.log' SIZE 10M
  LOGFILE  'log2.log' SIZE 10M
  DATAFILE 'df1.dbf'  AUTOEXTEND ON
           'df2.dbf'  AUTOEXTEND ON
                      NEXT 10M MAXSIZE UNLIMITED
  DEFAULT TEMPORARY TABLESPACE dts1
        TEMPFILE 'dts_1.f ' SIZE 60M
        EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M;
```

ORACLE

Create Default Temporary Tablespace at Database Creation Time

You can specify the DEFAULT TEMPORARY TABLESPACE clause of the CREATE DATABASE command to create a default temporary tablespace for the database. Oracle assigns to this temporary tablespace any users for whom you do not specify a different temporary tablespace. If you do not specify this clause, the SYSTEM tablespace is the default temporary tablespace.

You can also specify how you want this tablespace to be managed. In the above example, a locally managed tablespace is used.

Note: The default temporary tablespace must have a standard block size.

Alter Default Temporary Tablespace

- **The ALTER DATABASE command has been extended to allow changing the default temporary tablespace:**

```
SQL> ALTER DATABASE db1
  2  DEFAULT TEMPORARY TABLESPACE dts2;
```

- **Users using the old default temporary tablespace are automatically reassigned to the new default temporary tablespace**

ORACLE

Alter Default Temporary Tablespace

When migrating from earlier versions of Oracle, the DBA is able to assign any temporary tablespace, either dictionary or locally managed, as the default temporary tablespace using the ALTER DATABASE command.

Note: When a user is explicitly assigned to the default temporary tablespace, this same user is automatically assigned to the new default temporary tablespace whenever you change the default temporary tablespace by using the ALTER DATABASE command.

Restrictions on Default Temporary Tablespace

- **The default temporary tablespace cannot be dropped until after a new default is made available.**
- **Altering the default temporary tablespace to a permanent tablespace is not allowed (except for SYSTEM tablespace).**
- **The default temporary tablespace cannot be taken offline.**

ORACLE

Restrictions on Default Temporary Tablespace

- Dropping a default temporary tablespace: Dropping the default temporary tablespace is not allowed until after a new default is made available. The ALTER DATABASE command must be used to change the default temporary tablespace to a new default. The old default temporary tablespace is then dropped only after a new default temporary tablespace is made available. Users assigned to the old default temporary tablespace are automatically reassigned to the new default temporary tablespace
- Changing to a permanent type versus temporary type: Because a default temporary tablespace must be either SYSTEM tablespace or a temporary type tablespace, changing the default temporary tablespace to a permanent type is not allowed.
- Taking default temporary tablespace offline: Tablespaces are taken offline to make that part of the database unavailable to other users (that is, an offline backup, maintenance, or making a change to an application that uses the tablespace). Because none of these situations apply to a temporary tablespace, taking a default temporary tablespace offline is not allowed.

Summary

In this lesson, you should have learned how to:

- **Create and manage Oracle-Managed Files (OMF)**
- **Use the new `INCLUDING` clauses to remove files at the OS level**
- **Assign a database wide default temporary tablespace**

ORACLE

Practice 13-1 Overview

This practice covers the following topics:

- **Creating and managing OMF and non-OMF files in the same database**
- **Using the `INCLUDING` clause of the `DROP TABLESPACE` command to remove non-OMF files**
- **Using the new `db_create_file_dest` initialization parameter**

ORACLE

Practice 13-2 Overview

This practice covers the following topics:

- **Assigning a default temporary tablespace to users**
- **Using the `DATABASE_PROPERTIES` view**

ORACLE

14

Tablespace Management

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- **Understand the concept of Automatic Undo Management**
- **Create and maintain undo tablespaces**
- **Create and properly use multiple block sizes within a database**

ORACLE

Automatic Undo Management

- **Automatic Undo Management simplifies and automates rollback segment management.**
- **You can manage rollback segments automatically or manually by choosing a rollback mode.**
- **The mode is set with the `UNDO_MANAGEMENT` initialization parameter with these values:**
  - **`AUTO`: The instance manages rollback segments automatically.**
  - **`MANUAL` (default): You must create and manage rollback segments manually.**

ORACLE

Automatic Undo Management (AUM)

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* records.

When a rollback statement is issued, undo records are used to undo changes that were made to the database by the uncommitted transaction. During instance recovery, undo records are used to undo any uncommitted changes applied from the redo log to the data files. 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.

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.

You cannot use both methods in the same database instance, although for migration purposes it is possible, for example, to create undo tablespaces in a database that is using rollback segments, or to drop rollback segments in a database that is using undo tablespaces.

Automatic Undo Management Concepts

- **Rollback data is managed by means of an undo tablespace**
- **For each instance, you must allocate enough disk space for the workload of the instance in the undo tablespace versus allocating a number of rollback segments in different sizes.**
- **The notion of a single SYSTEM rollback segment is retained.**
  - **Created automatically within the SYSTEM tablespace**
  - **Automatically managed and cannot be taken offline**

ORACLE

Automatic Undo Management Concepts

With this design, you allocate undo space in a single undo tablespace, instead of maintaining a set of statically allocated rollback segments. For each Oracle instance, you only have to allocate enough disk space for the workload in that instance in an undo tablespace. You no longer need to decide on the different number and sizes of rollback segments to create, and on how to assign transactions (of different sizes) strategically to individual rollback segments.

Also, you don't need to adjust the attributes of rollback segments, in order to juggle between undo block contention and space utilization issues.

Automatic Undo Management Concepts

- **Rollback segments are still used but are internally created and maintained, and are called undo segments.**
- **With automatic undo management, you cannot `CREATE`, `DROP`, or `ALTER` undo segments.**
- **Undo segments have the same structure as normal rollback segments but they:**
  - **Support automatic creation**
  - **Use a modified allocation policy compared to Oracle8*i***
  - **Support dynamic extents transfer**
- **SMON shrinks undo segments when needed.**

ORACLE

Automatic Undo Management Concepts

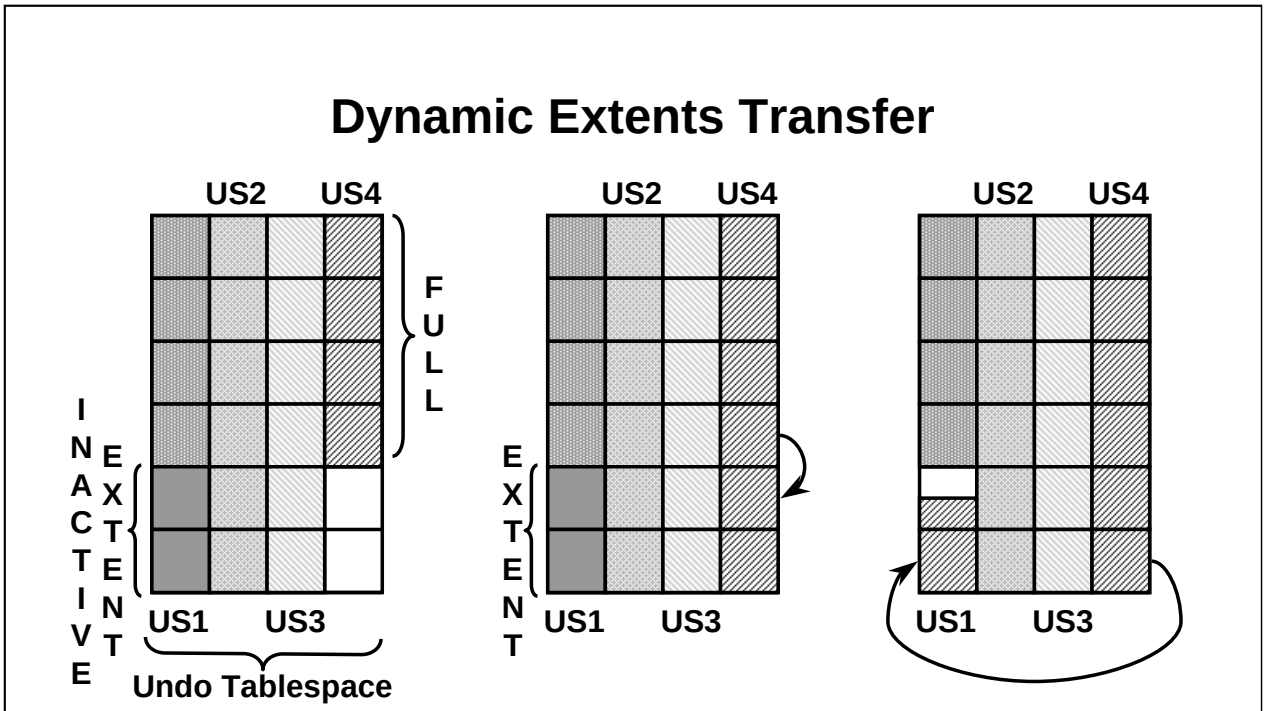
An undo tablespace is organized as a uniform bitmapped tablespace. It is composed of one or more files containing undo segments.

Each undo segment is assigned undo extents. The structure of these segments are internally the same as Oracle8*i* rollback segments, except that they cannot be manipulated by means of the rollback segment DDL statements. They are intentionally called undo segments here so that they will not be confused with rollback segments.

SMON is mainly responsible to shrink undo segments in the following situations:

- SMON will perform a shrink every 12 hours to remove undo space from idled undo segments
- SMON will be signaled by foreground processes to perform a shrink whenever they detect space pressure; that is, it has to steal space from another undo segment.

AUM mode supports automatic creation of undo segments. When the first DML operation in a transaction is executed, an undo segment is chosen. The AUM transaction-bind algorithm first attempts to bind one transaction per undo segment. If such a segment cannot be found, the system will attempt to online other undo segments in the current undo tablespace. If none is available, a new undo segment is created and brought online. If none of the above steps succeed (for example, an undo segment cannot be created because the undo tablespace is out of space), the transaction-bind algorithm then retries using the Oracle8*i* approach, find the least-used undo segment. In these cases, several transactions are executed in the same undo segment.



Dynamic Extents Transfer

AUM mode supports dynamic transfer of undo space between undo segments. If an executing transaction needs more undo space, space is reused either from the current undo segment or through an extension (such as today). If none of these steps results in enough free space for the transaction, inactive undo space is *stolen* from other undo segments. This dynamic scheme allows space to be reused efficiently, so that users do not see any ORA-30036 unless the undo tablespace is truly out of space.

Specifying the Mode for Undo Space Management

- **Starting in AUM mode:**
  - **UNDO_MANAGEMENT = AUTO**
  - **UNDO_TABLESPACE: Specifies a particular undo tablespace to be used. If it does not exists an error is raised. This parameter is dynamic.**
  - **If AUM is chosen and no undo tablespace is specified, Oracle9*i* uses the first available one. If none are available, Oracle9*i* uses the SYSTEM rollback segment.**
- **Starting in Rollback Segment Undo (RBU) Mode:**
  - **UNDO_MANAGEMENT = MANUAL (the default) or**
  - **Leave old initialization file unchanged**

ORACLE

Specifying the Mode for Undo Space Management

If you use the rollback segment method of managing undo space, you are said to be using the rollback segment undo (RBU) scheme. If you use the undo tablespace method, you are using the Automatic Undo Management (AUM) scheme. You determine whether to operate in RBU or AUM mode at instance startup using the UNDO_MANAGEMENT initialization parameter.

Starting an Instance in AUM Mode

The following initialization parameter setting causes the STARTUP command to start an instance in AUM mode: UNDO_MANAGEMENT = AUTO. The default value for this parameter is MANUAL.

When the instance starts up, it uses the undo tablespace specified by the UNDO_TABLESPACE dynamic initialization parameter. For example:

```
UNDO_TABLESPACE = undotbs01
```

The undo tablespace (in this example, undotbs01) must have already been created as explained in the next slides, or the STARTUP command will fail. If the UNDO_TABLESPACE parameter is omitted, the first available undo tablespace in the database is chosen. If there is no undo tablespace available, the instance starts, but uses the SYSTEM rollback segment. This is not recommended in normal circumstances, and an alert message is written to the alert file to warn the DBA that the system is running without an undo tablespace.

Note: If the initialization parameter file contains parameters relating to RBU mode, they are ignored.

Starting an Instance in Rollback Segment Undo (RBU) Mode

The following initialization parameter setting causes the STARTUP command to start an instance in RBU mode: UNDO_MANAGEMENT = MANUAL

If the UNDO_MANAGEMENT initialization parameter is not specified, the instance starts in RBU mode. If an UNDO_TABLESPACE initialization parameter is found, it is ignored. For DBAs who want to run their databases in RBU mode, their existing initialization parameter file can be used without any changes.

Note: You should continue to use old parameters such as ROLLBACK_SEGMENTS, TRANSACTIONS, TRANSACTIONS_PER_ROLLBACK_SEGMENT, and MAX_ROLLBACK_SEGMENTS, as previously.

Creating an Undo Tablespace at Database Creation Time

- **An undo tablespace can be created if the instance is started in AUM mode.**
- **If you do not specify an `UNDO TABLESPACE` clause, an undo tablespace with the name `SYS_UNDOTBS` is created.**
  - **Default file size: 10 MB, `AUTOEXTEND ON`**
  - **Default file name: `DBU1<ORACLE.SID>.dbf`**

```
SQL> CREATE DATABASE
  2  UNDO TABLESPACE undotbs01
  3  DATAFILE SIZE 50M;
```

ORACLE

Creating an Undo Tablespace at Database Creation Time

There are two methods of creating an undo tablespace. The first method creates the undo tablespace when the `CREATE DATABASE` statement is issued. It is used when you are creating a new database, and the instance is started in AUM mode (`UNDO_MANAGEMENT = AUTO`). The second method is used with an existing database. It uses the `CREATE UNDO TABLESPACE` statement. You cannot create database objects in an undo tablespace. It is reserved for automatic-managed undo data only.

To create an undo tablespace using the `CREATE DATABASE` statement, use the `UNDO TABLESPACE` clause. The above statement use the `UNDO TABLESPACE` clause in a `CREATE DATABASE` statement, and the undo tablespace is named `undotbs01`.

If the undo tablespace cannot be created successfully during `CREATE DATABASE`, the entire `CREATE DATABASE` operation fails. You must clean up the database files, correct the error, and retry the `CREATE DATABASE` operation.

If the `UNDO TABLESPACE` clause is not specified and the `CREATE DATABASE` statement is executed in AUM mode, a default undo tablespace is created with the name `SYS_UNDOTBS`. This tablespace is allocated from the default set of files used by the `CREATE DATABASE` statement and its attributes are determined by Oracle. The initial size is 10 MB, and it is auto extensible. This method of creating an undo tablespace is only recommended to users who do not have any specific requirements for allocation of undo space.

Note: The example assumes that OMF files are used; no file name is specified.

Creating an Undo Tablespace After Database Creation

```
SQL> CREATE UNDO TABLESPACE UNDOTBS1
  2  DATAFILE 'UNDOTBS1.DBF' SIZE 50M;
```

An undo tablespace:

- **Can be specified at instance startup using the `UNDO_TABLESPACE` dynamic parameter.**
- **Can only be used in the AUTOMATIC mode for storing rollback information.**
- **Is of permanent, locally managed type, read-write, and in logging mode.**

ORACLE

Creating an UndoTablespace After Database Creation

The `CREATE UNDO TABLESPACE` statement is the same as the `CREATE TABLESPACE` statement, but the `UNDO` keyword is specified. Oracle determines most of the attributes of the undo tablespace. You can specify only the `DATAFILE` clause.

An undo tablespace is a permanent, locally managed tablespace, read-write, and in logging mode with default block size. Values for `MINIMUM EXTENT` and `DEFAULT STORAGE` are system generated.

Note: One undo tablespace must be created per instance in the Real Application Clusters environment. Also, an undo tablespace can be created with a non-default block size (see later in this lesson).

Altering an Undo Tablespace

- **The `ALTER TABLESPACE` command can be used to make changes to undo tablespaces.**
- **Most parameters are system managed.**
- **The following example adds another data file to the undo tablespace:**

```
SQL> ALTER TABLESPACE UNDOTBS_1
  2  ADD DATAFILE 'UNDOTBS_2.DBF'
  3  AUTOEXTEND ON;
```

ORACLE

Altering an Undo Tablespace

Undo tablespaces are altered using the `ALTER TABLESPACE` statement. However, because most aspects of undo tablespaces are system managed, you need only be concerned with the following actions:

- Adding or resizing a data file
- Renaming a data file
- Bringing a data file online or taking it offline
- Beginning or ending an open backup

If an undo tablespace runs out of space, or if you want to prevent it from doing so, you can add more files to it or resize existing data files.

Dropping an Undo Tablespace

```
SQL> DROP TABLESPACE UNDOTBS_2;
```

- **This command has an implicit `INCLUDING CONTENTS` clause.**
- **An undo tablespace can only be dropped if not currently used by any instance.**
- **Readers needing information from dropped undo tablespaces may get `ORA-1555` error messages.**

ORACLE

Dropping an Undo Tablespace

Use the `DROP TABLESPACE` statement to drop an undo tablespace. The above example drops the `undotbs_2` undo tablespace.

An undo tablespace can only be dropped if it is not currently used by any instance. If the undo tablespace contains any outstanding transactions (for example, a transaction died but has not yet been recovered), the `DROP TABLESPACE` statement fails. However, because `DROP TABLESPACE` drops an undo tablespace even if it contains unexpired undo information (within the retention period). You must be careful not to drop an undo tablespace if undo information is needed by some existing queries because readers accessing transactions residing in dropped undo tablespaces may result in `ORA-1555`, if the snapshot is older than the DROP-SCN of the undo tablespace.

`DROP TABLESPACE` for undo tablespaces behaves like a `DROP TABLESPACE … INCLUDING CONTENTS` statement. All contents of the undo tablespace are removed.

Note: If you are using OMF files, then the corresponding files also get removed. You can enter: `DROP TABLESPACE … INCLUDING CONTENTS AND DATAFILES` in order to removed the corresponding non-OMF files.

Switching Undo Tablespaces

- **Only one undo tablespace can be used by an instance at a time.**
  - **Except for a `PENDING OFFLINE UNDO` tablespace**
- **Switching is done by using the `ALTER SYSTEM` command.**

```
ALTER SYSTEM SET UNDO_TABLESPACE=undotbs02;
```

```
ALTER SYSTEM SET UNDO_TABLESPACE='';
```

ORACLE

Switching Undo Tablespaces

You can switch from using one undo tablespace to another. Because the `UNDO_TABLESPACE` initialization parameter is a dynamic parameter, the `ALTER SYSTEM SET` statement can be used to assign a new undo tablespace. The following statement effectively switches to a new undo tablespace:

```
ALTER SYSTEM SET UNDO_TABLESPACE = undotbs02;
```

Assuming `undotbs01` is the current undo tablespace, after this command successfully executes, the instance uses `undotbs02` in place of `undotbs01` as its undo tablespace.

If any of the following conditions exist for the tablespace to which you are switching, an error is reported and no switching occurs:

- The tablespace does not exist
- The tablespace is not an undo tablespace
- The tablespace is already being used by another instance

The database is online while the switch operation is performed, and user transactions can be executed while this command is being executed. When the switch operation completes successfully, all transactions started after the switch operation began are assigned to transaction tables in the new undo tablespace.

Switching Undo Tablespaces (continued)

The switch operation does not wait for transactions in the old undo tablespace to commit. If there are any pending transactions in the old undo tablespace, the old undo tablespace enters into a PENDING OFFLINE mode (status). In this mode, existing transactions can continue to execute, but undo records for new user transactions cannot be stored in this undo tablespace.

An undo tablespace can exist in this PENDING OFFLINE mode, even after the switch operation completes successfully. A PENDING OFFLINE undo tablespace cannot be used by another instance, nor can it be dropped. Eventually, after all active transactions have committed, the undo tablespace automatically goes from the PENDING OFFLINE mode to the OFFLINE mode. From then on, the undo tablespace is available for other instances.

Note: If the parameter value for UNDO_TABLESPACE is set to '' (two single quotes), the current undo tablespace will be switched out without switching in any other undo tablespace. This can be used, for example, to unassign an undo tablespace in the event that you want to revert to RBU mode. The following example unassigns the current undo tablespace:

```
SQL> ALTER SYSTEM SET UNDO_TABLESPACE = '';
```

Changing the Retention Period for Undo Information

- **UNDO_RETENTION: The amount of rollback information to retain if possible**
  - **Dynamic initialization parameter**
  - **Default value: 30 seconds**
- **ORA-1555 is still possible if an undo tablespace is too small compared to the retention time**
- **Size the undo tablespace using this formula:**

```
UndoSpace = UR * UPS + overhead
```

ORACLE

Changing the Retention Period for Undo Information

Committed undo information normally is lost when its undo space is overwritten by a newer transaction. But for consistent read purposes, long-running queries may require old undo information for undoing changes and producing older images of data blocks. The UNDO_RETENTION initialization parameter provides a means of explicitly specifying the amount of undo information to retain. With a proper setting, long-running queries are more likely to complete without risk of receiving the *snapshot too old* error.

Retention is specified in units of seconds. It is persistent and can survive system crashes. That is, undo generated before an instance crash, is retained until its retention time has expired even across restarting the instance. When the instance is recovered, undo information is retained based on the current setting of the UNDO_RETENTION initialization parameter.

The UNDO_RETENTION parameter can be specified initially in the initialization parameter file, used by the STARTUP process. The UNDO_RETENTION parameter value can also be changed dynamically at any time using the ALTER SYSTEM command.

The effect of the UNDO_RETENTION parameter is immediate, but it can only be honored if the current undo tablespace has enough space for the active transactions. If an active transaction requires undo space and the undo tablespace does not have available space, the system starts reusing unexpired undo space. Such action can potentially cause some queries to fail with the *snapshot too old* error.

Note: If the UNDO_RETENTION initialization parameter is not specified, the default value is 30 seconds.

Changing the Retention Period for Undo Information (continued)

Given a specific UNDO_RETENTION parameter setting and some system statistics, the amount of undo space required to satisfy the undo retention requirement can be estimated using the following formula:

```
UndoSpace = UR * UPS + overhead
```

where:

- UndoSpace = number of undo blocks
- UR = UNDO_RETENTION in seconds
- UPS = undo blocks per second
- overhead = small overhead for metadata (transaction tables, bitmaps, and so on)

As an example, if UNDO_RETENTION (UR) is set to two hours, and the transaction rate (UPS) is 200 undo blocks per second, with a 4 KB block size, the required undo space (excluding the small overhead) is computed as follows:

`(2 * 3600 * 200 * 4 KB) = 5.8 GB`.

Such computation can be performed by using information in V$UNDOSTAT view explained in the next slides.

Data Dictionary View to Support Automatic Undo Management

- **V$UNDOSTAT contains information about how rollback segments are used by the current instance.**
- **It is available for both MANUAL or AUTO mode.**
- **DBA_UNDO_EXTENTS shows the commit time for each extent in the undo tablespace.**
- **You can still use the views V$ROLLSTAT and V$TRANSACTION in AUM mode.**

ORACLE

Data Dictionary View to Support Support Automatic Undo Management

The following views are available for obtaining undo space information:

- V$UNDOSTAT contains statistics for monitoring and tuning undo space. Use this view to help estimate the amount of undo space required for the current workload. Oracle also uses this information to help tune undo usage in the system. This view is available in both the AUM and the RBU mode. Statistics are available for undo space consumption, transaction concurrency, and length of queries in the instance. Each row in the view contains statistics collected in the instance for a certain period of time. The rows are in descending order by the BEGIN_TIME column value. Each row belongs to the time interval (generally about 10 minutes) marked by (BEGIN_TIME, END_TIME). Each column represents the data collected for the particular statistic in that time interval. The first row of the view contains statistics for the (partial) current time period, and his END_TIME is moving. The view ultimately reflects 24-hours before it is reset and begins the cycle again. Every time the instance is started, this view is reset.
- V$ROLLSTAT for AUM mode, information reflects behavior of the undo segments in the undo tablespace.
- V$TRANSACTION contains undo segment information.
- DBA_UNDO_EXTENTS shows the commit time for each extent in the undo tablespace.

New Data Dictionary View to Support Automatic Undo Management

This V$UNDOSTAT example illustrates how undo space is consumed in the system for the previous 24 hours from time 16:07.

```
End-    Undo     Txn    Txn  Query  Exten SSTooOld
Time  Blocks Concrcy  Total    Len Stolen    Error
----- ------ ------- ------ ------ ------ --------
16:07    252      15    151     25      2        0
16:00    752      16   1467    150      0        0
15:50    873      21   1954     45      4        0
15:40   1187      45   3210    633     20        1
15:30   1120      28   2498   1202      5        0
15:20    882      22   2002     55      0        0
...
```

ORACLE

New Data Dictionary View to Support Automatic Undo Management

Here is the detail of some of the important columns found in the V$UNDOSTAT view:

- BEGIN_TIME/END_TIME: Indicates the beginning/end of a time interval marked. The rows are ordered descending by END_TIME
- UNDOBLCKS: Represents the total number of undo blocks consumed. Used to obtain consumption rate of undo blocks, and to estimate the size of the undo tablespace needed to handle the workload of the system
- MAXCONCURRENCY: Represents the maximum number of transactions executed concurrently. Used as a reference for users to understand the level of concurrency in the current system
- TXNTOTAL: Shows the total number of transactions executed within the period
- MAXQUERYLEN: Represents the maximum length of queries executed in the instances
- SSOLDERRCNT: Shows the number of *snapshot too old* errors that have occurred within a period. You can use this statistic to decide whether or not the UNDO_RETENTION parameter is set properly given the size of the undo tablespace. Increasing the UNDO_RETENTION value can reduce the occurrence of this error.

Dictionary View to Support Automatic Undo Management (continued)

The example above illustrates the peak undo consumption happened at the interval of (15:30, 15:40), 1187 undo blocks were consumed in 10 minutes (or about 2 blocks/second). Also, the highest transaction concurrency occurred during that same period with 45 transactions executing at the same time. The longest query execution (1202 seconds) was executed in the period (15:20, 15:30).

New Parameters to Support Automatic Undo Management: Summary

- **UNDO_MANAGEMENT = {AUTO|MANUAL}**
- **UNDO_TABLESPACE: Specifies the undo tablespace to be used**
- **UNDO_SUPPRESS_ERRORS = TRUE: Suppresses errors while attempting to execute manual operations while in AUTO mode**
- **UNDO_RETENTION: Controls the amount of rollback information to retain**

ORACLE

New Parameters to Support Automatic Undo Management: Summary

UNDO_SUPPRESS_ERRORS enables users to suppress errors while executing RBU (rollback undo mode) operations (for example, ALTER ROLLBACK SEGMENT ONLINE) in AUM (Automatic Undo Managed) mode. Setting this parameter enables users to use the AUM feature before all application programs and scripts are converted to AUM mode. For example, if you have a tool that uses SET TRANSACTION USE ROLLBACK SEGMENT statement, you can add the statement "ALTER SESSION SET UNDO_SUPPRESS_ERRORS=TRUE" to the tool to suppress the error (OER 30019).

If you want to run in AUM mode, ensure that your tools or applications are updated to run in AUM mode.

Multiple Block Size Support

- **Oracle9*i* supports the creation of databases with multiple block sizes.**
- **The benefits to this feature include:**
  - **Ability to maximize I/O performance**
  - **Ability to transport tablespaces between databases with different block sizes**
- **A database can be created with a standard block size and up to four nonstandard block sizes.**
- **Block sizes can have any power-of-two value between 2 KB and 32 KB.**

ORACLE

Multiple Block Size Support

Oracle9*i* supports the creation of databases with multiple block sizes. This feature is useful in the following situations:

- When transporting a tablespace from an OLTP database to an enterprise data warehouse. Oracle9*i* facilitates transport between databases of different block sizes.
- When you require the ability to locate objects in tablespaces of appropriate block size in order to maximize I/O performance.

The block size of the SYSTEM tablespace is termed the standard block size. This is set when the database is created. With Oracle9*i* you can specify up to four nonstandard block sizes, in addition to a standard block size. In the initialization file, you can configure subcaches within the buffer cache for each of these block sizes. Subcaches can also be configured while an instance is running. You can create tablespaces having any of these block sizes. The standard block size is used for the system tablespace and most other tablespaces.

Standard Block Size

- **Set at database creation using the `DB_BLOCK_SIZE` parameter; cannot be changed without recreating the database**
- **Used for `SYSTEM` and `TEMPORARY` tablespaces**
- **`DB_CACHE_SIZE` specifies the size of the `DEFAULT` buffer cache for standard block size:**
  - **Minimum size = one granule (4 MB or 16 MB)**
  - **Default value = 48 MB**

ORACLE

Standard Block Size

The DB_BLOCK_SIZE initialization parameter specifies the standard block size for the database. This block size is used for the SYSTEM tablespace, and for any temporary tablespace. Unless specified otherwise, the standard block size is also used as the default block size for a tablespace. Oracle can support up to four additional nonstandard block sizes.

The most commonly used block size should be picked as the standard block size. In many cases, this is the only block size that you need to specify. Typically, DB_BLOCK_SIZE is set to either 4 KB or 8 KB. If not specified, the default data block size is operating system specific and is generally adequate.

The block size cannot be changed after database creation, except by re-creating the database.

The DB_CACHE_SIZE initialization parameter replaces the DB_BLOCK_BUFFERS initialization parameter that was used in previous releases. The DB_CACHE_SIZE parameter specifies the size of the cache of standard block size buffers, where the standard block size is specified by DB_BLOCK_SIZE.

For backward compatibility the DB_BLOCK_BUFFERS parameter still works, but it remains a static parameter and cannot be combined with any of the dynamic sizing parameters.

Standard Block Size (continued)

Note: A granule is a unit of contiguous virtual memory allocation. The size of a granule depends on the estimated total SGA size whose calculation is based on the value of the parameter `SGA_MAX_SIZE`: 4 MB if estimated SGA size is < 128 MB, 16 MB otherwise (for more details see later in this course).

Nonstandard Block Sizes

- **Configure additional caches with the following dynamic parameters:**
  - **DB_2K_CACHE_SIZE for 2K blocks**
  - **DB_4K_CACHE_SIZE for 4K blocks**
  - **DB_8K_CACHE_SIZE for 8K blocks**
  - **DB_16K_CACHE_SIZE for 16K blocks**
  - **DB_32K_CACHE_SIZE for 32K blocks**
- **DB_nK_CACHE_SIZE is not allowed if nK is the standard block size**
- **Minimum size for each cache: one granule**

ORACLE

Nonstandard Block Sizes

The buffer cache initialization parameters determine the size of the buffer cache component of SGA. You use them to specify the sizes of caches for the various block sizes used by the database. If you intend to use multiple block sizes in your database, you must have the DB_CACHE_SIZE and at least one DB_nK_CACHE_SIZE parameter set. Each parameter specifies the size of the buffer cache for the corresponding block size. The default value for DB_nK_CACHE_SIZE parameters is 0. Do not set this parameter to zero if there are any online tablespaces with an *n* KB block size.

Platform-specific block size restrictions apply. For example, you cannot set DB_32K_CACHE_SIZE if the maximum block size on the platform is less than 32 KB. Also, you cannot set DB_2K_CACHE_SIZE if the minimum block size is greater than 2 KB.

Note: These parameters cannot be used to size the cache for the standard block size. For example, if the value of DB_BLOCK_SIZE is 2 KB, it is illegal to set DB_2K_CACHE_SIZE. The size of the cache for the standard block size is always determined from the value of DB_CACHE_SIZE.

Creating a Nonstandard Block Size Tablespace

```
SQL> CREATE TABLESPACE tbs_1
  2  DATAFILE 'tbs_1.dbf'
  3  SIZE 10M BLOCKSIZE 4K;
```

```
SQL> desc dba_tablespaces
 Name                  Null?    Type
 ----------------- -------- ------------
 TABLESPACE_NAME    NOT NULL VARCHAR2(30)
 BLOCK_SIZE         NOT NULL NUMBER
 ...
```

ORACLE

Creating a Nonstandard Block Size Tablespace

Use the BLOCKSIZE clause to specify a nonstandard block size for the tablespace. You can specify the size in bytes or in kilobytes, using the "K" suffix.

In order to specify this clause, you must have the DB_CACHE_SIZE and at least one DB_nK_CACHE_SIZE parameter set, and the integer you specify in this clause must correspond with the setting of one DB_nK_CACHE_SIZE parameter setting.

Restriction: You cannot specify nonstandard block sizes for a temporary tablespace (that is, if you also specify TEMPORARY) or if you intend to assign this tablespace as the temporary tablespace for any users.

The first statement above creates a new tablespace called tbs_1 with the file tbs_1.dbf having a block size of 4 KB. In order for this statement to succeed, the buffers of size 4 KB must currently be configured in the buffer cache.

Note: A new column has been added to the *_TABLESPACES dictionary views in order to reflect the corresponding block size used in a particular tablespace.

Multiple Block Sizing Rules

- All partitions of a partitioned object must reside in tablespaces of the same block size.
- All temporary tablespaces, including permanent being used as default temporary tablespaces, must be of standard block size.
- Index-organized table overflow and out-of-line LOB segments can be stored in a tablespace with a block size different from the base table.

ORACLE

Summary

In this lesson, you should have learned how to:

- **Create undo tablespaces with the `UNDO` tablespace clause**
- **Use the `BLOCKSIZE` clause to create tablespaces with different block sizes**

ORACLE

Practice 14-1 Overview

This practice covers the following topics:

- **Creating a new undo tablespace and monitoring it**
- **Understanding the dynamic undo space transfer**
- **Switching undo tablespaces**

ORACLE

Practice 14-2 Overview

This practice covers the following topics:

- **Transporting a tablespace into a database having a different standard block size**
- **Using the new `db_8k_cache_size` initialization parameter**

ORACLE

15

Memory Management

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- **Set parameters to enable automatic and dynamic sizing of SQL working areas**
- **Use new columns and views to gather information regarding SQL execution memory management**
- **Describe the allocation and tracking of memory behind a dynamic SGA**

ORACLE

Automated SQL Execution Memory Management

- **Simplifies and improves memory allocation**
- **The size of the SQL working areas can be automatically and dynamically adjusted.**
- **Benefits include:**
  - **Ease of memory tuning**
  - **Reduction of time to tune memory**
  - **Better throughput when a large number of users are on the system**
  - **Improved response time on queries**

ORACLE

Automated SQL Execution Memory Management

The Automated SQL Execution Memory Management feature simplifies and improves the way memory is allocated for SQL execution without shutting down the databases.

Benefits

Manageability: In Oracle8*i*, a DBA can control the maximum size of the working areas using various parameters such as `SORT_AREA_SIZE, HASH_AREA_SIZE`, `BITMAP_MERGE_AREA_SIZE`, and `CREATE_BITMAP_AREA_SIZE`, but these parameter are difficult to set and tune. This feature releases the DBA from having to set the various `*_AREA_SIZE` parameters. It provides automatic and dynamic memory tuning, reducing the amount of time required to set and tune these parameters.

Performance: Because the parameters are automatically and dynamically adjusted, this feature can compensate for low or high memory usage, along with controlling the maximum amount of memory a query can use. In addition, memory-consuming operations such as hash-join, sort, and so on, dynamically alter their memory profile to ensure the best possible use of system memory and optimal system performance.

Usability: The above mentioned parameters can often waste PGA memory because more memory is allocated than is needed. By reducing the amount of memory used, the memory otherwise allocated can be better put to use by other queries.

PGA Memory Management

- **PGA memory is classified to differentiate between tunable and untunable memory:**
  - **Tunable memory: Memory consumed by SQL working areas**
  - **Untunable memory: Remaining memory**
- **Sizes the tunable fraction of memory when automatic mode is enabled**
- **Size of tunable portion allocated depends on an overall PGA memory target**

```
UNTUNABLE_MEMORY_SIZE + TUNABLE_MEMORY_SIZE <=
PGA_AGGREGATE_TARGET
```

ORACLE

PGA Memory Management

In order to define what part of the PGA memory was affected by the automatic tuning operation, the PGA was classified to differentiate between tunable and untunable memory. Tunable memory is the memory consumed by SQL working areas while untunable memory is the rest. This feature concentrates on sizing the tunable fraction of the PGA memory when the automatic mode is enabled. The size of the tunable portion allocated by an instance depends on an overall PGA memory target explicitly set by the DBA. Under this automatic mode the Oracle server tries to have:

```
UNTUNABLE_MEMORY_SIZE + TUNABLE_MEMORY_SIZE
<= PGA_AGGREGATE_TARGET
```

When the automatic mode is enabled, the Oracle server can only control the tunable fraction of the PGA memory. If this memory accounts for a very small percentage of the overall PGA memory, which is typically the case of OLTP workloads, it might be impossible to enforce the above equation. In an OLTP system, tunable memory consumed is a relatively limited (<1%) part of total PGA memory. Indeed, the tunable memory probably represents a major fraction of the PGA memory only under DSS workloads. For complex DSS workloads, the tunable memory accounts for most of the PGA memory (>90%), and most of the memory used by the instance. In that context, managing the tunable fraction is a very effective way to manage the overall PGA memory consumed by the instance automatically.

Enabling Automated SQL Execution Memory Management

- **Enabling automatic tuning requires the setting of two parameters:**
  - **PGA_AGGREGAGE_TARGET: A dynamic system parameter**
  - **WORKAREA_SIZE_POLICY: A dynamic session and system level parameter, with two values:**
    - **MANUAL: Tuning done using the %_AREA_SIZE parameters**
    - **AUTO: Tuning done automatically**

ORACLE

Enabling the Automated SQL Execution Memory Management

PGA_AGGREGATE_TARGET: A new system level initialization parameter set to specify the target aggregate amount of PGA memory available to the instance. This parameter is only a target and can be dynamically modified at the instance level by the DBA.

There is not a default value, and the range of values is from 10 MB to 4000 GB.

It must be set to a value before WORKAREA_SIZE_POLICY is set to AUTO. If you forget this, you get the following error:

```
ORA-04032: pga_aggregate_target must be set
           before switching to auto mode
```

Enabling the Automated SQL Execution Memory Management (continued)

WORKAREA_SIZE_POLICY: A new session and system level initialization parameter, which has two values that allow a user to select between automatic or manual tuning of work area sizes.

- MANUAL: This mode is the default. Under the manual mode, tuning is done using the existing %_AREA_SIZE parameters.
- AUTO: Under this mode, the work areas are sized to accomplish three goals:
  - Tuned so overall size of the PGA memory never exceeds the value of PGA_AGGREGATE_TARGET.
  - Tunable memory allocated by a process is regulated so that a process never runs out of memory.
  - Memory should be allotted to work areas to optimize both throughput and response time.

The default depends on the setting of PGA_AGGREGATE_TARGET. If set to a value, then the default is AUTO. If not set, then the default is MANUAL.

New Statistics and Columns

- **New statistics in V$SYSSTAT, V$SESSTAT, and V$MYSTAT:**

```
workarea memory allocated
workarea executions - optimal
workarea executions - onepass
workarea executions - multipass
```

- **New columns added to V$PROCESS**
  - **PGA_ALLOC_MEM**
  - **PGA_MAX_MEM**
  - **PGA_USED_MEM**

ORACLE

New Statistics and Columns

New statistics and columns have been added to accommodate the automatic mode of Automatic SQL Execution Management.

New Statistics in V$SYSSTAT, V$SESSTAT and V$MYSTAT are as follows:

- `workarea memory allocated`:
  Total amount of PGA memory in KB dedicated to work areas allocated on behalf of a given session (V$SESSTAT) or system (V$SYSTAT).
- `workarea executions - optimal size`:
  The cumulative count of work areas which had an optimal size. For example: optimal size is defined if the sort does not need to spill to disk.
- `workarea executions - onepass`:
  The cumulative count of work areas using the one pass size. One pass is generally used for big work areas where spilling to disk cannot be avoided.
- `workarea executions - multipass`:
  The cumulative count of work areas running in more than one pass. This should be avoided and is a symptom of a poorly tuned system.

New Statistics and Columns (continued)

New Columns in V$PROCESS are as follows:

- PGA_USED_MEM: PGA memory currently used by the process.
- PGA_ALLOC_MEM: PGA memory currently allocated by the process. Allocation includes free PGA memory not yet released to the OS by the server process.
- PGA_MAX_MEM: Maximum PGA memory ever allocated by the process.

Example

```
SQL> SELECT name, value FROM v$sysstat
  2  WHERE  name LIKE '%work area%';

NAME                                 VALUE
-------------------------------- -----
work area memory allocated               0
work area executions - optimal        1544
work area executions - onepass          11
work area executions - multipass         2
```

```
SQL> SELECT sum(pga_used_mem), sum(pga_alloc_mem)
  2  ,      sum(pga_max_mem)   FROM v$process;

SUM(PGA_USED_MEM) SUM(PGA_ALLOC_MEM) SUM(PGA_MAX_MEM)
----------------- ------------------ ---------------
          9421418           10345058        22853922
```

ORACLE

Example

In the example we used a PGA_AGGREGATE_TARGET of 10 MB and the WORKAREA_POLICY_SIZE is set to AUTO.

The SUM(PGA_ALLOC_MEM) will thus remain near constant when Automated SQL Execution Memory Management is enabled.

Note: The PGA_AGGREGATE_TARGET memory is not shared equally among the current sessions. Memory is adjusted whenever a session makes use of the adjustable PGA areas (sorting, bitmap creation, hash join). If sessions are created or destroyed, the memory is redistributed.

New Views to Support SQL Execution Memory Management

- **New views have been created to display SQL execution memory management information.**
  - **V$SQL_WORKAREA: Displays information about work areas used by SQL cursors.**
  - **V$SQL_WORKAREA_ACTIVE: Displays an instantaneous view of the work areas currently allocated by the system.**
  - **V$PGASTAT: Displays memory usage statistics.**

ORACLE

New Views to Support SQL Execution Memory Management

These views are only populated if `WORKAREA_SIZE_POLICY` is set to `AUTOMATIC`.

Example Use of V$SQL_WORKAREA

Example to find the top ten work areas requiring the most cache memory.

```
SQL> SELECT *
  2  FROM  (SELECT   workarea_address
  3          ,       operation_type
  4                  policy
  5          ,       estimated_optimal_size
  6          FROM    v$sql_workarea
  7          ORDER BY estimated_optimal_size DESC)
  8  WHERE  ROWNUM <= 10;
```

ORACLE

Example Use of V$SQL_WORKAREA

The example above gives the following output:

```
WORKAREA OPERATION_TYPE     POLICY   ESTIMATED_OPTIMAL_SIZE
------ -------------------- -------- ----------------------
820DD4E8 GROUP BY (SORT)    AUTO                2.9919E+10
821111C0 GROUP BY (SORT)    AUTO                     74752
   :                          :                        :
```

The size is the calculated ideal size, less may have been allocated.

Dynamic SGA

- **Implements an infrastructure allowing SGA configuration to change without shutting down the instance**
- **Benefits include:**
  - **The Oracle server can modify its physical address space use to respond to the operating system's use of physical memory**
  - **It provides an SGA that will grow and shrink dynamically in response to a DBA command.**

ORACLE

Dynamic SGA

Since its inception, the SGA has always been a static allocation of memory, which was shared across all Oracle threads of execution. The size of memory is calculated based on values in the `init.ora` parameter file. Once allocated, the amount of usable shared memory could not grow or shrink. If a DBA wanted to increase the number of database block buffers, the instance had to first be shut down, the initialization parameter file was modified, and then the instance was restarted.

The dynamic SGA implements an infrastructure allowing the SGA configuration to change while the instance is running. This allows the sizes of the buffer cache and shared pool to be changed without shutting down the instance.

The dynamic SGA infrastructure allows limits to be set at run time on how much physical memory will be used for the SGA. Conceivably, the buffer cache and shared pool could be initially underconfigured and would grow and shrink depending upon their respective work loads.

Additional buffer caches with different block sizes can be configured when importing or creating tablespaces that use other block sizes.

Unit of Allocation

- **The unit of allocation is a *granule*.**
  - **Contiguous virtual memory allocation**
  - **Size based on `SGA_MAX_SIZE`**
- **SGA memory is shown in granules by SGA components.**
- **`V$BUFFER_POOL` displays allocation and deallocation.**

ORACLE

Unit of Allocation

A granule is a unit of contiguous virtual memory allocation. The size of a granule depends on the estimated total SGA size whose calculation is based on the value of the `SGA_MAX_SIZE` parameter.

- 4 MB if estimated SGA size is < 128 MB
- 16 MB otherwise

The components (buffer cache and shared pool) are allowed to grow and shrink based on granule boundaries. For each component granules are tracked and displayed by the `V$BUFFER_POOL` view. The minimum SGA configuration is three granules (one granule for the fixed SGA (includes redo buffers); one granule for the buffer cache; one granule for the shared pool).

V$BUFFER_POOL Columns

Column	Description
ID	Buffer pool ID number
NAME	Buffer pool name. Possible values: DEFAULT, KEEP, RECYCLE. Note: Currently, KEEP and RECYCLE pools only exist for the standard block size. All non-standard block size pools are DEFAULT.
BLOCK_SIZE	Block size in bytes for buffers in this pool. Possible values: the standard block size, the power of 2 non-standard block sizes, 2048, 4096, 8192, 16384, 32768.
RESIZE_STATE	Current state of the resize operation STATIC: not being resized ALLOCATING: memory is being allocated (can be canceled by the user) ACTIVATING: new buffers are being created (user cannot cancel) SHRINKING: buffers are being deleted (can be canceled by the user)
CURRENT_SIZE	Present size of the sub-cache in megabytes
BUFFERS	Current instantaneous number of buffers
TARGET_SIZE	If a resize is in progress (state is not STATIC), records new target size in megabytes. If the pool is STATIC, the value in this column is the same as the current size of the pool.
TARGET_BUFFERS	If a resize is in progress, records new target size in terms of buffers. Otherwise, the value in this column is the same as the current number of buffers.
PREV_SIZE	Previous buffer pool size. If the buffer pool has never been resized, the previous size is zero.
PREV_BUFFERS	Previous number of buffers in the buffer pool. Value is zero if the buffer pool has never been resized.
LO_BNUM	Obsolete column
HI_BNUM	Obsolete column
LO_SETID	Obsolete column
HI_SETID	Obsolete column
SET_COUNT	Obsolete column

Growing a Component's SGA Memory Area

```
Init.ora parameter values:
SGA_MAX_SIZE = 128M
DB_CACHE_SIZE = 96M
SHARED_POOL_SIZE = 32M
```

```
SQL> ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;
-- insufficient memory error message

SQL> ALTER SYSTEM SET DB_CACHE_SIZE    = 64M;
SQL> ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;
-- insufficient memory error message, check
-- V$BUFFER_POOL to see if shrink has completed

SQL> ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;
Statement processed.
```

ORACLE

Growing a Component's SGA Memory Area

A DBA can grow a component's SGA use by issuing an `ALTER SYSTEM` command to modify the `init.ora` parameters value. Oracle takes the new size, rounds it up to the nearest multiple of 16 MB (the granule size in this case) and adds or takes away granules to meet the target size.

Adding a number of granules to a component (increasing the memory use of a component) with an with an `ALTER SYSTEM` command will succeed if Oracle has enough free granules to satisfy the request. Oracle does not start freeing another components granules for adding. Instead, the database administrator must ensure the instance has enough free granules to satisfy the increase of a component's granule use. If the current amount of SGA memory is less than `SGA_MAX_SIZE`, then Oracle is free to allocate more granules until the SGA size reaches `SGA_MAX_SIZE`.

The Oracle server, which invokes the `ALTER SYSTEM` command reserves a set of granules for the corresponding SGA component (init.ora parameter which defines the component's SGA use). After the reservation is complete, the foreground hands the completion to the background process. The background process completes the operation by taking the reserved granules and adding them to the component's granule list. This is also referred to as growing a components SGA memory area.

Dynamic Shared Pool

- **The Shared pool can be dynamically resized to grow or shrink using `ALTER SYSTEM`**

```
SQL> ALTER SYSTEM SET SHARED_POOL_SIZE = 64M;
```

- **Allocation size has the following limits:**
  - **Size must be an integer multiple of the granule size**
  - **Total SGA size cannot exceed `SGA_MAX_SIZE`**

ORACLE

Dynamic Shared Pool

Prior to Oracle9*i*, the SGA was allocated once, when the instance was started. If the shared pool could not find a large enough contiguous piece of memory, it signaled an error.

Example:

```
SQL> alter system set shared_pool_size = 128M;
alter system set shared_pool_size = 128M
*
ERROR at line 1:
ORA-02097: parameter cannot be modified because specified
value is invalid
ORA-04033: Insufficient memory to grow pool
```

Dynamic Buffer Cache

- The buffer cache can be dynamically resized to grow or shrink using `ALTER SYSTEM`

```
SQL> ALTER SYSTEM SET DB_CACHE_SIZE = 96M;
```

- The allocation size has the following limits:
  - Size must be an integer multiple of the granule size
  - Total SGA size cannot exceed `SGA_MAX_SIZE`
  - `DB_CACHE_SIZE` can never be set to zero

ORACLE

New Buffer Cache Parameters

- **New parameters define cache sizes for primary block size buffers:**
  - **DB_CACHE_SIZE**
  - **DB_KEEP_CACHE_SIZE**
  - **DB_RECYCLE_CACHE_SIZE**
- **New parameters for the other block sizes**
  - **DB_nK_CACHE_SIZE**
    n = 2, 4, 8, 16 or 32

ORACLE

New Buffer Cache Parameters

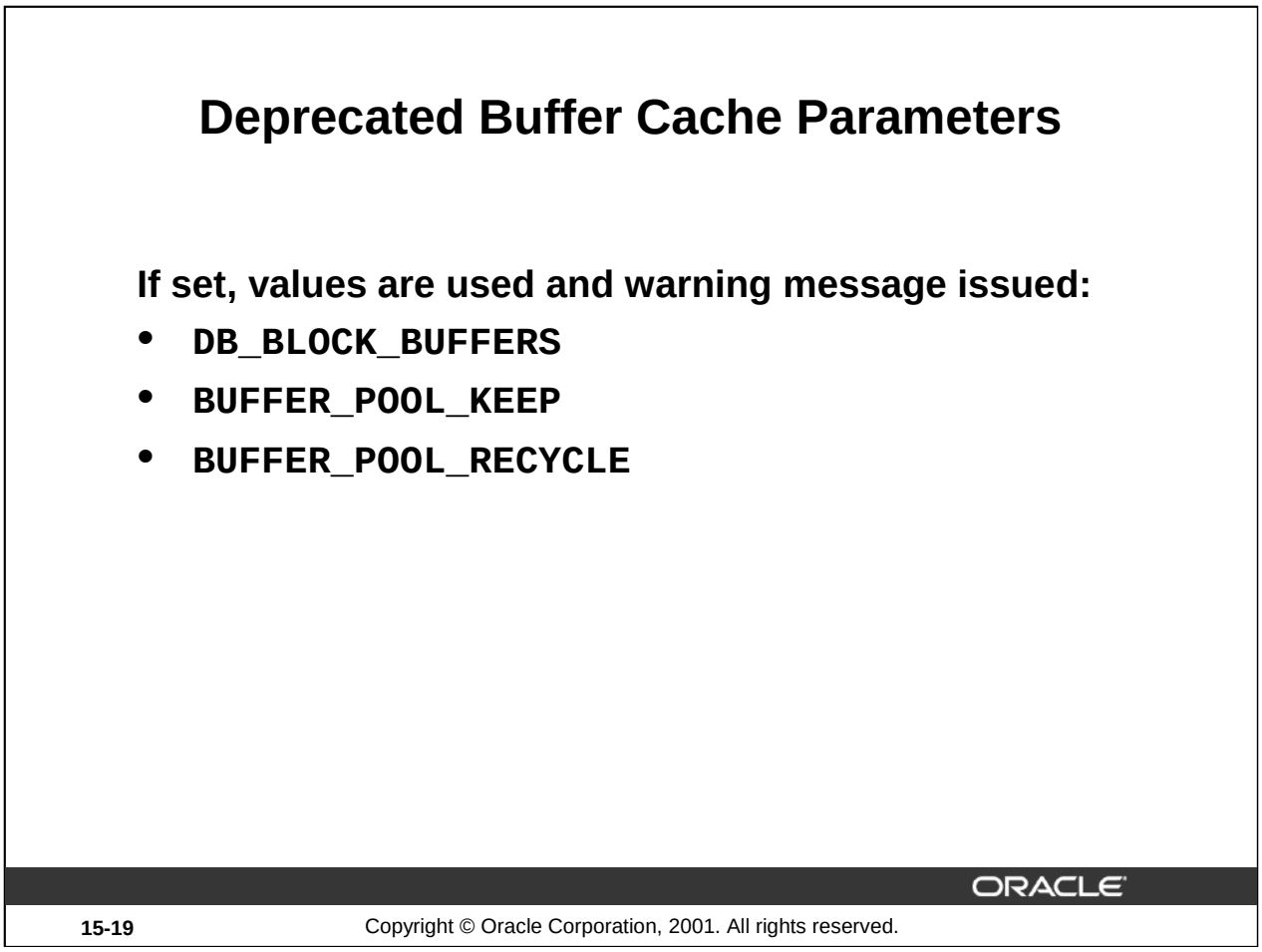
The buffer cache consists of independent subcaches for buffer pools and for multiple block sizes. The DB_BLOCK_SIZE parameter determines the primary block size, which is used for the SYSTEM tablespace. Three new parameters define the sizes of the caches for buffers for the primary block size.

- DB_CACHE_SIZE
- DB_KEEP_CACHE_SIZE
- DB_RECYCLE_CACHE_SIZE

The new value of DB_CACHE_SIZE refers only to the size of the DEFAULT buffer pool versus total size of the DEFAULT, KEEP, and RECYCLE buffer pools.

Multiple Block Size Parameters

If tablespaces with different block size are used, then buffers for the block size must be allocated. You must not use the DB_nK_CACHE_SIZE parameter that matches the DB_BLOCK_SIZE value.



Deprecated Buffer Cache Parameters

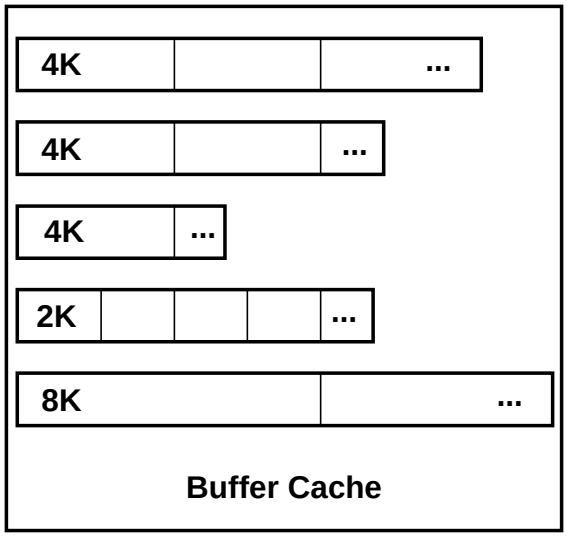
The Buffer Cache size parameters DB_BLOCK_BUFFERS, BUFFER_POOL_KEEP, and BUFFER_POOL_RECYCLE are deprecated, but have been maintained for backward compatibility, and will be made obsolete in the future. If the parameters are set, their values will be used, and a warning message will be made to encourage the user to migrate to the new parameter scheme.

Prior to Oracle9*i*, the syntax for the BUFFER_POOL_SIZE parameter allowed the user to optionally specify the number of LRU latches for the buffer pool in addition to the number of buffers in the buffer pool. These latches were allocated out of the total number of latches specified in DB_BLOCK_LRU_LATCHES. Since DB_BLOCK_LRU_LATCHES is now obsolete, the specification of the number of LRU latches for the buffer pool, if provided, is ignored and internally calculated.

These parameters will continue to be static parameters. Furthermore, these parameters cannot be combined with the dynamic size parameters. Combining them in the same parameter file produces an error.

Example Buffer Caches Setup

```
DB_CACHE_SIZE          =  48M
DB_KEEP_CACHE_SIZE     =  24M
DB_RECYCLE_CACHE_SIZE  =  12M
DB_2K_CACHE_SIZE       =  24M
DB_8K_CACHE_SIZE       =  96M
                          ------
Total Buffer Cache     = 204M
```



ORACLE

Example of a Buffer Caches Setup

The Standard block size is 4K as default or specified using DB_BLOCK_SIZE.

This instance is setup with a KEEP and RECYCLE buffer for the standard block size. The KEEP and RECYCLE can only be used with standard block size.

In addition there are block buffer caches for 2 Kb and 8 Kb block sizes.

The specified size for each buffer is rounded up to the nearest granule, in this example the granule size is 4 Mb.

Dynamic Buffer Cache Advisory Parameter

- **Enables and disables statistics gathering for predicting different cache size behavior**
- **Benefit:**
  - **Information enables sizing the buffer cache optimally for a given workload**
- **Enabled with `DB_CACHE_ADVICE`**
  - **Dynamic by means of `ALTER SYSTEM`**
  - **Three values available: `OFF`, `ON`, and `READY`**

ORACLE

Dynamic Buffer Cache Advisory Parameter

The buffer cache advisory feature enables and disables statistics gathering for predicting behavior with different cache sizes. The information provided by these statistics can help DBA size the buffer cache optimally for a given workload.

The buffer cache advisory is enabled by means of the initialization parameter `DB_CACHE_ADVICE`. It is a dynamic parameter by means of `ALTER SYSTEM`. Three values (`OFF, ON, READY`) are available.

`DB_CACHE_ADVICE` Parameter Values:

- `OFF:` Advisory is turned off and the memory for the advisory is not allocated
- `ON:` Advisory is turned on and both cpu and memory overhead is incurred
- `READY:` Advisory is turned off but the memory for the advisory remains allocated.

Attempting to set the parameter to `ON` when it is in the `OFF` state may lead to `ORA-4031: Inability to allocate from the shared pool`. If the parameter is in a `READY` state it can be set to `ON` without error because the memory is already allocated.

New View to Support Buffer Cache Advisory

- **V$DB_CACHE_ADVICE displays buffer cache statistics gathered**
- **Rows predict the estimated number of physical reads for different cache sizes**
- **Computes a physical read factor**

ORACLE

New View to Support Buffer Cache Advisory

The buffer cache advisory information is collected and displayed through a new view, V$DB_CACHE_ADVICE. The view contains different rows that predict the estimated number of physical reads for different cache sizes. The rows also compute a physical read factor, which is the ratio of the number of estimated reads to the number of reads actually performed during the measurement interval by the real buffer cache.

V$DB_CACHE_ADVICE Columns

- ID: Buffer pool ID (Ranges from 1-8)
- NAME: Buffer pool name
- BLOCK_SIZE: Block size in bytes for buffers in this pool, possible values are the standard block size, and the power of two nonstandard block sizes; 2048, 4096, 8192, 16384, 32768
- ADVICE_STATUS: Status of the advisory
- SIZE_FOR_ESTIMATE: Cache size for prediction (in megabytes)
- BUFFERS_FOR_ESTIMATE: Cache size for prediction (in terms of buffers)
- ESTD_PHYSICAL_READ_FACTOR: Physical read factor for this cache size; ratio of number of estimated physical reads to the number of reads in the real cache. If there are no physical reads into the real cache, the value of this column is NULL.
- ESTD_PHYSICAL_READS: Estimated number of physical reads for this cache size

V$DB_CACHE_ADVICE Example

```
SQL> SELECT name, block_size
  2  ,      buffers_for_estimate
  3  ,      estd_physical_read_factor
  4  FROM   v$db_cache_advice;

                        BUFFERS        ESTD
             BLOCK          FOR    PHYSICAL
NAME           SIZE    ESTIMATE READ_FACTOR
------- ---------- ---------- -----------
DEFAULT       4096        980           1
DEFAULT       4096       1078       .9777
DEFAULT       4096       1176       .9751
...
```

ORACLE

Example

The example above shows a sample output from V$DB_CACHE_ADVICE. To interpret this you need to know the current number of buffers; in this example it is 983.

The first line of output shows that at 980 buffers you should expect to see the same number of physical reads as we do now. If you increased the buffers to 1078, you should expect to get 97% of the physical reads, or about a 3% reduction in physical reads.

The view also shows how many more reads will be done if the number of buffers is reduced. You will have a total of 20 rows in the output showing buffer sizes from 10% to 200% of the current buffer.

Note: The output has been formatted.

Note: The OEM Console has a graphical display of this information

Summary

In this lesson, you should have learned how to:

- **Setup automatic tuning of work areas by using the `PGA_AGGREGATE_TARGET` and `WORKAREA_SIZE_POLICY`**
- **Dynamically modify the buffer caches and shared pool sizes using `ALTER SYSTEM` command**
- **Enable and disable statistics gathering for predicting different cache size behavior**
- **Monitor work area allocation with `V$DB_CACHE_ADVICE`**

ORACLE

16

Enterprise Manager Enhancements

ORACLE

Objectives

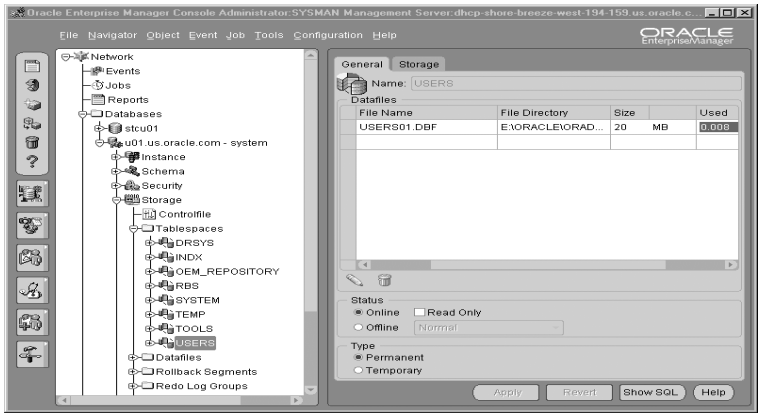
After completing this lesson, you should be able to do the following:

- **Describe the new look and feel of the Console**
- **Use the Console in standalone mode**
- **Explain Enterprise Manager functionality that supports Oracle9*i* database features**
- **Generate HTML reports**
- **Create user-defined events**

ORACLE

New Console Look and Feel

- **Two pane, master/detail view of the environment**
- **Events, Jobs, and Groups functionality integrated in the Navigator tree**
- **Database administration features fully integrated with Console**



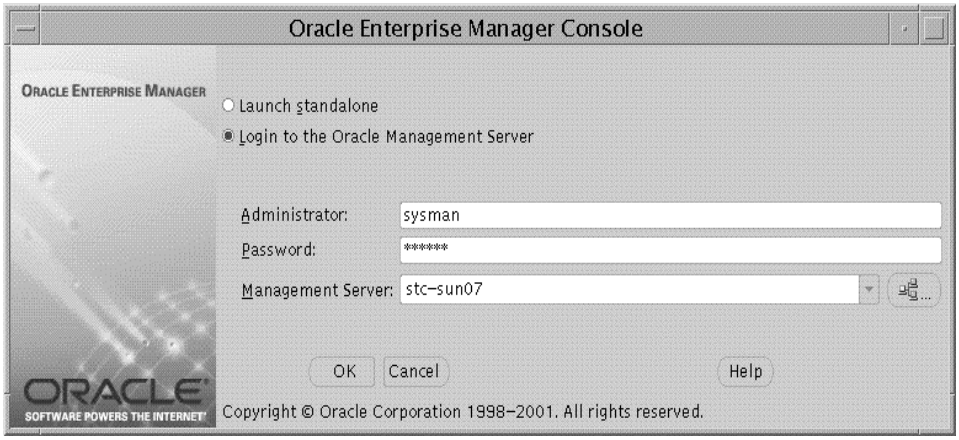
ORACLE

New Console Look and Feel

When user clicks on an item in the Navigator tree, the details of what has been selected appear on the right-hand side of the console.

For example, when user clicks on Events in the Navigator tree on the left, the detail side will show the Alerts, Registered, and History tabs, which contain details of Events for all targets.

Launching Enterprise Manager Console



ORACLE

Launching Enterprise Manager Console

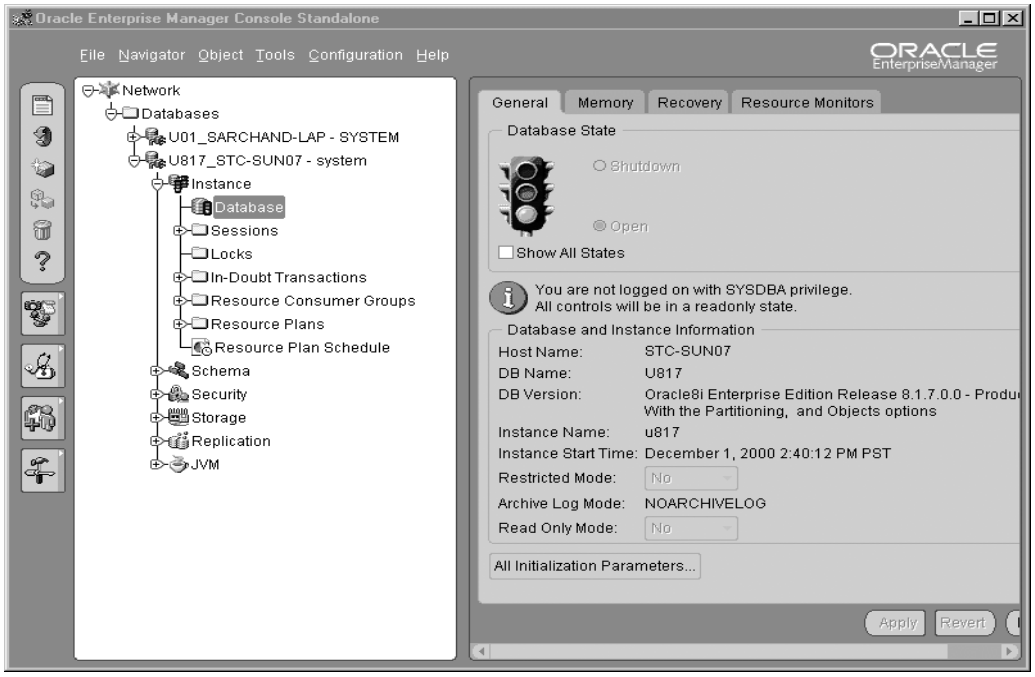
When you launch the Console, choose between standalone or logging to the middle-tier Oracle Management Server (OMS).

If the Management Server has not yet been installed nor configured, then you can launch the Console standalone and connect directly to target databases to perform administrative tasks such as deleting a table or creating a new database user. When launching standalone, you can connect directly only to Oracle databases; no other types of targets are currently supported for Oracle9*i*.

If the Management Server has been installed and configured, then you can launch the Console by logging into that Management Server. By logging into a Management Server, you have access to more comprehensive management capabilities.

Note: The choice between launching standalone or through the Oracle Management Server is also available through other Enterprise Manager applications.

Console Launched Standalone



ORACLE

16-5

Console Launched Standalone

When you launch the Oracle9*i* Console in standalone mode, it looks something similar to DBA Studio in release 2.1 or 2.2 but with more features and enhancements.

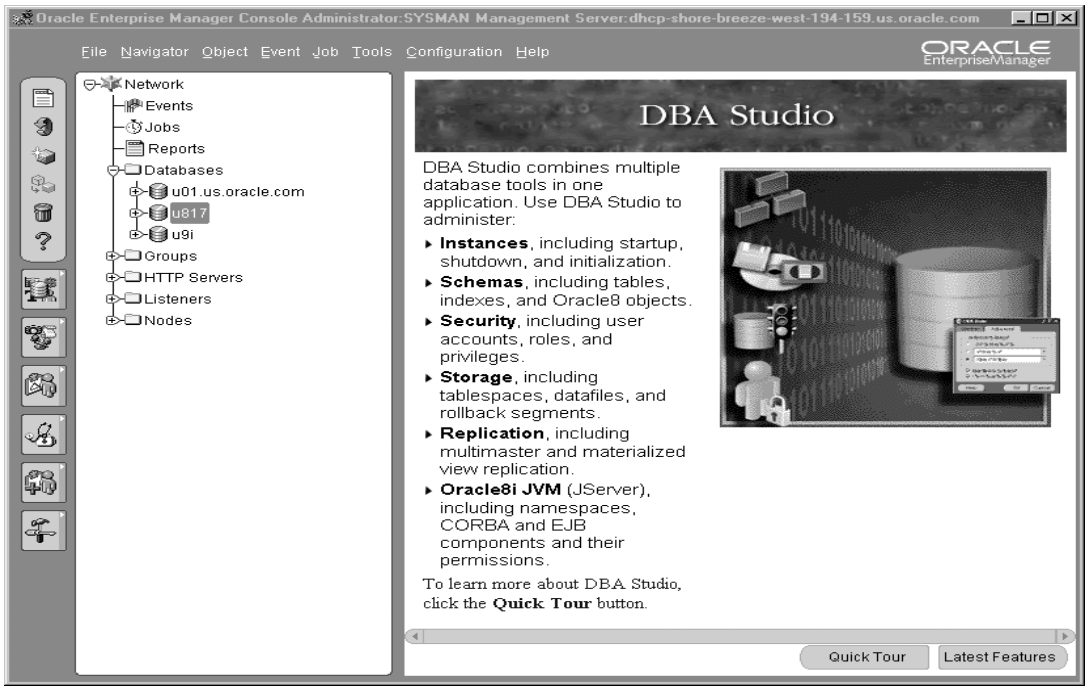
This is a screenshot of Console in standalone mode. The only targets in the navigator tree are databases. You add databases to the navigator tree as follows:

Adding Databases to the Navigator Tree

1. Launch Console standalone.
2. Add the databases you wish to manage using the Add Database to Tree dialog box. This dialog box appears automatically when you start the Console in standalone mode for the first time. It is also available from the Navigator main menu. You can manually enter the Net service names or add them from the local `tnsnames.ora` file.

There is no support for discovering nodes with Intelligent Agents when the Console is launched standalone.

Connections Using Management Server



ORACLE

Connections Using Management Server

There are several types of discovered targets in the navigator tree (for example, databases, HTTP Servers, Listeners, and so on) as well as Events, Jobs, and Reports. These additional types of targets and functionality would not be available if the Console were launched standalone; only connecting directly to databases is supported in standalone mode.

Standalone Connection Benefits

- **Available out-of-the-box**
- **Does not require installation or configuration of middle-tier Management Server**
- **Does not require installation of Intelligent Agent on administered database**
- **Connect directly to managed target (only supported for Oracle9*i* databases)**

ORACLE

Standalone Connection Benefits

A Standalone connection allows a single user to use one or more applications without the need of Oracle Management Server or Intelligent Agent. If you want to perform certain administrative tasks that do not require the job, event, or group system, the standalone Console can be used.

Standalone Connection Restrictions

- Only supports direct administration of Oracle databases; no management of other targets
- No access to Events, Jobs, and Groups
- Cannot share administrative information
- Cannot use Web-enabled applications
- No paging and e-mail blackouts
- Cannot store and access preferences
- Cannot use backup and data management tools
- Cannot customize, schedule, and publish reports

ORACLE

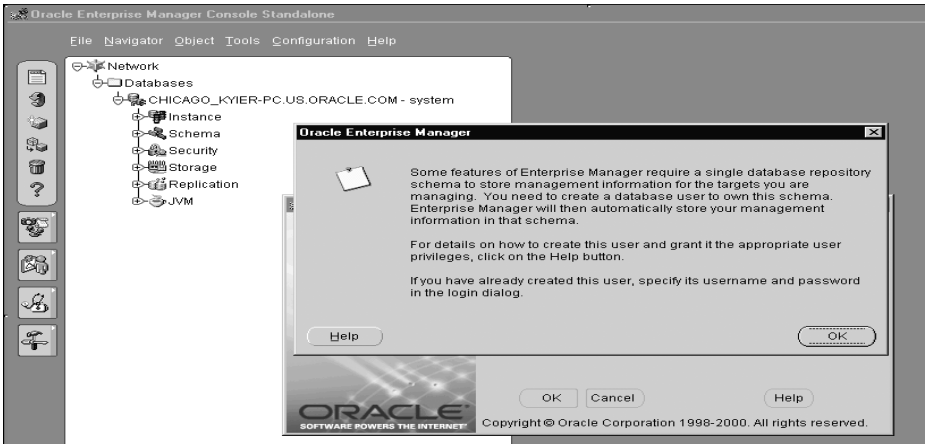
Standalone Connection Restrictions

If you need access to any features that are not available in Standalone mode, then you must install and configure the middle-tier Management Server and launch the Console by logging into the Management Server.

Note: As only databases are supported for Oracle9*i*, Management Pack for Oracle Applications and Management Pack for SAP R/3 do not support Standalone mode.

Standalone Repository

When launching Console standalone, some functionality requires standalone repository.



ORACLE

Standalone Repository

Standalone repository is a single user database repository schema that stores management information for the targets you are managing while using the Console in Standalone mode.

When launching the Console standalone and accessing certain Management Pack applications (for example, Oracle Change Manager, Oracle SQL Analyze, Oracle Index Tuning Wizard, Oracle Tablespace Map, and Oracle Expert) for the very first time, you will be prompted to create a single-user database repository schema to store management information for targets being managed. Once this standalone repository schema is created, it can be used by all five applications listed above; each application does not require its own standalone repository schema.

This single-user database repository schema is separate from the repository that is created when you install and configure a Management Server. This single-user database repository schema is for a single administrator and does not require a Management Server; while the repository used by a Management Server is for multiple administrators and is required by the middle tier. Furthermore, interaction, including migration or sharing of repository data, between the “standalone repository schema” and the “Management Server repository” is not supported. Database releases that support an Enterprise Manager release 9.0.1 standalone repository schema include:

- Enterprise Edition or standard edition, release 9.0.1
- Enterprise Edition or standard edition. release 8.1.7 and 8.1.6
- Enterprise Edition, release 8.0.6 (Objects Option must be installed and enabled)

Enterprise Manager Support for Oracle9*i* Database Features

- Server managed parameter file (`SPFILE`)
- Automatic Undo Management
- Unicode
- Dynamic Memory Management and Buffer Cache Sizing Advice
- Default temporary tablespace
- Multiple block sizes
- Mean Time to Recovery (MTTR)
- Backup and recovery enhancements
- Advanced Queuing

ORACLE

Enterprise Manager Support for Oracle9*i* Database Features

The Console has been enhanced to support new features within the database management system.

Creating the SPFILE

ORACLE

Creating the SPFILE

The SPFILE is initially created from the initialization parameter file. SYSDBA or SYSOPER privileges are required to create an SPFILE.

1. Right-click on the Configuration folder under Instance and click Import spfile, or select the Configuration folder and click Import spfile from the Object menu.
2. Enter the location of the PFILE. Use the Browse button to search for the files on the database machine. Enter the name of the SPFILE you want to create. Both the PFILE and the SPFILE must be located on the same machine as the database.
3. Click the OK button to create the SPFILE.

Creating a PFILE from SPFILE

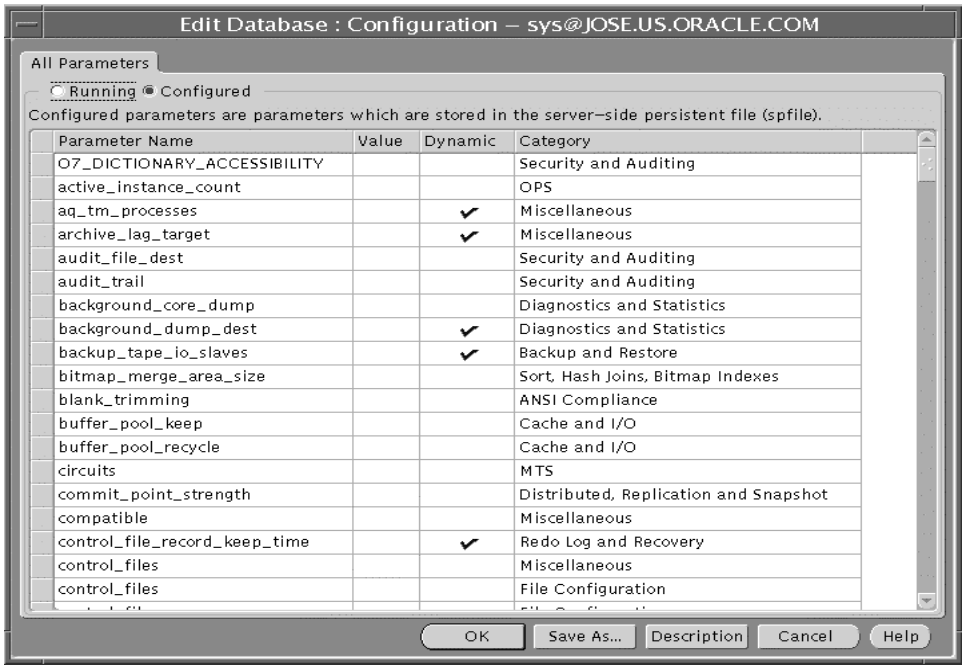
ORACLE

Creating a PFILE from SPFILE

You can export an SPFILE to create a text initialization parameter file. This may be necessary to modify the SPFILE, which is in binary format or create backups of the SPFILE. SYSDBA or SYSOPER privileges are required to create an `init.ora` parameter file from SPFILE.

1. Right-click on the Configuration folder under Instance and click Export spfile, or select the Configuration folder and click Export spfile from the Object menu.
2. Enter the name and location for the PFILE you want to create. Use the Browse button to search for the files on the database machine. Enter the name and location of the SPFILE. Both the PFILE and the SPFILE must be located on the same machine as the database.
3. Click the OK button to create the `init.ora` file.

Changing Parameters



ORACLE

Changing Parameters

The All Parameters page lists all initialization parameters.

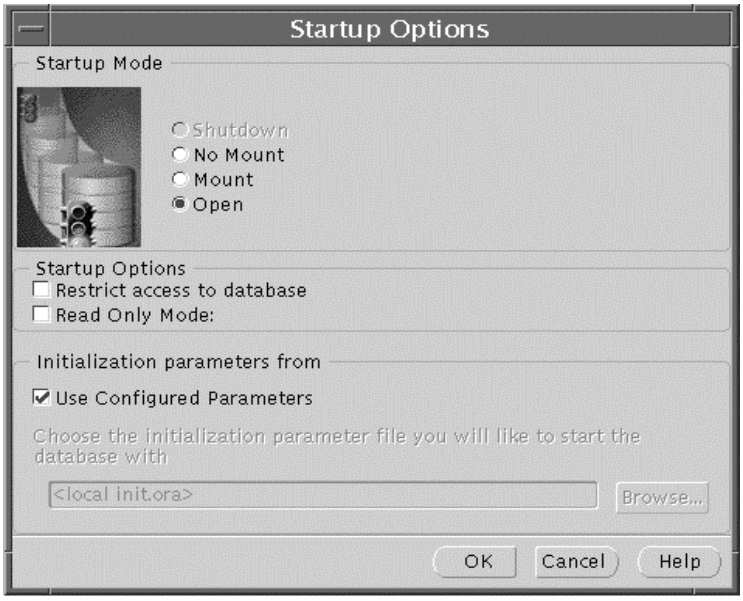
- For Oracle9*i* databases, you must be connected as SYSDBA to edit static initialization parameters. To edit dynamic parameters, SYSDBA privileges are not required.
- For non-Oracle9*i* databases you must be connected as SYSDBA to edit all parameters.

Changing Running and Configured Parameters

Running Parameters show the parameters that are currently running. If you change a dynamic parameter, the change is effected immediately. If the database is restarted, you start up with the original parameter value because it has not been saved to the SPFILE. If you change a static parameter, the changes are saved to a PFILE or SPFILE and take effect when the database is restarted.

Configured parameters are stored in the server-side parameter file. If you change a dynamic parameter, the changes are saved to an SPFILE and take place immediately.

Startup Using the SPFILE



ORACLE

Startup Using the SPFILE

You can use the startup dialog to start the instance.

1. Right-click on the database folder and select startup, or select startup from the Object menu.
2. If you want to startup the database using SPFILE, select the Use Configured Parameters check box. Oracle reads the initialization parameters from a server parameter file in a platform-specific default location.
3. If you want to use an `init.ora` file to start the database, deselect the Use Configured Parameters check box and supply the `init.ora` file to start the database.

Undo Tablespace Support

- **Simplifies and automates the management of rollback segments**
- **You have the choice of managing rollback segments or having Oracle automatically manage undo data in an undo tablespace.**
- **The rollback mode is set by the `UNDO_MANAGEMENT` parameter.**
- **New "undo" clause in the Create Tablespace dialog**
- **The Instance Management Undo tab displays all details about the undo tablespace.**

ORACLE

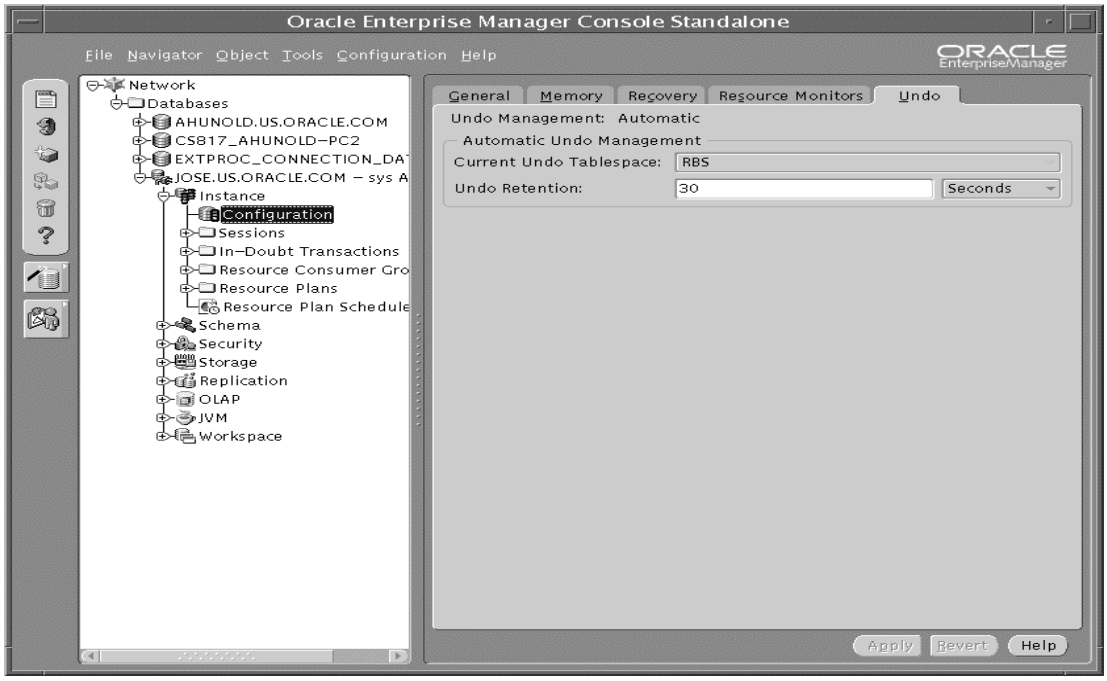
Undo Tablespace

With Enterprise Manager, you can determine if a database is in Automatic Undo Management mode. You specify whether or not the database is using rollback segments or an undo tablespace in the `init.ora` file. You should also specify which tablespace is the undo tablespace.

When you create a new tablespace, you can make it an undo tablespace by selecting the Undo option button in the Create Tablespace dialog. For an undo tablespace, all storage options are disabled because they are handled automatically.

In addition, in Instance Management a new Undo tab displays all details about the undo tablespace including the name of the current undo tablespace and retention time. The retention time is the time of the longest transaction or the farthest back you want to go with the server flashback feature.

Undo Tab

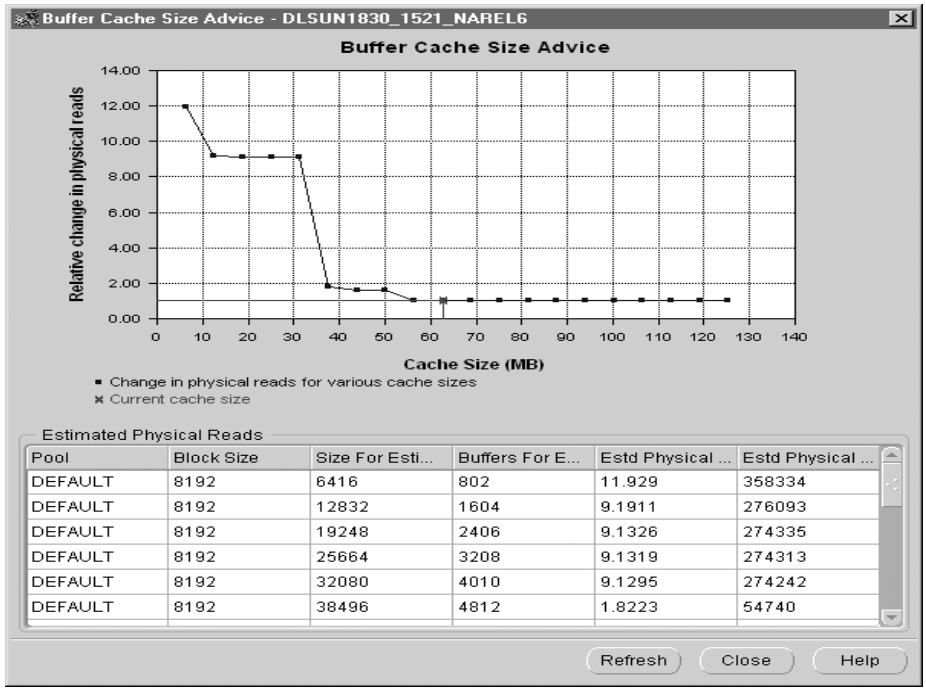


ORACLE

Creating an Undo Tablespace: Undo Tab

The Undo tab in Instance Management displays all details about the undo tablespace including the name of the current undo tablespace and the retention time. The retention time specifies the length of time to retain undo information. Committed undo information is normally lost when the undo space is overwritten by newer transactions. For read consistency purposes, if long running queries might require old undo information, the Undo Retention field provides a means of specifying the amount of undo information to retain.

Buffer Cache Size Advice View



ORACLE

Buffer Cache Size Advice View

You can view the Buffer Cache Size Advice information in Enterprise Manager as an alternative to querying `V$DB_CACHE_ADVICE`. The Enterprise Manager display includes a chart of the information which makes it much easier to view and interpret the data.

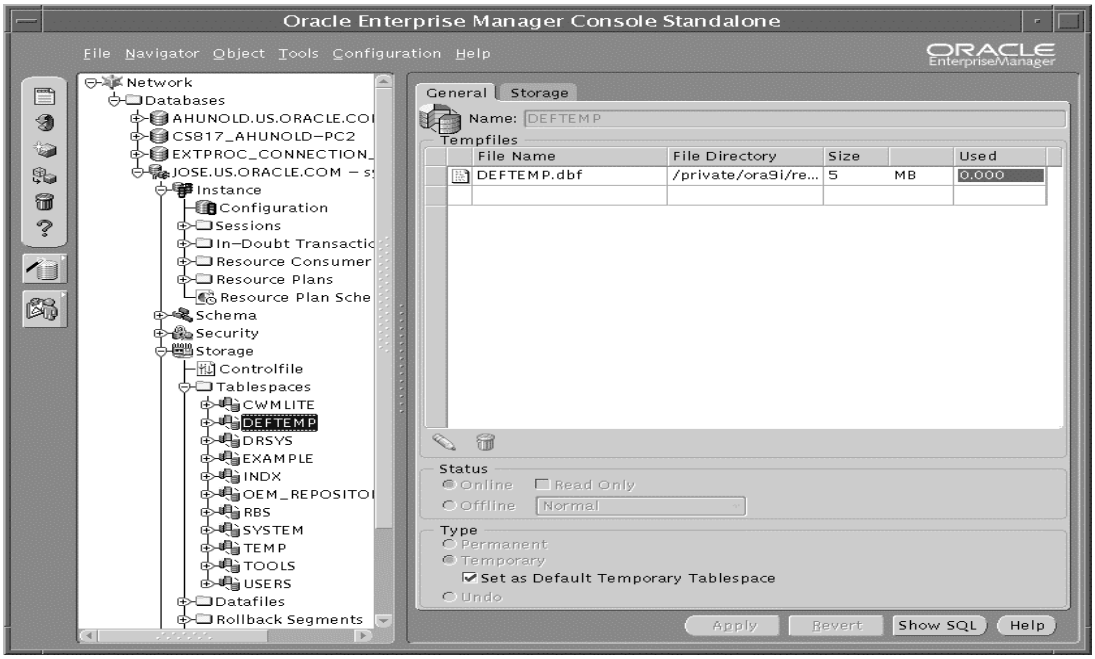
How to Launch Buffer Cache Size Advice

1. Click the Memory tab under Instance > Configuration
2. Click on Buffer Cache Size Advice to see the advice for choosing the size of the buffer cache
3. If the `db_cache_advice` initialization parameter has not been changed, a dialog box appears prompting whether you want to change the `db_cache_advice` initialization parameter. If Yes, the parameter will be changed.

How to Interpret the Graphical Display

As you move the mouse over the graphical display of Physical Reads versus Cache Size there will be text helping you interpret the data. For example, “If you set cache size to 5 MB the physical reads will increase by 2%.”

Creating Default Temporary Tablespace



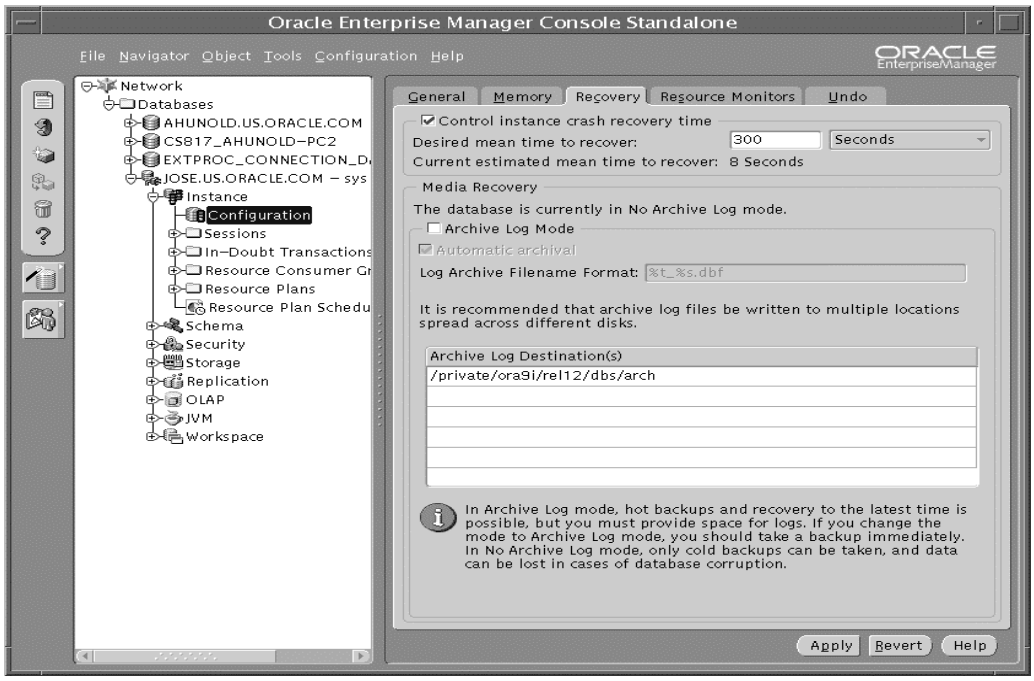
ORACLE

Creating Default Temporary Tablespace

Using Storage Management you can either create a new tablespace and set as the default temporary tablespace or define an existing temporary tablespace as the default temporary.

- Under the Create Tablespace folder, select the Type as temporary. This enables the Set as Default Temporary Tablespace check box.
- Select this check box to make this tablespace the Default temporary tablespace.
- You can also reassign the default temporary tablespace to another one.
- You cannot make the default temporary tablespace offline or make it permanent.

Mean Time to Recovery



ORACLE

Mean Time to Recovery

- Controls the time required to recover from an instance crash
- Specifies the maximum time you would like to spend to recover from an instance crash
- When you specify a value for this field, the FAST_START_MTTR_TARGET initialization parameter is changed automatically.

For example, you are running an e-commerce site and you want to recover from an instance crash within five minutes. You know the downtime can only be say, five minutes of unavailability. You would set the MTTR to be five minutes. The database then assures it can recover in that time.

Note: If this field is set to a short time, it can have negative impact on database performance.

Backup and Recovery Enhancements

- RMAN scripts can be submitted and scheduled through the Job system.
- RMAN Image Copy option support
- Requires Oracle9*i* Intelligent Agent
- Can be enabled in pre-Oracle9*i* Agents using `rman.tcl` file

ORACLE

Backup and Recovery Enhancements

To enhance the flexibility of the Backup and Recovery facility, the Job System supports an RMAN-specific job task. You can simply input any RMAN script and submit it through the Job system.

How to Enable RMAN Script in Pre-Oracle9*i* Agents

Add `rman.tcl` file into `$ORACLE_HOME/network/agent/jobs/oracle/rdbms/general` directory.

Advanced Queuing

- **Topology map displaying individual queues and their states on the database selected**
- **View errors and drill down to diagnostic detail**
- **View schedule details and messages**

ORACLE

Advanced Queuing

Advanced Queuing support is provided by introducing a new topology map that displays individual queues and their states on the database selected, including dblink status. You can see where there are errors and drill down to additional diagnostic detail.

HTML Database Reports

You can use report generation enhancements to generate fully formatted reports. You can generate:

- **Complete database configuration report**
- **Report of only the properties of an object selected in the Navigator tree**
- **Report of object dependencies**

ORACLE

HTML Database Reports

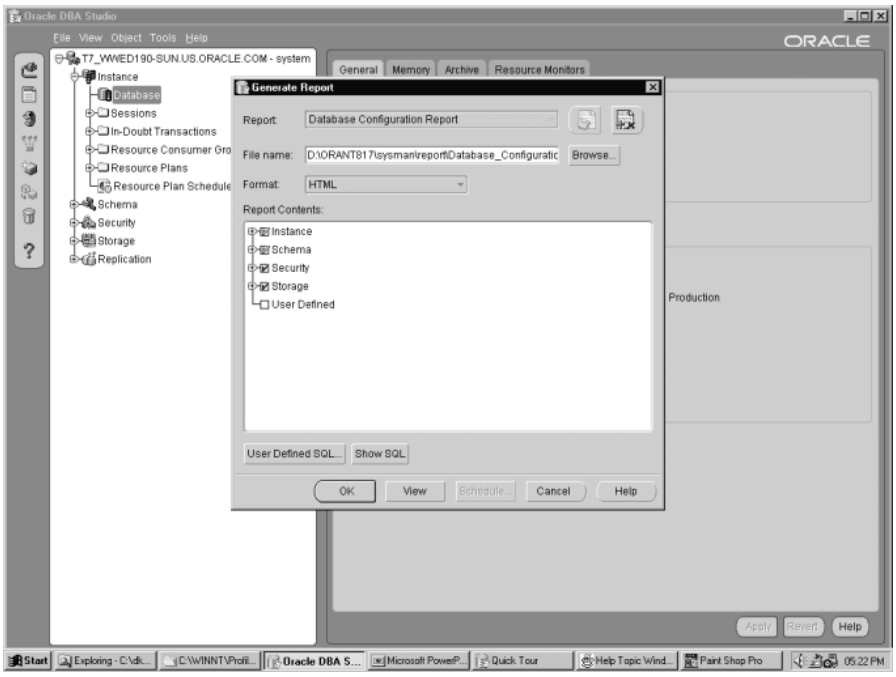
You can use report generation enhancements to generate fully formatted reports with just the information you need.

You can generate complete database configuration properties reports, or reports that contain only the properties of an object selected in the navigator tree.

When you launch report generation, the dialog menus that appear are appropriate to the currently selected object in the Navigator tree.

These reports are available in Standalone mode.

Database Configuration Report



ORACLE

Database Configuration Report

You can extract information from the database and save it in different formats (text, HTML, or Comma-Separated Values). For example, you can save the Instance, Storage, Schema information to an HTML file.

How to Generate an HTML Report

1. Highlight Database in the Navigator tree.
2. Click the Create Report icon in the toolbar, or right-click the database folder and click Create Report, or select Create Report from the Object menu.
3. Specify the desired contents of the report.
4. Click OK to generate the report or click View to generate and immediately view the report.

User-Defined SQL

Optionally, you can use user-defined SQL, to provide a SQL script to be added as an item under the Report Contents User Defined category.

Note: Each user defined SQL statement must be <= 2 KB.

User-Defined Events

- **Used to integrate any event monitoring script (in any language) with the Event System**
- **Allows customization of monitoring**
- **Extends the benefits of the Event System to existing user-defined monitoring scripts**
- **Requires Oracle9*i* Intelligent Agent**

ORACLE

User-Defined Events

You use user-defined events to integrate any monitoring script with the Event System. The Intelligent Agent will run any specified script that you may define to check for conditions specific to your environment. This allows much flexibility and customization to the type of event conditions monitored by the Intelligent Agent. This also extends the benefits of the Event System (cooperative monitoring among different administrators, notifications, history, and so on) to existing monitoring scripts.

The monitoring script itself should include the following:

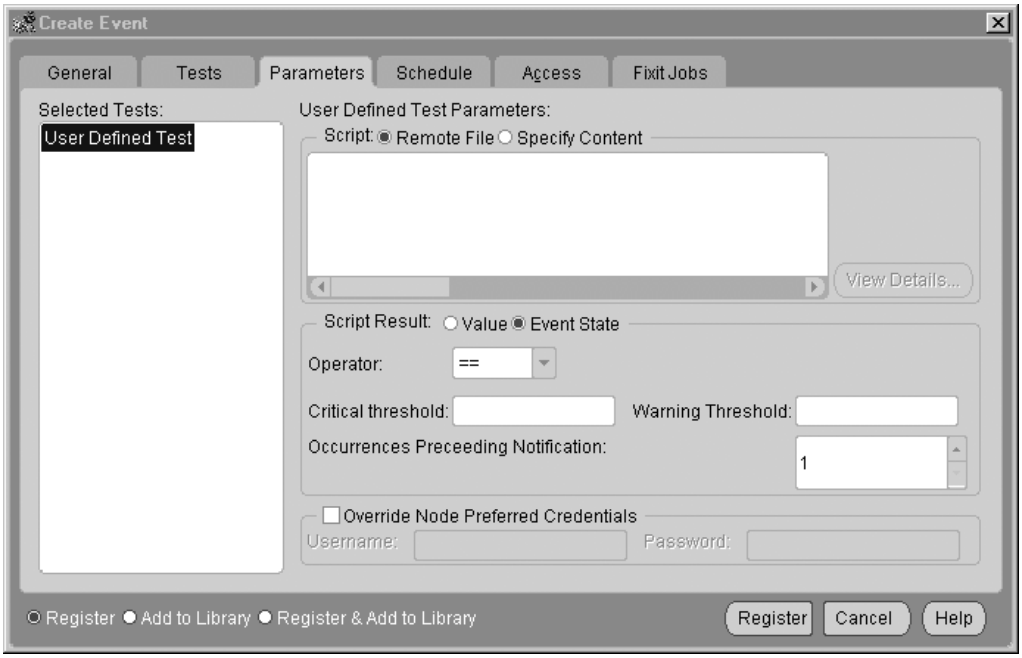
- Logic to check the specific event condition or value of the monitored metric (for example, the amount of free disk space)
- Logic to return the current value of the metric or status of the event and any associated message

If the current value of the metric is returned, then in the Create Event UI, you must also specify critical and warning thresholds against which the current value is compared.

Alternatively, the script can evaluate and return the status of the event (clear, critical, warning, error) which is then propagated back and triggered accordingly.

The script itself can be written in any language, as long as the run-time requirements needed by the Intelligent Agent to run the script are available on the monitored node.

User Defined Event Tests



ORACLE

User Defined Event Tests

Script: Specify the monitoring script. It can reside either on the monitored node or entered directly into the Create Event UI.

If the script resides on the monitored node, enter the fully qualified name of the script to be executed.

If the script is to be entered directly into the UI, enter the script commands. Alternatively, you can use your favorite editor to enter the script and save it to a file, then load it into the Create Event UI by means of the Browse button.

The Script Result

The script should return either the value of the monitored metric or the status of the event.

- Select Value if the script returns the value of the monitored metric.
- Select Event State if the script evaluates the event condition and returns an event status: clear, critical, warning, error.

User Defined Event Tests

If Value is selected for Script Result, the following additional parameters are required to determine how the event is evaluated:

- Operator: Specify the operator, Enterprise Manager should use to compare the monitored metric value against the thresholds. Comparison operators include == (equal to), < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), != (not equal to).
- Critical Threshold: The value against which the monitored metric is compared. If the comparison holds true, the event triggers at a Critical level.
- Warning Threshold: The value against which the monitored metric is compared. If the comparison holds true, the event triggers at a Warning level.
- Occurrences Preceding Notification: The number of times the event condition holds true before a notification is sent.

Override Node Credentials

Node credentials are required because the Intelligent Agent executes the script as the user specified in the node credentials. Default preferred credentials for the target node is used unless overridden in this field.

Refer to the online Help for more details.

Event Handler

- **Extends the Event System by allowing customizable responses**
- **Allows the user to respond to a triggered event by**
  - **Logging event information to a file and/or**
  - **Executing any O/S command**
- **Allows integration of third-party systems**
- **Service inside OMS**
  - **Takes advantage of OMS load-balancing and scalability features**

ORACLE

Event Handler

The Event Handler allows further processing when an event triggers. Specifically, it allows you to log event information to a file and/or execute any operating system command.

These two generic mechanisms provide much flexibility in allowing third-party integration. Any third-party application can then parse the event log for further processing.

Additionally, if you have a trouble-ticketing system that has a command line interface, the Event Handler can be used to execute the command required to open a trouble-ticket when the event triggers.

Summary

In this lesson, you should have learned how to:

- **Use the Console in Standalone mode**
- **Use Enterprise Manager functionality which supports Oracle9*i* database features**
- **Generate HTML reports**
- **Use user-defined events**

ORACLE

17

SQL Enhancements

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- **Use ISO/ANSI standard SQL syntax, such as joins, `CASE` expressions, `NULLIF`, `COALESCE`, scalar subqueries, `MERGE`, analytical functions**
- **Identify other SQL enhancements, such as constraint enhancements and `FOR UPDATE WAIT`**
- **Use the enhancements to LOBs and PL/SQL**

ORACLE

SQL:1999 Enhancements Overview

The most important enhancements are:

- **Full SQL:1999 join compliance**
- **Introduction of `CASE` expressions**
- **Introduction of scalar subqueries**
- **Support for explicit `DEFAULT` values**
- **The `MERGE` statement**
- **Additional analytical functions and grouping sets**
- **Naming query blocks with the `WITH` clause**

ORACLE

SQL:1999 Enhancements Overview

Throughout this lesson the phrase “SQL:1999” refers to the SQL:1999 notations of both ANSI and ISO/IEC standards, officially known as “ISO/IEC 9075-1:1999.”

SQL:1999 syntax compliance is important for the following reasons:

- Provides easier migration of third party applications without the need to modify existing SQL code
- Provides ANSI/ISO standard functionality within the Oracle9*i* database
- Provides easier learning curve when moving from other database products to Oracle9*i*

Note: For more details about Oracle and the ANSI/ISO SQL standard, please refer to one of the appendices of the *Oracle9i SQL Reference*.

SQL:1999 Joins

- **The *join type* is specified explicitly in the `FROM` clause in SQL:1999 syntax.**
- **The *join predicates* can be specified in the `ON` clause, separated from the `WHERE` clause.**
- **Join types:**
  - **Cross joins**
  - **Natural joins**
  - **Equijoins and the `USING` clause**
  - **Outer joins (full, left, right)**

ORACLE

SQL:1999 Joins

In Oracle9*i* the full SQL:1999 join syntax is implemented. Before Oracle9*i* you could only specify joins by listing multiple tables in the `FROM` clause (implicitly asking for a Cartesian product) and then "fix" the result in the `WHERE` clause by specifying appropriate join predicates. On top of that, you could specify a left or right outer join by adding the (+) syntax to your join predicates.

In Oracle9*i* the join is specified where it should be: in the `FROM` clause. The new syntax is easier to read and less error-prone, because you can separate the join predicates from the other (nonjoin) predicates. The new outer join syntax is more powerful because it also supports full outer joins.

Cross Joins

Equivalent with the Cartesian product of two tables

```
SQL> select c.country_name
  2  ,      r.region_name
  3  from   countries c
  4         CROSS JOIN
  5         regions r;
```

ORACLE

Cross Joins

The above example gives the same results as the following syntax (note the absence of a WHERE clause):

```
SQL> select c.country_name
  2  ,      r.region_name
  3  from   countries c
  4  ,      regions r;

COUNTRY_NAME REGION_NAME
------------ -----------
Argentina    Europe
Australia    Europe
Belgium      Europe
Brazil       Europe
...
Argentina    Americas
Australia    Americas
Belgium      Americas
Brazil       Americas
...
```

Natural Joins

- **The natural join is an equijoin based on all columns that have the same name.**
- **The join columns must contain compatible data.**
- **You cannot use an alias prefix (or table name) for a join column:**

```
ORA-25155: column used in NATURAL join
           cannot have qualifier
```

ORACLE

Natural Joins

In Oracle9*i* you can join two tables based on columns that have matching data types and names, by using the `NATURAL JOIN` syntax.

If two columns have the same name but different data types, then the `NATURAL JOIN` syntax can cause an error, depending on the actual data; a typical error is "ORA-01722: invalid number."

If you use the `SELECT *` syntax, then the common columns appear only once in the result set. In other words, the result set of a natural join does not have any ambiguous column names.

As a consequence, you do not need to use any table aliases (or prefix columns with table names) in your query. In fact, if you use an alias prefix (or table name) for a natural join column, you get an ORA-25155 error message. An example follows on the next page.

Natural Join Example

```
SQL> select department_id, location_id
  2  ,      city, country_id
  3  from   departments NATURAL JOIN
  4         locations;
```

DEPARTMENT_ID	LOCATION_ID	CITY	COUNTRY
10	1700	Seattle	US
20	1800	Toronto	CA
30	1700	Seattle	US
40	2400	London	UK
50	1500	South San Francisco	US
60	1400	Southlake	US
70	2700	Munich	DE
80	2500	Oxford	UK
...	...	...	...

ORACLE

Natural Join Example

In the above example the departments table is joined to the locations table by the location_id column, which is the only common column.

Note: In this example, you are not allowed to use an alias to qualify the location_id column. This is what happens:

```
SQL> select d.department_id, l.location_id
  2  ,      l.city, l.country_id
  3  from   departments d NATURAL JOIN
  4         locations   l;

select d.department_id, l.location_id
                        *
ERROR at line 1:
ORA-25155: column used in NATURAL join cannot have qualifier
```

Equijoins and the USING Clause

- **Apart from the natural join, you can also create a regular equijoin with the USING clause.**
- **Do not prefix columns referenced in the USING clause with a qualifier:**

```
ORA-25154: column part of USING clause
           cannot have qualifier
```

- **The NATURAL and USING keywords are mutually exclusive.**

ORACLE

Equijoins and the USING Clause

Natural joins unconditionally use all columns with matching names to join the tables. If two tables have several column names in common but you only want to join them on one of those matching columns, you can use the USING clause to specify only those columns that should be used for an equijoin.

Just like natural joins, join columns referenced in a USING equijoin should not have a qualifier (alias or table name) anywhere in the SQL statement. For example:

```
SQL> select l.city, d.department_name
  2  from   locations   l JOIN
  3         departments d USING (location_id)
  4  where  location_id = 1400;
```

is valid, but:

```
SQL> select l.city, d.department_name
  2  from   locations   l JOIN
  3         departments d USING (location_id)
  4  where  d.location_id = 1400;
```

is invalid because in the last line you have qualified location_id with a table alias.

Note: Both natural joins and equijoins with the USING clause do not allow join columns that are collections or LOBs.

USING Clause Example

```
SQL> select e.employee_id, e.last_name
  2  ,      d.location_id
  3  from   employees   e JOIN
  4         departments d USING (department_id)
  5  where  rownum < 6;
```

```
EMPLOYEE_ID LAST_NAME           LOCATION_ID
----------- ------------------- -----------
        100 King                       1700
        101 Kochhar                    1700
        102 De Haan                    1700
        103 Hunold                     1400
        104 Ernst                      1400
```

ORACLE

Join Predicates and the ON Clause

- **Used to separate join predicates from the other predicates**
- **The ON clause allows any predicate, including the usage of subqueries and logical operators.**

```
SQL> select e.employee_id,   e.last_name
  2  ,      d.department_id, d.location_id
  3  from   employees   e JOIN
  4         departments d ON
  5         (e.department_id = d.department_id)
  6  where  e.manager_id = 102;
```

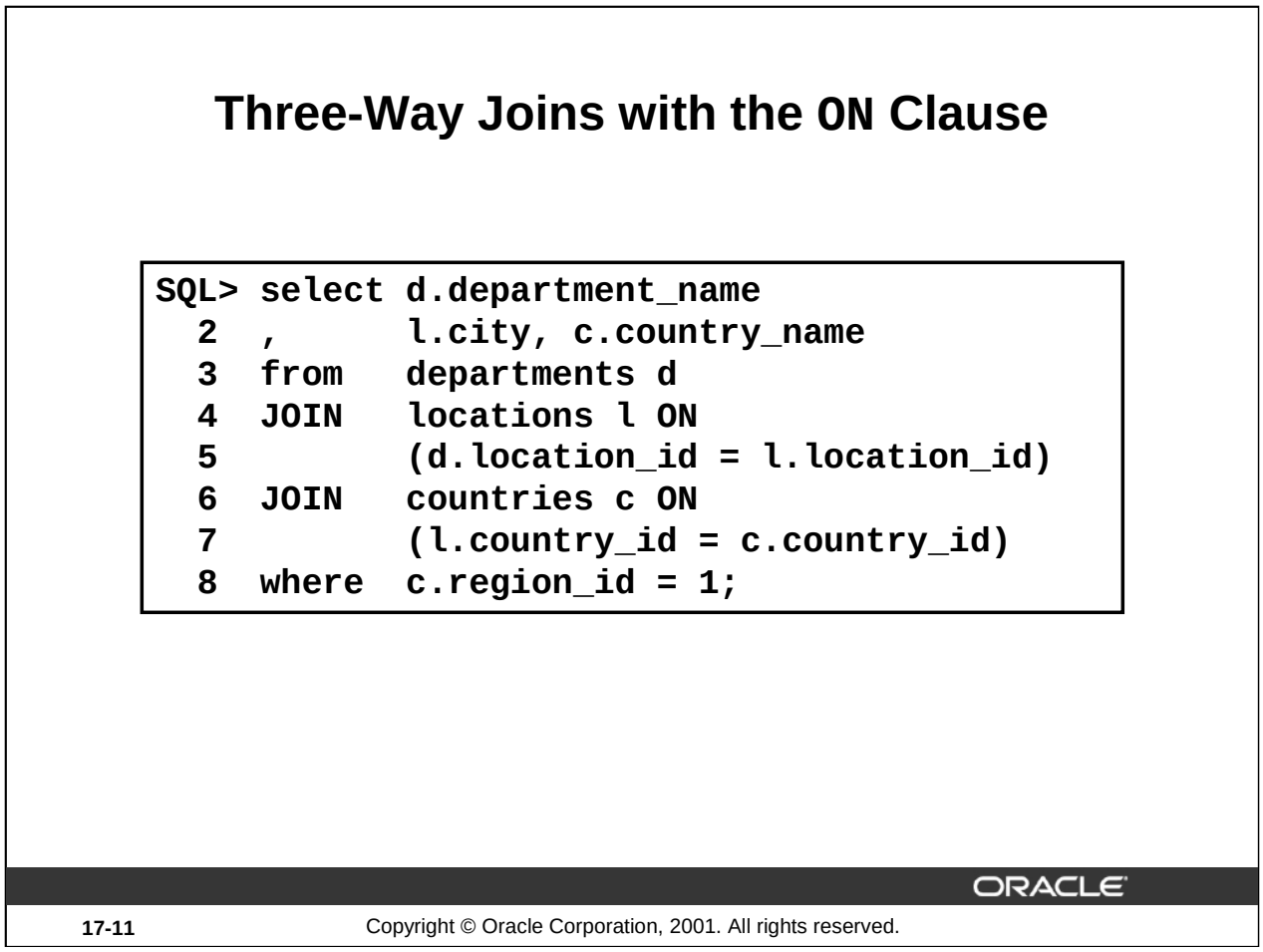
ORACLE

Join Predicates and the ON Clause

You can use the ON clause for any join to separate the join predicates from the nonjoin predicates. This makes your SQL code easier to understand:

```
SQL> select e.employee_id,   e.last_name
  2  ,      d.department_id, d.location_id
  3  from   employees   e JOIN
  4         departments d ON
  5         (e.department_id = d.department_id)
  6  where  e.manager_id = 102;

EMPLOYEE_ID LAST_NAME DEPARTMENT_ID LOCATION_ID
----------- --------- ------------- -----------
        103 Hunold               60        1400
```



Three-Way Joins with the ON Clause

In SQL:1999 compliant syntax, you can join multiple tables in a certain order by repeating ON clauses.

In the above example the departments and locations tables are joined first. The corresponding join predicate (on line 5) can only reference columns in those two tables; the second join predicate (on line 7) can reference columns from all three tables.

Note that the only nonjoin predicate is specified (on line 8) in the WHERE clause. Note also that although the SQL statement suggests a certain join ordering, the Oracle optimizer decides about the actual join order to be used at execution time.

The result of the above example is:

```
DEPARTMENT_NAME  CITY   COUNTRY_NAME
---------------- ------ --------------
Human Resources  London United Kingdom
Public Relations Munich Germany
Sales            Oxford United Kingdom
```

Outer Joins

- **Outer join types: `LEFT`, `RIGHT`, and `FULL`**
- **More powerful and readable than the (+) operator**

PROD	TYPE
P1	1
P2	2
P3	3
P4	3

TYPE	DESCR
2	D2
3	D3
4	D4

ORACLE

Outer Joins

Oracle has supported outer joins using the (+) syntax, for several versions. At the time this syntax was added to Oracle SQL, there was no ISO/ANSI standard for outer joins yet.

The (+) syntax is difficult to read, and it is also easy to make mistakes: if you forget to add one (+) string in your statement, you disable the outer join functionality. Moreover, the (+) syntax only allows a single sided (left or right) outer join.

Outer Join Types

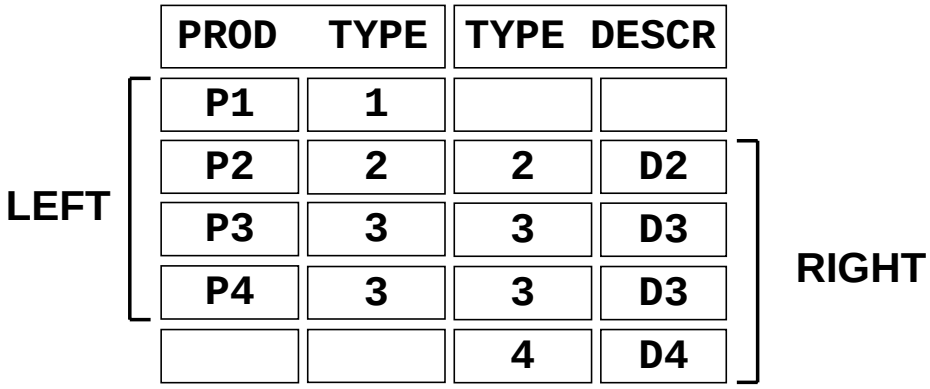
Suppose you have two tables: one with products (P) and one with product types (T).

To explain the concept of outer join types, suppose that these two tables are populated as above; note that you have one product (P1) with a nonexisting type (1) and also a type (4) without any corresponding products.

On the next page you see the differences between the three outer join types, using these two tables.

Outer Join Example

```
SQL> select p.prod, p.type
  2  ,      t.type, t.descr
  3  from   p {LEFT|RIGHT|FULL} OUTER JOIN
  4         t  ON (p.type = t.type);
```

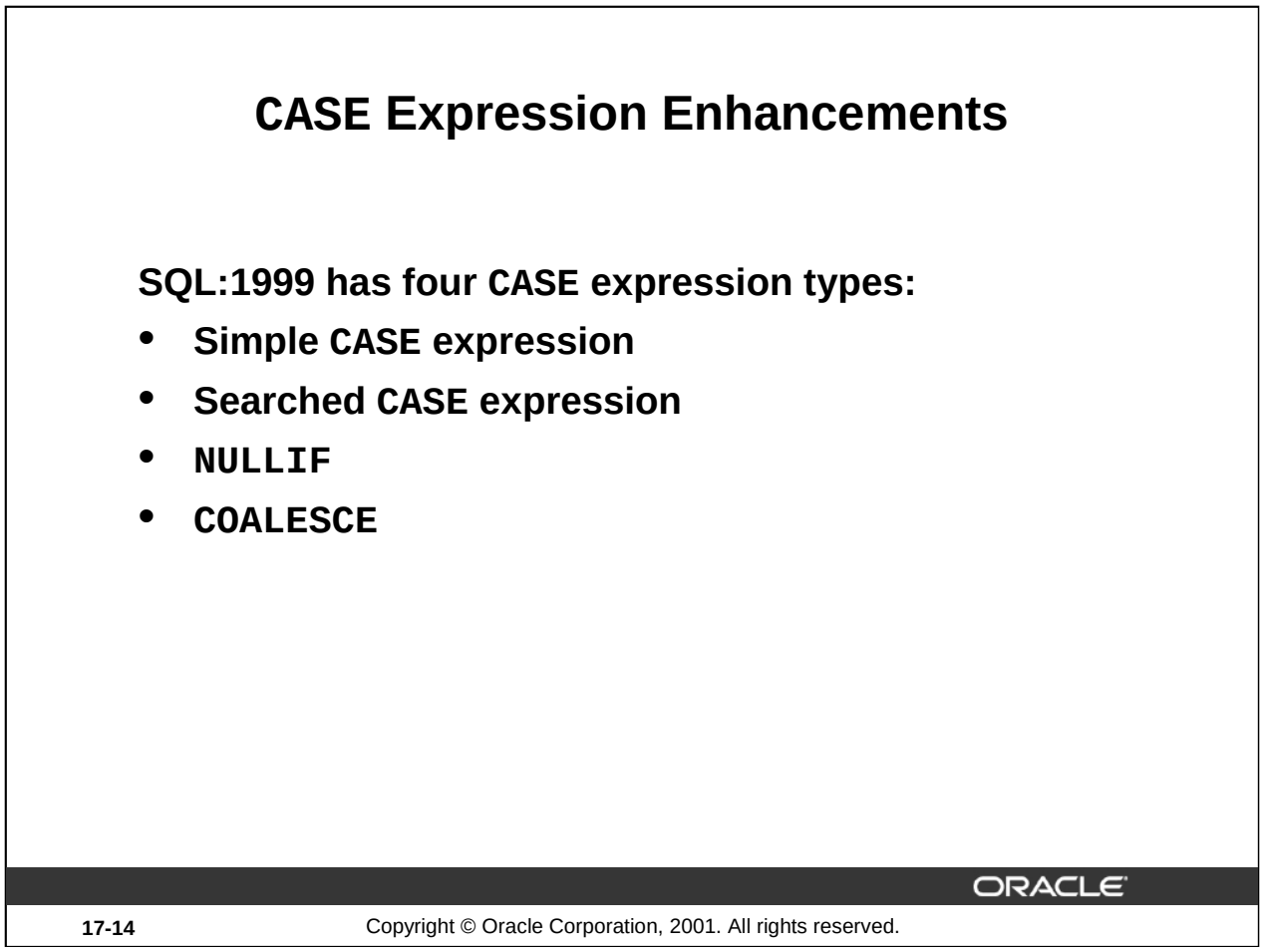


Outer Join Example

On the third line of the above statement, you must choose one of the three outer join types.

- The `LEFT` outer join retrieves the first four rows.
- The `RIGHT` outer join retrieves the last four rows.
- The `FULL` outer join retrieves all five rows.

Note: The keyword `OUTER` is optional; the keywords `LEFT`, `RIGHT`, and `FULL` implicitly assume an outer join.



CASE Expression Enhancements

The SQL:1999 standard has four types of CASE expressions:

- Simple CASE expression
- Searched CASE expression
- NULLIF
- COALESCE

The simple CASE expression was already available in Oracle8*i*. The other three types are new in Oracle9*i*.

Note: The CASE expression is also supported in PL/SQL. For more information please refer to *Oracle9i New Features for Developers*.

Simple CASE Expressions

- Similar to the DECODE function
- Search and replace values within an expression

```
SQL> select e.last_name
  2  ,     (CASE extract(year from e.hire_date)
  3         WHEN 1996 THEN ' 5 years of service'
  4         WHEN 1991 THEN '10 years of service'
  5         WHEN 1986 THEN '15 years of service'
  6                   ELSE '   maybe next year!'
  7         END )           as "Awards for 2001"
  8  from   employees e;
```

ORACLE

Simple CASE Expressions

In a simple CASE expression, each WHEN clause is followed by a literal value; in the example above you see the numbers 1996, 1991, and 1986. These literal values are compared with the CASE expression outcome for each selected row.

Note that the example also uses the ANSI/ISO standard EXTRACT(datetime) function to derive the four digit year from the hire_date column. The alternative expression would be:

```
TO_NUMBER(TO_CHAR(E.HIRE_DATE,'yyyy'))
```

Searched CASE Expressions

- Similar to an IF…THEN…ELSE construct
- Conditionally search and replace values within expressions

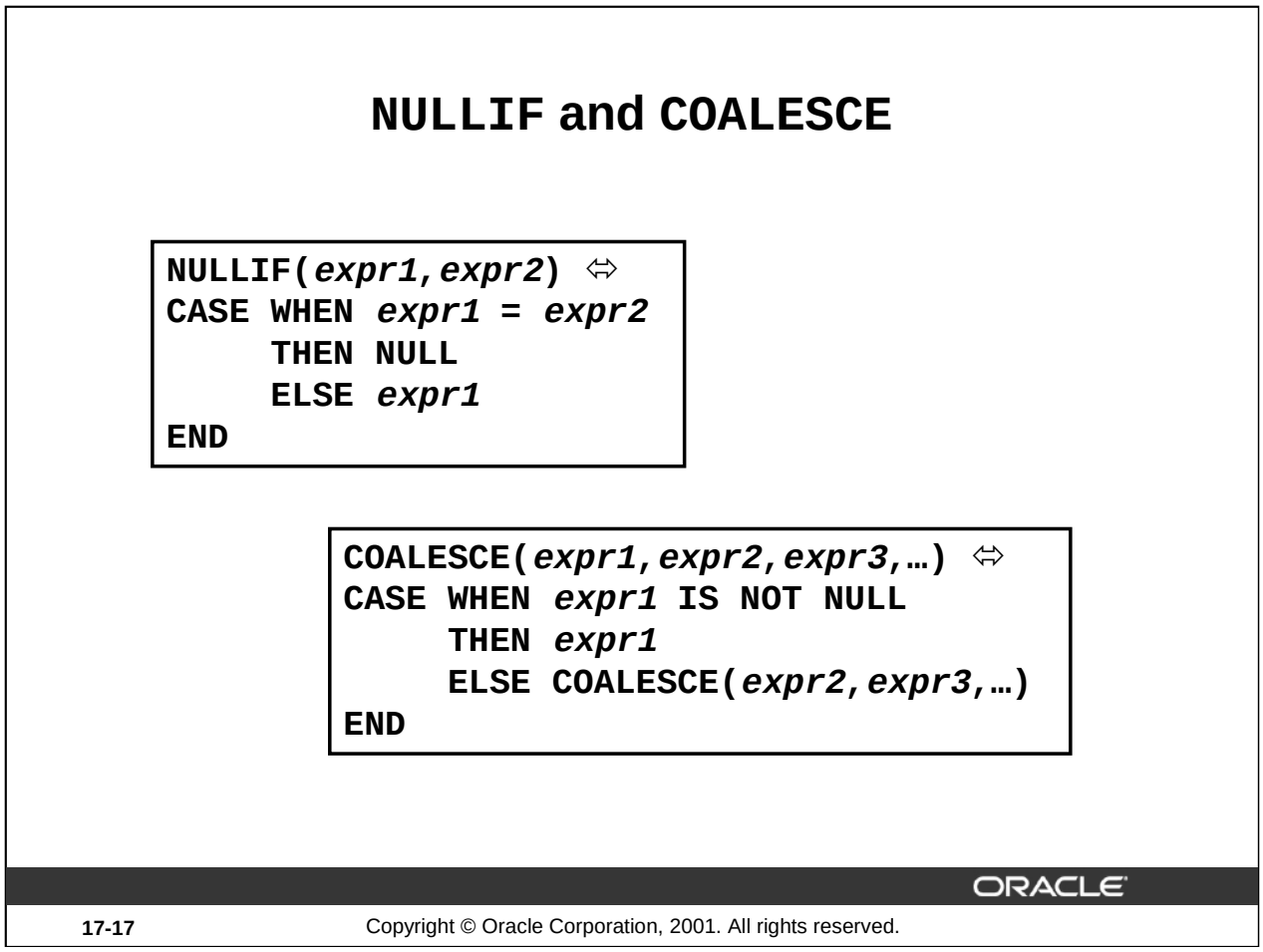
```
SQL> select e.first_name, e.last_name, e.job_id
  2  ,     (CASE
  3         WHEN e.job_id LIKE 'AD%'  THEN '10%'
  4         WHEN e.job_id LIKE 'IT%'  THEN '15%'
  5         WHEN e.first_name = 'Lex' THEN '18%'
  6                                   ELSE ' 0%'
  7         END )                     as "Raise"
  8  from   employees e;
```

ORACLE

Searched CASE Expressions

With a searched CASE expression, you can conditionally search and replace values; each WHEN clause is followed by a predicate. Each WHEN clause can have a different predicate, and you can use logical operators to combine multiple predicates.

Also note that in a searched CASE expression, the CASE clause is not followed by any expression but immediately by the first WHEN clause.



NULLIF and COALESCE

This slide introduces two new functions that are related to CASE expressions; the ANSI/ISO standard actually calls them "CASE abbreviations."

The double arrows in the slide denote the concept of "logical equivalence"; that is, you can use the CASE expressions as alternative formulations, regardless of the actual expressions you specify as arguments.

- The NULLIF function returns NULL if the first argument is equal to the second; otherwise, the value of the first argument is returned.
- The COALESCE function is a generalization of the NVL function; it accepts any number of arguments (two or more) and returns the first NOT NULL argument. If all expressions evaluate to NULL, the COALESCE function returns a NULL value.

The logical equivalence for the COALESCE function above is only true for three or more arguments. This logical equivalence is missing:

```
COALESCE(expr1,expr2) ⇔
CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END
```

As you can see, this is exactly the same as NVL(*expr1*,*expr2*).

The NULLIF function is useful in cases where a column contains coded default values. For example, if you have a default department 99 (with some application dependent meaning) and you want to treat the rows with that value as if they contained a NULL value when grouping, you can use a construct like NULLIF(d.department_id,99).

Scalar Subqueries

- **Return one row with one column value**
- **Limited support in Oracle8*i***
- **In Oracle9*i* they are allowed in any place where a scalar expression can be used.**
- **The data type of the return value must match the value being selected in the subquery.**

ORACLE

Scalar Subqueries

A scalar subquery is a single row subquery that selects only one expression. If a scalar subquery returns more than one row, an error is returned. If a scalar subquery returns no rows, the result of the scalar subquery is replaced by a NULL value.

Scalar subqueries are not supported in:

- Default values for column
- Returning clauses
- Hash expressions for clusters
- Function-based index expressions
- Check constraints on columns
- `WHEN` condition of triggers
- `GROUP BY` clauses

Note: The usage of scalar subqueries in the `SET` clause of an `UPDATE` statement and in the `VALUES` list of an `INSERT` statement was already supported in Oracle8*i*.

Scalar Subquery Example

```
SQL> select d.department_name
  2  ,     (select count(*)
  3         from   employees e
  4         where  e.department_id =
  5                d.department_id ) as empcount
  6  from   departments d;
```

ORACLE

Scalar Subquery Example

The results of the above example are as follows:

```
DEPARTMENT_NAME  EMPCOUNT
---------------- --------
Administration          1
Marketing               2
Purchasing              6
Human Resources         1
Shipping               45
IT                      5
Public Relations        1
Sales                  35
Executive               3
Finance                 6
Accounting              2
...
```

Explicit Defaults

```
SQL> insert into employees
  2  (employee_id, first_name, department_id)
  3  values (1, 'Scott', DEFAULT);
```

```
SQL> update employees
  2  set    department_id = DEFAULT
  3  where  department_id = 10;
```

ORACLE

Explicit Defaults

You can use the DEFAULT keyword in SQL statements to reference the default value that is stored with a column definition in the data dictionary.

You can leave out the DEFAULT keyword for insert statements, although it increases readability; you need it for update statements if you want to update a column to the default value.

DEFAULT provides extra flexibility because you can change the default value in the data dictionary, without the need to change your SQL statements.

The MERGE Statement

Also informally known as "upsert"

- **Perform an *update* if the row exists; otherwise, do an *insert***
- **Important in Data Warehousing applications**
- **Better performance; fewer statements and source table scans needed**

```
MERGE INTO  t1
      USING t2 ON (join_predicate)
WHEN  MATCHED     THEN UPDATE SET ...
WHEN  NOT MATCHED THEN INSERT(...) VALUES(...)
```

ORACLE

The MERGE Statement

Suppose `t1` is a large table and `t2` is a smaller table with rows that need to be inserted into `t1` conditionally.

Each row from `t2` is checked for a matching row in `t1` by evaluating the join condition specified by the `ON` clause:

- If there is a match, the row in `t1` is updated using the `UPDATE SET` clause
- Otherwise, rows are inserted as specified by the `INSERT` clause

Note that you can only use the `MERGE` statement with two tables; that is, you cannot merge a row unless it is already part of a table. However, the source table (`t2` in the example above) could be an external table.

Analytical Function Enhancements

- **Inverse percentile functions:**
  - **PERCENTILE_CONT**
  - **PERCENTILE_DISC**
- **What-if rank and distribution functions:**
  - **RANK, DENSE_RANK**
  - **PERCENT_RANK, CUME_DIST**
- **FIRST and LAST aggregate functions**
- **WIDTH_BUCKET function**
- **Grouping sets**

ORACLE

Analytical Function Enhancements

You can use inverse percentile functions to find the data that corresponds to a specified percentile value. The PERCENTILE_DISC function works with discrete values; the PERCENTILE_CONT function uses linear interpolation. For more details and examples, refer to the *Oracle9i SQL Reference.*

With what-if rank and distribution functions you can find out what rank or percentile value a hypothetical value would have if it were added to an existing data set.

You can use the FIRST and LAST functions to specify sorted aggregate groups and return the first or last value of each group.

The WIDTH_BUCKET function returns the histogram bucket in which a certain input value would fall, given a certain histogram specification.

Grouping sets are extensions to the GROUP BY clause that you can use to determine which result set rows are subtotals, and to specify the exact level of aggregation for a given subtotal.

All these analytical function enhancements provide significantly better query performance, in particular for ROLAP products such as Oracle Express. You can perform complex data analysis with much clearer and more concise SQL code; tasks which in the past required multiple SQL statements (or the use of procedural languages) can now be expressed using single SQL statements.

The syntax leverages existing aggregate functions, such as SUM and AVG, so that these well-understood keywords can be used in extended ways.

What-if Rank and Distribution Functions

In certain applications, such as financial planning, you may want to know how a data value would rank if it is added to a data set. For instance, a new employee is hired at a salary of $100,000; where would his salary rank compared to the other salaries in the company?

The hypothetical rank and distribution functions support this form of what-if analysis. They return the rank or percentile value which a row would get if the row was hypothetically inserted into an existing set of rows.

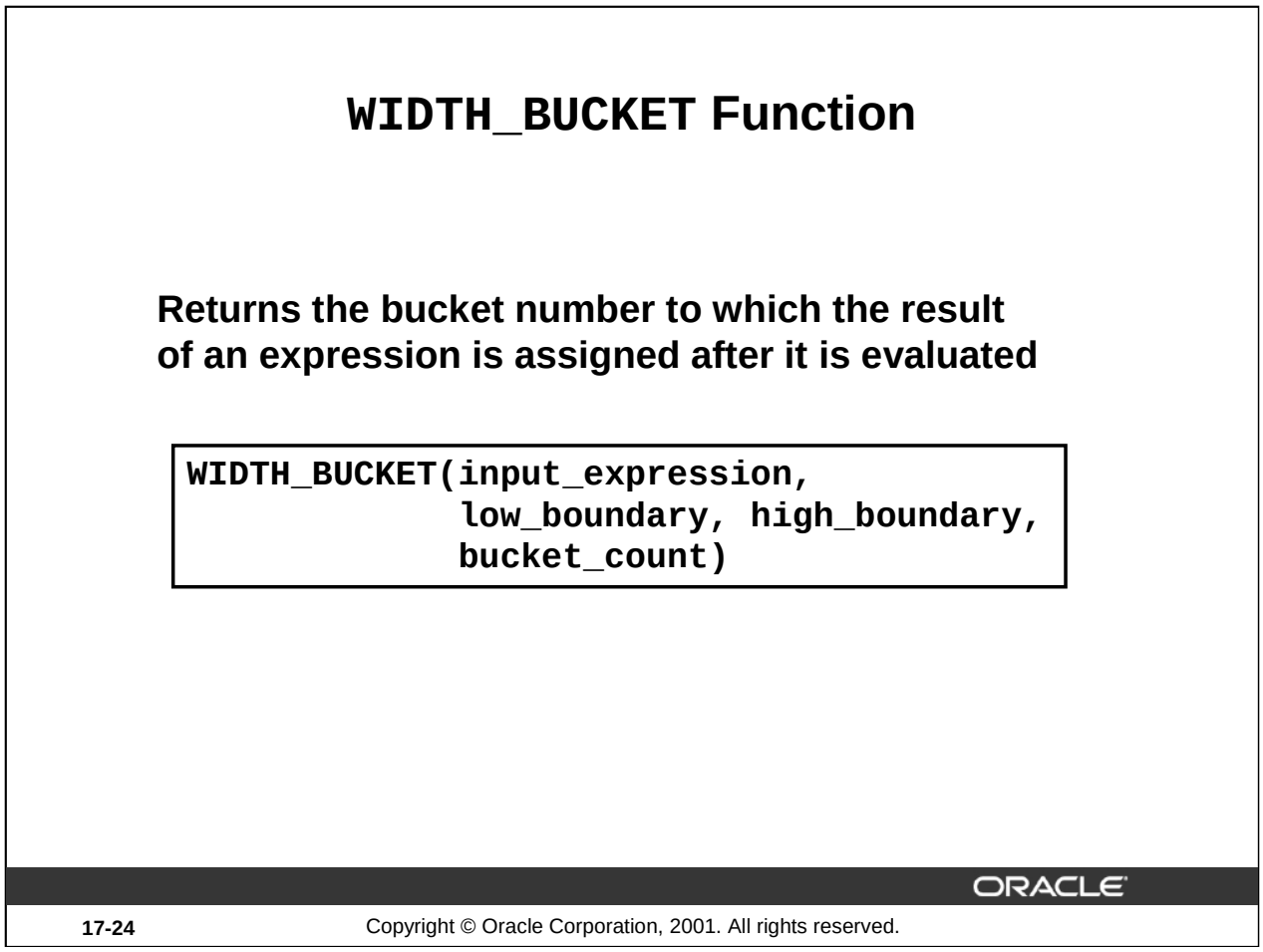
The hypothetical functions can calculate `RANK`, `DENSE_RANK`, `PERCENT_RANK`, and `CUME_DIST`. They use a `WITHIN GROUP` clause containing an `ORDER BY` specification.

Note: The `RANK` and `DENSE_RANK` functions are already available with Oracle8*i*, but in Oracle9*i* they are enhanced to accept additional arguments for what-if analysis.

First and Last Aggregate Values

You can use `FIRST` and `LAST` aggregate functions to specify an order within the aggregated groups and then return the first or last row of each group.

While an equivalent query can be created using a join or subquery, the SQL syntax is cumbersome and performance can be inefficient. The `FIRST` and `LAST` functions do this work with simpler SQL syntax and greater performance.



WIDTH_BUCKET Function

For any given expression, the WIDTH_BUCKET function returns the bucket number to which the result of this expression is assigned after it is evaluated.

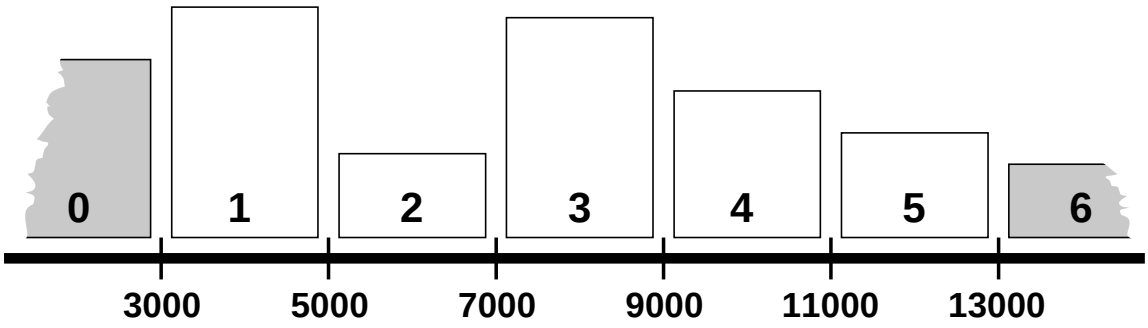
You can generate equiwidth histograms with this function. Equiwidth histograms divide data sets into buckets with an equal interval size.

You provide the input expression, the minimum boundary value, the maximum boundary value, and the number of buckets.

Note: If you ask for (*n*) buckets, you actually get (*n* + 2) buckets; the two additional buckets will hold any values below the low boundary value or above the high boundary value.

WIDTH_BUCKET Example

```
SQL> select last_name, salary,
  2         WIDTH_BUCKET(salary,3000,13000,5)
  3  from   employees;
```



ORACLE

WIDTH_BUCKET Example

Salaries less than 3000 are placed in bucket 0; salaries greater than 13000 are placed in bucket 6. The other salaries are placed in buckets 1 to 5, depending on the salary value.

```
LAST_NAME       SALARY WIDTH_BUCKET
--------------- ------ ------------
King             24000            6
Kochhar          17000            6
De Haan          17000            6
Hunold            9000            4
Ernst             6000            2
Austin            4800            1
Pataballa         4800            1
Lorentz           4200            1
Greenberg        12000            5
Faviet            9000            4
Sciarra           7700            3
Popp              6900            2
Raphaely         11000            5
Khoo              3100            1
Baida             2900            0
...
```

Grouping Sets

- **Superset of `GROUP BY {ROLLUP|CUBE}`**
- **Produces a single result set, which is equivalent to a `UNION ALL` approach**

```
SQL> select time_id, channel_id, prod_id
  2  ,      sum(amount_sold) as amount
  3  from   sales
  4  where  ...
  5  group by GROUPING SETS
  6       ((time_id, channel_id, prod_id)
  7       ,(time_id, channel_id)
  8       ,(channel_id, prod_id)
  9       );
```

ORACLE

Grouping Sets

A grouping set is a set of groupings that you want the system to perform, in order to apply aggregate functions on those groupings. This is done in the `GROUP BY` clause.

In Oracle8*i* the `CUBE` and `ROLLUP` enhancements to the `GROUP BY` clause were introduced; grouping sets in Oracle9*i* add more flexibility.

Without the enhancements in Oracle9*i*, you need multiple queries combined together with `UNION ALL` to achieve these tasks. A multiquery approach is inefficient, because it requires multiple scans of the same data. Grouping sets allow the optimizer to choose better plans, resulting in better query performance.

The `GROUPING SET` clause allows you to identify exactly the groups you are interested in. For example, `GROUP BY CUBE (a,b,c)` produces eight groupings in total; but maybe you are only interested in three of those eight groupings.

In the example, you specify three groupings (on lines 6, 7, and 8):

- (time_id, channel_id, prod_id)
  This results in aggregate information per day per channel per product
- (time_id, channel_id)
  This results in aggregate information per day per channel
- (channel_id, prod_id)
  This results is aggregate information per channel per product

Grouping Sets

TIME_ID	CHAN	PROD_ID	AMOUNT	
08-DEC-99	I	20	112	Day, channel, and product
08-DEC-99	I	45	79	
08-DEC-99	S	20	42	
08-DEC-99	S	45	1422	
09-DEC-99	I	20	168	
09-DEC-99	T	20	70	
08-DEC-99	I		191	Day and channel
08-DEC-99	S		1464	
09-DEC-99	I		168	
09-DEC-99	T		70	
	I	20	280	Channel and product
	I	45	79	
	S	20	42	
	S	45	1422	
	T	20	70	

ORACLE

Grouping Sets (continued)

You can read the first five rows of the result above as follows:

- On December 8, through internet sales, product 20 was sold for an amount of 112

The second section (rows 7, 8, 9, and 10) is interpreted as:

- On December 8, through internet sales, the total amount was 191 (112 + 79)
- On December 8, through direct sales, the total amount was 1464 (42 + 1422)
- On December 9, through internet sales, the total amount was 168
- On December 9, through telesales, the total amount was 70

The last five rows should be interpreted as:

- Through internet sales, product 20 was sold for a total of 280 (112 + 168)
- Through internet sales, product 45 was sold for a total of 79
- Through direct sales, product 20 was sold for a total of 42
- Through direct sales, product 45 was sold for a total of 1422
- Through telesales, product 20 was sold for a total of 70

Composite Columns

Treat a group of columns as a unit:

- **GROUP BY ROLLUP (a,b,c) gives four groupings**
- **GROUP BY ROLLUP (a,(b,c)) gives three groupings**

```
SQL> select ...
  2  from   sales
  3  where  ...
  4  group by rollup
  5  (prod_id,(channel_id, time_id));
```

ORACLE

Composite Columns

A composite column is a collection of columns that are treated as a unit during the computation of groupings. With CUBE and ROLLUP, you do not have full control over the aggregation levels. For example, the following statement:

```
SQL> select ...
  2  from sales
  3  where ...
  4  group by rollup(prod_id, channel_id, time_id);
```

would result in computing four groupings:

- (product, channel, time)
- (product, channel)
- (product)
- (grand total)

However, the example in the slide above results in only three groups:

- (product, channel, time)
- (product)
- (grand total)

Composite Columns

PROD_ID	C	TIME_ID	AMOUNT	
20	C	02-DEC-99	14	Totals per product/channel/time
20	I	01-DEC-99	266	
20			280	Total for product 20
45	C	01-DEC-99	79	Totals per product/channel/time
45	C	02-DEC-99	4187	
45	S	01-DEC-99	316	
45	T	01-DEC-99	790	
45	T	02-DEC-99	158	
45			5530	Total for product 45
			5810	Grand total

ORACLE

Composite Columns (continued)

This slide shows the results for the example query on the previous page. As you can see, the result shows groupings for:

- (product, channel, time)
- (product)
- (grand total)

Concatenated Groupings

- **Produce cross-products of multiple groupings**
- **Concatenated groupings are specified by listing multiple grouping sets, cubes, and rollups**

```
SQL> select   prod_id, channel_id, time_id
  2  ,        sum(amount_sold)
  3  from     sales
  4  where    ...
  5  GROUP BY prod_id
  6  ,        cube(channel_id)
  7  ,        rollup(time_id);
```

ORACLE

Concatenated Groupings

Concatenated groupings offer a concise way to generate useful combinations of groupings. Groupings specified with concatenated groupings yield the cross-product of groupings from each grouping set. The cross-product operation enables even a small number of concatenated groupings to generate a large number of final groups. The concatenated groupings are specified by listing multiple grouping sets, cubes, rollups, and separating them with commas.

The example on the slide results in the following groupings: (product, channel, time), (product, channel), (product, time), and (product).

WITH Clause

- **Name a query block in a SELECT statement to reference it more than once within a query**
- **The WITH clause can hold multiple query blocks, separated by commas.**
- **Resolved as in-line views or temporary tables**

ORACLE

WITH Clause

Using the WITH clause, you can reuse the same query block when it is expensive to evaluate the query block and it occurs more than once within a complex query.

You can use the WITH clause to define a query block before using it in a query. This can improve performance, and it certainly improves readability and maintainability of your SQL statements; using the WITH clause, you can isolate the business question from the data gathering.

A query name is visible to all WITH element query blocks (including their subquery blocks) defined after it and the main query block itself (including its subquery blocks).

A query name can be the same as some persistent table name or query name in the WITH list of another query block. If this happens, the parser searches for the right definition inside out (in terms of query block nesting). The inner-most query name definition is used for resolution. This is similar to the C programming language, in which a local variable has priority when its name is the same as some global variable.

The WITH clause is internally resolved as an in-line view or a temporary table; the cost-based optimizer chooses the appropriate resolution.

WITH Clause Example

```
SQL> WITH summary as (
  2  select   d.department_name
  3  ,        sum(e.salary) as dept_total
  4  from     employees e, departments d
  5  where    e.department_id = d.department_id
  6  group by d.department_name)
  7  select   department_name, dept_total
  8  from     summary
  9  where    dept_total >
 10          (select sum(dept_total) * 1/8
 11           from   summary )
 12  order by dept_total desc;
```

ORACLE

WITH Clause Example

The query on the slide retrieves all departments whose total salary cost is above 1/8 the total salary cost of the whole company.

Alternative Formulation

Without using the WITH clause, the query would typically look like:

```
SQL> select   d.department_name
  2  ,        sum(e.salary) as dept_total
  3  from     employees e, departments d
  4  where    e.department_id = d.department_id
  5  group by d.department_name
  6  having   sum(e.salary) >
  7          (select sum(e.salary) * 1/8
  8           from   employees e, departments d
  9           where  e.department_id = d.department_id)
 10  order by sum(e.salary) desc;
```

Using the WITH clause, you can materialize the query block that does the GROUP BY, thus avoiding summarizing the department total cost more than once.

Constraint Enhancements

- **Explicit index control**
- **Less foreign key locking overhead**
- **Cached primary key lookup**
- **Primary keys and foreign keys on views**

ORACLE

Constraint Enhancements

These four constraint enhancements are discussed in more detail in the following pages.

Explicit Index Control

```
SQL> create table employees
  2  ( employee_id number(6)
  3                constraint emp_pk
  4                primary key USING INDEX
  5               (CREATE INDEX emp_pk
  6                ON employees(employee_id))
  7  , first_name  varchar2(15)
  8  , ...                                  );
```

```
ALTER TABLE table_name
{DROP|DISABLE} CONSTRAINT constraint_name ...
{DROP|KEEP} INDEX
```

ORACLE

Explicit Index Control

In the above example the CREATE INDEX clause is used in the CREATE TABLE statement to create an index for the primary key constraint explicitly. The full CREATE INDEX syntax can be used, including the parallel and partition clauses, to make index creation more efficient.

You can specify a CREATE INDEX clause in a similar way for UNIQUE constraints.

When dropping or disabling (primary or unique) constraints, you can explicitly specify whether you want the associated index to be dropped. By default, unique indexes that are implicitly created during the creation of primary and unique keys are dropped when the constraints are dropped.

Less Foreign Key Locking Overhead

If a foreign key column is unindexed:

- **A table-level share lock is still needed when doing an update or delete on the primary key**
- **However, the lock is released immediately after obtaining it**

ORACLE

Less Foreign Key Locking Overhead

A table-level share lock is placed on unindexed foreign keys when doing an update or delete on the primary key. However, this lock is released immediately after obtaining it. If multiple primary keys are updated or deleted, the lock is obtained and released once per row. The obtaining and releasing of the shared lock are coded as follows:

- Get a savepoint
- Obtain a share lock
- Rollback to savepoint when needed

The share lock has only one purpose, to check whether you have pending transactions against any rows. If that is the case, the share lock request would fail because of exclusive row locks.

Cached Primary Key Look Up

- **Primary key values are cached:**
  - **Reduced primary key lookup time**
  - **Faster foreign key creation**
- **Only done for multirow DML statements**

ORACLE

Cached Primary Key Look Up

The look up of matching primary keys at the time of foreign key insertion takes time. In Oracle9*i* the first 256 primary key values can be cached, so the addition of multiple foreign keys becomes significantly faster.

The cache is only set for the second row processed; this avoids the overhead of setting up a cache for single row DML statements.

Constraints on Views

- **Constraints help to identify cubes; this is a problem with views in Oracle8*i*.**
- **Constraints on views enable:**
  - **DW applications to identify view-based cubes**
  - **Additional query rewrites for materialized views based on views**
- **Constraints on views are only declarative; only table constraints are enforced.**

ORACLE

Constraints on Views

Data warehouse applications recognize multidimensional cubes by identifying referential integrity constraints in the data dictionary. However, this does not work in an environment where you have views on top of your fact and dimension tables. In such environments, applications cannot identify the cubes properly.

By allowing constraint definitions on views in Oracle9*i*, you can propagate base table constraints to the views thereby allowing applications to recognize the cubes.

View constraint definitions are declarative in nature; therefore, DML operations on views are subject to the constraints defined on the underlying base tables.

Defining constraints on base tables is necessary, not only for data correctness and cleanliness, but also for materialized view query rewrite purposes.

View Constraint Types

- **The following constraint types are supported:**
  - **Primary key**
  - **Unique**
  - **Referential integrity: Tables can reference view constraints and vice versa**
- **Supported constraint states:**
  - **`DISABLE NOVALIDATE` only**
  - **`RELY` or `NORELY` both allowed**

ORACLE

View Constraint Types

Oracle9*i* supports the following constraints on views: primary key, unique, and foreign key constraints. A regular table can reference a primary (or unique) key constraint defined on a view, and a view can reference a constraint defined on a table.

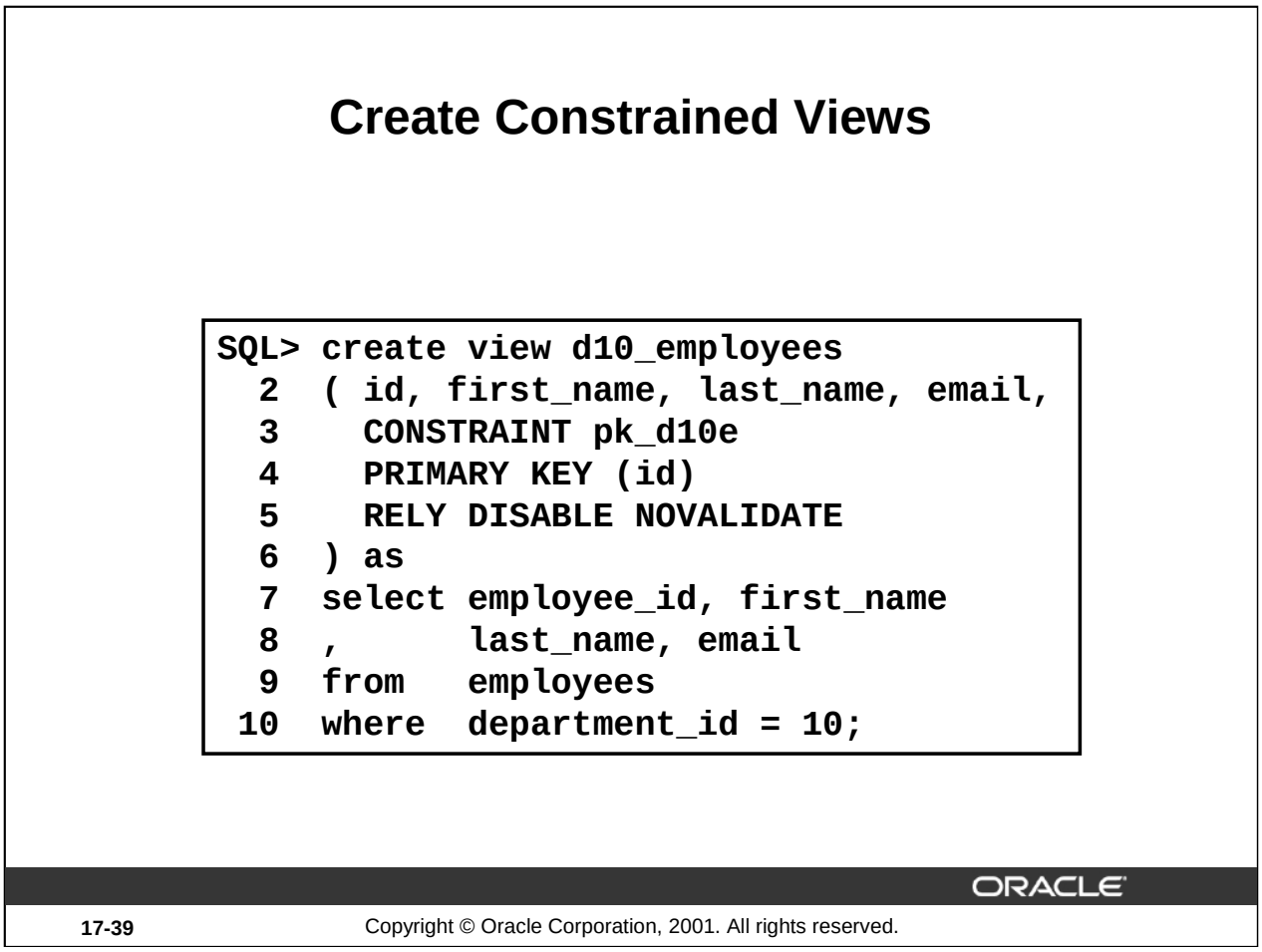
Check constraints are not supported. Note that views have a `WITH CHECK OPTION` clause that has the same semantics of check constraints.

`NOT NULL` constraints on views are not supported explicitly, as they are propagated implicitly from the base table constraints.

Given that view constraints are declarative, `DISABLE NOVALIDATE` is the only valid state for a view constraint. You can also choose between a `RELY` or `NORELY` state, which affects query rewrites. A view constraint in `RELY` state allows query rewrites to occur when the integrity level is set to `TRUSTED` mode.

Because view constraints are purely declarative, there are some more restrictions:

- No deferrability status allowed
- No physical attributes allowed, such as the `USING INDEX` clause
- No `ON DELETE` actions for foreign key constraints
- No `EXCEPTIONS INTO` clause



View Constraints Example

As you can see in the above example, the syntax for defining view constraints using the CREATE VIEW statement is quite similar to defining constraints on tables using the CREATE TABLE...AS SELECT (CTAS) statement. However, unlike the CTAS command, which does not allow referential integrity constraints, the CREATE VIEW statement supports such constraint definitions.

As for tables, you can also define a view constraint at the column level (in-line) or at the view level (as shown in the above example.)

Note: For each view constraint you must specify [RELY|NORELY] DISABLE NOVALIDATE. Otherwise, you get an error message.

Constrained View Maintenance

```
SQL> alter view d10_employees
  2  DROP CONSTRAINT pk_d10e;
```

```
SQL> alter view d10_employees
  2  ADD CONSTRAINT u_d10e
  3  unique(email) RELY DISABLE NOVALIDATE;
```

```
SQL> alter view d10_employees
  2  MODIFY CONSTRAINT u_d10e NORELY;
```

```
SQL> drop view d10_employees
  2  CASCADE CONSTRAINTS;
```

ORACLE

Constrained View Maintenance

You cannot drop a unique or primary key constraint if it is part of a referential integrity constraint on views, even if the referential integrity constraint is in NORELY DISABLE NOVALIDATE mode.

You cannot change the state of a unique or primary key constraint on a view from RELY to NORELY if it is part of a RELY referential integrity constraint, without also dropping the foreign key or changing its state from RELY to NORELY.

The ALTER VIEW clause checks if the view is valid before processing a constraint definition.

Because a view constraint can be referenced by other table or view constraints, Oracle9*i* provides the CASCADE CONSTRAINTS clause for the DROP VIEW command.

The CASCADE CONSTRAINTS option drops all referential integrity constraints that refer to primary and unique keys in the dropped view. If you omit this clause, and such referential integrity constraints exist, you get an error message.

Index Scans and Function-Based Indexes

- **B*-tree indexes do not contain NULL entries; therefore, index scans are sometimes impossible.**
- **All built-in operators know whether their result is guaranteed to be NOT NULL when their inputs are NOT NULL.**
- **This knowledge allows CBO to use index-only scans on function-based indexes.**

ORACLE

Index Scans and Function-Based Indexes

In prior releases, index-only scans could be done only when an indexed column was known to be NOT NULL; expressions could not be marked NOT NULL. This meant that function-based indexes could not be used for index-only scans unless the query somehow disallowed NULL values in the indexed expressions.

The deduction of NULL values in expressions based on the underlying columns in Oracle9*i* allows queries to do index-only scans on function-based indexes. All built-in operators know whether their result is guaranteed to be NOT NULL when all their inputs are NOT NULL.

Example

```
SQL> create index sal_comm_idx
  2  on employees (salary * (1 + commission_pct));
```

Consider the follwing query:

```
SQL> select salary * (1 + commission_pct)
  2  from   employees;
```

This query can do an index-only scan on the function-based index when both salary and commission_pct columns are defined as NOT NULL.

SELECT . . . FOR UPDATE WAIT

- **You can specify how long to wait if the rows being selected are locked.**
- **If NOWAIT is specified the behavior is still the same.**

```
SELECT ... FROM ... WHERE ...
[FOR UPDATE [NOWAIT|WAIT n]]
```

ORACLE

SELECT . . . FOR UPDATE WAIT

In prior releases the SELECT … FOR UPDATE statement has only two alternatives when the rows being selected are already locked: wait for the lock to be released or return immediately with an error message. Another alternative has been added in Oracle9*i* so that you can specify the time interval to wait before returning with the error.

An integer can be specified after the WAIT keyword to indicate the number of seconds to wait for a lock.

This feature prevents the indefinite waits on locked rows and allows more control on the wait time for locks in applications.

The typical workaround probably used by many programmers so far has been to execute a SELECT ... FOR UPDATE NOWAIT, upon failure followed by a short sleep (for example, three seconds) followed by a second attempt to lock the rows. This can now be achieved in a single statement.

Multitable INSERT Statement

- Allows you to insert rows into multiple tables as part of a single DML statement
- Two types: conditional and unconditional
- Can be used to transfer data from one or more source tables to a set of target tables
- Can be used to refresh materialized views
- Performance improvement: no materialization and repeated scan costs on the source tables

ORACLE

Multitable INSERT Statement

The multitable INSERT statement inserts rows in one or more tables, derived from the evaluation of a subquery.

There are two forms of the multitable INSERT statement. For the *unconditional* form, an INTO clause list is executed once for each row returned by the subquery. For the *conditional* form, the INTO clause entries are preceded by WHEN clauses that determine whether the corresponding INTO clause should be executed or not.

Note: The multitable INSERT statement can be executed in parallel and used with the direct load mechanism.

Multitable INSERT Syntax

```
INSERT ALL INTO t1 [ VALUES(...)]
           INTO t2 [ VALUES(...)]
           ...
SELECT ...
```

```
INSERT {ALL|FIRST}
        WHEN c1 THEN INTO t1 [ VALUES(...)]
        WHEN c2 THEN INTO t2 [ VALUES(...)]
        ...
       [        ELSE INTO t3 [ VALUES(...)]]
SELECT ...
```

ORACLE

Multitable INSERT Syntax

The INTO target can be any table expression that is legal for a regular INSERT INTO ... SELECT statement; however, aliases cannot be used. The same table can be specified as the target for multiple INTO clauses.

An INTO clause also provides the value of the row to be inserted using a VALUES clause. You can use any valid expression in the VALUES clause, but you can only refer to columns returned by the select list of the subquery. If the VALUES clause is omitted, the select list of the subquery provides the values to be inserted. If a column list is given, each column in the list is assigned a corresponding value from the VALUES clause or the subquery. If no column list is given, the computed row must provide values for all columns in the target table.

The first INSERT ALL command (without a WHEN predicate) is an unconditional multitable INSERT statement.

When you use the INSERT ALL syntax with multiple WHEN clauses, it is possible that rows are inserted into *multiple* tables for each row returning from the SELECT clause and *none* for others, depending on the outcome of the WHEN predicates.

When you use the INSERT FIRST syntax, only the first matching INTO clause is executed. This means that although you can insert into different tables, each row resulting from the SELECT clause can only lead to an insert into *one* of them. Note also that the INSERT FIRST syntax supports an optional ELSE clause.

LONG to LOB Migration

- ALTER TABLE has been enhanced:
  - Modify LONG column to CLOB
  - Modify LONG RAW column to BLOB
- During conversion the space required for both the LONG and LOB data must be available
- Most SQL functions and operators that accept VARCHAR2 accept CLOB as well
- SQL functions that accept RAW accept BLOB too
- LOB columns in partitioned index-organized tables and in function-based indexes

ORACLE

LONG to LOB Migration

Oracle8*i* introduced the TO_LOB( ) operator to copy LONG to LOB. You can use this operator in CREATE TABLE ... AS SELECT and INSERT INTO ... SELECT statements to copy existing data from a LONG into a LOB column.

Oracle9*i* LONG to LOB Migration

LOB migration is made even simpler: You can use the ALTER TABLE command to change a LONG to a LOB. Default values can be specified for the LOB column, and a LOB storage clause can be specified for storing the LOB segment. The NOT NULL constraints of the old LONG column are maintained for the new LOB column. Other ALTER TABLE commands are not allowed within the same command.

Note: With the ALTER TABLE … MODIFY command you can only modify a LONG or LONG RAW column to a LOB column. It will not modify a VARCHAR or a RAW column.

The ALTER TABLE ... MODIFY command temporarily doubles the space requirements. During the migration, the redo changes for the table are logged only if the table has logging on. The redo changes for the column being converted from LONG to LOB are logged only if the storage characteristics of the LOB indicate LOGGING.

You can now create LOB columns in partitioned index-organized tables, and function-based indexes can be created on LOB columns too.

PL/SQL Support for LOB Migration

- All standard package functions that accepted LONG now accept CLOB as well.
- All predefined functions accepting LONG RAW now accept BLOB as well.
- Implicit conversion for LOB to VARCHAR and RAW and vice versa
- You can define and bind LOB columns as VARCHAR and RAW.

ORACLE

Restrictions on LOB Migration

- **LOB columns are not allowed in clustered tables.**
- **Materialized views on the table being migrated must be rebuilt manually.**
- **No LOB support in UPDATE OF trigger clauses**
- **If a view has an INSTEAD OF trigger you cannot specify strings for INSERT/UPDATE of LOB columns.**
- **Indexes must be rebuilt manually; domain indexes on LONG columns must be dropped before migration.**

ORACLE

Restrictions on LOB Migration

LOB columns are not allowed in clustered tables, whereas LONG columns are allowed. So if a table is a part of a cluster, a LONG or LONG RAW column cannot be changed to LOB.

If a table is replicated or has materialized views, and you migrate a column from LONG to LOB, then all the replicas must be manually migrated too.

LOB columns are not allowed in the UPDATE OF list in the update triggers. Hence triggers that were intended to update LONG columns become invalid after the column is migrated to a LOB and are not recompiled. Also, if a view with a LOB column has an INSTEAD OF trigger which attempts to INSERT/UPDATE into the LOB, the trigger fails because implicit conversion from LOB is not allowed in INSTEAD OF triggers.

All indexes (including function-based and domain indexes) must be manually rebuilt. The ALTER TABLE ... MODIFY command acts like a MOVE command, and therefore does not convert the indexes. Domain indexes for the LOB column must be dropped before altering a LONG column to LOB.

Domain indexes on the LONG columns must be dropped before migration.

Common SQL Parser

- **Reduces duplication of SQL analysis**
- **Allows PL/SQL to pick up new SQL features as they are implemented in the RDBMS**
- **Some consequences:**
  - **Wrap files expose static SQL statement text**
  - **PLS error messages become ORA error messages**

ORACLE

Common SQL Parser

The common SQL parser replaces PL/SQL's compile-time analysis of a static SQL statements with analysis using a SQL component shared with the RDBMS. This reduces duplication of SQL analysis, allowing PL/SQL to pick up new SQL features as they are implemented in the RDBMS and eliminating bugs due to differences in SQL analysis between SQL and PL/SQL.

PL/SQL Wrap Files

Prior to Oracle9*i* wrap files exposed column and variable names and string literals occurring in PL/SQL program units, but not entire static SQL statements. In Oracle9*i* wrap files expose the text of static SQL statements.

Error Messages

Several error messages raised during static SQL statement analysis change from PLS errors to ORA errors. The common SQL parser raises errors during SQL compilation that PL/SQL analysis did not catch in Oracle8*i*. This allows earlier detection of static SQL statement errors.

Native PL/SQL Execution

- **Faster execution of PL/SQL programs by generating native C code instead of byte code**
- **Performance improvements:**
  - **Eliminate byte code interpretation overhead**
  - **Faster control flow in native code**
  - **Compiled code corresponding to a PL/SQL program is mapped to a PGA as opposed to SGA**
  - **PL/SQL without SQL references is 2 to 10 times faster**

ORACLE

Native PL/SQL Execution

A PL/SQL program is compiled to native code in two phases: the program is translated to C code which is subsequently compiled to native code. For PL/SQL fixed packages, the compiled code is statically linked to the Oracle executable whereas for end-user PL/SQL programs, the compiled code is dynamically linked to an Oracle process.

Control flow is much faster in native code than in interpreted code, because jumps are label based in C code. Function calls to targets in the same compilation unit are mapped to C function calls. The cost of setting up a frame is also cheaper because the primary memory for the frame is allocated from the C stack as opposed to the PGA in the case of MCODE. Exception handling is also much faster in native code because exceptions are implemented as jumps to switch statements containing the exception handler code. There is no run-time overhead associated with looking up an exception handler table as in the case of MCODE based exception handling.

The compiled code corresponding to a PL/SQL program is mapped into the PGA as opposed to the SGA into which the MCODE is loaded. This should result in less contention for SGA and thus better scalability. Also in the case of large PL/SQL programs, the startup time should be faster on demand paged systems. The MCODE is loaded in its entirety into SGA.

The constant pool is also mapped into the PGA as constant data. The constant pool items (such as string literals) can be arbitrarily long as the PGA is paged by the operating system. Similarly, the handle segment need not be paged in the case of native compilation. PL/SQL execution is 2 to 10 times faster; however, it does not speed up SQL execution.

Summary

In this lesson, you should have learned how to:

- **Use ISO/ANSI standard SQL syntax for joins, `CASE` expressions, `NULLIF`, `COALESCE`, scalar subqueries, `MERGE`, analytical functions**
- **Identify other SQL enhancements, such as:**
  - **Constraint enhancements**
  - **`FOR UPDATE WAIT`**
  - **Constraints on views**
  - **LOB enhancements**
  - **Common SQL parser and native PL/SQL execution**

ORACLE

Practice 17-1 Overview

This practice covers the following topics:

- **ANSI/ISO SQL:1999 standard joins**
- **The `WIDTH_BUCKET` function**
- **`CASE` expressions**
- **Scalar subqueries**

ORACLE

Practice 17-1 Overview

This practice concentrates on using the ANSI/ISO SQL:1999 standard enhancements in Oracle9*i*.

Practice 17-2 Overview

This practice covers the following topics:

- **Explicit constraint index control**
- **Less foreign key locking overhead**
- **`SELECT ... FOR UPDATE [WAIT n]`**

ORACLE

Practice 17-2 Overview

This practice covers the other Oracle9*i* enhancements, that are not part of the ANSI/ISO SQL:1999 standard.

18

Globalization Support

Formerly National Language Support

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- **Describe and use the new date and time data types**
- **Describe the Unicode enhancements**
- **List the enhanced sorting functionality**
- **Use the Character Set Scanner**
- **Explain the new byte and character semantics**
- **Use the Locale Builder**

ORACLE

Objectives

Describe and use the new date and time data types:
Time zones, their use for globalization, Daylight Saving Time, and the corresponding new SQL functions.

Understand the Unicode enhancements:
The new Unicode character sets, National Character Set limitations, Unicode character implementations, and SQL*Loader enhancements.

Use the Character Set Scanner:
Identify character conversion problems, Database summary report, and Individual Exception Reports.

Use the Locale Builder:
User defined Locales, and manipulating existing NLS locale data.

Globalization and NLS

- **Globalization support is the new name for National Language Support (NLS)**
- **NLS still used in the names of views and variables**

ORACLE

Globalization Support

The term *Globalization Support* is now used in preference to *National Language Support* to reflect the much broader application of the features in the Oracle RDBMS. NLS originally only covered language setting.

Globalization Support is not a product; it is a database feature. It is a suite of features that you use to develop multilingual applications that can be accessed and run anywhere in the world simultaneously, without modification, rendering content in the native user's language and locale preferences.

The abbreviation NLS is still used as part of the name of the parameters and functions.

New Time and Interval Data Types

There are five new data types:

- **TIMESTAMP**
- **TIMESTAMP WITH TIME ZONE**
- **TIMESTAMP WITH LOCAL TIME ZONE**
- **INTERVAL YEAR TO MONTH**
- **INTERVAL DAY TO SECOND**

ORACLE

New Date and Time Data Type

The new time and date data types store both date and time, like the DATE data type, but with fractional seconds, up to 9 digits precision, and time zone information.

TIMESTAMP stores a date and time. You optionally specify the precision of fractional seconds, the default is 6 digits, maximum 9. For example:

```
CREATE TABLE … (occurred_at TIMESTAMP(4))
```

TIMESTAMP WITH TIME ZONE includes the time zone the time component refers to. This is stored in extra bytes. You might think of this as an "absolute" time. You specify precision as with TIMESTAMP, for example:

```
CREATE TABLE … (entry_time TIMESTAMP(2) WITH TIME ZONE)
```

TIMESTAMP WITH LOCAL TIME ZONE normalizes the time to the database time zone. You thus cannot see which time zone the date was entered with, but it will be recalculated and displayed in the correct time zone of the client retrieving the data. You can think of this as a "relative" time. An example:

```
CREATE TABLE … (mail_sent TIMESTAMP WITH LOCAL TIME ZONE)
```

New Time and Interval Data Type Example

```
SQL> CREATE TABLE happenings
  2  (occurred_at TIMESTAMP(4)
  3             -- Absolute, non-globalized timestamp
  4  ,entry_time  TIMESTAMP(2) WITH TIME ZONE
  5             -- Absolute, globalized timestamp
  6  ,mail_sent   TIMESTAMP WITH LOCAL TIME ZONE
  7             -- Relative, globalised
  8  ,guarantee   INTERVAL YEAR(1) TO MONTH
  9             -- Maximum is 9 years 11 months
 10  ,workhours   INTERVAL DAY(0) TO SECOND(0)
 11             -- Maximum is 23:59:59s
 12  );
```

ORACLE

New Interval Data Type

The interval time types are used to store a length of time, as opposed to a point in time.

`INTERVAL YEAR TO MONTH` allows intervals to be specified in (years, months). You specify the number of digits to give to the year component, for example:

```
SQL> CREATE TABLE … (guarantee INTERVAL YEAR(1) TO MONTH);
```

Default precision is 2, and the maximum is 9. Specifying 0 disables the year component.

`INTERVAL DAY TO SECOND` allows intervals to be specified in (days, hours, minutes, seconds). You can specify the number of digits to allow in the day (default 2, maximum 9, and zero disables the field) and precision of fractional seconds, for example:

```
SQL> CREATE TABLE … (workhours INTERVAL DAY(0) TO SECOND(0),
  2                  holidays  INTERVAL DAY(2) TO SECOND(0));
```

TIMESTAMP Literals

```
SQL> INSERT INTO happenings
  2  (occurred_at,entry_time,mail_sent) VALUES
  3  ('21-MAY-01 12:23:45.87'     -- Fractional secs
  4  ,'22-MAY-01 08:50:12 AM GMT' -- Named timezone
  5  ,'23-MAY-01 05:00:00 PM'     -- Client timezone
  6  );
```

```
SQL> ALTER SESSION SET
  2  nls_timestamp_tz_format = 'DDMMYYHH24MISSFF TZH';
SQL> INSERT INTO happenings(entry_time)
  2  VALUES ('24050110061578 +3');
SQL> INSERT INTO happenings(entry_time)
  2  VALUES (TIMESTAMP '2001-05-25 18.30.26.12345'
  3          AT TIME ZONE 'Europe/London');
```

ORACLE

TIMESTAMP Literals

The format of entering timestamps is controlled by the corresponding NLS timestamp format mask, with optional fractional seconds and an optional time zone field. The latter can be entered as a named time zone, or as an offset, for example: `'-3:00'`.

Alternatively you can use the following format:

`TIMESTAMP 'YYYY-MM-DD HH24:MI:SS.FF' AT TIME ZONE <tz spec>`

That is, the keyword `TIMESTAMP` is followed by a string, where the format is fixed as shown above. The `AT TIME ZONE` is optional.

When specifying the time zone, there is a difference between entering it with an offset or entering it with a region name. This is stored as part of the value. This is reflected when selecting from `TIMESTAMP WITH TIME ZONE`. If it was defined with an offset, then it does not show as a region, even if the format calls for one.

Entering a `TIMESTAMP WITH LOCAL TIME ZONE` is identical to entering a `TIMESTAMP`.

INTERVAL Literals

```
SQL> UPDATE happenings SET
  2    guarantee = INTERVAL '1-6'   YEAR TO MONTH
  3  , workhours = INTERVAL '08:00' HOUR TO MINUTE
  4  , mail_sent = mail_sent +
  5                INTERVAL '1000'  MINUTE;
```

ORACLE

INTERVAL Literals

The interval has the INTERVAL keyword followed by the literal text string that specifies the amount and ends with the interval type keywords.

`INTERVAL '<year>-<month>' YEAR TO MONTH`

`INTERVAL '<day> <hr>:<min>:<sec>.<fraction>' DAY TO SECOND`

Note the hyphen in the YEAR TO MONTH and the space between DAY and HOUR. The field value can be too large, if it can be converted into a higher field.

You can abbreviate the literal and omit corresponding fields as shown in the examples.

More Examples

`INTERVAL '1 2:3:4.5' DAY TO SECOND`
One day, two hours, three minutes, four and a half seconds.

`INTERVAL '1' YEAR + INTERVAL '1' DAY`
One year and one day.

Formatting NLS Variables

- **NLS_TIMESTAMP_FORMAT**
  Default: 'DD-MON-RR HH.MI.SSXFF AM'
- **NLS_TIMESTAMP_TZ_FORMAT**
  Default: 'DD-MON-RR HH.MI.SSXFF AM TZR'

ORACLE

NLS Formatting Variables

The new data types have their own formatting variables. The default value is shown. As with other formatting masks these apply to input and output, whether entered by the user or in PL/SQL code.

The time zone field values are formatted either numerically with TZH or TZM for hour and minute respectively, or TZR for a region name. Specifying TZR in the formatting mask still allows numeric entry. When formatting for display with TZR it show only a region, if a region was entered. If specified with TZD, the string indicates if daylight saving (summer time) is in effect.

The format mask for timestamp includes FF for fractional second. You cannot specify the width of the field, only its presence or absence. Specifying FFFF does not specify four digits, but the fractional seconds field is repeated twice on output. The X specifies the radix separator.

Using Time Zones

- **The *database* operates in a time zone:**
  - **Defined at `CREATE DATABASE` time**
  - **Can be altered with `ALTER DATABASE`**
  - **Current value given by `DBTIMEZONE`**
- **The *session* operates in a time zone:**
  - **Defined with `ORA_SDTZ` environment variable**
  - **Can be altered with `ALTER SESSION SET TIME_ZONE`**
  - **Current value given by `SESSIONTIMEZONE`**
- **`TIMESTAMP WITH LOCAL TIME ZONE` adjusts for time zones**

ORACLE

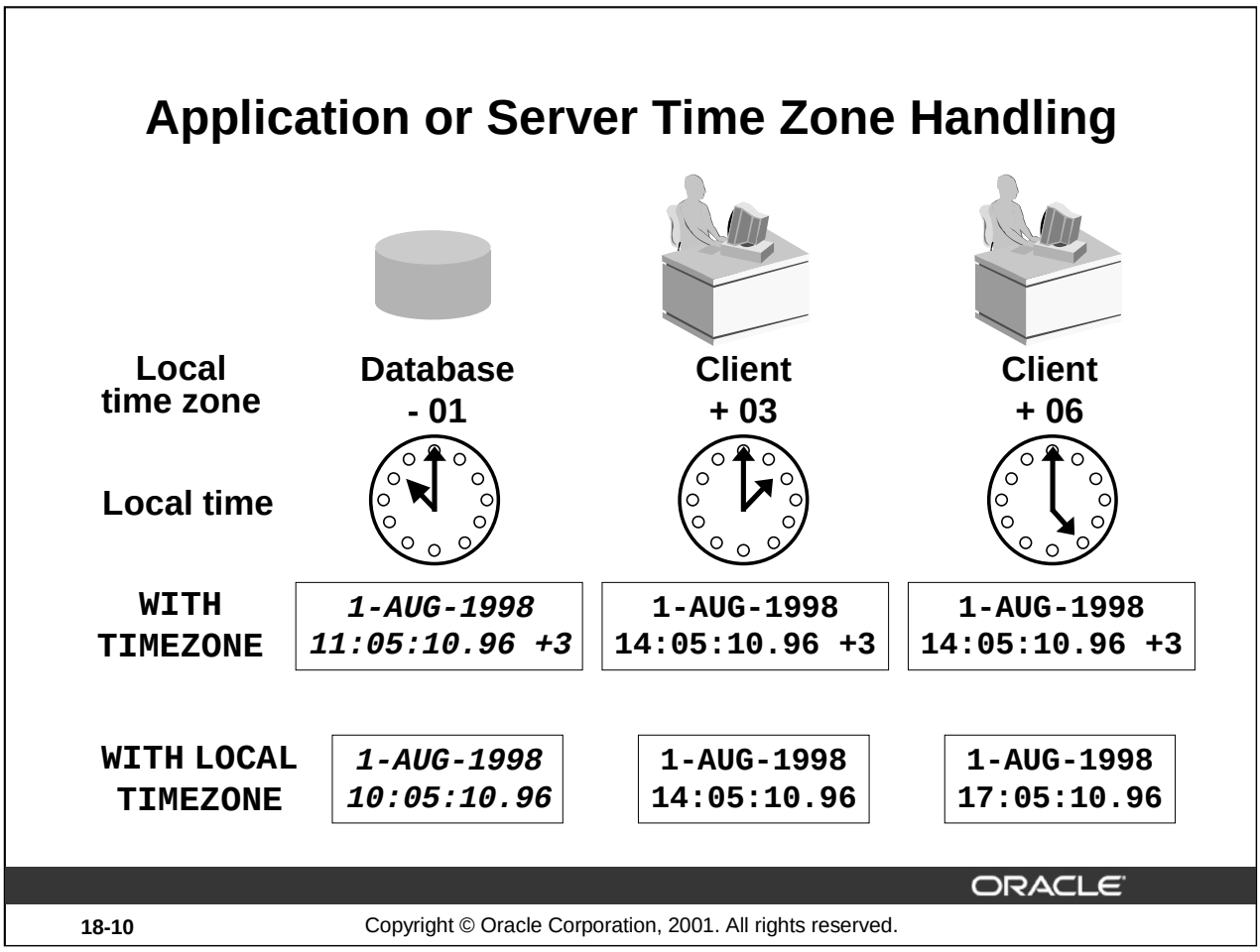
Using the Time Zones

The time zone information is for applications that operate in several time zones simultaneously; for example, mail programs.

The client session takes its local time zone from the `'ORA_SDTZ'` environment variable which can have values `'OS_TZ'` (use operating system time zone), `'DB_TZ'` (use time zone of database, effectively negates the function of `LOCAL TIME ZONE`), or a specified offset or named time zone. The local time zone of the session can be altered with

```
ALTER SESSION SET TIME_ZONE=
{<tz name>|<offset value>|DBTIMEZONE|SESSIONTIMEZONE}
```

The time zone of the client is included in the time value that is stored in a `TIMESTAMP WITH TIME ZONE` column if not specified. In other words, if you do not specify a time zone in your date time literal, an implicit conversion is made from `DATE` or `TIMESTAMP` to `TIMESTAMP WITH TIME ZONE`, and thus adding the client time zone.



Application Based Time Zone Handling

The TIMESTAMP WITH TIME ZONE data type stores the time zone offset with the time. The application code can examine this field and perform application specific conversion or adjustment.

This is shown by the upper row. The time was entered by client 1. Client 2's application may adjust the time value with the data and session time zone for display or sorting purposes.

You can determine the time zone offset of column data by using either of the following:

```
SQL> SELECT TO_CHAR(tsz,'TZH') HR, TO_CHAR(tsz,'TZM') MI
  2  FROM   happenings;

SQL> SELECT EXTRACT(TIMEZONE_REGION FROM tsz) FROM happenings;
```

Server Based Time Zone Handling

The TIMESTAMP WITH LOCAL TIME ZONE data type stores times in the database with the same time zone as the database is defined with. The times are adjusted, when displayed at a client, to be in the client's local time zone. Clients in different time zones therefore see different results when selecting from this data type. Inside the database the time is stored in the database time zone.

The second row shows the same time entered by client 1. Note the return value for client 2.

No Time Zone

The TIMESTAMP data type stores the time value and does not make any adjustments.

Datetime/Interval Arithmetic

All the sensible manipulations are supported

- **You can add or subtract an interval from a timestamp or date**
- **You can add or subtract intervals from each other**
- **You can multiply or scale an interval**
- **You can subtract dates to yield an interval**

ORACLE

Date/Time Arithmetic

You can perform arithmetic on date and time variables and interval types. Any sensible combination works. If you use a number in a date time expression, this is converted to a DATE type and coerce the expression result into a DATE type (possibly loosing precision). To avoid this, explicitly specify an INTERVAL type. Consider the following:

```
SQL> SELECT entry_time, entry_time + 4,
  2         entry_time + INTERVAL '4' DAY
  3  FROM   happenings;
```

Returns for each expression, respectively:

The original value:	22-MAY-01 08:50:12.87000 PM PST
4 days later, but result in DATE:	22-MAY-01 08:50:12
As above, but in TIMESTAMP:	22-MAY-01 08:50:12.87000000 PM PST

You loose both fractional precision and time zone information in the second case. Possible errors if you attempt impossible calculations include:

```
ORA-30081: invalid data type for datetime/interval arithmetic
```

which may be an indication of an unexpected conversion.

```
ORA-30087: Adding two datetime values is not allowed
```

when trying to add two datetime values. Subtraction is supported.

Note: Oracle8*i* already supported adding or subtracting an interval literal from a date.

Datetime Functions

To get current session time and database time for all datetime data types

CURRENT_DATE	Session date and time (DATE)
CURRENT_TIMESTAMP	Session date and time (TIMESTAMP WITH TIME ZONE)
LOCALTIMESTAMP	Session date and time (TIMESTAMP)
SYSTIMESTAMP	Server date and time (TIMESTAMP WITH TIME ZONE)
DBTIMEZONE	Server time zone (VARCHAR2)
SESSIONTIMEZONE	Session time zone (VARCHAR)
SYSDATE	Server date and time (DATE)

ORACLE

Get Information Functions

CURRENT_DATE: Returns the current date and time in the session time zone, in a value in the Gregorian calendar of data type DATE, which means without fractional seconds.

LOCALTIMESTAMP: Returns the same value, but in TIMESTAMP data type, that is, with fractional seconds.

CURRENT_TIMESTAMP: Returns the same value as LOCALTIMESTAMP but in the TIMESTAMP WITH TIME ZONE data type, and thus includes the session time zone.

SYSTIMESTAMP: Returns date and time of the server in TIMESTAMP data type.

By contrast, the SYSDATE function is the server system date and time, without time zone adjustment.

DBTIMEZONE: Returns the value of the database time zone. The return type is a time zone offset or a time zone region name, depending on how you defined the database time zone.

SESSIONTIMEZONE: Returns the value of the current session's time zone. The return type is a time zone offset or a time zone region name, depending on how you specified the session time zone value in the most recent ALTER SESSION statement.

Examples

```
SQL> SELECT DBTIMEZONE, SESSIONTIMEZONE   FROM DUAL;
SQL> SELECT LOCALTIMESTAMP - SYSTIMESTAMP FROM DUAL;
```

Datetime Conversion Functions

Function:	From:	To:
TO_DSINTERVAL	String	INTERVAL DAY TO SECOND
TO_YMINTERVAL	String	INTERVAL YEAR TO MONTH
TO_TIMESTAMP	String	TIMESTAMP
TO_TIMESTAMP_TZ	String	TIMESTAMP WITH TIME ZONE
FROM_TZ	TIMESTAMP	TIMESTAMP WITH TIME ZONE
TO_CHAR, TO_NCHAR	All date and time types to character; extended for new formats	

ORACLE

Conversion functions

TO_DSINTERVAL: Converts a string to an INTERVAL DAY TO SECOND data type.

TO_YMINTERVAL: Converts a string to an INTERVAL YEAR TO MONTH data type.

TO_TIMESTAMP: Converts a string to a TIMESTAMP data type.

TO_TIMESTAMP_TZ: Converts a string to a TIMESTAMP WITH TIME ZONE data type.

FROM_TZ: Converts a TIMESTAMP value to a TIMESTAMP WITH TIME ZONE value with a specified time zone.

TZ_OFFSET: Returns the time zone offset of the value entered, and adjusts the return value on the date the statement is executed (that is, Daylight Saving Time)

The TO_CHAR function is extended to support the new datetime and interval types and converts these into a character string with the default or specified format mask. There is also a TO_NCHAR function where the result is in the national character set.

Examples

```
SQL> SELECT hire_date,
  2         hire_date + TO_YMINTERVAL('01-02') "14 months"
  3  FROM   employees;

SQL> SELECT TO_TIMESTAMP_TZ('1999-12-01 11:00:00 -8:00',
  2                         'YYYY-MM-DD HH:MI:SS TZH:TZM')
  3  FROM   dual;
```

Datetime Extract Function

Gets appropriate field from datetime or interval data type, and returns a NUMBER value

```
SQL> SELECT EXTRACT(YEAR FROM '07-MAR-1998')
  2  FROM   dual;

EXTRACT
-------
   1998
```

ORACLE

Extract functions

EXTRACT returns the value as a NUMBER of a specified datetime field from a datetime or interval value expression.

You can extract YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR, TIMEZONE_MINUTE, TIMEZONE_REGION, and TIMEZONE_ABBR (abbreviation). The value must contain the field; for example, you cannot extract SECOND from a INTERVAL YEAR TO MONTH.

When you extract a TIMEZONE_REGION or TIMEZONE_ABBR, the value returned is a string containing the appropriate time zone name or abbreviation. When you extract any of the other values, the value returned is in the Gregorian calendar format. When extracting from a datetime with a time zone value, the value returned is in Coordinated Universal Time, formerly Greenwich Mean Time (UTC).

```
SQL> SELECT EXTRACT(YEAR FROM DATE '1998-03-07')
  2  FROM   DUAL;
```

Daylight Saving Time

- **Daylight Saving Time (DST) in named regions**
  - **List of named regions in `V$TIMEZONE_NAMES`**
  - **Not all regions have DST**
- **Arithmetic on `TIMESTAMP WITH TIME ZONE` adjusts for DST:**
  - **The skipped and inserted hour is skipped or counted twice, respectively.**
  - **The skipped hour cannot be stored.**
  - **The inserted hour has to be specified in timezone if it is the first or second occurrence.**

ORACLE

Daylight Saving in Regions

Some localities have defined Daylight Saving Time (DST) (also known as Summer Time) for a period of the year, where the clocks are advanced by one hour. The dates and times this occurs is different from place to place, and not consistent in a time zone. Therefore there are many named regions for a given time zone. `'+01:00'`, `'CET'` and `'Poland'` all refer to the same time zone, but treatment of DST varies.

When adding or subtracting `INTERVAL` from `TIMESTAMP WITH TIME ZONE` values, the Daylight Saving Hour in spring and autumn, respectively, is compensated for. This only works where the time zone was entered as a named region, not a hour and minute offset.

The format mask `'TZD'` in `NLS_TIMESTAMP_TZ_FORMAT` will show if the time is in DST or not.

Result for Special Cases

Inserting invalid values (the skipped hour in springtime) will insert it as a standard time, that is, ignoring daylight saving rules.

If the `ERROR_ON_OVERLAP_TIME` session variable is set to TRUE, however, an error is raised. You set it to TRUE or FALSE with `ALTER SESSION`.

Unicode

- **Contains all major living scripts and supports legacy data**
- **Develop, deploy, and host multiple languages in a single instance**
- **Enable worldwide interchange of data**
- **Conforms to International Standards**

ORACLE

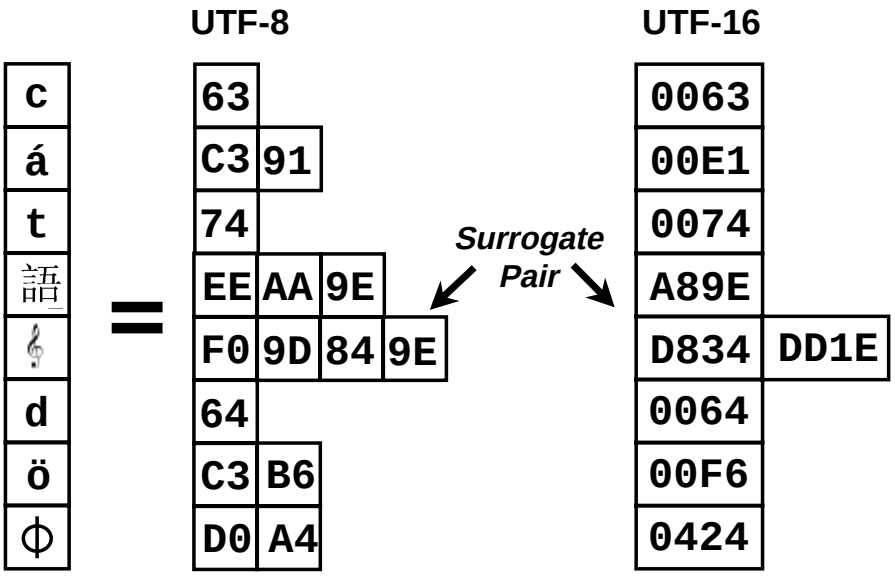
What Is Unicode?

Unicode is a universal character encoding scheme that you use to store information from any major language using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language.

Oracle started supporting Unicode as a database character set in Oracle7. In Oracle9*i*, Unicode support has been expanded so that you can find the right solution for your globalization needs. Oracle9*i* supports Unicode 3.0, the third version of the Unicode Standard. For more information about the Unicode Standard Version 3.0, see *The Unicode Standard Version 3.0*, published by Addison-Wesley, or on `http://www.unicode.com`

Unicode specifies both which characters are encoded, and several different methods to represent these in binary. The list of defined characters is independent of the method to encode the binary numbers, which is only of interest when exchanging data. This is why there are several Unicode character sets in Oracle9*i*, one for each binary encoding format, although the list of which characters are represented is the same. There is the varying width byte method, and the fixed-length two bytes and four byte methods. Lastly there is a method to extend the first two methods with surrogate pairs.

Unicode Encodings



ORACLE

UTF-16 Encoding

This is the 16-bit encoding of Unicode. It is a fixed-width multibyte encoding in which the character codes 0x00 through 0x7F have the same meaning as ASCII. One Unicode character is 2-bytes in this encoding. Characters from both European and Asian scripts are represented in 2 bytes.

UTF-8 Encoding

This is the 8-bit encoding of Unicode. It is a variable-width multibyte encoding in which the character codes 0x00 through 0x7F have the same meaning as ASCII. One Unicode character can be 1-byte, 2-bytes, or 3-bytes in this encoding. Characters from the European scripts are represented in either 1 or 2 bytes, while characters from most Asian scripts are represented in 3 bytes.

Surrogate Pairs

You can extend Unicode to encode more than 1 million characters. These extended characters are called surrogate pairs. Surrogate pairs are designed to allow representation of characters in future extensions of the Unicode Standard. Surrogate pairs require 4 bytes in UTF-8 and UTF-16. There are no such currently assigned characters in the current version of the standard, but it is widely expected that such characters will be introduced in the near future. One current proposal would assign Western Musical Symbols, like the treble clef shown above, in the surrogate area.

UCS-2 is a subset of UTF-16. UTF-16 differs from UCS-2 in that it can access additional encodings with the use of surrogate pairs.

Unicode Character Forms

- **Unicode also defines some character forms, irrespective of the encoding used**
  - **Composed**
  - **Decomposed**
- **The SQL functions `COMPOSE` and `DECOMPOSE` perform this action.**

ORACLE

Composed Versus Decomposed

Characters which have diacritical marks, such as 'ö', can either be represented as two characters, the base character followed by the mark (or this example as an 'o' and '¨') or as a single character. The visual output and the intended lexical meaning is the same character, but they store differently. This is due to the legacy character sets all being representable in Unicode. This is true in all Unicode character sets. Unicode character sets only differ in which binary bytes are used; the repertoire of characters is the same.

Comparisons should use the decomposed format consistently. Either strings are always stored in decomposed format, or an extra conversion must be done.

Examples

```
SQL> SELECT COMPOSE('o'||UNISTR('\0308'))
  2  FROM   DUAL;

CO
--
Ö

SQL> SELECT DECOMPOSE('Châteaux')
  2  FROM   DUAL;

DECOMPOSE
---------
Cha^teaux
```

Enhanced Unicode Support

- More Unicode character sets
- Hexadecimal character specification possible with `UNISTR` function
- Character set choice for National Character Set is limited to Unicode character sets. This can affect migration of databases.
- Also supported in database character set

ORACLE

More Unicode Character Sets

The Oracle Unicode now supports the fixed-width character encoding. This enables easier loading of multinational data.

`UNISTR` Function

UNISTR converts a string to a Unicode string. In the string you can specify the hexadecimal character number (not the byte representation) of “difficult” characters.

```
SQL> SELECT UNISTR('M\00b2')
  2  FROM   DUAL;
```

National Character Set Choices

The National Character Set can only contain Unicode character sets. This is a change to previous versions. You can now declare both varying width and fixed width character sets, but they must be one of the Unicode character sets. The current implementation restricts the National Character Set to either UTF8 or AL16UTF16 only. This is covered more in the section on the Character Set Scanner.

Oracle9*i* Unicode Globalization Design

Future schema design should put all character strings, which may contain language specific information, in the data types NCHAR, NVARCHAR2 and NCLOB, where they can utilize the full character set of Unicode. The DATABASE CHARACTER SET needs only to be a character set that is compliant with the operating system, to refer to data dictionary objects, and filenames. For performance this should be a fixed width one byte character set.

The previous style of having the varying-width Unicode character set as the DATABASE CHARACTER SET is still supported, both for new and migrated databases.

Migration and Unicode Issues

- **Unicode conversion**
- **Conversion required during migration**

ORACLE

Conversion to Unicode for National Character Set

An Oracle8*i* database with a non-Unicode character set in National Character Set has to convert the National Character Set to a Unicode character set. The Character Set Scanner should be used prior to migration to identify any potential problems to identify any conversion problems. These are no different to the other character conversion problems. The migration utility has a facility to do the conversion of `NCHAR`, `NVARCHAR2` and `NCLOB` without use of export or import. Using export and import is also supported, particularly if the database is migrated using these facilities.

See the *Oracle9i Migration Manual* for details.

New Unicode Character Sets

- **Two new Unicode character sets:**
  - **AL16UTF16**
  - **AL32UTF8**
- **Existing character set:**
  - **UTF8**
  - **UTFE**
- **Desupported character set:**
  - **AL24UTFFSS**

ORACLE

Unicode Character Sets

AL16UTF16 is the fixed width two-byte character set of Unicode. Surrogate pairs are not supported, as they are not ratified in the Unicode standard at this time, but these will occupy 4 bytes. It can only be used in the National Character Set. This is also sometimes referred to as UTF16 or UCS-2. Even though this uses more storage for English text compared to the UTF8 based sets, processing is faster because it is fixed size.

AL32UTF8 is an enhanced version of UTF8. It strictly implements the UTF-8 standard. When defining new databases AL32UTF8 should be used instead of UTF8 if this is required as the Database Character Set.

UTF8 is still supported, in order to allow migration of databases from Oracle8*i* or earlier versions. UTFE is still supported, as it is an EBCDIC based variant of UTF8 for use on platforms where EBCDIC is the operating system character set.

AL24UTFFSS is desupported in Oracle9*i*. This is similar to UTF8 but contains fewer characters and is based on an old Unicode standard.

Space Tradeoffs

Your choice of different Unicode character sets affects the space usage. Storing non-English European strings occupy two bytes; Far East characters occupy three bytes in the varying width character set. In the fixed width every character occupies two bytes. So the proportion of Asian characters is influential. Using a non-Unicode 8 bit character set limits the repertoire of characters, but all characters occupy one byte.

Choosing a Unicode Solution Scenario Unicode Database

Scenario

- **Current character set: WE8ISO8859P1**
- **Will need to expand language requirements beyond the current capacity. The database contains existing PL/SQL code.**

Solution

- **Database Character Set: AL32UTF8**
- **National Character Set: Default**

ORACLE

Scenario

The current database is configured for character set WE8ISO8859P1. The database will need to expand language requirements beyond the current character set. Also the database contains PL/SQL code for existing applications.

Considerations

The need to support additional languages requires a change at the database or column level. Adding SQL NCHAR data type columns to tables would require significant code changes. Migrating to a Unicode database character set would require few changes to existing code.

Solution

Choose AL32UTF8 as the database character set.

Choosing a Unicode Solution Scenario Unicode Data Type

Scenario

- **Current character set: JA16EUC (Japanese)**
- **Need support for additional Asian and Western European Languages in the form of customer names and addresses.**

Solution

- **Database Character Set: JA16EUC**
- **National Character Set: AL16UTF16**

ORACLE

Scenario

The current database is configured for character set JA16EUC (Japanese). Access to additional Asian and Western European languages is needed.

Considerations

Only customer names and addresses require Unicode support. Using NCHAR columns meet the requirement without migrating the entire database.

JA16EUC is a variable width 16-bit character set. Moving text data to NCHAR datatypes in UTF-16 would save space by storing data in two bytes; also this would improve performance by using a fixed-width character set.

Solution

Choose AL16UTF16, the default for Unicode storage for the National Character Set. Leave the database character set at JA16EUC.

More Character Sets, Languages, and Territories

Character Set	Language	Territory
BLT8ISO8859P13	Assamese	Chile
CEL8ISO8859P14	Bangla	Colombia
CL8ISOIR111	Gujarati	Costa Rica
CL8KOI8U	Kannada	El Salvador
WE8EBCDIC924	Malayalam	Guatemala
WE8EBCDIC1047E	Marathi	Macedonia
ZHS32GB18030	Oriya	Nicaragua
	Punjabi	Panama
	Telugu	Peru
		Puerto Rico
		Venezuela
		Yugoslavia

ORACLE

Character Sets

BLT8ISO8859P13	ISO 8859-13 Baltic character set
CEL8ISO8859P14	ISO 8859-14 Celtic character set
CL8ISOIR111	SOIR111 Cyrillic character set
CL8KOI8U	KOI8 Ukrainian Cyrillic character set
WE8EBCDIC924	
WE8EBCDIC1047E	
ZHS32GB18030	

These new character sets are primarily intended to support more client locales, as a Unicode database can store all characters.

Languages

The language definition includes: Language of message (Note; a language definition does not necessarily imply that the message files have been translated), date names, and default sorting order.

Territories

The Territory definition includes: Default formatting of numbers and dates, currency symbols, calendar settings.

Extending Definitions

Defining new or modifying new character sets, languages and territories can be done easier in the Locale builder, covered later in this lesson.

Enhanced Linguistic Sorting

The sorting now uses four-level sorting:

- **Enables more complex sorting rules**
- **New sorting definitions added, which utilize three levels**
- **Result of sort of the old definitions unchanged**
- **User can refine sorting rules by defining the fourth level**

ORACLE

Enhanced Linguistic Sorting

The Oracle server sorts characters by converting each character into its sorting value (see NLSSORT function), and then sorting on that. Versions prior to Oracle9*i* generated two numbers per character. If the first number, the major sorting value, is the same for a pair of characters, then the second value, the minor, value was compared. Oracle9*i* has up to four sorting values per character, so that the lower levels are used if the first two levels are identical. Defining a sort is thus the list of characters and their sorting values. The characters are defined in Unicode code points, so that the sort can be applied to all character sets, there being an implicit conversion to Unicode.

There are also some additional entries that define character swapping, character expansion and contraction which modifies the basic sorting in some languages.

You specify which sort to use, as before, with `NLS_SORT`. Sort definitions from before Oracle9*i* have not changed, to maintain compatibility. There are new sort definitions that utilize the more complex sorting tables.

If you want to refine the sort even further, you can define your own sorts, using the Locale Builder.

New Linguistic Sorts

- **GENERIC_M**
- **SPANISH_M**
- **FRENCH_M**
- **CANADIAN_M**
- **DANISH_M**
- **BIG5**
- **HKSCS**
- **GBK**
- **THAI_M**
- **SCHINESE_STROKE_M**
- **SCHINESE_PINYIN_M**
- **TCHINESE_RADICAL_M**
- **TCHINESE_STROKE_M**
- **JAPANESE_M**
- **KOREAN_M**
- **SCHINESE_RADICAL_M**

ORACLE

Multilinguistic Sorting

It is not possible to make a generic multilingual sort, as letters sort differently in different languages. In a given single-language sort, characters that are not part of the language are often not sorted. For example, when sorting in Chinese, any Arabic strings are not sorted. The multilinguistic sort has a defined basic sorting order for all characters, which is used for the characters outside this language.

New Sorts

- GENERIC_M: Generic sorting order which is based on ISO14651 and Unicode canonical equivalence rules but excluding compatible equivalence rules

These other new sort definitions use the ISO standard 14651 as the basic sorting, but otherwise sort correctly for the language. In addition they implement more special language specific sorting rules.

- CANADIAN_M: Canadian French sort supports reverse secondary, special expanding characters
- CYRILLIC_M: Cyrillic sort
- DANISH_M: Danish sort supports sorting lower case characters before upper case characters
- FRENCH_M: French sort supports reverse sort for accented characters

New Sorts (continued)

- JAPANESE_M: Japanese sort supports SJIS character set order and EUC characters which are not included in SJIS
- KOREAN_M: Korean sort: Hangul characters are based on Unicode binary order. Hanja characters based on pronunciation order. All Hangul characters are before Hanja characters
- SPANISH_M: Traditional Spanish sort supports special contracting characters
- THAI_M: Thai sort supports swap characters for some vowels and consonants
- SCHINESE_STROKE_M: Simplified Chinese sort uses number of strokes as primary order and radical as secondary order
- SCHINESE_PINYIN_M: Simplified Chinese PinYin sorting order
- TCHINESE_RADICAL_M : Traditional Chinese sort based on radical as primary order and number of strokes order as secondary order
- TCHINESE_STROKE_M: Traditional Chinese sort uses number of strokes as primary order and radical as secondary order
- SCHINESE_RADICAL_M: Simplified Chinese sort based on radical as primary order and number of strokes order as secondary order

Byte and Character Semantics

- **Length specification can be byte or character:**
  - **When defining column**
  - **Values in SQL function**
- **Byte or character semantics is an attribute of the column, stored in data dictionary**
- **Can be specified for `CHAR` and `VARCHAR2`:**
  - **With column**
  - **As a session default by setting `NLS_LENGTH_SEMANTICS`**
- **`NCHAR` and `NVARCHAR2` are always character semantics.**

ORACLE

Byte and Character Semantics

The default sizing of character data types (CHAR, VARCHAR2, and LONG) are in bytes by default. CHAR(10) in a table definition means 10 bytes not 10 characters. For a single-byte character set encodings the character and byte length are the same. However, multi-byte character set encodings do not correspond to the bytes, making sizing the column more difficult.

Explicitly setting NLS_LENGTH_SEMANTICS to CHAR, either as an environment variable for the client, or in an ALTER SESSION statement, enables the new character length semantics. Alternatively you can specify the semantic mode in the column definition, for example:

```
SQL> CREATE TABLE semantic
  2  (bytewide CHAR(10 BYTE)
  3  ,charwide CHAR(10 CHAR)
  4  );
```

NLS_LENGTH_SEMANTICS has no effect on tables owned by SYS or SYSTEM. These are always treated with byte semantics.

Character-Length Semantics: Data Dictionary

The USER_TAB_COLUMNS and USER_IND_COLUMNS data dictionary tables have been extended to show which semantics have been used in the table (and dependant index) definition.

This also applies to the DBA_* and ALL_* version of the above views.

The CHAR_LENGTH column computes the maximum character length that can be accommodated.The CHAR_USED column indicates with the value 'B' or 'C' whether the column was created with byte or character length semantics, respectively.

```
SQL> SELECT column_name, data_length,
  2         char_length, char_used
  3  FROM   user_tab_columns
  4  WHERE  table_name = 'STRINGS';

SQL> SELECT column_name,
  2         char_length, char_used
  3  FROM   user_ind_columns
  4  WHERE  table_name = 'STRINGS';
```

Implicit Type Conversion for NCHAR Data Type

- **SQL functions that take CHAR/VARCHAR2 as arguments also take NCHAR/NVARCHAR2 as arguments.**
- **You can use the TO_NCHAR function instead of TO_CHAR for output in NCHAR/NVARCHAR2 data type.**

ORACLE

New Implicit Character Data Type Conversion

SQL functions that require a character string now accept either CHAR, NCHAR, VARCHAR2, or NVARCHAR2.

Implicit conversion in assignments is also supported. This reduces the need for the TRANSLATE(…USING…) function.

Prior to Oracle9*i* you had to explicitly specify a text string as being database or national by omitting or prefixing the string with an N (example: N'National'), respectively. Assigning a database string to a national data type or vice versa failed. The implicit conversion hides the need.

The advantage of the implicit conversions is that you can convert existing data from using CHAR to NCHAR columns and only need some minimal change in application code.

The disadvantage is that an unintended conversion may alter the string if some characters are not convertible.

Explicit Conversion

You can still perform explicit conversion, by using the TRANSLATE (… USING …) function, or the new TO_NCHAR function where you would use TO_CHAR.

SQL*Loader Unicode Support

- **Can load UTF-16 data:**
  - **Character-length semantics**
  - **Byte ordering**
- **Character- and byte-length semantics**

ORACLE

SQL*Loader Unicode Support

SQL*Loader is now able to process data files with UTF-16, sometimes called UCS-2, data.The control file specifies character length in characters, not in bytes, for this character set. You specify whether the byte ordering is big- or little-endian. The default is the same as the platform SQL*Loader is running on.

To load UCS-2 or UTF16 data, you add two clauses in the control file:

- `CHARACTERSET UTF16`
- `BYTEORDER {BIG|LITTLE} ENDIAN`

The `CHARACTERSET` keyword is not required if loading into `NCHAR`, `NVARCHAR2`, or `NCLOB` columns, and the National Character Set is AL16UTF16.

String lengths of `CHAR`, `VARCHAR2` and `LOB` character data can be specified in characters instead of bytes by adding the clause:

- `LENGTH SEMANTICS CHAR`

The default is taken from the environment `NLS_LENGTH_SEMANTICS`.

SQL*Loader for Unicode Sample Control File

```
LOAD DATA
CHARACTERSET UTF16
BYTEORDER LITTLE ENDIAN
INFILE TEST.DAT
TRUNCATE
INTO TABLE TEST_EMP (
  EMPNO INTEGER EXTERNAL(4),
  ENAME CHAR(10),
  JOB VARCHARC(2,10),
  EMGRNAME VARCHAR (20),
  RAW_DATA RAW(200)
)
```

ORACLE

Example

The above is an example of a SQL*Loader control file used to load rows containing data in the UTF-16 character set in little-endian format, into a table defined like this:

```
SQL> CREATE TABLE test_emp
  2  (empno    NUMBER(5)
  3  ,ename    NVARCHAR2(30)
  4  ,job      NVARCHAR2(10)
  5  ,emgrname NVARCHAR2(20)
  6  ,raw_data RAW(200) ) ;
```

In this example, the character set of the SQL*Loader data file `TEST.DAT` is UTF-16. The integer external and character data in the file and the length field for the `VARCHAR` field are byte swapped if SQL*Loader is running on a system which is is big endian).

The lengths of the various character fields (`INTEGER EXTERNAL(4)`, `CHAR(10)`, `VARCHARC(2,10)`, and `VARCHAR(20)`) are interpreted in characters because of the character set choice. For example, for the `ENAME` field 20 bytes will be read, as 10 characters in the file are specified, even though the column has room for 30. The length of the `VARCHARC` length field is read as 2 UTF-16 characters (4 bytes), and is the length in characters. The maximum length of the `VARCHARC` field is 10 characters. The length of the `VARCHAR` field will be read as a `SMALLINT` from the data file and is interpreted as a length in characters. The length value bytes for the `VARCHAR` field will likewise be byte swapped if SQL*Loader is running on a system which is big endian. The `RAW` data will be read as 200 bytes and no byte swapping will be done on that data.

Character Set Scanner

- **Used before converting database character sets**
- **Scans database character data for problems**
- **Does not convert data in the database**
- **Scans for any character conversion, not limited to Unicode**
- **Output is a report of potential conversion faults.**
- **Available since Oracle8*i* Release 3 (8.1.7)**

ORACLE

Character Set Scanner

This is a utility to scan all the character data and the related column definitions in the database, to identify data loss if the data is converted to a new character set. It does not alter any character data.

When converting the character data in the database to a different character set, or redeclaring the character set you should:

- Run the Character Set Scanner
- Correct the problems it finds, or find another solution than character set conversion
- Do the conversion

The version of the Character Set Scanner must match the database version. A version for Release 8.1.7 is distributed with the Oracle8*i* Release 8.1.7 database. Versions for earlier Oracle8*i* releases can be requested for backport through Oracle Support Services.

Further Reading

The Character Set Scanner is described in detail in a separate chapter in the *Globalization Support Guide*.

Common Character Conversion Problems

- **Data truncation**
- **Data loss or data corruption**
- **Character set mismatch**

ORACLE

Data Truncation

New byte codes for existing characters may or may not fit into the current columns. For example, migrating from `WE8ISO8859P1`, a single-byte character set, to `AL32UFT8`, a multibyte variable-width character set, some characters will translate into two byte characters. If the migration is performed prior to fixing the column width, the data will be truncated.

Data Loss or Data Corruption

If the source and target character sets are not compatible (subset/superset), characters may be replaced with another character during conversion. For example:

```
Character Set A Character Set B
--------------- ---------------
              €               ?
              ä               a
              ©               ?
```

Character Set Mismatch

If the database character set does not match the usage of the character strings, there is a mismatch, or a erroneous declaration. For example, a database with character set `US7ASCII` can receive data from a Chinese Windows NT client with `NLS_LANG` set to `SIMPLIFIED CHINESE_CHINA.US7ASCII`. Because both the client and database character sets are set to `US7ASCII`, no conversion is performed on the data between the client and server; however, the data is handled differently on each side.

Character Set Mismatch (continued)

This can lead to two data inconsistency problems. First, the server will treat these characters as single-byte USC7ASCII characters, hence all SQL string manipulation functions such as SUBSTR or LENGTH will be based on single bytes rather than (two-byte) characters. Second, when the database is migrated to another character set, for example, UTF8, character codes are converted as if they were in US7ASCII. Each of the two bytes of a ZHS16GBK character will be converted separately, yielding garbage values in UTF8.

The Character Set Scanner cannot directly detect character mismatch situations.

Character Set Scanner Operation

- **Checks all CHAR, VARCHAR2, LONG, CLOB, NCHAR, NVARCHAR2, and NCLOB data**
  - **Done for specified table, user or whole database**
  - **Parallel operation**
- **Checks if:**
  - **Characters change codes**
  - **Characters are representable in new set**
  - **Length of new strings exceeds column definition**
- **Outputs report:**
  - **Three text files**

ORACLE

Operation Mode

The table, user, and whole database modes limit what to scan. Parallellization is very simple. You specify how many scanning processes you want started simultaneously. Each process scan one table.

Check of All Character Data

There are two types of character data in the database: user data, and data dictionary data, such as the names of tables and columns, PL/SQL code, and view definitions. Only in the full database scan are data dictionary character strings checked. The data dictionary information such as columns lengths, is checked in all modes, for the scanned objects.

If the database contains data that is not part of the declared character sets, that is, there is a mismatch situation as described earlier, then either correct the situation, or force the Character Set Scanner to interpret the data in the correct character set by using the FROMCHAR parameter for the tables concerned.

Output

The output is in three text files. The `log` parameter is used to give the path and file name; the scanner generates files with extensions `.out`, `.txt`, and `.err` . The default name is `scan`.

Character Set Scanner Command

- **Utility name is `csscan`**
- **Command-line interface**
- **Interactive prompts**

```
$ csscan system/manager FULL=y TOCHAR=utf8
         ARRAY=102400 PROCESS=5
```

```
$ csscan system/manager PARFILE=filename
```

ORACLE

Character Set Scanner Command

The command line used to invoke the Character Set Scanner is similar to that used by other Oracle utilities. To get the list of all program parameters, use:

```
$ csscan help=y
```

The essential parameters are:

- `USERID`: Standard username/password[@alias] connect string to the database; the user must have DBA privileges.
- `FULL`: If Y, scan the entire database. `FULL` is exclusive to `USER` and `TABLE`.
- `USER`: Owner of tables to scan (Only if `FULL=N`)
- `TABLE`: Names of tables to scan in `(table1,table2,...)` format
- `TOCHAR`: Target `CHAR` character set name; the test conversion of `CHAR`, `VARCHAR2`, `LONG` and `CLOB` data will be performed to this character set.
- `PROCESS`: Number of processes to read from database
- `ARRAY`: Size of fetch array

Optional parameters:

- `LOG`: Name and path of the three output files.
- `TONCHAR`: Target character set of `NCHAR`, `NVARCHAR` and `NCLOB` data
- `PARFILE`: Parameter file that contains all parameters

Character Set Scanner Output

- **scan.out: A copy of the screen log**
- **scan.txt: Database Scanner summary**
  - **Parameters used**
  - **Size of tablespaces**
  - **Amount of data scanned**
  - **List of all table and columns which contain data with problems**
- **scan.err: Errors**
  - **List of rows that have problems**
    - **Unconvertible characters**
    - **Exceed column length**

ORACLE

scan.out

```
Character Set Scanner: Release 9.0.1.0.0 – Production
(c) Copyright 2000 Oracle Corporation. All rights reserved.

Connected to:
Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

Enumerating tables to scan...

. process 1 scanning OE.CUSTOMERS
. process 1 scanning OE.WAREHOUSES
. process 1 scanning OE.ORDERS
. process 1 scanning OE.PRODUCT_INFORMATION
. process 1 scanning OE.PRODUCT_DESCRIPTIONS

Creating Database Scan Summary Report...

Creating Individual Exception Report...

Scanner terminated successfully.
```

scan.txt

```
Database Scan Summary Report

Time Started  : 2001-07-15 23:12:16
Time Completed: 2001-07-15 23:12:18

Process ID         Time Started       Time Completed
---------- -------------------- --------------------
         1  2001-07-15 23:12:16  2001-07-15 23:12:17
---------- -------------------- --------------------

[Database Size]

Tablespace              Used(MB)   Free(MB)  Total(MB)
----------------------- -------- ---------- ----------
SYSTEM                   266,000      8,000    274,000
USERS                      1,000     19,000     20,000
TEMP                         ,000     20,000     20,000
COMSCHEME                  3,000      7,000     10,000
----------------------- -------- ---------- ----------
Total                    302,000    122,000    424,000

[Database Scan Parameters]

Parameter                      Value
------------------------------ ----------------------
Scan type                      User tables
User name                      oe
Scan CHAR data?                YES
Current database character set WE8MSWIN1252
New database character set     we8iso8859p1
Scan NCHAR data?               YES
Current NCHAR character set    WE8MSWIN1252
New NCHAR character set        utf8
Array fetch buffer size        100000
Number of processes            1
------------------------------ ----------------------

[Scan Summary]

Some character type application data are not
convertible to the new character set

[Application Data Conversion Summary]

Datatype      Changeless Convertible Exceptional  Total
------------- ---------- ----------- ----------- ------
VARCHAR2           4.790           0           0  4.790
CHAR                 319           0           0    319
LONG
CLOB
NVARCHAR2            380         287           1    668
NCHAR
NCLOB
------------- ---------- ----------- ----------- ------
Total              5.489         287           1  5.777
```

scan.txt (continued)

```
[Distribution of Convertible Data per Table]
USER.TABLE                              Convertible Exceptional
--------------------------------------- ----------- -----------
OE.PRODUCT_DESCRIPTIONS                         287           1

[Distribution of Convertible Data per Column]
USER.TABLE|COLUMN                       Convertible Exceptional
--------------------------------------- ----------- -----------
OE.PRODUCT_DESCRIPTIONS|TRANSLATED_NAME          16           1
OE.PRODUCT_DESCRIPTIONS|TRANSLATED_DESCRIPTION 271           0
--------------------------------------- ----------- -----------
[Indexes to be Rebuilt]
USER.INDEX on USER.TABLE(COLUMN)
-------------------------------------------------------------
OE.PROD_NAME_IX on OE.PRODUCT_DESCRIPTIONS(TRANSLATED_NAME)
-------------------------------------------------------------
```

scan.err

```
Database Scan Individual Exception Report
[Database Scan Parameters]
Parameter                     Value
----------------------------- ------
```

(omitted here)

```
[Application data individual exceptions]
User  : OE
Table : PRODUCT_DESCRIPTIONS
Column: TRANSLATED_NAME
Type  : NVARCHAR2(50)
Number of Exceptions         :  1
Max Post Conversion Data Size: 51
ROWID              Exception Type      Size Cell Data   ...
------------------ ------------------ ----- ----------------
AAAGFaAAIAAAAEvAAi exceed column size    51 Blåbærgrød  ...
------------------ ------------------ ----- ----------------
```

The three reports have been slightly edited to fit and for clarity.

Conclusion from these reports: An Export and Import is required, because some data has to be converted. Also one row will not fit. Either the column can be made longer, or you can edit the character data (as it is a description field, it can perhaps be abbreviated).

Character Conversion

- **This step is done separately.**
- **Use one of the following:**
  - **Full Export and Import (recommended)**
  - **`ALTER DATABASE`, if no characters change codes**
  - **A combination of the two; doing export and import on the few tables that contain changing character codes, and an `ALTER DATABASE`**

ORACLE

The Character Conversion

The character conversion process is done after you have considered the output of the Character Set Scanner. The choice of how to perform the conversion are the same as in previous versions of the server, and are made after the Character Set Scanner has completed. They are briefly summarized here.

Export and Import

This is the recommended way to convert a database character set. While this is a lengthy process, it has some other beneficial reorganization of the database (index rebuild, compacting deleted space, and so forth). You still have to be aware of column size changes, as reported by the Character Set Scanner.

- Export the database.
- Recreate the database with the new character sets.
- Create the tables which require longer columns.
- Import the data (`IGNORE=YES` to import into tables created in previous step).

ALTER DATABASE [NATIONAL] CHARACTER SET

This is a faster alternative, if the Character Set Scanner has confirmed that no characters change their code. In addition, check the *Globalization Support Guide* if the change is permissible, that is, if the new character set is a superset of the current character set.

The ALTER DATABASE CHARACTER SET command is sometimes used to change the declaration to reflect the true character set of data previously entered, that is, to correct a mis-match situation described earlier. Discussion of this situation is outside the scope of this lesson.

Combination Export/Import and ALTER DATABASE

This reduces the time needed for export and import, if the amount of data that needs converting is a small part of the database.

- Export the tables with contentious data.
- Truncate those tables.
- ALTER DATABASE … CHARACTER SET …
- Import the table data again.

Oracle Locale Builder

A GUI based tool to build your own locale:

- **Language**
- **Territory**
- **Character set**
- **Collation**
- **Does not include building own message files**

ORACLE

Expanded Locale Coverage and User Definable Locales

Oracle9*i* provides a large number of locales. If you need another locale, or make an adjustment to an existing locale, then you can do so using the Locale Builder. Viewing the definitions in the current supplied locales gives you an accurate and easy-to-read description of it, without having to understand the formatting of the `.NLT` files.

The globalization definitions are stored in files in `$ORACLE_HOME/ocommon/nls/admin/data` with extensions `.NLT` (readable source text) and `.NLB` (translated binary) and are read by the server. You use the Locale Builder to view and modify these. You can also create your own.

Starting the Locale Builder

Start the executable `lbuilder`. It is located in `$ORACLE_HOME/ocommon/nls/lbuilder`.

Further Reading

The Locale Builder is fully documented in the *Globalization Support Guide,* chapter 11.

Locale Builder Examples

- **Default sorting order of a language**
- **Default date formatting of a territory**
- **Conversion of characters between character sets**
- **Collation order of a language**

ORACLE

Default Sorting Order

Setting a language (using the `NLS_LANG` environment variable) the value of `NLS_SORT`, among others, is set to a corresponding value at client start up. This can be altered, thus avoiding the need to perform an explicit `ALTER SESSION SET NLS_SORT=value` after starting the client. Other language subordinate values can also be altered.

Default Date Formatting

Setting a territory will similarly set `NLS_DATE_FORMAT` to a corresponding value. You can use the Local Builder to define your own format to be applied for the territory.

Character Definitions, Conversion

When the server has to convert characters between client and server, and the source character is not available in the destination set, you can define what the replacement character is to be, either the default or the specific replacement character can be defined in the character set. For example, you can define that `'ä'` be replaced by `'a'` instead of `'?'` in a variation of the US7ASCII character set.

Collation

In the sorting order (collation) defined above, you choose which sorting order to apply. For a collation you can define precisely how to sort the characters.

New or Modified Globalization Settings

- **Do not alter the supplied files, preferably.**
- **Create new definitions, based on the supplied files.**
- **Generate binary files on each platform using `lxinst`**
- **Use the new definitions**

ORACLE

New or Modified Globalization Settings

The recommendation is to define new definitions, rather than altering the existing ones. For example you call the modified language `DUTCH EXT`, if it is a variation of the default `DUTCH`, rather than modifying the existing language. You have to define a numeric identification for your new language. It is recommended you name the files in the same convention.

It is recommended that you save your modifications into a new file, and thus create your own named language, territory, character set or collation, rather than into the existing file, thus altering the Oracle-supplied globalization definitions.

Generation of NLB files

After all changes have been made, you must generate the binary `.NLB` version of the `.NLT` file. This can be done in the Locale Builder, if it is the same platform, otherwise you must copy the `.NLT` file to the target system, and use the `lxinst` utility to compile them. This will also update the `lxboot`-file, so you must have an appropriate backup of the NLS files before making changes. You need to restart the instance for the new files to become effective.

Using the New or Modified Globalization Settings

You select your new language with the same methods. In the example above, you use the environment variable `NLS_LANG="DUTCH EXT_NETHERLANDS.WE8ISO8859P1"`, or `ALTER SESSION SET NLS_LANGUAGE='DUTCH EXT';` The quotation marks are necessary here, because the name contains a space.

Summary

In this lesson, you should have learned about:

- **The new time and date data types**
- **Unicode enhancements**
- **New localization definitions**
- **The Character Set Scanner**
- **The Locale Builder**

ORACLE

Practice 18-1 Overview

This practice covers the following topics:

- **Creating and manipulating timestamp data between time zones**
- **Displaying and altering the client time zone**

ORACLE

19

Database Workspace Management

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- **Identify the Workspace Manager role**
- **Version enable a table**
- **Disable workspace participation for a table**
- **Create and assign a workspace**
- **Understand Import and Export considerations**

ORACLE

What Is a Database Workspace?

A shared, transactionally consistent view of database tables, that:

- **Captures changes to version-enabled tables as new row versions within the workspace**
- **Makes versions invisible outside the workspace until explicitly merged with the parent workspace**
- **Changes made to a table that is not version enabled are made directly to the table**

ORACLE

Workspaces

A workspace is a virtual environment that provides a transactionally consistent snapshot of the entire database. One or more users can share this environment to version data in the database.

The unit of versioning is the table. When a user in a workspace updates a row in a version-enabled table, a new version of the row is created. Versions are only accessible within the workspace until explicitly merged with the parent workspace. This also applies to inserts and deletes made within a workspace to a version-enabled table.

There can be one or more versions of a row in a workspace from one or more version-enabled tables. The current or active version of a row in a workspace refers to the version of a row to which changes are currently being made.

Tables which are not version enabled will be changed like normal regardless of which workspace a user might be in. Access to any table is still controlled by the privileges a user has, not which workspace a user is in.

It is possible to create workspaces and not version enable any tables, and it is possible to version enable tables and not create any workspaces. But doing either of these provides no additional capabilities to the database. To fully use this feature you must do both. Certainly there will be instances where the application may be viewing and or modifying both version-enabled and non-version-enabled tables.

How Does Workspace Manager Work

- **Automatically installed with Oracle9*i***
- **Allows for version enabling tables by running a packaged procedure**
- **Automatically versions only changed rows**
- **Provides mechanism to identify and resolve conflicts**
- **Merges changes with parent or discard changes**

ORACLE

Versioning

By using Database Workspace Manager you can selectively version enable some or all tables in an existing database.

Once a table is version enabled, changes to any row in that table are only viewable to the workspace in which the change was made. The changes to tables within a workspace are done with conventional short transaction mechanisms. All DML and DCL is available while in a workspace.

DDL is not supported on version-enabled tables. If necessary you can disable versioning, perform any DDL needed, then version enable the table again. You will lose any versioned rows when you disable versioning.

The changes to a row are stored in the same table. When version-enabled, new columns are added to the table to allow for multiple version of each row. So the growth of the table depends on how many versions of each row are created.

Eventually the different versions of the row will either be merged into any parent workspace (and ultimately the LIVE workspace) or discarded. Conflicts are automatically detected and must be resolved before a merge can take place.

Workspace Manager Administrator Role

- **Oracle Installer automatically installs Workspace Manager and creates the `WM_ADMIN_ROLE`.**
- **The `WM_ADMIN_ROLE` role has all Workspace Manager privileges.**
- **By default, the DBA role is granted the `WM_ADMIN_ROLE`.**
- **The DBA can grant privileges or can grant the `WM_ADMIN_ROLE` role to one or more users to grant the privileges.**

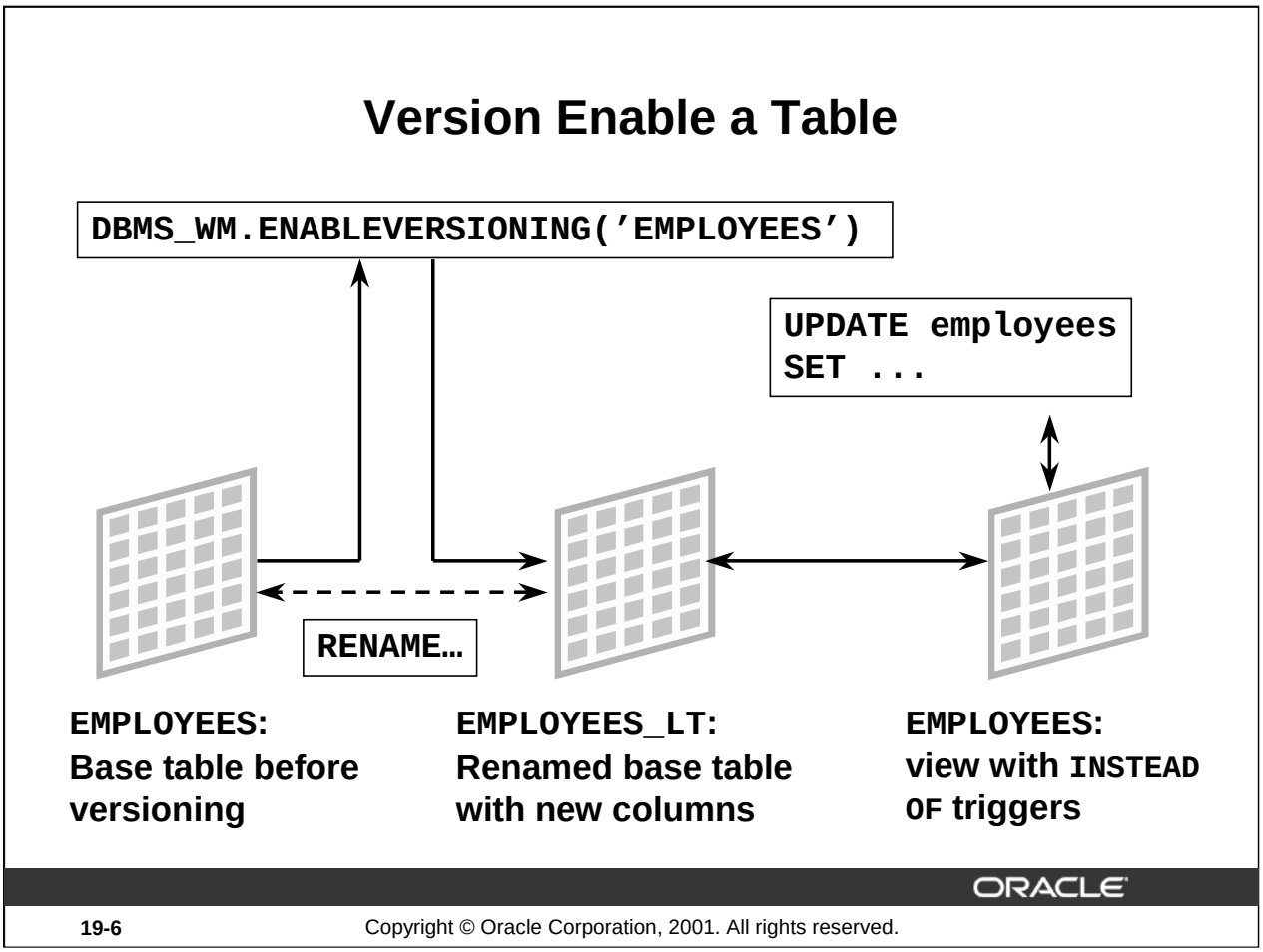
ORACLE

Workspace Manager Administrator Role

The `WM_ADMIN_ROLE` role has all Workspace Manager privileges with the grant option.

By default, the DBA role is granted the `WM_ADMIN_ROLE`.

Decide which users should be granted which privileges. Then, either the DBA can grant the privileges, or the DBA can grant the `WM_ADMIN_ROLE` role to one or more selected users and these users can grant the privileges.



Workspace Enabling a Table

The example on the slide version enables the EMPLOYEES table. ENABLEVERSIONING performs the following tasks:

- Augments the table with four workspace metadata columns to store the following data:
  - VERSION: The row version ID
  - LTLOCK: The lock status
  - DELSTATUS: The delete status
  - NEXTVER: The next row version ID
- Renames the table by adding the suffix _LT to the table name
- Creates a view with the same name as the original table
- Creates INSTEAD OF triggers on the view for insert, update, and delete of versioned rows. Once version-enabled, all rows in the table can support multiple versions of data.

When the view is accessed, it uses the workspace metadata to show only the row versions relevant to the current workspace of the user. The workspace infrastructure is not visible to the end-users.

Changes Due to Versioning

- **A new table <table_name>_AUX for conflicts**
- **Index changes:**
  - **The PK index is changed**
  - **New index <table_name>_PKI$**
  - **New index <table_name>_TI$**
  - **New PK index AUX_<table_name> on the <table_name>_AUX table**
- **Several views, most for internal use only**

ORACLE

Changes Due to Versioning

Tables

Not only is the base table changed as already described, also a new table is added with the name of <table_name>_AUX. This table maintains information about rows that have been synchronized after conflicts have been resolved. Its size is therefore directly dependent on the number of conflicts that get resolved.

Indexes

The primary key is changed and three indexes are created when ENABLEVERSIONING is executed on a table:

- The primary key has the VERSION column added, if history is enabled (the VIEW_WO_OVERWRITE/VIEW_W_OVERWRITE options) then the CREATETIME column is also added.
- <table_name>_PKI$: This is a nonunique index that is the original primary key index without the version column. This is needed because several workspace operations do queries on the <table_name>_LT without the version column. For any given primary key, the number of index entries equal the number of versions where this primary key row has been versioned.

Changes Due to Versioning (continued)

Indexes (continued)

- <table_name>_TI$: This is nonunique index created to support history. It is made up of these two columns: CREATETIME, RETIRETIME. It is needed for support of the <table_name>_HIST view. For any given primary key, the number of index entries equal the number of versions where this primary key row has been versioned, because each version of the row will have distinct CREATETIME, RETIRETIME values if the table was versioned with VIEW_W_OVERWRITE option. If the table was versioned with VIEW_WO_OVERWRITE, the number of index entries will be the number of DML changes made to this primary key because in this case every DML operation is captured as a distinct row with its own CREATETIME, RETIRETIME.
- Also a primary key is created on the <table_name>_AUX table. The index is named AUX_<table_name>. This is relatively small index with the number of entries dependent on the number of conflicts resolved.

Views

There are several views created when a table is version enabled. Most are used internally by the procedures and functions of the DBMS_WM package. Some of the views created are:

- Conflict view, each having a name in the form <table_name>_CONF.
- Difference view, each having a name in the form <table_name>_DIFF.
- Lock view, each having a name in the form <table_name>_LOCK.
- History view (if history tracking is enabled), each having a name in the form <table_name>_HIST.
- Multiworkspace view, each having a name in the form <table_name>_MW.

Note: Because of these changes and additions, the length of the table name that is being version enable is limited to 25 characters. If you try to version enable a table with a longer name, you will get: ORA-20136: table names are limited to 25 characters.

The History Option

- **Optionally, the history of versions of rows can be tracked.**
- **Parameter of the `ENABLEVERSIONING` procedure**
- **Possible values:**
  - **`NONE`: No tracking, the default**
  - **`VIEW_W_OVERWRITE`: Most recent only**
  - **`VIEW_WO_OVERWRITE`: All modifications**

```
DBMS_WM.ENABLEVERSIONING
('employees','VIEW_WO_OVERWRITE')
```

ORACLE

The History Option

This option tracks modifications to `table_name` using a view named `<table_name>_HIST`. It must be set to one of the following values:

- `NONE:` No modifications to the table are tracked. This is the default.
- `VIEW_W_OVERWRITE`: Shows only the most recent modification, subsequent changes to a row in the same version, overwrite earlier changes.
- `VIEW_WO_OVERWRITE`: Shows all modifications, subsequent changes to a row in the same version, do not overwrite earlier changes.

If history is enabled, then two more columns are added to the `<table_name>_LT` table:

- `CREATETIME`: Date and time this version of the row was created
- `RETIRETIME`: Date and time this version of the row was replace by another in the same workspace.

Guidelines for Tables Participating in a Workspace

- A version-enabled table must have a primary key.
- A table can be version enabled by the table owner or by a user with `WM_ADMIN_ROLE`.
- Tables owned by `SYS` cannot be version enabled.
- Referential integrity constraints are supported on version-enabled tables.
- Triggers are supported on version-enabled tables with some restrictions.

ORACLE

Guidelines for Tables Participating in a Workspace

Referential Integrity Constraints

Version-enabled tables can have referential integrity constraints, including constraints with the `CASCADE` and `RESTRICT` options; however, the following considerations and restrictions apply:

- If the parent table in a referential integrity relationship is version enabled, the child table must be version enabled also. The child table is the one on which the constraint is defined.
- Referential integrity constraints cannot be added when a table is version enabled; they must be defined before a table is version enabled.
- A child table in a referential integrity relationship is allowed to be version enabled without the parent table being version enabled.
- A version-enabled table cannot be both a child and a parent in a referential integrity relationship, unless it is a self-referential constraint.

Guidelines for Tables Participating in a Workspace (continued)

Triggers

Version-enabled tables can have triggers defined; however, the following considerations and restrictions apply:

- The triggers must be defined before the table is version enabled.
- Only row level triggers are supported. Statement level triggers are not supported.
- Only whole row triggers are supported, that is, before and after update triggers for specific columns are not supported.
- Triggers on nested table columns are not supported.
- The only call-out supported is to PL/SQL procedures. That is, the `ACTION_TYPE` must be PL/SQL.
- Any triggers that are not supported for version-enabled tables are deactivated when versioning is enabled, and are activated when versioning is disabled.

Disabling Workspace Participation for a Table

- **Reverses workspace enabling:**

```
DBMS_WM.DISABLEVERSIONING('employees')
```

- **It can be done by the table owner or by a user with the `WM_ADMIN_ROLE`.**
- **Workspace hierarchy and savepoints remain.**
- **The latest version of each row in `LIVE` workspace remains.**
- **The optional `FORCE` argument allows the user to disable versioning on a table even if workspaces have modified data in the table.**

ORACLE

Disabling Workspace Participation for a Table

The `DISABLEVERSIONING` procedure is used to reverse the effect of the `ENABLEVERSIONING` procedure. It deletes the Workspace Manager infrastructure for versioning of rows, but does not affect any user data in the `LIVE` workspace. The workspace hierarchy and any savepoints still exist, but all rows are the same as in the `LIVE` workspace. If there are multiple versions of a row in the table for which versioning is disabled, only the most recent version of the row is kept.

The example on the slides disables versioning on the employees table.

Workspace Savepoints

- **The concept of a savepoint in the Workspace environment is similar to that of other savepoints.**
- **They are two types of savepoints:**
  - **Explicit savepoints are created to roll back changes in workspaces**
  - **Implicit savepoints are created automatically whenever a new workspace is created**
- **The savepoint takes no space, but can cause more versions of a row to be created.**

ORACLE

Savepoints

Savepoints are used in similar ways to save points used in transactional operations. One of the main differences is that once a savepoint is created any row modified after the creation of the savepoint will cause a new version of the row to be stored in the table.

Like transactional savepoints they can be used for rollback. Unlike transitional ones, the user does have access to the data before and after the savepoint.

When calculating the amount of space that might be needed you must consider the following: each row could be potentially duplicated in the table for each workspace in the database and for each savepoint within a workspace.

For example, if a database has five workspaces created and each workspace has three savepoints within them, then for each version-enabled table that could be 15 copies of each row. This should be extreme because it is unlikely that each workspace is going to modify every row in the table for each savepoint.

The activities that the developer or user can do with savepoints include: Create, Go to savepoint, Go to date, Alter, Compare difference, and Delete.

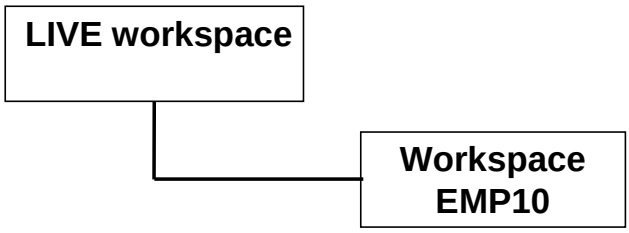
Savepoint are not covered in detail in this course.

Create a Workspace

- **Create a workspace named `EMP10`:**

```
DBMS_WM.CREATEWORKSPACE('EMP10')
```

- **A new workspace is a child of the current workspace.**
- **It creates implicit savepoint in the current workspace.**



ORACLE

Create a Workspace

If the session has not explicitly entered a workspace, it is in the `LIVE` database workspace, and the new workspace is a child of the `LIVE` workspace.

This procedure does not implicitly go to the workspace created. To go to the workspace, use the `GOTOWORKSPACE` procedure.

An implicit savepoint is created in the current version of the current workspace when a new workspace is created. This allows the child workspace to have a transactionally consistent view of the database. The current version for which the savepoint is created does not have to be the latest version in the current workspace

An exception is raised if a workspace already exists or the user does not have the privilege to create a workspace.

Workspace names are case sensitive.

Assign Workspace: Associate a User

- At login, the user is placed in the `LIVE` workspace.
- The `GOTOWORKSPACE` procedure moves the current user session to the destination workspace.
- To include the user in the `EMP10` workspace:

```
DBMS_WM.GOTOWORKSPACE('EMP10')
```

- All subsequent modifications to data by the user take place on the latest versions in the `EMP10` workspace.

ORACLE

Assign Workspace

By using the GOTOWORKSPACE procedure current user is then "in" the named workspace. From this point on all changes to version-enabled tables are only viewable in the named workspace. A user must have the ACCESS_WORKSPACE privilege for the named workspace or the ACCESS_ANY_WORKSPACE system privilege.

Because many operations are prohibited when any users are in the workspace, it is often convenient to go to the LIVE workspace before performing operations on created workspaces.

To go to the LIVE database, specify workspace as LIVE.

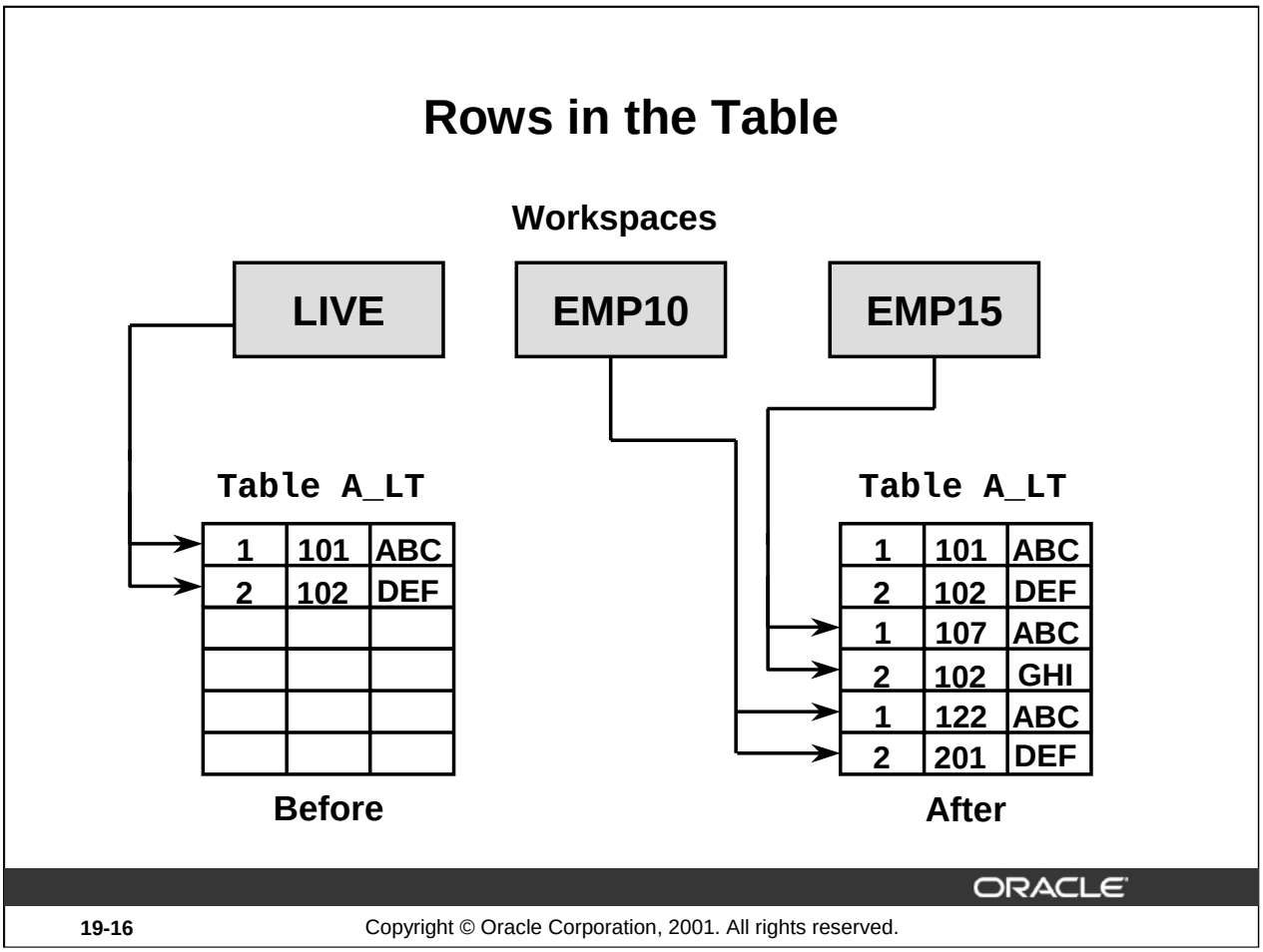
For example:

```
SQL> EXECUTE DBMS_WM.GOTOWORKSPACE ('LIVE');
```

To see which workspace your session is currently in, use the GETWORKSPACE function, for example:

```
SQL> SELECT DBMS_WM.GETWORKSPACE FROM DUAL;

GETWORKSPACE
------------------------
EMP10
```



Rows in the Table

In this example you have a version-enabled table with two rows. There are two workspaces created under the `LIVE` workspace, named `EMP10` and `EMP15`.

Then a user in the `EMP10` workspace does an update on the two rows in the table, and so does a user in workspace `EMP15`.

Now each workspace has its own version of each row. The net effect is that there are now three copies of each row in the table. One for the `LIVE` workspace and one each for the `EMP10` and `EMP15` workspaces.

The table here has been simplified for the discussion. It doesn't show the extra columns that are added by the versioning of the table.

Assign Workspace: Grant Privileges

- **Workspace privileges: `ACCESS`, `CREATE`, `REMOVE`, `MERGE`, and `ROLLBACK`**
- **Privileges in the form of `priv_WORKSPACE` allow the user to affect a specified workspace.**
- **Privileges in the form: `priv_ANY_WORKSPACE` allow the user to affect any workspace.**

```
GRANTWORKSPACEPRIV
( 'ACCESS_WORKSPACE,
   MERGE_WORKSPACE',
  'EMP10',
  'SMITH',
  'YES' );
```

ORACLE

Workspace Privileges

Workspace Manager implements a set of privileges in addition to standard Oracle database privileges. There are two types of privileges:

- Workspace-level privileges have names in the form `priv_WORKSPACE`. They allow the user to affect a specified workspace.
- System-level privileges have names in the form `priv_ANY_WORKSPACE`. They allow the user to affect any workspace.

Each privilege can be granted with or without the grant option. The grant option allows the user to which the privilege is granted to grant the privilege to other users.

The example in the slide enables user Smith to access the `EMP10` workspace and merge changes in that workspace, and allows Smith to grant the two specified privileges on `EMP10` to other users.

Workspace Privileges (continued)

The following are the available privileges for working with workspaces.

Privilege	Description
ACCESS_WORKSPACE	Allows the user to go to a specified workspace. ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other privileges.
ACCESS_ANY_WORKSPACE	Allows the user to go to any workspace. ACCESS_WORKSPACE or ACCESS_ANY_WORKSPACE privilege is needed for all other privileges.
CREATE_WORKSPACE	Allows the user to create a child workspace in a specified workspace.
CREATE_ANY_WORKSPACE	Allows the user to create a child workspace in any workspace.
REMOVE_WORKSPACE	Allows the user to remove a specified workspace.
REMOVE_ANY_WORKSPACE	Allows the user to remove any workspace.
MERGE_WORKSPACE	Allows the user to merge the changes in a specified workspace to its parent workspace.
MERGE_ANY_WORKSPACE	Allows the user to merge the changes in any workspace to its parent workspace.
ROLLBACK_WORKSPACE	Allows the user to roll back the changes in a specified workspace.
ROLLBACK_ANY_WORKSPACE	Allows the user to roll back the changes in any workspace.

To grant most of the above, the GRANTWORKSPACEPRIV procedure is used. This grants workspace-level privileges to users and roles. These are the above privileges without the "any" in the name. The grant_option parameter enables the grantee to then grant the specified privileges to other users and roles.

```
DBMS_WM.GrantWorkspacePriv(
  priv_types    IN VARCHAR2
 ,workspace     IN VARCHAR2
 ,grantee       IN VARCHAR2
[,grant_option  IN VARCHAR2 DEFAULT 'NO']
[,auto_commit   IN BOOLEAN DEFAULT TRUE]);
```

For the system privileges, the ones with the "any" in the name, the GRANTSYSTEMPRIV procedure is used:

```
DBMS_WM.GrantSystemPriv(
   priv_types   IN VARCHAR2
  ,grantee      IN VARCHAR2
 [,grant_option IN VARCHAR2 DEFAULT 'NO']
 [,auto_commit  IN BOOLEAN DEFAULT TRUE]);
```

The AUTO_COMMIT parameter functions in the following way: if TRUE (the default), causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. If FALSE,causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.

Assign Workspace: Set Locks

- **Sets session override to exclusive locking:**

```
DBMS_WM.SETLOCKINGON ('E')
```

- **Shared locking on for EMP10 workspace:**

```
DBMS_WM.SETWORKSPACELOCKMODEON('EMP10','S',TRUE)
```

- **Displays the locking mode in effect for the session:**

```
SELECT DBMS_WM.GETLOCKMODE FROM DUAL;
```

- **Locks the row in employees table as shared where DEPARTMENT_ID=30 in EMP10**

```
DBMS_WM.LOCKROWS
('EMP10','EMPLOYEES','DEPARTMENT_ID = 30','S')
```


Locks

Workspace Manager locks eliminate row conflicts between a parent and child workspace. Locking is enabled at a session level and is a session property independent of the workspace that the session is in. When locking is enabled for a session, it locks rows in all workspaces in which it participates. Locking activities include:

- `SetLockingOn` or `SetLockingOff` sets locking for the session
- `SetWorkspaceLockModeOn` or `SetWorkspaceLockModeOff` is the default mode for workspace row-level locking. If OFF, it enables access to versioned rows in the specified workspace and to corresponding rows in the parent workspace.
- `GetLockMode` returns the locking mode for the parent and child workspace.
- `LockRows` or `UnlockRows` specifies access to versioned rows in a specified table and to corresponding rows in the parent workspace. Either used to proactively lock rows before they are updated or automatically locks row after it is updated be a SQL statement.

In addition to locks provided by conventional Oracle short transactions (transactions without workspaces), Workspace Manager provides two types of version locks:

- Exclusive locks are similar to short transaction locks in that once an exclusive lock has been placed on a record, no other user in the database can change the record except for the session that locked it.
- Shared locks ensure that only users in the workspace in which the row was locked are allowed to modify it.

Freeze a Workspace

- **Specifies the access allowed to the workspace:**
  - **NO_ACCESS is the default**
  - **READ_ONLY for all workspace users**
  - **1WRITER enables a single writer, all readers**
  - **WM_ONLY is for workspace operations**
- **Examples:**

```
FreezeWorkspace('EMP10','READ_ONLY')
```

```
UnFreezeWorkSpace('EMP10')
```

ORACLE

Freeze a Workspace

A workspace can be frozen or not frozen. If a workspace is frozen, no changes can be made to data in version-enabled tables, and access to the workspace is restricted.

To make a workspace frozen, use the `FreezeWorkspace` procedure. To make a frozen workspace not frozen, use the `UnfreezeWorkspace` procedure.

In addition, some procedures automatically freeze one or more workspaces.

`WM_ONLY` allows workspace operations such as merge, rollback, refresh, and others.

Roll Back a Workspace

- **Discards all changes made in the workspace, a table or after a specified savepoint; the workspace structure is not affected.**
- **Examples: Roll back all workspace changes, roll back changes since a savepoint, and roll back all changes to a table**

```
DBMS_WM.RollbackWorkspace('EMP10')
```

```
DBMS_WM.RollbackToSP('EMP10','SAVEPOINT1')
```

```
DBMS_WM.RollbackTable('EMP10','HR.EMPLOYEES')
```

ORACLE

Roll Back a Workspace

Rolling back a workspace involves deleting either all changes made in the workspace or all changes made since an explicit savepoint.

Before rolling back to a savepoint:

- Remove any implicit savepoints created since the specified savepoint by merging or removing the descendent workspaces that caused the implicit savepoints to be created.
- Remove all active users.

Rolling back a workspace leaves behind the workspace structure for future use; only the data in the workspace is deleted. To completely remove a workspace, use the `RemoveWorkspace` procedure.

Refresh a Workspace

- **Applies all changes made in the parent to the child since the child was created or last refreshed**
- **Refresh changes made to a single table:**

```
DBMS_WM.RefreshTable
('EMP10','EMPLOYEES','DEPARTMENT_ID = 30')
```

- **Refresh all workspace changes:**

```
DBMS_WM.RefreshWorkspace('EMP10')
```

- **Before refreshing a table:**
  Regular (nonworkspace) transactions must be committed and conflicts must be resolved.

ORACLE

Refresh a Workspace

The first example refreshes `EMP10` by applying changes made to the `EMPLOYEES` table where `DEPARTMENT_ID = 30` in its parent workspace.

The second example refreshes `EMP10` by applying changes made in its parent workspace.

Resolve Workspace Conflicts

- **Conflict: The same row is changed in two or more workspaces.**
- **Conflicts are detected when a workspace merge or refresh operation is attempted.**
- **Conflicts must be resolved before merge or refresh operations succeed.**
- **Resolve conflicts by choosing a row value from:**
  - **BASE**
  - **CHILD**
  - **PARENT**

ORACLE

Resolve Workspace Conflicts

Rows that are changed in the child and parent workspace or in two peer workspaces can lead to data conflicts. Conflicts are discovered during a merge or refresh operation:

- During a merge operation, the changes in a child workspace are incorporated in its parent workspace.
- During a refresh operation, changes made in the parent workspace are incorporated in the child workspace

Conflicts are presented to the user in conflict views, with one conflict view per table. The conflict view lists the primary key of the rows in conflict and also the column values of the rows in the two workspaces that form the conflict.

Conflicts have to be resolved by using the `ResolveConflicts` procedure. During this procedure the user chooses the row value from the base, child, or parent table.

When there are no conflicts between the parent and child workspaces, the data in the two workspaces can be merged.

Conflicts must be resolved before a `MergeWorkspace` or `RefreshWorkspace` operation can be performed.

Conflict Resolution Example: Check for Existence of Conflicts

- **Check for conflicts between child and parent:**

```
DBMS_WM.SetConflictWorkspace('EMP10_focus_2')
```

- **View conflicts in EMPLOYEES table:**

```
SELECT * FROM employees_conf;
```

ORACLE

Conflict Resolution Example: Check for Existence of Conflicts

The general process to check for the existence of conflicts is as follows:

`SetConflictWorkspace`: The example checks for any conflicts between child workspace, `EMP10_FOCUS_2`, and its parent workspace, `EMP10`, and activates a view for every version-enabled table where the view name is `<table_name>_CONF`.

The query on the screen shows the conflicts for that table.

Merge a Workspace

- **Applies changes in a child workspace or table to its parent workspace or table.**
- **Merge either entire workspace, table, or specific rows**
- **Conflicts must be resolved first.**
- **`MergeTable` can roll back to initial workspace state.**
- **`MergeWorkspace` can remove the workspace.**

```
EXEC DBMS_WM.MERGEWORKSPACE('EMP10',
REMOVE_WORKSPACE=>TRUE);
```

ORACLE

Merge a Workspace

Merging a workspace applies changes made in a workspace to its parent workspace and optionally removes it. While a merge is in progress the child workspace is frozen in `NO_ACCESS` mode and the parent workspace is frozen in `READ_ONLY` mode.

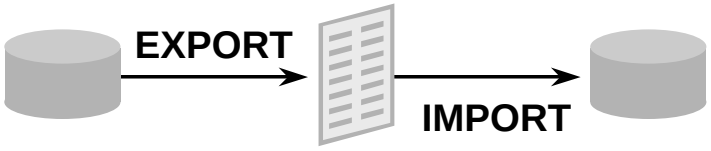
`MERGEWORSPACE` has the following parameters:

- `WORKSPACE`: Name of the workspace. The name is case sensitive.
- `CREATE_SAVEPOINT`: A Boolean value (TRUE or FALSE). TRUE creates an implicit savepoint in the parent workspace before the merge operation. FALSE (the default) does not create an implicit savepoint in the parent workspace before the merge operation.
- `REMOVE_WORKSPACE`: A Boolean value (TRUE or FALSE). TRUE removes workspace after the merge operation. FALSE (the default) does not remove workspace after the merge operation; the workspace continues to exist.
- `AUTO_COMMIT`: A Boolean value (TRUE or FALSE). TRUE (the default) causes the operation to be executed as an autonomous regular transaction that will be committed when it finishes. FALSE causes the operation to be executed as part of the caller's open regular transaction (if one exists). If there is no open regular transaction, the operation is executed in a new regular transaction. In either case, the caller is responsible for committing the transaction.

If there are conflicts between the workspace being merged and its parent workspace, the merge operation fails and the user must manually resolve conflicts using the `<table_name>_CONF` view.

Import and Export Considerations

- **A database with version-enabled tables can be exported only if the database being imported into does not have any version-enabled tables or workspaces.**
- **Only database-wide import and export operations are supported for version-enabled databases.**
- **For an import operation, you must specify `IGNORE=Y`.**
- **The `FROMUSER` and `TOUSER` import options are not supported with version-enabled databases.**



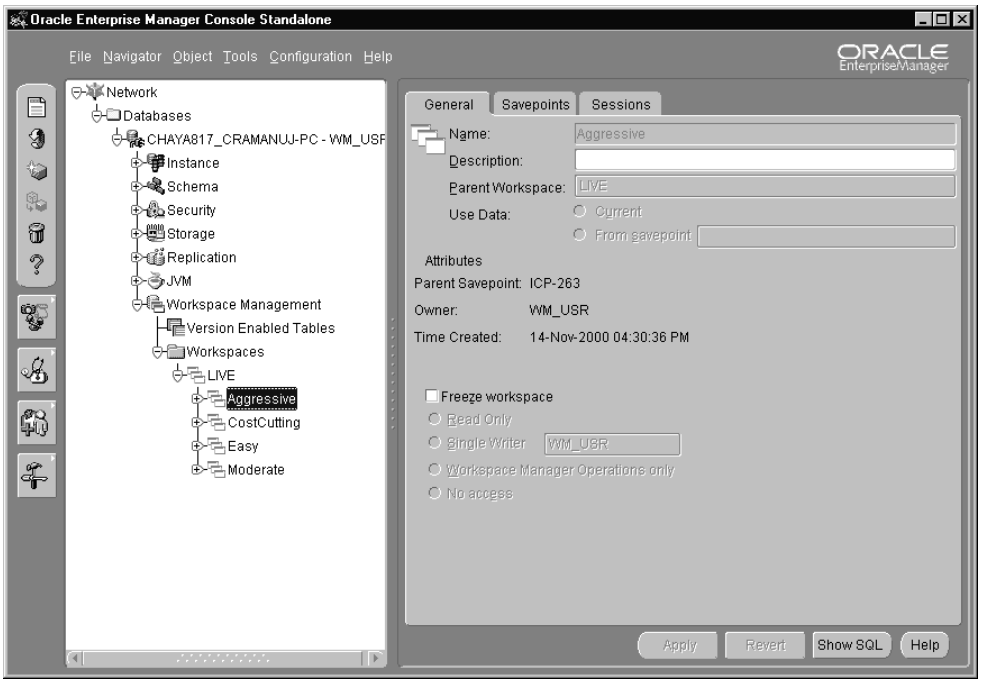
ORACLE

Import and Export Considerations

Standard Oracle database import and export operations can be performed on version-enabled databases; however, the following considerations and restrictions apply:

- A database with version-enabled tables can be exported to another Oracle database only if the other database has Workspace Manager installed and does not currently have any version-enabled tables or workspaces (that is, other than the LIVE workspace).
- Only database-wide import and export operations are supported for version-enabled databases. No other export modes (such as schema, table, partition, and workspace) are supported.
- For an import operation, you must specify `IGNORE=Y`.
- The `FROMUSER` and `TOUSER` capabilities of the Oracle9*i* Import utility are not supported with version-enabled databases.

Enterprise Manager Interface



ORACLE

Enterprise Manger

Database Workspace Manager is integrated in with Oracle's Enterprise Manager (OEM) Console. While connected to a database, you will see a folder called Workspace Management that can be expanded to see two subfolders: Version Enabled Tables and Workspaces.

The OEM Version Enabled tables subfolder allows you to view table status and set tables as version-enabled.

Enterprise Manager (continued)

The OEM Workspaces folder allows you to:

- Create and view workspace hierarchies and attributes. In the slide four workspaces have been created with variations on the live data for what-if cost cutting analysis. The `AGGRESSIVE` workspace is currently selected.
- Set and view workspace access modes. The user access modes for a workspace are:
  - No access, is the default
  - Read only
  - Single writer, allowing all other users to read
  - Workspace operations only, such as merge and rollback
- Set and view Savepoints
  - Implicit savepoint created by the system when child workspace is created
  - Explicit savepoint created by a user
- Roll back changes since last explicit savepoint
- Resolve differences between any two workspaces or between two savepoints in a workspace
- Refresh an entire workspace, a table, or rows with data from the parent workspace. Refreshing a workspace may not succeed if there are conflicts.
- Merge all changes made in the workspace or changes made to a specific table.
- Set Privileges to access, create, delete, rollback and merge workspaces.

Workspace Metadata Views

- **Metadata views hold information about:**
  - **Version-enabled tables**
  - **Workspaces**
  - **Savepoints**
  - **Users, privileges**
  - **Locks, conflicts**
- **Views are read-only to users.**
- **Views are used to administer the Workspace environment and diagnose problems.**

ORACLE

Workspace Metadata Views

Workspace Manager creates and maintains metadata views to hold information that helps to manage the workspace environment and diagnose problems. These views are read-only to users.

Views that span the whole workspace environment are:

- `[DBA|USER|ALL]_WM_VERSIONED_TABLES` contain information about version-enabled tables.
- `[USER|ALL]_WM_MODIFIED_TABLES` contain information about version-enabled tables that have been modified.
- `[DBA|USER|ALL]_WORKSPACES` contain information about the workspaces a user owns or can access.
- `[DBA|USER|ALL]_WORKSPACE_SAVEPOINTS`, `[DBA|USER|ALL]_WORKSPACE_PRIVS` contain information about users' privileges.
- `USER_WM_PRIVS` contains information about the current user has in each workspace.
- `ROLE_WM_PRIVS` contains information about privileges that all roles granted to the current user have in each workspace.

Workspace Metadata Views (continued)

Views that span the whole workspace environment (continued):

- [USER|ALL]_WM_LOCKED_TABLES contains information about locks placed in the current workspace on rows in version-enabled tables.
- DBA_WORKSPACE_USERS contains information about user information for workspaces other than LIVE.
- [USER|ALL]_WM_RIC_INFO contains referential integrity constraints.
- [USER|ALL]_WM_TAB_TRIGGERS contains information about triggers defined on version-enabled tables.
- ALL_VERSION_HVIEW contains information about the version hierarchy.

Summary

In this lesson, you should have learned how to:

- **Identify the Workspace Manager role**
- **Version enable a table**
- **Disable workspace participation for a table**
- **Create and assign a workspace**
- **Understand Import and Export considerations**

ORACLE

Practice 19-1 Overview

This practice covers the following topics:

- **Version enabling a table**
- **Creating workspaces**
- **Using workspaces to make changes and view these changes**
- **Merging and dropping workspaces**
- **Version disabling a table**

ORACLE

20

Advanced Replication

Objectives

After completing this lesson, you should be able to do the following:

- **Add a new master site without quiescing**
- **Decide when to use row-level system change numbers**
- **Use new materialized view fast refresh abilities**
- **List which object-relational constructs can be replicated**
- **Configure job queue initialization parameters**
- **List miscellaneous Oracle9*i* changes**

ORACLE

Extended Availability of Replication Environment

You can add new masters to the replication group without impacting end users:

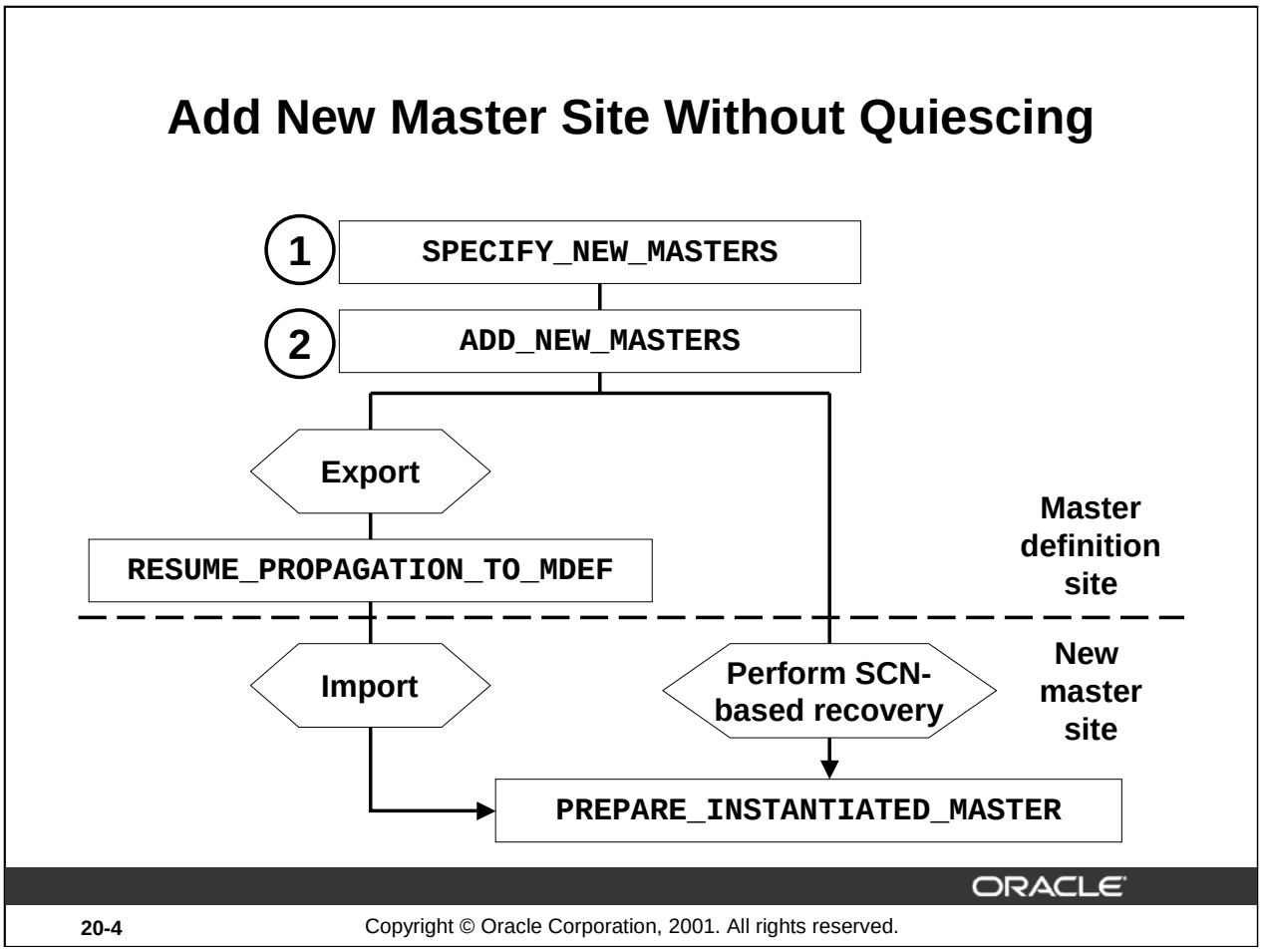
- **The replication group does not have to be quiesced**
- **Users can continue to execute DML on the existing masters**
- **Implemented using a multistep procedure**

ORACLE

Extended Availability of Replication Environment

Some operations in a replication environment require you to quiesce (stop all replication activity in) the related replicated groups. This can be a slow process if there are deferred transactions in the replication queues because you have to propagate these before you can quiesce the master group. Quiescing also prevents data manipulation language (DML) statements on the tables in the master group until you activate the group again.

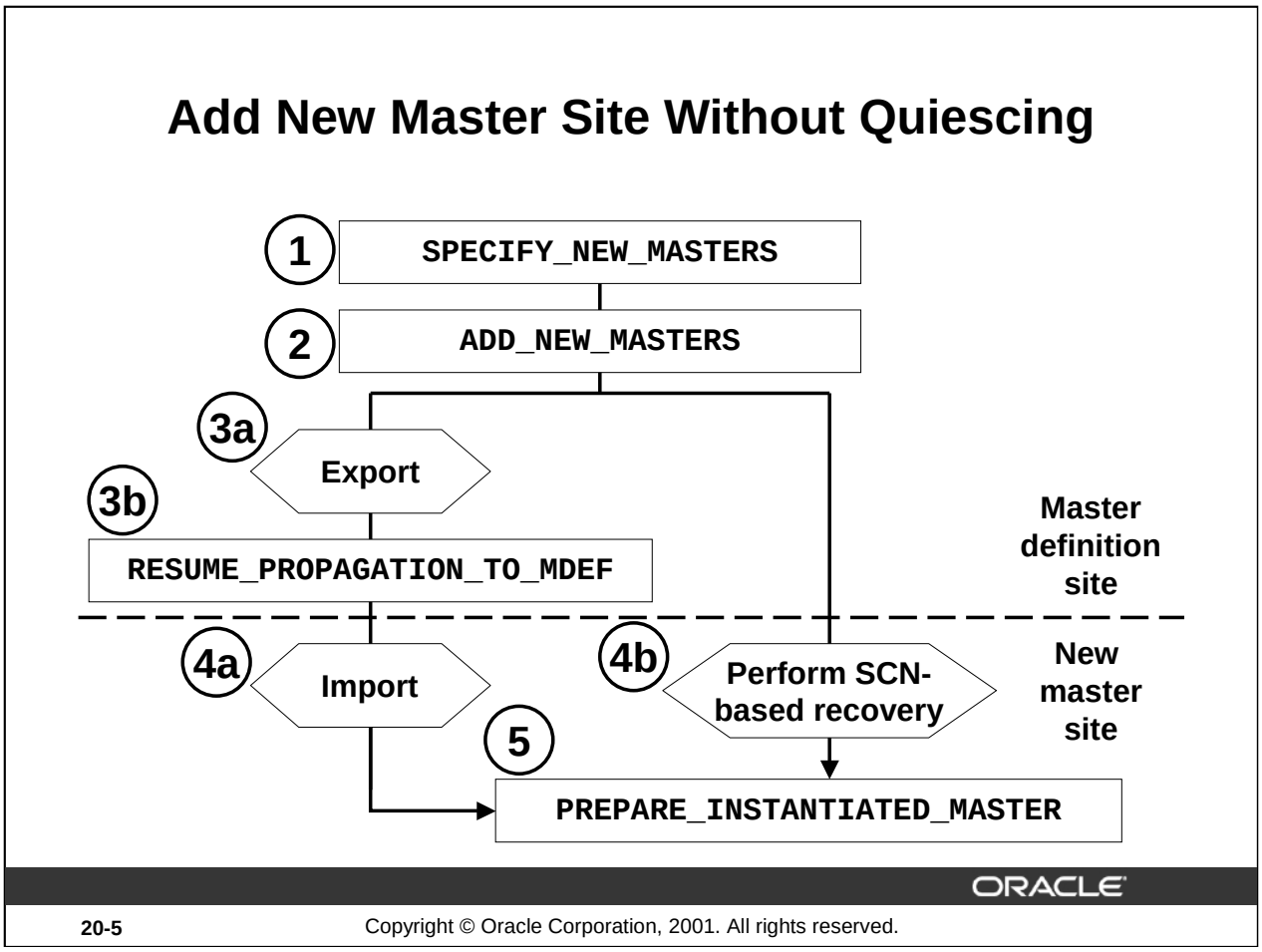
Recent releases have reduced the number of operations that require quiescing. In Oracle9*i*, you can now add databases to a master group without impacting your end users. Unlike in previous releases, your users can continue to execute DML on the replicated tables while you configure a new master site.



Adding a New Master Site Without Quiescing

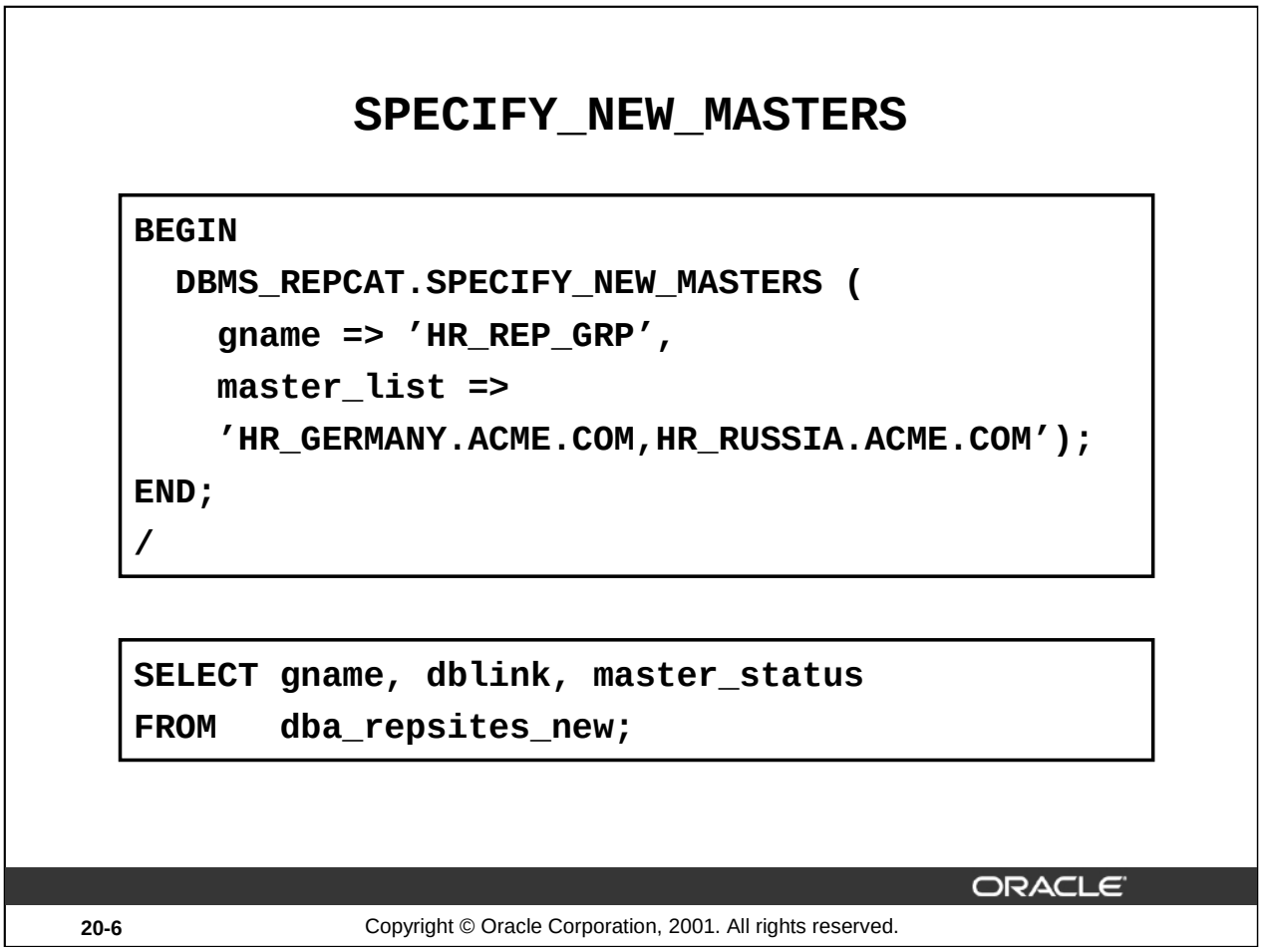
Use the following steps to add new masters without quiescing the existing master sites:

1. Specify your intention to add one or more new masters in an existing replication group with a new master definition site procedural call, SPECIFY_NEW_MASTERS. The new sites to be created are added to the view DBA_REPSITES_NEW. If you specify no new master sites, all sites previously listed in DBA_REPSITES_NEW will be removed.
2. After specifying all replication group extensions with `SPECIFY_NEW_MASTERS`, you then invoke a new master definition site procedural call, `ADD_NEW_MASTERS`, which does the following:
   - Adds new masters to the replication catalog at all existing masters and at all new masters to be instantiated with table-level export/import
   - Adds replicated objects to the replication catalog at new masters to be instantiated with table-level export/import
   - Disables, either partially or completely, propagation to each new master at each existing master, including the master definition site
   - Returns, to the calling session, the system change number (SCN) to which the new masters should be instantiated



Adding a New Master Site Without Quiescing (continued)

3. If you are using change-based recovery to instantiate all new masters, skip to step 4b. Otherwise, perform the next two operations:
   a. Perform a full database or table level export, specifying the SCN obtained in step 2 for the FLASHBACK_SCN export parameter.
   b. Call RESUME_PROPAGATION_TO_MDEF to indicate that the export has finished.
4. At each new master site, instantiate the database using one of the following two steps:
   a. Import the data.
   b. Perform change-based recovery to the SCN obtained in step 2.
5. After the database is instantiated, make sure that the correct database links exist between the new masters sites and the master definition site. If you are using the full export/import method for change-based recovery to add new masters, execute the PREPARE_INSTANTIATED_MASTER procedure to set the appropriate global_name for the new master and to remove (clear) the deferred transactions and error log from the new master site. For sites being added with object-level export/import, verify that all requests in the DBA_REPCATLOG view have completed successfully before executing PREPARE_INSTANTIATED_MASTER. In addition to the activities already described, the PREPARE_INSTANTIATED_MASTER procedure enables propagation of deferred transactions between all of your master sites.



SPECIFY_NEW_MASTERS

Use SPECIFY_NEW_MASTERS, a new routine in DBMS_REPCAT, to specify the master sites you intend to add to an existing replication group. This routine is called on the master definition site for the given replication group. The routine replaces any masters in the local DBA_REPSITES_NEW view for the given replication group with the list of masters specified.

In the example above, the routine is used to identify two new masters sites, HR_RUSSIA and HR_GERMANY, to be added to the replication group HR_REP_GRP. This query shows the results after executing the PL/SQL in the example:

```
SQL> SELECT gname, dblink, master_status
  2  FROM   dba_repsites_new;

GNAME          DBLINK               MASTER_STATUS
-------------- -------------------- -------------
HR_REP_GRP     HR_GERMANY.ACME.COM  READY
HR_REP_GRP     HR_RUSSIA.ACME.COM   READY
```

The replication group HR_REP_GRP maintains the READY status, so DML operations are still allowed on the tables within this replication group.

Note: If master_list is empty, all masters for the given replication group will be removed from the DBA_REPSITES_NEW view.

ADD_NEW_MASTERS

```
VARIABLE masterdef_flashback_scn VARCHAR2(15);
VARIABLE extension_id VARCHAR2(32);
BEGIN
 DBMS_REPCAT.ADD_NEW_MASTERS (
  export_required => TRUE,
  available_master_list => NULL,
  masterdef_flashback_scn =>
   :masterdef_flashback_scn,
  extension_id => :extension_id,
  break_trans_to_masterdef => TRUE,
  break_trans_to_new_masters => TRUE,
  percentage_for_catchup_new => 60,
  cycle_seconds_new => 300);
END;
```


ADD_NEW_MASTERS

The code in this example instantiates the new master site with the following options:

- Export/import is being used for at least one new master site (`export_required => TRUE`)
- Instantiate all new masters using either full export/import (if `export_required` is `TRUE`) or change-based recovery (`available_master_list => NULL`)
- Allow existing masters to propagate their deferred transactions to the master definition site for replication groups that are not adding master sites (`break_trans_to_masterdef => TRUE`)
- Allow existing masters to continue to propagate deferred transactions to the new master sites for replication groups that are not adding master sites (`break_trans_to_new_masters => TRUE`)
- Use 60 percent of propagation resources to catch up to new masters (`percentage_for_catchup_new => 60`)
- Allow a maximum of five minutes for extended replication groups to be propagated to new masters before switching to the unaffected replication group (`cycle_seconds_new => 300`)

ADD_NEW_MASTERS

```
PRINT masterdef_flashback_scn

MASTERDEF_FLASHBACK_SCN
-----------------------
3456871
```

```
$ exp system/manager FILE=fulldb_orc1.dmp
  FULL=y DIRECT=n GRANTS=y ROWS=y COMPRESS=y
  INDEXES=y CONSTRAINTS=y STATISTICS=compute
  LOG=export.log FLASHBACK_SCN='3456871'
```

ORACLE

ADD_NEW_MASTERS (continued)

You use the SQL*Plus `masterdef_flashback_scn` bind variable to display the SCN provided in the return variable after the procedure has executed, as shown in the example above. You can also view this SCN value by querying the `DBA_REPSITES_NEW` and `DBA_REPEXTENSIONS` data dictionary views:

```
SQL> SELECT flashback_scn
  2  FROM   dba_repextensions;

FLASHBACK_SCN
------------------------------
3456871
```

Using this information, you can perform an export consistent with this SCN by including the `FLASHBACK_SCN` option:

```
$ exp system/manager FILE=fulldb_orc1.dmp FULL=y DIRECT=n
  GRANTS=y ROWS=y COMPRESS=y INDEXES=y CONSTRAINTS=y
  STATISTICS=compute LOG=export.log FLASHBACK_SCN='3456871'
```

RESUME_PROPAGATION_TO_MDEF

```
SELECT extension_id
FROM   dba_repsites_new;
```

```
EXTENSION_ID
--------------------------------
75AFDBD77CD83C4EE034080020CAFF6B
```

```
DBMS_REPCAT.RESUME_PROPAGATION_TO_MDEF (
extension_id =>
'75AFDBD77CD83C4EE034080020CAFF6B');
```

ORACLE

RESUME_PROPAGATION_TO_MDEF

This procedure indicates that export has finished and that propagation to the master definition site, for both extended and unaffected replication groups existing at master sites, can be enabled. The EXTENSION_ID argument identifies the current pending request to add master databases to a master group without quiesce. You can find the value for EXTENSION_ID by querying the DBA_REPSITES_NEW and DBA_REPEXTENSIONS data dictionary views.

```
SQL> SELECT extension_id, dblink, master_status
  2  FROM   dba_repsites_new;
EXTENSION_ID                     DBLINK     MASTER_STATUS
-------------------------------- ---------- -------------
75AFDBD77CD83C4EE034080020CAFF6B REP3.WORLD READY
```

Perform Import or Change-Based Recovery

```
$ imp system/manager FILE=fulldb_orc1.dmp
  FULL=y BUFFER=30720 IGNORE=y GRANTS=y ROWS=y
  DESTROY=y COMMIT=y LOG=import.log
```

```
SQL> RECOVER DATABASE UNTIL CHANGE 3456871;
```

```
RMAN> run {
2>    SET UNTIL SCN = 3456871;
3>    DUPLICATE TARGET DATABASE TO hr_russia
4>    NOFILENAMECHECK;
5>    }
```

ORACLE

Perform Import or Change-based Recovery

Perform import or change-based recovery at each new master site. If you are using export/import, then complete the import of the database you exported at each new master site that is being added with the database export/import.

The following is an example of a full database import statement:

```
$ imp system/manager FILE=fulldb_orc1.dmp FULL=y BUFFER=30720
  IGNORE=y GRANTS=y ROWS=y DESTROY=y COMMIT=y LOG=import.log
```

If you are using change-based recovery, then perform the recovery using the SCN returned by the `MASTERDEF_FLASHBACK_SCN` parameter. You can also query the `DBA_REPEXTENSIONS` data dictionary view for the `MASTERDEF_FLASHBACK_SCN` value.

You can perform a change-based recovery in one of the following ways:

- Issue a SQL*Plus `RECOVER` command similar to the one shown in the example. See the *Oracle9i User-Managed Backup and Recovery Guide* for instructions.
- Use the Recovery Manager (RMAN) `DUPLICATE` command like the one shown in the example. See the *Oracle9i Recovery Manager Reference Guide* for instructions.

PREPARE_INSTANTIATED_MASTER

- **If you are using the full import or change-based recovery techniques, then:**
  - **Set `JOB_QUEUE_PROCESSES = 0` in new master's parameter file.**
  - **Ensure that the new master has the correct `global_name`.**
  - **Create database links to the master definition site.**
- **Execute the `PREPARE_INSTANTIATED_MASTER` procedure.**

```
DBMS_REPCAT.PREPARE_INSTANTIATED_MASTER (
extension_id => :ext_id);
```

ORACLE

PREPARE_INSTANTIATED_MASTER

This procedure enables the propagation of deferred transactions from other prepared master sites and existing master sites to the invocation master site. It also enables propagation of deferred transactions from the invocation master site to the other prepared new master sites and existing master sites.

Before executing this procedure, you must perform the following steps if you are using either the full database import or change-based recovery techniques:

- Set `JOB_QUEUE_PROCESSES = 0` in the new master's parameter initialization file. This must be done before the import or change-based recovery is performed.
- Ensure that the new master has the correct `global_name`, which must be the same name you specified when invoking `DBMS_REPCAT.SPECIFY_NEW_MASTERS` at the master definition site. Use the `ALTER DATABASE RENAME GLOBAL_NAME` command to change the global name if necessary.
- Create database links to the master definition site for the replication administrator's account.

When you execute the `PREPARE_INSTANTIATED_MASTER` procedure, the `EXTENSION_ID` argument is required. This argument identifies the current pending request to add master databases to a master group without quiesce. You can find the value for `EXTENSION_ID` by querying the `DBA_REPSITES_NEW` and `DBA_REPEXTENSIONS` data dictionary views.

When to Add New Masters Without Quiescing

- **You want to perform data manipulation language (DML) operations on replicated tables while the new master sites are being added.**
- **Each existing master site has enough free space to store a large unpropagated deferred transaction queue while the master sites are being added.**
- **You have a stable connection to the new master site, or your database meets the requirements for using Export/Import or SCN-based recovery.**
- **Your system does not meet any of the restrictions identified later.**

ORACLE

When to Add New Master Sites without Quiescing

As your replication environment expands, you may need to add new master sites to a master group. You can either add new master sites to a master group that is running normally or to a master group that is quiesced. If the master group is not quiesced, then users can perform data manipulation language (DML) operations on the data while the new master sites are being added. However, more administrative actions are required when adding new master sites if the master group is not quiesced.

When you add new master sites without quiescing the replication group, propagation of deferred transactions to the new master site is partially or completely disabled while the new master sites are being added. Therefore, make sure each existing master site has enough free space to store the largest unpropagated deferred transaction queue that you may encounter.

When using full database export/import and change-based recovery to add all of the replication groups at the master definition site to the new master sites, the following conditions apply:

- The new master sites cannot have any existing replication groups.
- The master definition site cannot have any materialized view groups.
- The master definition site must be the same for all of the master groups.
- The new master site must include all of the replication groups in the master definition site when the extension process is complete.

Restrictions on Adding New Masters Without Quiescing

ADD_NEW_MASTERS restrictions:

- **Synchronous replication is not supported.**
- **The master definition site cannot be relocated for any replication group being extended.**
- **The connection qualifier for all master group database links must be the same.**
- **The option to use serial or parallel propagation is only available when queue is disabled.**
- **Serial propagation is only supported when queue propagation is completely disabled.**
- **All master sites must use the Oracle9*i* release.**

ORACLE

Restrictions on Adding Masters without Quiescing

Restrictions on using the ADD_NEW_MASTERS routine include:

- Asynchronous replication, but not synchronous replication, is supported.
- If a replication group is being extended, the master definition site cannot be relocated.
- All database links for the affected master group must have the same connection qualifier or not use any connection qualifier.
- Parallel propagation (parallelism set to 1 or greater), but not serial propagation (parallelism equal to zero), is supported when queue propagation is not completely disabled. Both parallel and serial propagation are supported when queue propagation is completely disabled through procedure arguments.
- Once you begin adding a set of masters to replication groups with the same master definition site, you must wait until the new masters have been added before you can add another set of master sites to any of the affected master groups.
- The COMPATIBLE initialization parameter for all affected master sites must be set to "9.0" or higher.

If you need to remove the changes you made to a particular master site by the SPECIFY_NEW_MASTERS and ADD_NEW_MASTERS procedures, you should use the UNDO_ADD_NEW_MASTERS_REQUEST procedure.

Row-Level System Change Numbers

- **Dependent SCN is used to determine dependencies for parallel propagation.**
- **The default is to calculate the dependent SCN at the data block level.**
- **In Oracle9*i*, the dependencies can be narrowed to the row level.**

ORACLE

Row-Level System Change Numbers

When Advanced Replication enables parallel propagation, dependent SCNs are used to determine dependencies based on constraint definitions. In older releases, and as the default in Oracle9*i*, dependent SCNs are calculated at the block level. That is, every entry on a block is considered to have the same SCN, regardless of when the entry actually changed. This can result in false dependencies being defined for parallel propagation.

By allowing you to implement row level SCNs, Oracle9*i* can narrow dependencies from the block level to the row level. This reduces the number of false dependencies for parallel propagation which, in turn, can improve the replication throughput because more transactions can be streamed in parallel.

Features of Row-Level SCNs

- **A table can be created so that the commit SCN is stored in the head piece of each table row.**

```
CREATE TABLE master_rep … ROWDEPENDENCIES;
```

- **Each row has an SCN that:**
  - **Requires six bytes of additional storage per row**
  - **Is greater than or equal to the commit time of the last transaction that modified the row**
  - **Represents the cleanout time of the last transaction for each row**
  - **Is updated during delayed block clean out**

ORACLE

Features of Row-Level SCNs

The CREATE TABLE statement syntax includes a clause to control whether the table uses row-level SCNs:

```
CREATE TABLE ...   [NO]ROWDEPENDENCIES
CREATE CLUSTER ... [NO]ROWDEPENDENCIES
```

The row-level dependency feature and, therefore, the use of row level SCNs, applies only to tables and clusters that are created with the ROWDEPENDENCIES options. By default, segments are created without row-level SCNs. You cannot use the ALTER TABLE command to change between dependent and row-level SCNs in an existing segment.

When you use row-level SCNs, each row:

- Requires an additional six bytes to store the SCN information in the row header
- Contains an SCN that represents the clean out time of the last transaction for that row, that is, a commit time no earlier than the last transaction that modified the row
- Is updated with the row level SCN when the delayed block clean out occurs for a transaction

Parallel Propagation and Row-Level SCNs

- **When using parallel propagation, Oracle9*i* orders the execution of dependent transactions to preserve data integrity.**
- **The order of the transactions is determined by the SCN.**
- **More granular propagation occurs when you use row-level SCNs because rows can be ordered independently rather than based on a more generic, block-dependent SCN.**

ORACLE

Parallel Propagation and Row-Level SCNs

If you use row-level SCNs, you can maintain for each row that last commit time of the changes made to that row. Therefore, two rows changed by different transactions on the same block can have different row-level SCNs (depending of course on the commit times of those transactions), whereas the dependent SCN for parallel propagation would have been computed solely based on the block state. Using the fine-grained row level SCNs to calculate dependent SCNs can enable better dependency tracking between transactions in the replication queue.

Constraint SCNs

- **When constraints exist on a table, they create additional replication dependencies for the table.**
- **Constraint SCNs represent the highest commit SCN of all transactions that a particular row depends on.**
  - **Index entries for unique constraints**
  - **Referential constraints**
- **Using constraint SCNs helps to reduce false dependencies which can occur when using the default dependent SCN calculation method.**
- **Constraint SCNs are activated automatically when the table is defined with row level SCNs and require an extra six bytes of storage on each index leaf block.**

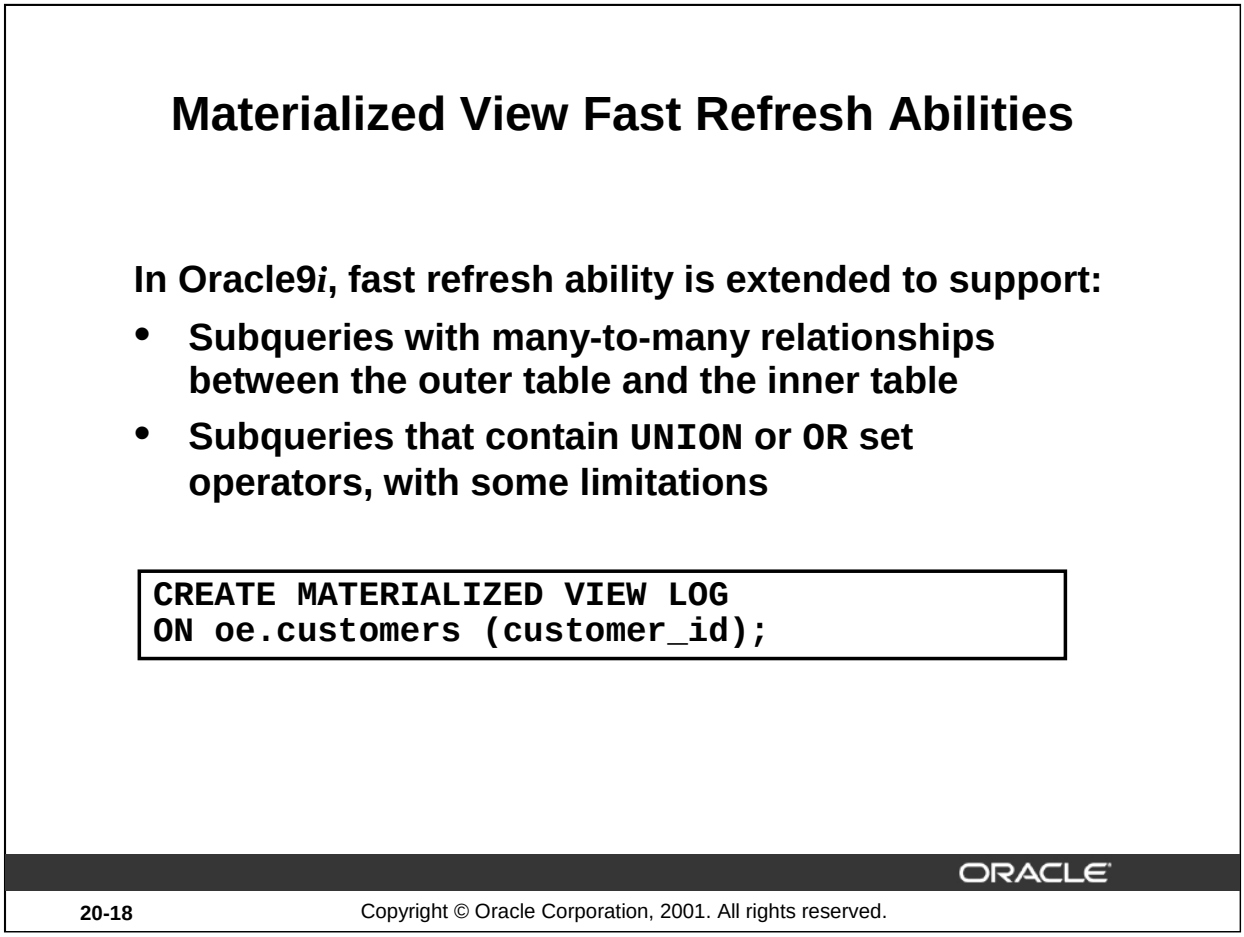
ORACLE

Constraint SCNs

A constraint SCN is the SCN that is larger than the commit times of all the transactions that a particular row operation depends on due to unique or referential integrity constraints. For example, if there is a unique index on a table, a successful row insertion is dependent on all the previously committed transactions that deleted the given unique key. The constraint SCN captures this dependency.

By default, a dependency SCN is computed using the last committed transaction in a block. Dependent SCNs can cause false dependencies to be tracked. For example, the insertion of a single key into an index leaf block will depend on all other transactions that commit in the same block, including the insertion of other keys.

False dependencies add unnecessary overhead when the rows from the table are replicated because unrelated row changes are also checked for the constraint dependencies. Constraint SCNs allow more exact tracking of when changes were made and which dependencies really need to be checked. Constraint SCNs are automatically activated when a table is defined with the `ROWDEPENDENCIES` option (or it is created in a cluster defined with this option). They add an extra six bytes of header information in each leaf block of any index defined on the table.



New Materialized View Fast Refresh Abilities

In older releases, if your materialized view (MV) contained a join subquery without a unique key constraint on the inner table, you could not update the MV with a fast refresh. This meant that you could not perform fast refreshes on your MV if it included a many-to-many join between the outer and inner tables. In Oracle9*i*, your MVs with subqueries can be fast refreshable even if the relation between the outer table and the inner table is many to many. You must store the join column in the materialized view log to enable this capability, as in the following statement:

```
SQL> CREATE MATERIALIZED VIEW LOG
  2  ON oe.customers (customer_id);
```

Primary key (PK) MVs containing `UNION` or `OR` set operators can be both fast refreshable and updatable in Oracle9*i*. However, because the changes are propagated based on row changes identified by the PK, it is necessary that a given row can only be derived from one table. This ensures that local updates are applied to the correct base table. Thus, the set operators must be over an identical list of columns in the `SELECT` clause, based on the same table.

Fast Refresh Example

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
 SELECT * FROM oe.orders@dbs1.acme.com o
 WHERE EXISTS
  (SELECT * FROM oe.customers@dbs1.acme.com c
   WHERE o.customer_id = c.customer_id
   AND c.credit_limit > 50)
 UNION
 SELECT * FROM oe.orders@dbs1.acme.com o
 WHERE EXISTS
  (SELECT * FROM oe.customers@dbs1.acme.com c
   WHERE o.customer_id = c.customer_id
   AND c.account_mgr_id = 30);
```

ORACLE

Fast Refresh Example

The statement creates the OE.ORDERS materialized view. The query selects all orders that are associated with the account manager whose ID number is 30, as well as all orders for customers who have a credit limit greater than 50.

To make this MV fast refreshable, you would have to create a materialized view log and enable filtering on the CUSTOMER_ID column of the OE.CUSTOMERS table as shown in the earlier example. Otherwise, the materialized view is fast refreshable because the UNION operates on the same columns (all of them) from the same table, OE.ORDERS.

Explain Materialized View

- **Sometimes it can be difficult to determine why a materialized view cannot fast refresh.**
- **The `DBMS_MVIEW.EXPLAIN_MVIEW` procedure:**
  - **Populates the `MV_CAPABILITIES_TABLE`**
  - **Explains why an existing or proposed materialized view is not fast refreshable**
- **The `MV_CAPABILITIES_TABLE` contains:**
  - **The capabilities of the MV**
  - **Whether or not each capability is possible**
  - **Why a capability is not possible**

ORACLE

Explain Materialized View

In previous releases, it could be difficult to determine which of several restrictions was preventing you from performing a fast refresh.

In Oracle9*i*, you can determine why a materialized view or query is not fast refreshable by executing the `DBMS_MVIEW.EXPLAIN_MVIEW` procedure to populate the `MV_CAPABILITIES_TABLE` table. You then query `MV_CAPABILITIES_TABLE` to see the refresh capabilities for your MV. The `POSSIBLE` column indicates whether you can perform the operation for each option listed in the `CAPABILITY_NAME` column:

`REFRESH`: Can do at least complete refresh

`REFRESH_FROM_LOG_AFTER_INSERT`: Can do fast refresh from an MV log or change capture table at least when change operations are restricted to `INSERT`

`REFRESH_FROM_LOG_AFTER_ANY`: Can do fast refresh from an MV log or change capture table after any combination of changes, `INSERT`, `UPDATE`, or `DELETE`

`REWRITE`: Can do at least full text match rewrite

`REWRITE_PARTIAL_TEXT_MATCH`: Can do at least full and partial text match rewrite

`REWRITE_GENERAL`: Can do all forms of rewrite

MV_CAPABILITIES_TABLE

```
Name                      Null? Type
------------------------- ----- --------------
STATEMENT_ID                    VARCHAR2(30)
MVOWNER                         VARCHAR2(30)
MVNAME                          VARCHAR2(30)
CAPABILITY_NAME                 VARCHAR2(30)
POSSIBLE                        CHAR(1)
RELATED_TEXT                    VARCHAR2(2000)
RELATED_NUM                     NUMBER
MSGNO                           NUMBER(38)
MSGTXT                          VARCHAR2(2000)
SEQ                             NUMBER
```

ORACLE

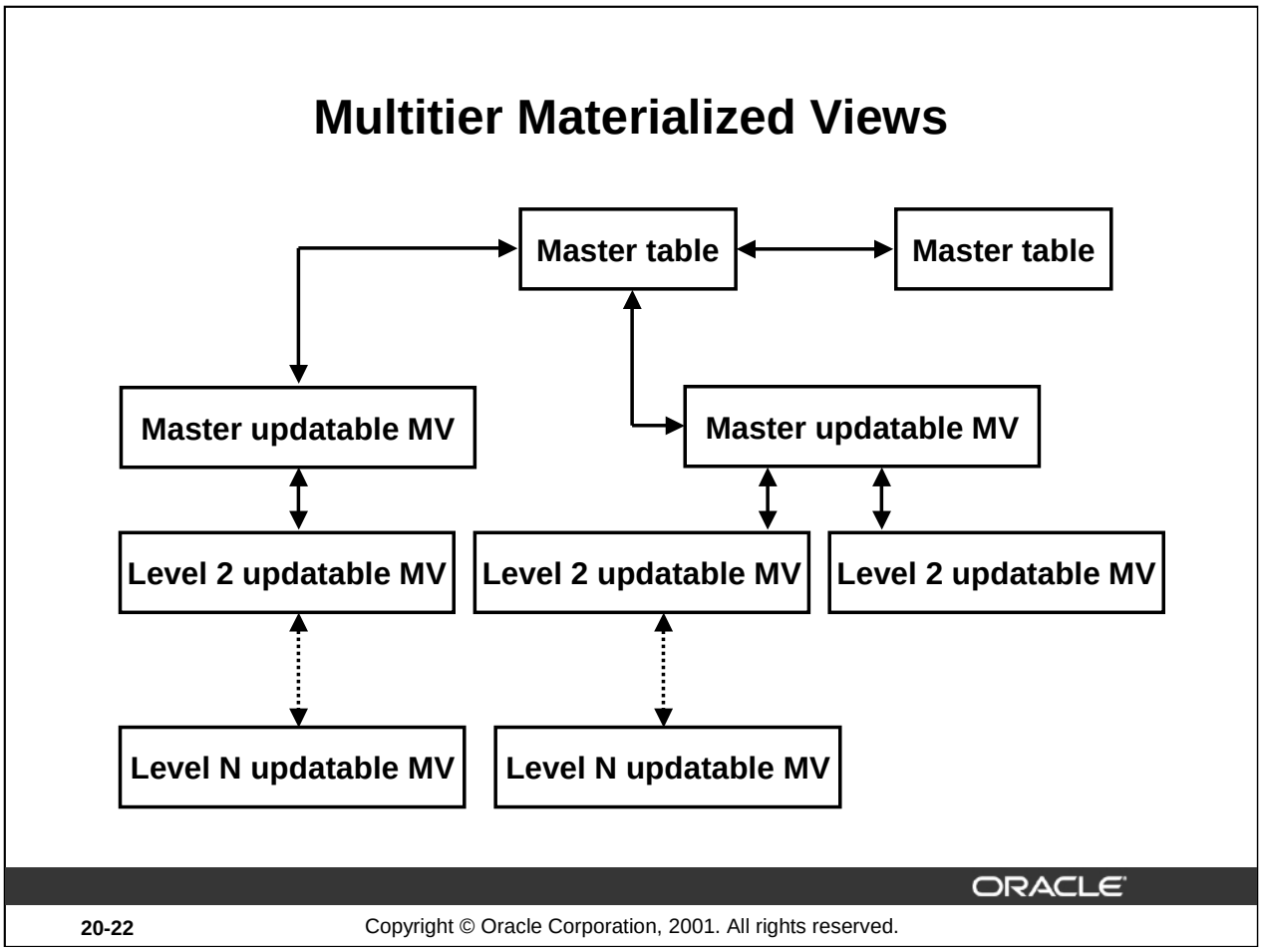
MV_CAPABILITIES_TABLE

The MV_CAPABILITIES_TABLE table is created from the utlxmv.sql script. The examples show the execution of the EXPLAIN_MVIEW procedure and a subsequent query against the contents of the MV_CAPABILITIES_TABLE table.

For example, to determine the capabilities of the oe.orders materialized view, enter:

```
SQL> EXECUTE DBMS_MVIEW.EXPLAIN_MVIEW ('oe.orders');
```

```
SQL> SELECT capability_name, possible
  2  ,      related_txt, msgtxt
  3  FROM   mv_capabilities_table;
```



Multitier Materialized Views

You use the multitier materialized view (MV) feature to create an MV based on another MV rather than on a base table. Oracle9*i* supports the creation of both read-only and updatable multitier materialized views.

Multitier updatable MVs are useful in scenarios where you do not need or want a complete instantiation of a table at every site because you do not need to see all the data or you may not have the storage capacity. Also, if you have limited network bandwidth, you may want to limit the amount of data transmitted between sites. Multitier MVs let you use lower bandwidth networks because you do not need to refresh from the master site each time.

For example, suppose a company is structured with three levels: international, national, and local offices. Many nodes at both the national and local office level are required. A possible solution for such a company would be to setup a master site at the headquarters office site, an updatable MV at each national subsidiary, and another updatable MV at each local office, based on the MV at the national level office.

In the graphic, the level one MVs are labeled “Master updatable MV.” A materialized view at any level can be a master MV and can have one or more MVs based on it. However, only an updatable master MV can have an updatable materialized view based on it. Because the level one updatable MV, or master MV, acts like a master for the level two updatable MV, a receiver user must be created at the level one updatable MV site. Similarly, a database link must be created between the propagator at the level two site to the receiver at the level one site.

Creating Multitier Materialized Views

```
CREATE MATERIALIZED VIEW regional_orders
   REFRESH FAST FOR UPDATE
AS SELECT * FROM oe.orders@hq.acme.com
   WHERE EXISTS
  (SELECT customer_id
   FROM   oe.customers@hq.acme.com c
   WHERE  c.cust_address.postal_code = '19555');
```

```
CREATE MATERIALIZED VIEW my_orders
   REFRESH FAST FOR UPDATE
AS SELECT * FROM oe.regional_orders@reg1.acme.com
   WHERE sales_rep_id = 205;
```

ORACLE

Creating Multitier Materialized Views

This example shows the commands used to create a level one materialized view, REGIONAL_ORDERS, and a level two materialized view, MY_ORDERS, based on REGIONAL_ORDERS.

The level one MV, created in the first statement, is based on the OE.ORDERS master table at headquarters, HQ.ACME.COM. The MV is created at the regional office REG1.ACME.COM and contains only orders whose postal code is 19555. The MVs at the regional office are in the headquarters MV group.

The second statement creates a level two MV based on the MV at the regional office, REG1.ACME.COM, containing only orders for postal code 19555. Additionally, this MV restricts the data to show only those orders assigned to the salesperson with the ID number 205.

Replication of Objects

Oracle9*i* gives you the ability to replicate the following:

- **Tables with:**
  - **Object columns**
  - **Collection type columns (nested tables, VARRAYs)**
  - **REF columns**
- **Object tables**
- **Object views and INSTEAD OF triggers**
- **Index types, operators, and domain indexes**

ORACLE

Replication of Objects

The replication of objects is a new feature of Oracle9*i*. Multimaster replication treats an object table like a regular relational table except that it preserves the object identifier (OID) of the object table (EOID) and the OID for each row.

For object views, any type information for VARRAYs, nested tables, and object types involved in the view definition, must be replicated explicitly.

You cannot specify alternate key columns for a replicated object table in the following circumstances:

- If the OID is system-generated because replication uses this OID column as the key
- If the OID is user-defined because replication uses the underlying primary key as the key

Note: When you replicate tables with nested tables, the storage tables of the nested tables are automatically replicated. You cannot explicitly replicate the storage tables themselves.

Replicating Objects with Materialized Views

- **Materialized Views for Oracle9*i* now allow you to replicate:**
  - **Column objects**
  - **Object views**
  - **REF columns**
  - **Collection columns**
- **To support replication of object tables, the materialized view logs can now record the object identifier (OID).**
- **All user-defined types required for MV creation must exist at the MV site prior to creation of the MV.**

ORACLE

Replicating Objects with Materialized Views

All of the object types which can be replicated using multimaster replication can also be replicated using materialized views.

Some new terminology for materialized views in Oracle9*i*:

- Materialized views created on tables which contain column-level objects are called Object-Relational materialized views.
- Materialized views created on object tables are called Object materialized views.
- Object materialized views created on relational tables (similar to Object Views), which are known as Derived Object materialized views, are not supported in the first release of Oracle9*i*.

The Materialized View Log construct has been updated to support objects. Object-relational materialized view logs will track either the ROWID or primary key of the master table. The primary key can consist of scalar object attributes. If you are creating an object materialized view (created against an object table), the MV log must also log the OID for each row. This is specified with the clause `WITH OBJECT ID`. The `WITH OBJECT ID` clause can be in addition to the primary key or ROWID.

```
SQL> CREATE MATERIALIZED VIEW LOG ON obj_table
  2  WITH OBJECT ID, PRIMARY KEY;
```

Monitoring and Managing Replication Environments

- **New dynamic performance views for Oracle9*i* replication environments:**
  - **V$MVREFRESH**
  - **V$REPLPROP**
  - **V$REPLQUEUE**
- **Enhancements to the Replication Management tool in Oracle9*i* include:**
  - **Reporting capabilities**
  - **Redesigned Deployment Template Wizard**
  - **Redesigned Template Script Generation Wizard**

ORACLE

Performance Monitoring in Replication Environments

The following dynamic performance views are new for performance monitoring in Oracle9*i* replication environments:

V$MVREFRESH: Contains information about local materialized views that are currently being refreshed. This information is visible only on the materialized view site.

V$REPLPROP: Contains information about the parallel propagation currently in progress at the replication site

V$REPLQUEUE: Contains statistics about the deferred transactions queue

In addition, columns related to performance monitoring are included in the DEFSCHEDULE data dictionary view in Oracle9*i*.

Replication Management Tool
The Replication Management tool in Oracle9*i* has been enhanced with the following features:

- Reporting capabilities
- Redesigned Deployment Template Wizard and Template Script Generation Wizard (formerly called the Offline Instantiation Wizard)

Job Queue Changes

- **The JOB_QUEUE_INTERVAL parameter is obsolete.**
- **The SNP background process is obsolete.**
- **JOB_QUEUE_PROCESSES determines the maximum number of processes (J000, J001, …) that can be started.**
  - **Support for multiple child processes**
  - **Less overhead than in previous releases**
- **You can dynamically alter the number of job queue processes used by the database.**

ORACLE

Job Queue Changes

Job queue processes process requests created by DBMS_JOB. Replication automatically creates jobs to handle tasks such as asynchronous data propagation at scheduled intervals and purging the deferred transaction queue.

The SNP process has been retired and replaced with a single-parent family of processes that work similarly to the parallel query family of processes. The head of the family is called the Coordinator, Job Queue (CJQ0). The child processes are named J*nnn* where *nnn* starts at 000 and increments by 1.

If the JOB_QUEUE_PROCESSES initialization parameter is greater than 0, the CJQ0 process remains active, scanning the job queue for available work, so the JOB_QUEUE_INTERVAL process is no longer needed. When CJQ0 finds work waiting to be done, it starts more Jnnn processes automatically, up to the limit specified by the initialization parameter JOB_QUEUE_PROCESSES. When a Jnnn process finishes execution of a job, it polls for another job to execute. If there are no jobs selected for execution, it enters an idle state, but wakes up periodically to poll again. If, after a predetermined number of tries, it still finds no jobs to execute, it terminates.

You can dynamically change the value of JOB_QUEUE_PROCESSES with a command such as the following:

```
SQL> ALTER SYSTEM SET JOB_QUEUE_PROCESSES = 12;
```

For more information, refer to the *Oracle9i: Databases Administrator's Guide* and the *Oracle9i: Replication Management API Reference* guide.

LONG to LOB Migration

- The ALTER TABLE command allows you to modify:
  - A LONG column to a CLOB
  - A LONG_RAW column to a BLOB
- Applications that perform DML on tables with migrated LONG columns do not have to be modified.
- Converting LONG columns to LOB columns means the data can now be replicated.
  - May increase bandwidth requirements
  - Tables at remote sites have to be regenerated
  - Materialized Views defined on a LOB migrated table have to be manually rebuilt

ORACLE

LONG to LOB Migration

In Oracle9*i*, you can use the ALTER TABLE command to change a LONG data type to a LOB data type or a LONG RAW data type to a BLOB data type.

The migration of a LONG column to a LOB column using the ALTER TABLE command on a replicated table cannot itself be replicated. This is because the LONG column has not been maintained by replication and may not be synchronized between sites. Once the LONG data has been synchronized between sites, then the modification of the LONG to the LOB column can be performed using replication's administrative engine. If a table you change this way is replicated, or has materialized views, you need to regenerate the replicated tables at the remote sites manually.

Because the converted LONG data can now be replicated, this may increase network bandwidth requirements, and you need to be aware of the impact this change may have for your replication system.

Changes for Related Oracle9*i* Features

- **The term *snapshot* is deprecated.**
  - **The name is no longer used in Oracle9*i* documentation.**
  - **The terms *materialized view* and *mview* have replaced the term in the documentation and the data dictionary.**
  - **SQL syntax currently retains the term *snapshot* for backwards compatibility.**
- **Support for new date and time data types**

ORACLE

Changes Related to Oracle9*i* Features

Over the past few releases, the term *snapshot*, the original name, has been used synonymously with materialized view. In Oracle9*i*, the old name has been dropped from documentation, view names, procedure names, and so on. You will find the term *materialized view*, used in text, or the term *mview*, used in names of views, packages, and procedures, in place of the term *snapshot*. For example, the name of the package previously known as `DBMS_SNAPSHOT` has changed to `DBMS_MVIEW`. For backwards compatibility, SQL commands in this release retain the keyword `SNAPSHOT`.

A number of new data types to support date, time, and intervals between two date and time stamps, are introduced in Oracle9*i*. Advanced replication supports these data types but their long names cannot be handled. Consequently, you need to use the abbreviations defined below when using the replication management API:

- TSTZ for Timestamp with Time Zone
- TSLTZ for Timestamp with Local Time Zone
- IYM for Interval Year to Month
- IDS for Interval Day to Second

Installation and Upgrade

- **Check the Replication section of the `README_rdbms.htm` file for special instructions on using the new Oracle9*i* features of Advanced Replication.**
- **The script `catrep.sql` is called from `catproc.sql` in Oracle9*i*.**
- **If upgrading to Oracle9*i* and your existing database uses National Character Set columns, additional steps may be needed during the upgrade to Oracle9*i*.**

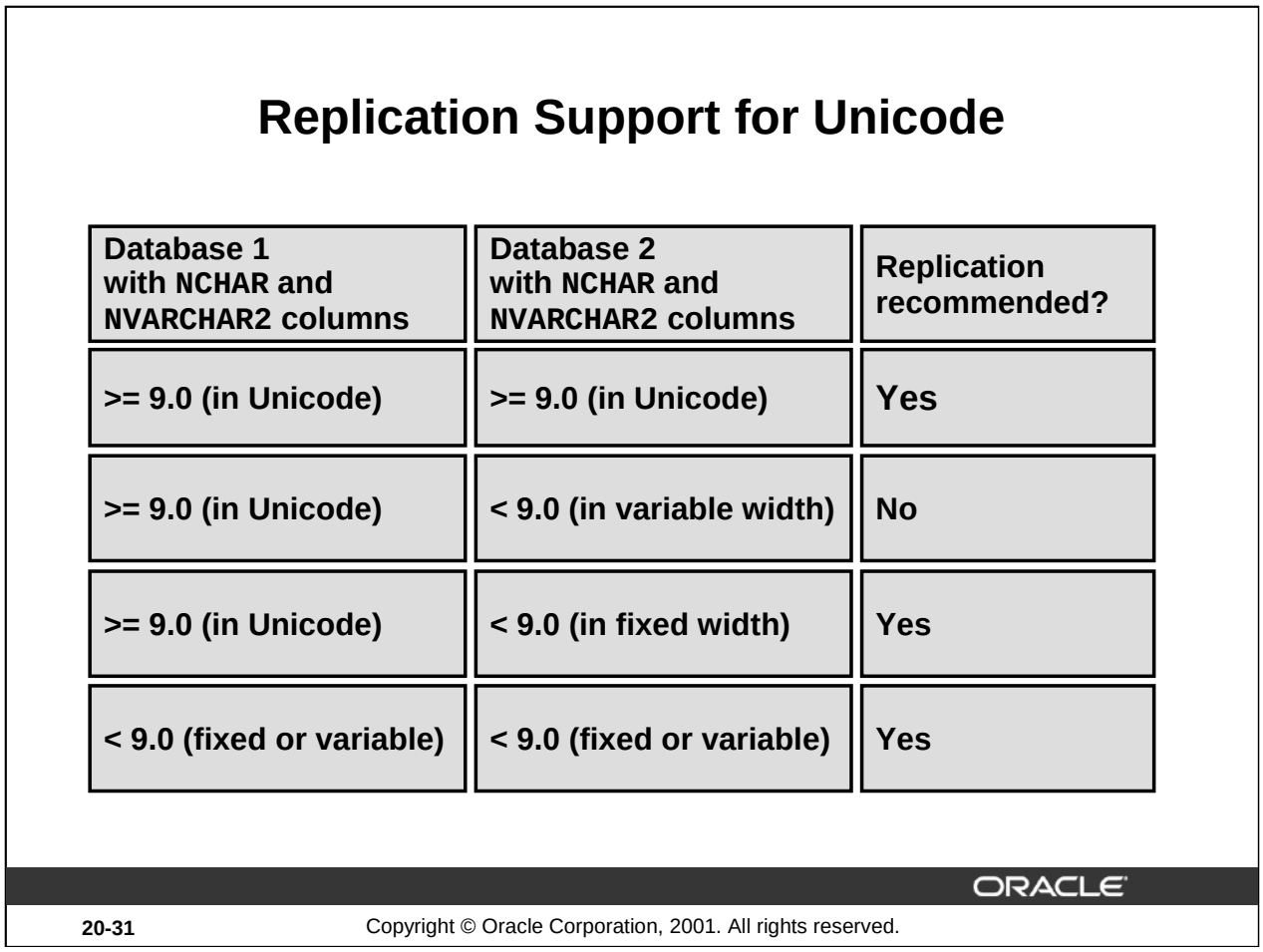
ORACLE

Installation and Upgrade

Before upgrading to Oracle9*i*, you should check for any special preparatory instructions documented in the section on Replication in the README file provided with the release. ($ORACLE_HOME/rdbms/doc/README_rdbms.htm)

After upgrading to Oracle9*i*, it is no longer necessary to run the CATREP.SQL script as in previous versions. CATREP.SQL is now executed as part of CATPROC.SQL script and no longer requires special handling after the upgrade.

The next page shows the recommended upgrades based on compatibility of various globalization support character sets, including Unicode. This, and similar information about upgrading with different column length semantics, is discussed in more detail in the *Oracle9i: Replication* manual.



Replication Support for Globalization Support Character Sets

Unicode is a universal encoded character set that enables you to store information from any language using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language. Unicode is supported in both multimaster and materialized view replication environments.

In Oracle9*i*, all columns specified as NCHAR or NVARCHAR2 data type are stored in Unicode format.

For both master sites and materialized view sites, replication is possible in an environment with different releases of Oracle using an NCHAR or NVARCHAR2 data type.

However, replication is not recommended when one of the replication sites is a release prior to Oracle9*i* and uses a variable width character set because, in this case, there is a possibility of data loss. The Oracle server does not detect an error when you set up replication between the two sites, but data loss may occur later. If data loss occurs, then an error is raised.

Summary

In this lesson, you should have learned how to:

- **Increase availability by adding new master site without quiescing**
- **Implement row level system change numbers**
- **Enable materialized view fast refresh abilities**
- **Create multitier materialized views**
- **Configure job queues with Oracle9*i* initialization parameters**

ORACLE

A
Practices

Practice 1-1: Security

Your instructor will give you the account details of your UNIX account (telnet login) and the SID of your instance.

1. Establish a command window (telnet) on your UNIX account. Connect to your instance in privileged mode, without any knowledge of any Oracle account.

2. Start up the instance. The parameter file you must use lies in your `$HOME/ADMIN/PFILE` directory. Specifying the init file is optional, because there is a link from the default init file name and location to your file.

3. Check that the `remote_login_passwordfile` parameter is set to `exclusive`, and the `orapwd` file exists (located in `$ORACLE_HOME/dbs`). Create a substitute DBA user, DBA2 using a password identification. Give the DBA2 privileged operator rights.

4. Start SQL*Plus and connect as the new user on one line. Stop and start the instance to test the privilege.

5. **Windows version:** Start SQL*Plus on the client (not the UNIX server) and connect as the new user. You can use either of the Windows based SQL*Plus, the SQL*Plus worksheet, or the command line version of SQL*Plus. Verify that shutting down the instance requires a privileged login, and that the DBA2 user only has SYSOPER rights.
   Note: Starting the instance would require a local copy of the parameter file in this configuration; `SPFILE` useage comes in a later chapter.

   UNIX version: Log in to the unprivileged UNIX account, which is not part of the DBA group. Start SQL*Plus and connect as the new user. Verify that shutting down the instance requires a privileged login, and that the DBA2 user only has SYSOPER rights.

Practice 2-1: Using Flashback

1. Connected as SYSTEM under SQL*Plus, ensure that you are in Automatic Undo Management mode. Then create a simple table T1 with only one NUMBER column called C.

2. Set the UNDO_RETENTION parameter to one hour. Determine the date and time of the system; then, using the DBMS_FLASHBACK package, determine the current SCN.

3. Wait for at least five minutes and then insert two different values inside table T1 (1 and 2 for example), and determine again the date, time, and current SCN as in the previous step.

4. Enable Flashback at the date and time calculated in step 2 and select data from table T1. Explain what you see on your screen.

5. Still connected in the same session under SYSTEM, try to insert a new row into table T1. What happens and why? Try to fix the problem without disconnecting from the current session, and insert a third row (for example: value 3).

6. Now, enable Flashback at the date and time calculated in step 3. Then select data from table T1. What happens, why, and what could have happened?

7. How would you get around the above possible problem?

8. Connected as SYSDBA, try to enable Flashback. What happens?

9. Connected as SYSTEM, drop table T1.

Practice 2-2: Use of Resumable Space Allocation and Import

1. Connected as SYSTEM under SQL*Plus, create a new dictionary managed tablespace called TEST1 with one 200 KB data file. Then, create a table TEST in tablespace TEST1 as a select of the PRODUCT_INFORMATION table of the OE schema.

2. Now, export the TEST table using the Export utility.

3. Drop tablespace TEST1 and its contents and re-create it but this time with only one 80 KB data file. Once done, create again the TEST table inside tablespace TEST1 with the same structure as TEST in step 1 but without the corresponding rows (create only the structure of the TEST table).

4. Import the table TEST by using the generated dump file from step 2.
   For this, use the Import utility with the following new parameters:
   - RESUMABLE=YES
   - RESUMABLE_NAME="TEST"
   - RESUMABLE_TIMEOUT=60

   What happens and why?

5. Do steps 3 and 4 again in the previous session. Create a separate session connected as SYSDBA, and before the timeout is reached during the import phase in the first session, query the dba_resumable view, and identify the problem. Wait for the timeout period to expire and query again the dba_resumable view. What is your conclusion?

6. From the second session, create a simple AFTER SUSPEND ON DATABASE trigger called ADD_SPACE that resizes the unique TEST1 tablespace data file to 10 KB.

7. From the first session, execute steps 3 and 4 again. What happens and why?

8. From the second session, recreate the ADD_SPACE trigger with a 2 MB file size increase instead of 10 KB.

9. From the first session, repeat steps 3 and 4. What happens and why?

10. Connected as SYSDBA, drop tablespace TEST1 and its data file as well as the add_space trigger.

Practice 3-1: LogMiner Enhancements

1. Prepare for using LogMiner by creating a flat file dictionary file called `dictionary.ora`. Check your `UTL_FILE_DIR` parameter and use the same directory.

2. Switch logfiles and note the current logfile name and the current time. This is the redo you will analyze.

3. Do some user activity. Create a table consisting of a few rows and columns selected from `HR.EMPLOYEES` (for example ID, last name, and salary in department 30). Add an extra column to `EMP30`. Update the new column with values. Roll back that update and do another update which you commit. Drop the table. Note the time.

4. Initialize LogMiner with the current redo file. Start mining between the two time points noted above.

5. Display the committed changes made by user HR on segment `EMP30`.

6. End the LogMiner session.

Practice 4-1: Backup and Recovery

1. Set the database into ARCHIVELOG mode, while in MOUNT mode. Enable automatic archiving to the `$HOME/ORADATA/ARCHIVE1` directory. Edit the parameter file as needed.

2. Create a tablespace, `RMANTEST`, of size 75 KB. Put this in the same directory as your other tablespaces. Exit SQL*Plus.

3. Start RMAN and connect to the database. Do not use a recovery catalog. Use the `show all` command to see the current configuration settings.

4. Configure a channel for back up, set the file location to your home directory, and use `<SID>_%U.bak` for the file name. Configure the snapshot control file to be written to your home directory as well. Keep the default name.

5. Back up your database. Exit from RMAN.

6. As SYSTEM/MANAGER in SQL*Plus make a copy of the `HR.EMPLOYEES` table in the `RMANTEST` tablespace calling the table `EMP`. Note how many rows are in the new table.

7. Connect as a privileged user and shut down the instance. Exit from SQL*Plus.

8. Rename the data file of the `RMANTEST` tablespace, simulating data file loss.

9. Log in to SQL*Plus and attempt to start the instance. Exit SQL*Plus.

10. Start RMAN, connect without a recovery catalog, and do a restore of the missing data file. Exit RMAN.

11. Use SQL*Plus to do a test recovery.

12. Because the test shows no errors, do the recovery. Test by examining the row count of the `EMP` table.

13. Clean up by dropping the tablespace including the data file.

Practice 7-1: Online Operations

1. Log in as SYSTEM and create a table EMP30 in the HR schema, which consists of all the columns of all employees in department 30 from the EMPLOYEES table. Add a primary key to the EMPLOYEE_ID column.

   The new company standard is that all key columns will now have the suffix of "_KEY" not "_ID" and all these key columns must be the first set of columns, with primary key always first. (The order of the key columns following the PK is not important.)
   The LAST_NAME column must be in upper case. There is a new column FULL_NAME which consists of the first name, a space, and then the last name. Here the last name must be in the same case as it was in the original employee table. The table must be partitioned on EMPLOYEE_KEY with three partitions; the high values for each are 116, 118, and MAXVALUE. To keep things simple do not worry about any other constraints that are on the original table. This redefinition of the table must be done online, without loss of data.

2. Create the interim table for the redefinition including above changes. Call it INT_EMP30 and it must be in the HR schema.

3. Verify that EMP30 can be redefined, then execute the redefinition.

4. Check that the work was completed as expected.

5. Clean up and drop the tables:

```
SQL> DROP TABLE HR.EMP30;
SQL> DROP TABLE HR.INT_EMP30;
```

Practice 7-2: New Parameter File

1. Start up the instance with the supplied text parameter file.

2. Now create the SPFILE from the PFILE used to start up the instance.

3. Shut down the instance and restart using the SPFILE. Again, examine the `SPFILE` parameter.

4. Examine the `OPEN_CURSORS` parameter and increase it with 50, but only change the SPFILE and add a comment of “Test Run 1.” Check to see that the change has only happened in the SPFILE.

5. In preparation for removing the SPFILE create a PFILE from the current SPFILE. Then shut down the instance, remove the SPFILE and delete the SPFILE. Restart the instance using the newly created PFILE.

Practice 8-1: Creating an External Table

1. Connected as SYSTEM under SQL*Plus, create a copy of the OE.ORDER_ITEMS table (only the first ten rows) in the SYSTEM schema.

2. Create an Oracle directory called EXT_TABLES pointing to your $HOME/STUDENT/LABS/ O/S directory (specify full path!). Then, create an external table called ORDER_ITEMS_EXT mapping to the previously created ORDER_ITEMS table. Use the two precreated files, order_items1.dat and order_items2.dat, stored in your $HOME/STUDENT/LABS directory as the LOCATION for the external table. Once done, verify that you can select from the created external table.

3. Use the two new dictionary views, DBA_EXTERNAL_TABLES and DBA_EXTERNAL_LOCATIONS, to verify the various characteristics of the newly created external table.

4. If you don't know the exact syntax you should use to create an external table, you can use SQL*Loader as follows.
   Under the precreated $HOME/STUDENT/LABS directory, you should find the two data files used in the previous steps plus a SQL*Loader control file called order_items.ctl. View the contents of the SQL*Loader control file. As you can see, it describes the ORDER_ITEMS table structure.

5. Execute the SQL*Loader utility under the SYSTEM account with the order_items.ctl control file as parameter. Use also the brand new external_table parameter with the GENERATE_ONLY value. This will generate the various external table commands inside the SQL*Loader log file.

6. Look at the generated SQL*Loader log file.

7. Connected as SYSTEM, drop the tables ORDER_ITEMS_EXT, ORDER_ITEMS and then drop directory EXT_TABLES.

Practice 8-2: Creating a List-Partitioned Table

1. Connected as SYSTEM under SQL*Plus, create the list partitioned table SALES_LIST using the lab_08_01.sql script.

2. Use the DBA_PART_TABLES and DBA_TAB_PARTITIONS dictionary views to identify the SALES_LIST table characteristics.

3. Reconstitute the complete CREATE TABLE SALES_LIST command using only the data dictionary.

4. Execute the lab_08_02.sql script in order to insert some rows into the SALES_LIST table. What happens and why?

5. Analyze table SALES_LIST with the COMPUTE STATISTICS option and then create the PLAN_TABLE into the SYSTEM account. Once done, explain the following statements (you should use the explain.sql script to print the result of each explain plan command):

```
SQL> select * from sales_list;
SQL> select * from sales_list where sales_state = 'a';
SQL> select * from sales_list where sales_state in ('a','e');
SQL> select * from sales_list where sales_state = 'a'
  2                               or sales_state = 'd';
SQL> select * from sales_list where sales_state > 'e';
SQL> select * from sales_list where sales_state < 'e';
```

   For each of the above statements, give your interpretation of the generated execution plans.

6. How would you prove that the Oracle server really performed partition pruning in the above six cases?

7. Connected as SYSTEM, and drop the sales_list table.

Practice 9-1: Using Automatic Free Space Management Segments

1. Connected as SYSTEM under SQL*Plus, create a new tablespace called BITMAP_SEGS with the following characteristics:
   - One 5 MB data file
   - Locally managed with uniform extents sizes of 100 KB
   - Containing only automatic free space management segments

   Then, query the DBA_TABLESPACES view in order to identify the different characteristics of each tablespace.

2. Drop the already existing TEST table and create a new table called TEST in the BITMAP_SEGS tablespace. This table should only have one NUMBER column called C, and only one extent. Verify the size of the newly created extent.

3. Execute the lab_09_01.sql script in order to insert rows into the TEST table. Once done, determine the number of rows stored in each block of the table.

4. Delete all rows of the TEST table having C < 70. Then, commit your changes. Once done, determine the number of rows stored in each block of the table. What do you observe?

5. Although this step is optional, it is recommended that you do it in order to be sure of the lab's result. Connect as SYSDBA and fix the data blocks bimap status of the test table by using the SEGMENT_FIX_STATUS procedure of the DBMS_REPAIR package. Connect again as SYSTEM and execute the lab_09_02.sql script in order to insert back rows into the TEST table. Once done, determine and note the number of rows stored in each block of the table. What do you observe?

6. Create a new tablespace called USERS2 with only one 5 MB data file and with manual segment space management set. Create a new table called TEST1 with the same structure as the TEST table but stored in the USERS2 tablespace. Then edit the lab_09_01.sql script and change all occurrence of the word "TEST" to the word "TEST1." Save the edited script and execute it. Once done, determine the number of rows stored in each block of the table. What do you observe?

7. Repeat step 4 but this time for the TEST1 table instead of the TEST table. What do you observe?

8. Edit the lab_09_02.sql script and change all occurrences of the word "TEST" to the word "TEST1." Save the edited script and execute it. Once done, determine the number of rows stored in each block of the table. What is your conclusion?

9. Connected as SYSTEM, drop the BITMAP_SEGS and USERS2 tablespaces. Remove also their corresponding data files.

Practice 10-1: Monitoring Indexes

1. The goal of this lab is to explain how to monitor index usage.
   First of all connect as user HR under SQL*Plus.

2. Create the PLAN_TABLE table using the `utlxplan.sql` script located under `$ORACLE_HOME/rdbms/admin`.

3. Create the EMP table as a simple select from the EMPLOYEES table.

4. Create the IEMAIL index on the EMAIL column of the EMP table.

5. Use the V$OBJECT_USAGE view in order to determine if there are indexes currently monitored in the HR account.

6. Monitor IEMAIL index usage.

7. Verify that the IEMAIL index is being monitored.

8. Connected as SYSDBA, try to verify that IEMAIL is being monitored. What is your conclusion?

9. Connect again under the HR account and delete all rows of the EMP table. Do not commit or roll back your changes.

10. Verify if the index was considered as being used by the previous command. Then roll back your changes and verify again the usage status of the IEMAIL index. What is your conclusion?

11. Execute a select statement against the EMP table that uses the IEMAIL index. Then verify the usage status of the IEMAIL index. What is your conclusion?

12. Verify that the contents of the V$OBJECT_USAGE view persist across an instance crash.

13. Drop the EMP table and analyze the impact on the monitored index. What is the consequence?

14. Re-create the same objects as in previous steps and activate monitoring usage again on the IEMAIL index.

15. Use the EXPLAIN PLAN command to analyze the query you executed in step 11. Again analyze monitoring usage on the IEMAIL index. What is your conclusion?

16. Turn off monitoring usage on the IEMAIL index and look at the V$OBJECT_USAGE view. What is your conclusion?

17. Drop the IEMAIL index and verify that the corresponding entry in the V$OBJECT_USAGE view has been deleted.

18. Re-create the IEMAIL index and activate its monitoring usage again. Then reexecute the query from step 11.

19. Turn off monitoring usage on the IEMAIL index, then activate it again. What are the consequences on the V$OBJECT_USAGE view?

    In the previous step, the MONITORING and USED columns were set to YES. Turning monitoring off updates *only* the MONITORING flag. Activating monitoring again resets both columns.

20. Connected as HR, drop the EMP table and the PLAN_TABLE table.

Practice 10-2: Cursor Sharing Enhancements

1. The goal of this lab is to explain the usage of the new cursor sharing optimization. Create the `PLAN_TABLE` under the `SYSTEM` account. Then, create a simple table `T` under the `SYSTEM` user in the `USERS` tablespace. It is enough to create a table with only one column (C) of `NUMBER` data type. Once done, insert the same value (`1111111111111111`) many times into this table. Once done, commit your modification (in order to insert the data you should use the `lab_09_01.sql` script located in your LABS directory). The idea is to generate data skew in the table.

2. In order to generate data skew in the table, insert a last row with a completely different value (`222222222222222`) from the previous ones. Commit this modification also. You are now left with an unbalanced repartition of the rows inside the `T` table.

3. Create a simple index called `IT` on the `C` column of the `T` table in the `USERS` tablespace. Then, gather statistics for table `T` with the `dbms_stats.gather_table_stats` procedure.

4. Explain the following query with the two different values residing inside the `T` table:
   SQL> `select count(*) from t where c = ...;`
   What is your conclusion?

5. Change the value of the `CURSOR_SHARING` initialization parameter for your session so that it uses the new possible value: `SIMILAR`. Once done, flush the contents of the shared pool.

6. Now, execute the same select statement as in step 4 with the two different values.

7. Identify in the shared pool the corresponding SQL texts generated for the execution of the two statements above. What happened and why?

8. In order for the optimizer to possibly find data skew in the `T` table, gather table statistics again and also generate a histogram with 75 buckets for column `C`. After that, flush the shared pool again.

9. Repeat steps 6 and 7. What is your conclusion?

10. How would you prove that two different optimizer plans were used by the Oracle server to execute the previous two queries?

11. Connect as `SYSTEM`, and drop table `T`.

Practice 13-1: Create OMF and Non-OMF Files in the Same Database

1. Under your HOME directory, create two new subdirectories called OMF1 and OMF2 respectively. Then connect as SYSDBA under SQL*Plus.

2. Try to create tablespace OMF1 with the following command:

   ```
   SQL> CREATE TABLESPACE omf1 DATAFILE SIZE 10M AUTOEXTEND OFF;
   ```

   What happens and why?

3. How would you modify your environment to allow the previous command to create OMF files in the $HOME/OMF1 directory?
   Create tablespace OMF1 and verify that an OMF file was created in the $HOME/OMF1 directory.

4. Now, change the DB_CREATE_FILE_DEST value to point to $HOME/OMF2 directory. Then drop tablespace OMF1. What happens?

5. How would you verify all this?

6. Create the same OMF1 tablespace again and verify that the Oracle server creates the corresponding OMF file in the $HOME/OMF2 directory.

7. In order to demonstrate how you can migrate an OMF file to a non-OMF file:

   - Take the OMF1 tablespace offline
   - Make an O/S copy of the created OMF file in the same directory. Call it omf1.dbf
   - Use the appropriate ALTER TABLESPACE command in order to rename the OMF data file to the newly created file
   - Bring the OMF1 tablespace back online

   What happens and why?

8. How would you drop tablespace OMF1, including its data file, at the O/S level?

Practice 13-2: Using the Default Temporary Tablespace Assignment

1. Switch the database default temporary tablespace to SYSTEM. How would you determine the current default temporary tablespace used in your database?

2. Create a user without assigning a temporary tablespace. Then, verify that this user was automatically assigned the previous default temporary tablespace.

3. Now, change the default temporary tablespace for your database to the TEMP tablespace and verify that the previously created user inherits the new database default temporary tablespace.

4. Create a tablespace called DEF1. Make sure that this tablespace is dictionary managed with a file size of 10 MB. Once done, make it the database default temporary tablespace. What happens and why?

   Answer: Because the DEF1 tablespace is not defined as TEMPORARY, it is not possible to use it as the database default temporary tablespace.

5. How would you modify this situation in order to make the DEF1 tablespace the database default temporary tablespace?

6. Once tablespace DEF1 is the database default temporary tablespace, how would you drop tablespace DEF1?

Practice 14-1: Maintaining Automatic Undo Management Tablespaces

1. Connected as SYSTEM under SQL*Plus, execute the lab_14_01.sql script in order to create some tables into the SYSTEM account. It is assumed that the USERS tablespace exists and has at least 1 MB free space. After executing the script, change the retention period of your instance to one hour.

2. Create a new undo tablespace called UNDO02 containing only one 60 KB data file. What happens and why?

3. Repeat the previous step but this time with a data file size of 80 KB instead of 60 KB.

4. Determine how many undo segments are currently created. Also, determine their current status. Then, change the active undo tablespace to be the UNDO02 tablesapce. Once done, try to delete the SYSTEM.T table. What happens and why?

5. Now switch back to the UNDO1 tablespace as the active undo tablespace. Then drop and re-create the UNDO02 tablespace but this time with a 400 KB data file. Once done, determine the number of undo segments and their statuses. What is your conclusion?

6. Change the active undo tablespace to be the UNDO02 tablespace. Then, look at the undo segments statuses. What happened?

7. Execute the lab_14_02.sql script in order to delete rows as well as creating and populating one additional table in the SYSTEM schema. This is done to get assigned to a transaction table in the UNDO02 tablespace. Using V$TRANSACTION, check to which transaction table you were assigned. Once done, commit your changes.

8. Determine the number of extents used by each undo segment in both undo tablespaces (UNDO02, and UNDO1). What do you observe?

9. Delete table T1 and find out which transaction table was assigned to this new transaction. Once done, try to delete table T3. What happens and why?

10. Reexecute the query you used in step 8. What is your conclusion?

11. Commit your modifications. Then, check the number of rows currently stored inside table T3. Once done, insert one new row inside table T3 and do not commit your modification.

12. From a second session connected as SYSTEM, switch the active undo tablespace to UNDO1, and then look at all undo segment statuses. What is your conclusion?

13. From the first session, commit your changes.

14. Still in the first session, check the undo segments status many times during five minutes. Once done, drop tablespace UNDO02 and its datafile. What are your conclusions?

15. Connected as SYSTEM, drop the tables T, T1, T2, and T3.

Practice 14-2: Transporting a Tablespace with Nondefault Block Size

1. Connected as SYSDBA under SQL*Plus, determine the default block size used for this database and also the various parameters used to configure the buffer cache of this instance.

2. Drop tablespace TEST1, including its data files, if it already exists. Determine what is inside the $HOME/STUDENT/LABS directory. Then, copy the test1.dbf file into the $HOME/ORADATA/u04 directory. Also, drop the SYSTEM.T table.

3. Try to plug the TEST1 tablespace using the plug.dmp export dump file. What happens and why?

4. How would you get around this problem?

5. After fixing the problem, plug in tablespace TEST1 using the same procedure as in step 3.

6. Verify that you can access table SYSTEM.T and look at the DBA_TABLESPACES view to see the characteristics of the plugged tablespace TEST1.

7. Connected as SYSDBA, drop tablespace TEST1 including its data file.

Practice 17-1: ANSI/ISO SQL:1999

1. With the natural join, you do not need to specify any join predicate or join columns. What happens if you apply the natural join to two tables that do not have any common column names?
   Think about this first, then connect to the HR schema, and try the following statement:

   ```
   SQL> select * from regions NATURAL JOIN jobs;
   ```

2. What is the difference between the cross join and the natural join, if two tables do not have any column names in common? Change NATURAL into CROSS in the previous example and compare the results.

3. Write a six-table join by using the USING and ON syntax, to show the last name, department name, city, and region name for all employees reporting to Steven King.

4. Look at the example of the WIDTH_BUCKET function (on page 17-25):

   ```
   SQL> select last_name, salary
     2  ,      WIDTH_BUCKET(salary,3000,13000,5)
     3  from   employees;
   ```

   Rewrite this query as a searched CASE statement.

5. Write a query to retrieve the average salary, excluding any employees that have a salary of 2500; try to find a solution without a WHERE clause, using the NULLIF function.
6. Write a single query, using scalar subqueries, to return the current system date if you have more employees than jobs.

Practice 17-2: Other SQL Enhancements

1. Look at the `JOB_HISTORY` table, and check the primary key constraint and the associated index. Drop the primary key constraint, and make sure that the associated index is not dropped. Restore the original situation by creating the primary key constraint.

   For the remaining steps of this practice, you need two SQL*Plus sessions connected to the HR schema (referred to as session A and session B respecively).

2. a. Make sure that you don't have an index on the `department_id` column of the `EMPLOYEES` table.

   b. From session A, delete any department from the `DEPARTMENTS` table without issuing a commit; to avoid foreign key violations, choose a department without employees. Save the delete statement, because you will need it again in the next exercise.

   c. From the same session A, query `v$locked_object` (joined with the `user_objects` view to display the object name) to see which locks are held by your session. In Oracle8*i* you would see at least an additional shared lock on the `EMPLOYEES` table, because you do not have an index on the foreign key column. Save the query for the next exercise.

   d. From session B, try to update any employee; as you see, this is possible. Save this update statement in a script file too.

   e. Roll back the changes you made in both sessions (but stay connected.)

3. To show that the shared lock is needed, you repeat the previous exercise in the opposite order. You can use the three SQL statements you saved during the previous exercise.

   a. From session B, update any employee; do not commit.

   b. From session B, query the data dictionary for locks; note that you have an exclusive row lock on the `EMPLOYEES` table.

   c. From session A, try to delete the same department again.
   Now you are unable to delete any department, because the exclusive row lock in session B causes the shared lock request from session A to wait.

   d. Roll back the changes you made in both sessions (but stay connected.)

4. From session A, update the salary of any employee without issuing a commit; then, try to select the same row for update from the other session. First try the default `FOR UPDATE` clause; then, try to specify a wait time of five seconds.

   Roll back the changes you made.

Practice 18-1: Globalization Support

1. Log on as user HR. Create a table TIMES, with the following four columns:
   - Column TS, data type TIMESTAMP
   - Column TSZ, data type TIMESTAMP WITH TIME ZONE
   - Column TLZ, data type TIMESTAMP WITH LOCAL TIME ZONE
   - Column OLDTIME, data type DATE

2. Check your database timezone. Check and possibly adjust your session timezone to be 'Europe/London.'

3. Determine the hour offset for this region.

4. Insert a row that populates all columns in the table. For the time component use one quarter of a second after 10 a.m. Specify the CET timezone where appropiate.

5. Alter your session timezone to be 'America/LosAngeles' and display the values. Explain what you see.

6. Drop the TIMES table.

Practice 19-1: Workspaces

1. Connected as SYSDBA under SQL*Plus, create a user called WM_DEVELOPER.

2. Grant CONNECT, RESOURCE roles to WM_DEVELOPER. Also directly grant the CREATE TABLE privilege to WM_DEVELOPER.

3. Grant the WM-specific privileges (with grant_option = YES) to WM_DEVELOPER. Specifically:
   ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, CREATE_ANY_WORKSPACE, REMOVE_ANY_WORKSPACE, and ROLLBACK_ANY_WORKSPACE.

4. Connected as WM_DEVELOPER, create a table for the annual marketing budget for several cola (soft drink) markets in a given geography (such as a city or a state). Each row will contain budget data for a specific cola.

   Note: This table does not reflect recommended database design. For example, a manager ID should be used, not a name. It is deliberately oversimplified for purposes of illustration. Budget is in Millions. In order to help you create this table, simply execute the lab_19_01.sql script located in your LABS directory.

5. Version-enable the table. Specify the hist option of VIEW_WO_OVERWRITE so that the COLA_MARKETING_BUDGET_HIST view contains complete history information. Once done, populate the table with some rows using the lab_19_02.sql script locate in your LABS directory.

6. Now create workspaces for the following scenario: a major marketing focus in the cola_b area. Managers and budget amounts for each market can change, but the total marketing budget cannot grow. The B_focus_1 scenario features a manager with more expensive plans (which means more money taken from other areas' budgets). The B_focus_2 scenario features a manager with less expensive plans (which means less money taken from other areas' budgets). Two workspaces (B_focus_1 and B_focus_2) are created as child workspaces of the LIVE database workspace.

7. Enter the B_focus_1 workspace and change the cola_b manager to Beasley and raise the cola_b budget amount by 1.5 to bring it to 3.0. Reduce all other area budget amounts by 0.5 to stay within the overall budget. In order to make the previous changes, you can use the lab_19_03.sql script.

8. Enter the B_focus_2 workspace and change the cola_b manager to Burton and raise the cola_b budget amount by 0.5 to bring it to 2.0. Reduce only the cola_d amount by 0.5 to stay within the overall budget. You can use the lab_19_04.sql script in order to make the changes. Once done, select every column from the COLA_MARKETING_BUDGET table.

9. Assume that you have decided to adopt the scenario of the B_focus_2 workspace using that workspace's current values; Go to the LIVE workspace, and remove the B_focus_1 workspace. Once done, apply changes in the second workspace to the LIVE database

workspace. Note that the workspace is removed by default after `MergeWorkspace`. Once done, select every column from the `COLA_MARKETING_BUDGET` table.

10. Disable versioning on the table because you are finished testing scenarios. Also, users with version-enabled tables cannot be dropped, in case you want to drop the `WM_DEVELOPER` user. Set the `force` parameter to `TRUE` if you want to force the disabling even if changes were made in a non-`LIVE` workspace. Also, remove the `B_focus_2` workspace.

11. Connected as `SYSDBA`, drop user `WM_DEVELOPER`.

B
Solutions

Practice 1-1 Solution: Security

Your instructor will give you the account details of your UNIX account (telnet login) and the SID of your instance.

1. Establish a command window (telnet) on your UNIX account. Connect to your instance in privileged mode, without any knowledge of any Oracle account.

```
$# Logged into an account in the DBA group
$ sqlplus
SQL*Plus: Release 9.0.1.0.0 – Production
(c) Copyright 2000 Oracle Corporation.  All rights reserved.

Enter user-name: / AS SYSDBA
Connected to an idle instance.
```

2. Start up the instance. The parameter file you must use lies in your `$HOME/ADMIN/PFILE` directory. Specifying the init file is optional, because there is a link from the default init file name and location to your file.

```
SQL> STARTUP PFILE=$HOME/ADMIN/PFILE/initU01.ora
ORACLE instance started.

Total System Global Area   93094040 bytes
Fixed Size                   278680 bytes
Variable Size              58720256 bytes
Database Buffers           33554432 bytes
Redo Buffers                 540672 bytes
Database mounted.
Database opened.
```

3. Check that the `remote_login_passwordfile` parameter is set to `exclusive`, and the `orapwd` file exists (located in `$ORACLE_HOME/dbs`). Create a substitute DBA user, DBA2 using a password identification. Give the DBA2 privileged operator rights.

```
SQL> SHOW PARAMETER remote_login
NAME                           TYPE     VALUE
------------------------------ -------- ---------
remote_login_passwordfile      string   EXCLUSIVE

SQL> HOST ls $ORACLE_HOME/dbs

orapwdU01

SQL> CREATE USER dba2 IDENTIFIED BY dba2;
User created.

SQL> GRANT CONNECT, SYSOPER TO dba2;
Grant succeeded.
```

4. Start SQL*Plus and connect as the new user on one line. Stop and start the instance to test the privilege.

```
$ sqlplus "dba2/dba2 as sysoper"
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> SHUTDOWN IMMEDIATE
Database closed.

SQL> STARTUP PFILE=$HOME/ADMIN/PFILE/initU01.ora
ORACLE instance started.
Total System Global Area   93094040 bytes ...
```

Note: As you are logged into a DBA account, it does not matter what you type as the user and password name, as long as you add `as sysdba` to the connection string. Anything other than above would yield an error for an account not in the DBA-group, or a network connection, as shown below.

5. **Windows version:** Start SQL*Plus on the client (not the UNIX server) and connect as the new user. You can use either of the Windows based SQL*Plus, the SQL*Plus worksheet, or the command line version of SQL*Plus. Verify that shutting down the instance requires a privileged login, and that the DBA2 user only has SYSOPER rights.
   Note: Starting the instance would require a local copy of the parameter file in this configuration; `SPFILE` useage comes in a later chapter.

```
C:\temp>sqlplus dba2/dba2@sun_U01
SQL*Plus: Release 9.0.1.0.0 - Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> SHUTDOWN IMMEDIATE
ORA-01031: insufficient privileges

SQL> CONNECT dba2/dba2@sun_U01 AS SYSDBA
ERROR: ORA-01031: insufficient privileges

SQL> CONNECT dba2/dba2@sun_U01 AS SYSOPER
Connected.

SQL> SHUTDOWN IMMEDIATE
Database closed.
Database dismounted.
ORACLE instance shut down.
```

5. **UNIX version:** Log in to the unprivileged UNIX account, which is not part of the DBA group. Start SQL*Plus and connect as the new user. Verify that shutting down the instance requires a privileged login, and that the DBA2 user only has SYSOPER rights.

```
$ sqlplus dba2/dba2
SQL*Plus: Release 9.0.1.0.0 - Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> SHUTDOWN IMMEDIATE
ORA-01031: insufficient privileges

SQL> CONNECT dba2/dba2@sun_U01 AS SYSDBA
ERROR: ORA-01031: insufficient privileges

SQL> CONNECT dba2/dba2@sun_U01 AS SYSOPER
Connected.

SQL> SHUTDOWN IMMEDIATE
Database closed.
Database dismounted.
ORACLE instance shut down.
```

Practice 2-1 Solution: Using Flashback

1. Connected as SYSTEM under SQL*Plus, ensure that you are in Automatic Undo Management mode. Then create a simple table T1 with only one NUMBER column called C.

```
$ sqlplus system/manager
SQL*Plus: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> show parameter undo

NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------
undo_management                      string      AUTO
undo_retention                       integer     900
undo_suppress_errors                 boolean     FALSE
undo_tablespace                      string      UNDO1

SQL> create table t1(c number);
Table created.
```

2. Set the UNDO_RETENTION parameter to one hour. Determine the date and time of the system; then, using the DBMS_FLASHBACK package, determine the current SCN.

```
SQL> alter system set undo_retention = 3600;
System altered.

SQL> select to_char(sysdate,'DD-MON-YYYY:HH24:MI:SS') from dual;

TO_CHAR(SYSDATE,'DD-
--------------------
26-JUL-2001:05:32:56

SQL> select dbms_flashback.get_system_change_number from dual;

GET_SYSTEM_CHANGE_NUMBER
------------------------
                  205050
```

3. Wait for at least five minutes and then insert two different values inside table T1 (1 and 2 for example), and determine again the date, time, and current SCN as in the previous step.

```
SQL> insert into t1 values(1);
1 row created.

SQL> insert into t1 values(2);
1 row created.

SQL> commit;
Commit complete.

SQL> select to_char(sysdate,'DD-MON-YYYY:HH24:MI:SS') from dual;

TO_CHAR(SYSDATE,'DD-
--------------------
```

```
26-JUL-2001:05:33:17

SQL> select dbms_flashback.get_system_change_number from dual;

GET_SYSTEM_CHANGE_NUMBER
------------------------
                  205056
```

4. Enable Flashback at the date and time calculated in step 2 and select data from table T1. Explain what you see on your screen.

   Answer: Although table T1 currently contains two rows, the date and time calculated at step 2 correspond to the time where T1 was empty. Thus, after enabling Flashback at that time, a SELECT * FROM T1 will return no rows. If you did not wait five minutes before inserting T1 rows in step 3, you might see the current result (both rows) due to the SCN/timestamp mapping.

```
SQL> execute dbms_flashback.enable_at_time(-
>    TO_TIMESTAMP('26-JUL-2001:05:32:56', 'DD-MON-YYYY:HH24:MI:SS'));
PL/SQL procedure successfully completed.

SQL> select * from t1;
no rows selected

-- Or:

   C
----
   1
   2
```

5. Still connected in the same session under SYSTEM, try to insert a new row into table T1. What happens and why? Try to fix the problem without disconnecting from the current session, and insert a third row (for example: value 3).

   Answer: Because your session is still in Flashback mode, it is not possible to modify data. In order to insert a new row into table T1, you need to first disable Flashback.

```
SQL> insert into t1 values(3);
insert into t1 values(3)
            *
ERROR at line 1:
ORA-08182: operation not supported while in Flashback mode

SQL> commit;
Commit complete.

SQL> insert into t1 values(3);
insert into t1 values(3)
*
ERROR at line 1:
ORA-08182: operation not supported while in Flashback mode
```

```
SQL> execute dbms_flashback.disable;
PL/SQL procedure successfully completed.

SQL> insert into t1 values(3);
1 row created.

SQL> commit;
Commit complete.
```

6. Now, enable Flashback at the date and time calculated in step 3. Then select data from table `T1`. What happens, why, and what could have happened?

   Answer: You may see two possible outcomes from the select statement. This depends on the time to SCN correspondence `SMON` keeps track in the `smon_scn_time table`. Generally, you will observe that each time interval is about five minutes. Thus, depending on the time you selected in previous steps, it is possible that there are more than five minutes between the time you created table `T1` and the time you committed the inserts. If this is the case, you should now see the two inserted rows from step 3. If it is not the case, you should see no row selected.

```
SQL> execute dbms_flashback.enable_at_time(-
>    TO_TIMESTAMP('26-JUL-2001:05:33:17', 'DD-MON-YYYY:HH24:MI:SS'));
PL/SQL procedure successfully completed.

SQL> select * from t1;

   C
----
   1
   2

-- Or:

no rows selected
```

7. How would you get around the above possible problem?

   Answer: Using SCNs instead of times is probably the best solution in this case because date and time is only precise at five minutes. Thus, in order to be sure to see the two inserted rows, you should enable Flashback using the SCN calculated in step 3.

```
SQL> execute dbms_flashback.enable_at_system_change_number(205056);
BEGIN dbms_flashback.enable_at_system_change_number(205056); END;
*
ERROR at line 1:
ORA-08184: attempting to re-enable Flashback while in Flashback mode
ORA-06512: at "SYS.DBMS_FLASHBACK", line 0
ORA-06512: at line 1

SQL> execute dbms_flashback.disable;
PL/SQL procedure successfully completed.

SQL> execute dbms_flashback.enable_at_system_change_number(205056);
```

```
PL/SQL procedure successfully completed.

SQL> select * from t1;

    C
-----
    1
    2

SQL> execute dbms_flashback.disable;
PL/SQL procedure successfully completed.
```

8. Connected as SYSDBA, try to enable Flashback. What happens?

   Answer: It is not possible to enable Flashback under user SYS.

```
SQL> connect / as sysdba;
Connected.

SQL> execute dbms_flashback.enable_at_time(-
>    TO_TIMESTAMP('26-JUL-2001:05:33:17', 'DD-MON-YYYY:HH24:MI:SS'));
BEGIN dbms_flashback.enable_at_time('26-JUL-2001:05:33:17'); END;
*
ERROR at line 1:
ORA-08185: Flashback not supported for user SYS
ORA-06512: at "SYS.DBMS_FLASHBACK", line 0
ORA-06512: at line 1
```

9. Connected as SYSTEM, drop table T1.

```
SQL> connect system/manager;
Connected.

SQL> drop table t1;
Table dropped.
```

Practice 2-2 Solution: Use of Resumable Space Allocation and Import

1. Connected as SYSTEM under SQL*Plus, create a new dictionary managed tablespace called TEST1 with one 200 KB data file. Then, create a table TEST in tablespace TEST1 as a select of the PRODUCT_INFORMATION table of the OE schema.

```
$ sqlplus system/manager
SQL*Plus: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> create tablespace test1 datafile '$HOME/ORADATA/u04/test1.dbf'
  2  size 200K extent management dictionary;
Tablespace created.

SQL> create table test tablespace test1
  2  as select * from oe.product_information;
Table created.
```

2. Now, export the TEST table using the Export utility.

```
SQL> host exp system/manager tables=TEST
Export: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

Export done in US7ASCII character set and AL16UTF16 NCHAR character set
server uses WE8ISO8859P1 character set (possible charset conversion)

About to export specified tables via Conventional Path ...
. . exporting table TEST  288 rows exported
Export terminated successfully without warnings.
```

3. Drop tablespace TEST1 and its contents and re-create it but this time with only one 80 KB data file. Once done, create again the TEST table inside tablespace TEST1 with the same structure as TEST in step 1 but without the corresponding rows (create only the structure of the TEST table).

```
SQL> drop tablespace test1 including contents and datafiles;
Tablespace dropped.

SQL> create tablespace test1
  2  datafile '$HOME/ORADATA/u04/test1.dbf' size 80K
  3  extent management dictionary;
Tablespace created.

SQL> create table test (product_id number(6,0),
  2  product_name varchar2(50), product_description varchar2(2000),
  3  category_id number(2,0), weight_class number(1,0),
  4  warranty_period interval year(2) to month,
  5  supplier_id number(6,0), product_status varchar2(20),
  6  list_price number(8,2), min_price number(8,2),
  7  catalog_url varchar2(50))
  8  tablespace test1;
Table created.
```

4. Import the table TEST by using the generated dump file from step 2. For this, use the Import utility with the following new parameters:
   - RESUMABLE=YES
   - RESUMABLE_NAME="TEST"
   - RESUMABLE_TIMEOUT=60

   What happens and why?

   Answer: Because the size of the TEST1 tablespace is not big enough to fit the imported rows, Oracle fails to allocate a new extent for the TEST table during the import. This would have immediately blocked the import process if you had not used the two new parameters above. Here, because of the timeout sets, Oracle waits for 60 seconds before signaling the error. Because this error was not fixed during the period, the import process stops after a while.

```
SQL> host imp system/manager file=expdat.dmp tables=test ignore=yes -
>    RESUMABLE=YES RESUMABLE_NAME="TEST" RESUMABLE_TIMEOUT=60
Import: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

Export file created by EXPORT:V09.00.01 via conventional path
import done in US7ASCII character set and AL16UTF16 NCHAR character set
import server uses WE8ISO8859P1 character set (possible charset
conversion)
. importing SYSTEM's objects into SYSTEM
. . importing table "TEST"
IMP-00058: ORACLE error 30032 encountered
ORA-30032: the suspended (resumable) statement has timed out
ORA-01653: unable to extend table SYSTEM.TEST by 8 in tablespace TEST1
IMP-00028: partial import of previous table rolled back:
           163 rows rolled back
Import terminated successfully with warnings.
```

5. Do steps 3 and 4 again in the previous session. Create a separate session connected as SYSDBA, and before the timeout is reached during the import phase in the first session, query the dba_resumable view, and identify the problem. Wait for the timeout period to expire and query again the dba_resumable view. What is your conclusion?

   Answer: After the timeout period expired, the error message listed in the dba_resumable view is automatically purged.

```
SQL> drop tablespace test1 including contents and datafiles;
Tablespace dropped.

SQL> create tablespace test1
  2  datafile '$HOME/ORADATA/u04/test1.dbf' size 80K
  3  extent management dictionary;
Tablespace created.

SQL> create table test (product_id number(6,0),
  2  product_name varchar2(50), product_description varchar2(2000),
  3  category_id number(2,0), weight_class number(1,0),
```

```
  4  warranty_period interval year(2) to month,
  5  supplier_id number(6,0), product_status varchar2(20),
  6  list_price number(8,2), min_price number(8,2),
  7  catalog_url varchar2(50))
  8  tablespace test1;
Table created.

SQL> host imp system/manager file=expdat.dmp tables=test ignore=yes -
>    RESUMABLE=YES RESUMABLE_NAME="TEST" RESUMABLE_TIMEOUT=60
Import: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

Export file created by EXPORT:V09.00.01 via conventional path
import done in US7ASCII character set and AL16UTF16 NCHAR character set
import server uses WE8ISO8859P1 character set (possible charset
conversion)
. importing SYSTEM's objects into SYSTEM
. . importing table "TEST"

-- From a separate session:

$ sqlplus /nolog
SQL*Plus: Release 9.0.1.0.0 – Production

SQL> connect / as sysdba;
Connected.

SQL> select name,resume_time,error_msg from dba_resumable;

NAME   RESUME_TIME ERROR_MSG
------ ----------- -------------------------------------
TEST               ORA-01653: unable to extend table
                   SYSTEM.TEST by 8 in tablespace TEST1

SQL> /
no rows selected
```

6. From the second session, create a simple AFTER SUSPEND ON DATABASE trigger called ADD_SPACE that resizes the unique TEST1 tablespace data file to 10 KB.

```
SQL> create or replace trigger add_space
  2  after suspend on database
  3  DECLARE
  4   PRAGMA AUTONOMOUS_TRANSACTION;
  5  BEGIN
  6    execute immediate
  7    'alter database datafile ''$HOME/ORADATA/u04/test1.dbf''
  8     resize 10K';
  9  END;
 10  /
Trigger created.
```

7. From the first session, execute steps 3 and 4 again. What happens and why?

Answer: The main difference with the previous step is that the trigger was fired automatically but did not fix the problem because 10 KB was not enough to accommodate the import. Thus, the import aborted as well.

```
SQL> drop tablespace test1 including contents and datafiles;
Tablespace dropped.

SQL> create tablespace test1
  2  datafile '$HOME/ORADATA/u04/test1.dbf' size 80K
  3  extent management dictionary;
Tablespace created.

SQL> create table test (product_id number(6,0),
  2  product_name varchar2(50), product_description varchar2(2000),
  3  category_id number(2,0), weight_class number(1,0),
  4  warranty_period interval year(2) to month,
  5  supplier_id number(6,0), product_status varchar2(20),
  6  list_price number(8,2), min_price number(8,2),
  7  catalog_url varchar2(50))
  8  tablespace test1;
Table created.

SQL> host imp system/manager file=expdat.dmp tables=test ignore=yes -
>    RESUMABLE=YES RESUMABLE_NAME="TEST" RESUMABLE_TIMEOUT=60
Import: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

Export file created by EXPORT:V09.00.01 via conventional path
import done in US7ASCII character set and AL16UTF16 NCHAR character set
import server uses WE8ISO8859P1 character set (possible charset
conversion)
. importing SYSTEM's objects into SYSTEM
. . importing table "TEST"
IMP-00058: ORACLE error 604 encountered
ORA-00604: error occurred at recursive SQL level 1
ORA-03297: file contains used data beyond requested RESIZE value
ORA-06512: at line 4
ORA-01653: unable to extend table SYSTEM.TEST by 8 in tablespace TEST1
IMP-00028: partial import of previous table rolled back:
           163 rows rolled back
Import terminated successfully with warnings.
```

8. From the second session, recreate the ADD_SPACE trigger with a 2 MB file size increase instead of 10 KB.

```
SQL> create or replace trigger add_space
  2  after suspend on database
  3  DECLARE
  4   PRAGMA AUTONOMOUS_TRANSACTION;
  5  BEGIN
  6    execute immediate
  7    'alter database datafile ''$HOME/ORADATA/u04/test1.dbf''
  8     resize 2M';
  9  END;
 10  /
```

```
Trigger created.
```

9. From the first session, repeat steps 3 and 4. What happens and why?

   Answer: Now the trigger is able to fix the problem by sufficiently raising the data file size in order to allow the import process to proceed without having problems.

```
SQL> drop tablespace test1 including contents and datafiles;
Tablespace dropped.

SQL> create tablespace test1
  2  datafile '$HOME/ORADATA/u04/test1.dbf' size 80K
  3  extent management dictionary;
Tablespace created.

SQL> create table test (product_id number(6,0),
  2  product_name varchar2(50), product_description varchar2(2000),
  3  category_id number(2,0), weight_class number(1,0),
  4  warranty_period interval year(2) to month,
  5  supplier_id number(6,0), product_status varchar2(20),
  6  list_price number(8,2), min_price number(8,2),
  7  catalog_url varchar2(50))
  8  tablespace test1;
Table created.

SQL> host imp system/manager file=expdat.dmp tables=test ignore=yes -
>    RESUMABLE=YES RESUMABLE_NAME="TEST" RESUMABLE_TIMEOUT=60
Import: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

Export file created by EXPORT:V09.00.01 via conventional path
import done in US7ASCII character set and AL16UTF16 NCHAR character set
import server uses WE8ISO8859P1 character set (possible charset
conversion)
. importing SYSTEM's objects into SYSTEM
. . importing table "TEST" 288 rows imported
Import terminated successfully without warnings.
```

10. Connected as SYSDBA, drop tablespace TEST1 and its data file as well as the add_space trigger.

```
SQL> connect / as sysdba;
Connected.

SQL> drop tablespace test1 including contents and datafiles;
Tablespace dropped.

SQL> drop trigger add_space;
Trigger dropped.
```

Practice 3-1 Solution: LogMiner Enhancements

1. Prepare for using LogMiner by creating a flat file dictionary file called `dictionary.ora`. Check your `UTL_FILE_DIR` parameter and use the same directory.

```
SQL> CONNECT / AS SYSDBA
Connected.

SQL> SHOW PARAMETER utl

NAME                 TYPE        VALUE
-------------------- ----------- ----------------------------
utl_file_dir         string      /oracourse/user/st01/ORADATA

SQL> EXECUTE dbms_logmnr_d.build –
> (DICTIONARY_FILENAME => 'dictionary.ora', -
>  DICTIONARY_LOCATION => '/oracourse/user/st01/ORADATA', -
>  OPTIONS => dbms_logmnr_d.store_in_flat_file);
PL/SQL procedure successfully completed.
```

The name of the directory will be different for your site.

2. Switch logfiles and note the current logfile name and the current time. This is the redo you will analyze.

```
SQL> ALTER SYSTEM SWITCH LOGFILE;

SQL> SELECT member FROM v$logfile, v$log
  2  WHERE  v$logfile.group# = v$log.group#
  3  AND    v$log.status = 'CURRENT';

MEMBER
-------------------------------------------------
/oracourse/user/st01/ORADATA/u03/log_01_01_u01.rdo

SQL> SELECT CURRENT_TIMESTAMP FROM DUAL;

CURRENT_TIMESTAMP
---------------------------------
03-JUL-01 11.57.38.654612 AM +02:00
```

The name of the directory will be different for your site.

3. Do some user activity. Create a table consisting of a few rows and columns selected from `HR.EMPLOYEES` (for example ID, last name, and salary in department 30). Add an extra column to `EMP30`. Update the new column with values. Roll back that update and do another update which you commit. Drop the table. Note the time.

```
SQL> CREATE TABLE emp30 AS
  2  SELECT employee_id,last_name,salary FROM employees
  3  WHERE department_id = 30 ;
Table created.

SQL> ALTER TABLE emp30 ADD (new_salary NUMBER(8,2));
```

```
Table altered.

SQL> UPDATE emp30 SET new_salary = salary * 1.5;
6 rows updated.

SQL> ROLLBACK;
Rollback complete.

SQL> UPDATE emp30 SET new_salary = salary * 1.2;
6 rows updated.

SQL> COMMIT;
Commit complete.

SQL> DROP TABLE emp30;
Table dropped.

SQL> SELECT CURRENT_TIMESTAMP FROM DUAL;

CURRENT_TIMESTAMP
---------------------------------
03-JUL-01 12.07.34.111612 AM +02:00
```

4. Initialize LogMiner with the current redo file. Start mining between the two time points noted above.

```
SQL> CONNECT / AS SYSDBA
Connected.

SQL> EXECUTE dbms_logmnr.add_logfile ( -
> LOGFILENAME => -
> '/oracourse/user/st01/ORADATA/u03/log_01_01_u01.rdo', -
> OPTIONS => dbms_logmnr.new)
PL/SQL procedure successfully completed.

SQL> EXECUTE dbms_logmnr.start_logmnr ( -
> DICTFILENAME => '/oracourse/user/st01/ORADATA/dictionary.ora',    -
> STARTTIME => TO_DATE('03-JUL-01 11:57:38','DD_MON-RR HH24:MI:SS'),-
> ENDTIME =>   TO_DATE('03-JUL-01 12:07:34','DD_MON-RR HH24:MI:SS'),-
> OPTIONS =>
dbms_logmnr.ddl_dict_tracking+dbms_logmnr.committed_data_only)
PL/SQL procedure successfully completed.
```

The name of the directory will be different for your site.

5. Display the committed changes made by user HR on segment EMP30.

```
SQL> SELECT timestamp, username, operation, sql_redo, sql_undo
  2  FROM   v$logmnr_contents
  3  WHERE  username='HR'
  4  AND    (seg_name = 'EMP30' OR seg_name IS NULL);

TIMESTAMP                USERNAME  OPERATION
------------------------ --------  --------------------------------
SQL_REDO                           SQL_UNDO
---------------------------------  --------------------------------
03-JUL-01 11:58:11             HR  START
set transaction read write;

03-JUL-01 11:58:11             HR  DDL
CREATE TABLE emp30 AS              (Note: null for UNDO)
SELECT employee_id,last_name,
salary FROM employees
WHERE department_id = 30;

03-JUL-01 11:58:11             HR  COMMIT
commit;

03-JUL-01 12:01:23             HR  DDL
ALTER TABLE emp30 ADD              (Note: null for UNDO)
(new_salary NUMBER(8,2));

03-JUL-01 12:03:05             HR  UPDATE
update "HR"."EMP30" set            update "HR"."EMP30" set
"NEW_SALARY" = '16500' where       "NEW_SALARY" = NULL where
"NEW_SALARY" IS NULL and ROWID     "NEW_SALARY" = '16500' and ROWID
= 'AAABnFAAEAAALkUAAA';            = 'AAABnFAAEAAALkUAAA';

03-JUL-01 12:05:58             HR  DDL
DROP TABLE emp30;                  (Note: null for UNDO)
```

The listing is much reduced and formatted. There should be about 45 rows.

6. End the LogMiner session.

```
SQL> EXECUTE dbms_logmnr.end_logmnr
PL/SQL procedure successfully completed.
```

Practice 4-1 Solution: Backup and Recovery

1. Set the database into ARCHIVELOG mode, while in MOUNT mode. Enable automatic archiving to the `$HOME/ORADATA/ARCHIVE1` directory. Edit the parameter file as needed.

```
SQL> CONNECT / AS SYSDBA
Connected.

SQL> SELECT NAME,LOG_MODE FROM V$DATABASE;
NAME      LOG_MODE
--------- ------------
DB01      NOARCHIVELOG

SQL> SHUTDOWN IMMEDIATE
Database shutdown.

SQL> STARTUP PFILE=$HOME/ADMIN/PFILE/initU01.ora
ORACLE instance started.

SQL> ALTER DATABASE ARCHIVELOG;
Database altered.

SQL> ALTER DATABASE OPEN;
Database altered.

SQL> SHOW PARAMETER LOG_ARCHIVE_DEST_1
log_archive_dest_1 string   location=/databases/ed27/ORADATA/ARCHIVE1

SQL> SHOW PARAMETER LOG_ARCHIVE_START
log_archive_start  boolean  TRUE
```

2. Create a tablespace, `RMANTEST`, of size 75 KB. Put this in the same directory as your other tablespaces. Exit SQL*Plus.

```
SQL> CREATE TABLESPACE rmantest DATAFILE
  2  '/databases/ed27/ORADATA/u01/rmantest_01.dbf' SIZE 75K;
Tablespace created.

SQL> EXIT
```

3. Start RMAN and connect to the database. Do not use a recovery catalog. Use the `show all` command to see the current configuration settings.

```
$ rman target /
Recovery Manager: Release 9.0.1.0.0 – Production
connected to target database: ED27 (DBID=868643342)

RMAN> show all;

RMAN configuration parameters are:
CONFIGURE RETENTION POLICY TO REDUNDANCY 1;
CONFIGURE BACKUP OPTIMIZATION OFF;
CONFIGURE DEFAULT DEVICE TYPE TO DISK;
CONFIGURE CONTROLFILE AUTOBACKUP OFF;
```

```
CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK TO '%F';
CONFIGURE DEVICE TYPE DISK PARALLELISM 1;
CONFIGURE DATAFILE BACKUP COPIES FOR DEVICE TYPE DISK TO 1;
CONFIGURE ARCHIVELOG BACKUP COPIES FOR DEVICE TYPE DISK TO 1;
CONFIGURE MAXSETSIZE TO UNLIMITED;
CONFIGURE SNAPSHOT CONTROLFILE NAME TO
'/databases/oracle9i/dbs/snapcf_ed27.f';
```

4. Configure a channel for back up, set the file location to your home directory, and use `<SID>_%U.bak` for the file name. Configure the snapshot control file to be written to your home directory as well. Keep the default name.

```
RMAN> CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT
      '/databases/ed27/ed27_%U.bak';

old RMAN configuration parameters:
CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT '';
new RMAN configuration parameters:
CONFIGURE CHANNEL 1 DEVICE TYPE DISK FORMAT   '/databases/ed27/ed27_%U';
new RMAN configuration parameters are successfully stored

RMAN> CONFIGURE SNAPSHOT CONTROLFILE NAME TO
'/databases/ed27/snapcf_ed27.f';

snapshot controlfile name set to: /databases/ed27/snapcf_ed27.f
new RMAN configuration parameters are successfully stored
```

5. Back up your database. Exit from RMAN.

```
RMAN> backup database;

Starting backup at 02-JUL-01
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=8 devtype=DISK
channel ORA_DISK_1: starting full datafile backupset
channel ORA_DISK_1: specifying datafile(s) in backupset
including current controlfile in backupset
input datafile fno=00001 name=/databases/ed27/ORADATA/u01/system01.dbf
input datafile fno=00002 name=/databases/ed27/ORADATA/u01/undotbs01.dbf
input datafile fno=00005 name=/databases/ed27/ORADATA/u01/example01.dbf
...
input datafile fno=00010 name=/databases/ed27/ORADATA/u01/rmantest_01.dbf
channel ORA_DISK_1: starting piece 1 at 02-JUL-01
channel ORA_DISK_1: finished piece 1 at 02-JUL-01 with 2 copies
piece handle=/databases/ed27/ed27_0ccopgm9_1_1.bak comment=NONE
piece handle=/databases/ed27/ed27_0ccopgm9_1_2.bak comment=NONE
channel ORA_DISK_1: backup set complete, elapsed time: 00:01:18
Finished backup at 02-JUL-01

RMAN> EXIT
```

6. As SYSTEM/MANAGER in SQL*Plus make a copy of the `HR.EMPLOYEES` table in the `RMANTEST` tablespace calling the table `EMP`. Note how many rows are in the new table.

```
SQL> CONNECT system/manager
Connected.

SQL> CREATE TABLE emp TABLESPACE rmantest
  2   AS (SELECT * FROM hr.employees);
Table created.

SQL> select count(*) from emp;
  COUNT(*)
----------
       107
```

7. Connect as a privileged user and shut down the instance. Exit from SQL*Plus.

```
SQL> connect / as sysdba
Connected.

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

SQL> EXIT
```

8. Rename the data file of the RMANTEST tablespace, simulating data file loss.

```
$ cd $HOME/ORADATA/u01/
$ ls rmantest*
rmantest_01.dbf
$ mv rmantest_01.dbf rmantest_01.bad
```

9. Log in to SQL*Plus and attempt to start the instance. Exit SQL*Plus.

```
$ sqlplus "/ as sysdba"
SQL*Plus: Release 9.0.1.0.0 – Production
Connected to an idle instance.

SQL> startup
ORACLE instance started.

Total System Global Area   21790412 bytes
Fixed Size                   278220 bytes
Variable Size              16777216 bytes
Database Buffers            4194304 bytes
Redo Buffers                 540672 bytes
Database mounted.
ORA-01157: cannot identify/lock data file 10 - see DBWR trace file
ORA-01110: data file 10: 'databases/ed27/ORADATA/u01/rmantest_01.dbf'

SQL> EXIT
```

10. Start RMAN, connect without a recovery catalog, and do a restore of the missing data file. Exit RMAN.

```
$ rman target /
Recovery Manager: Release 9.0.1.0.0 – Production
connected to target database: ED27 (DBID=868643342)

RMAN> RESTORE DATAFILE 'databases/ed27/ORADATA/u01/rmantest_01.dbf';
Starting restore at 02-JUL-01

using target database controlfile instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: sid=10 devtype=DISK
channel ORA_DISK_1: starting datafile backupset restore
channel ORA_DISK_1: specifying datafile(s) to restore from backup set
restoring datafile 00010 to databases/ed27/ORADATA/u01/rmantest_01.dbf
channel ORA_DISK_1: restored backup piece 1
piece handle=/databases/ed27/ed27_0ccopgm9_1_1.bak tag=null params=NULL
channel ORA_DISK_1: restore complete
Finished restore at 02-JUL-01

RMAN> EXIT
```

11. Use SQL*Plus to do a test recovery.

```
SQL> CONNECT / as sysdba
Connected.

SQL> RECOVER DATABASE TEST;
ORA-10574: Test recovery did not corrupt any data block
ORA-10573: Test recovery tested redo from change 492498 to 492507
ORA-10572: Test recovery canceled due to errors
ORA-10585: Test recovery can not apply redo
           that may modify control file
```

12. Because the test shows no errors, do the recovery. Test by examining the row count of the EMP table.

```
SQL> RECOVER DATABASE;
Media recovery complete.

SQL> ALTER DATABASE OPEN;
Database altered.

SQL> SELECT COUNT(*) FROM system.emp;

  COUNT(*)
----------
       107
```

13. Clean up by dropping the tablespace including the data file.

```
SQL> DROP TABLESPACE rmantest INCLUDING CONTENTS;
Tablespace dropped.

SQL> host rm –i /databases/ed27/ORADATA/u01/rmantest_01*
```

Practice 7-1 Solution: Online Operations

1. Log in as SYSTEM and create a table EMP30 in the HR schema, which consists of all the columns of all employees in department 30 from the EMPLOYEES table. Add a primary key to the EMPLOYEE_ID column.

```
SQL> CONNECT System/manager
Connected.

SQL> CREATE TABLE hr.emp30 AS
  2  SELECT * FROM hr.employees WHERE department_id = 30;
Table created.

SQL> ALTER TABLE hr.emp30 ADD PRIMARY KEY (employee_id);
Table altered.
```

   The new company standard is that all key columns will now have the suffix of "_KEY" not "_ID" and all these key columns must be the first set of columns, with primary key always first. (The order of the key columns following the PK is not important.)
   The LAST_NAME column must be in upper case. There is a new column FULL_NAME which consists of the first name, a space, and then the last name. Here the last name must be in the same case as it was in the original employee table. The table must be partitioned on EMPLOYEE_KEY with three partitions; the high values for each are 116, 118, and MAXVALUE. To keep things simple do not worry about any other constraints that are on the original table. This redefinition of the table must be done online, without loss of data.

2. Create the interim table for the redefinition including above changes. Call it INT_EMP30 and it must be in the HR schema.

```
SQL> CREATE TABLE hr.int_emp30 (
  2  employee_key    NUMBER(6) PRIMARY KEY,
  3  job_key         VARCHAR2(10),
  4  manager_key     NUMBER(6),
  5  department_key  NUMBER(4),
  6  first_name      VARCHAR2(20),
  7  last_name       VARCHAR2(25),
  8  full_name       VARCHAR2(50),
  9  email           VARCHAR2(25),
 10  phone_number    VARCHAR2(20),
 11  hire_date       DATE,
 12  salary          NUMBER(8,2),
 13  commission_pct  NUMBER(2,2))
 14  PARTITION BY RANGE(employee_key) (
 15    PARTITION emp116 VALUES LESS THAN (116)
 16                     TABLESPACE sample,
 17    PARTITION emp118 VALUES LESS THAN (118)
 18                     TABLESPACE sample,
 19    PARTITION empmax VALUES LESS THAN (MAXVALUE)
 20                     TABLESPACE sample);
Table created.
```

3. Verify that EMP30 can be redefined, then execute the redefinition.

```
SQL> EXECUTE DBMS_REDEFINITION.CAN_REDEF_TABLE('HR','EMP30');
PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_REDEFINITION.START_REDEF_TABLE( -
>   'HR', 'EMP30', 'INT_EMP30', -
>   'EMPLOYEE_ID EMPLOYEE_KEY, -
>    JOB_ID JOB_KEY, -
>    MANAGER_ID MANAGER_KEY, -
>    DEPARTMENT_ID DEPARTMENT_KEY, -
>    FIRST_NAME FIRST_NAME, -
>    UPPER(LAST_NAME) LAST_NAME, -
>    FIRST_NAME||CHR(32)||LAST_NAME FULL_NAME, -
>    EMAIL EMAIL, -
>    PHONE_NUMBER PHONE_NUMBER, -
>    HIRE_DATE HIRE_DATE, -
>    SALARY SALARY, -
>    COMMISSION_PCT COMMISSION_PCT');
PL/SQL procedure successfully completed.

SQL> EXECUTE –
> DBMS_REDEFINITION.FINISH_REDEF_TABLE('HR','EMP30','INT_EMP30');
PL/SQL procedure successfully completed.
```

4. Check that the work was completed as expected.

```
SQL> DESCRIBE hr.emp30
 Name                              Null?    Type
 --------------------------------- -------- --------------
 EMPLOYEE_KEY                      NOT NULL NUMBER(6)
 JOB_KEY                                    VARCHAR2(10)
 MANAGER_KEY                                NUMBER(6)
 DEPARTMENT_KEY                             NUMBER(4)
 FIRST_NAME                                 VARCHAR2(20)
 LAST_NAME                                  VARCHAR2(25)
 FULL_NAME                                  VARCHAR2(50)
 EMAIL                                      VARCHAR2(25)
 PHONE_NUMBER                               VARCHAR2(20)
 HIRE_DATE                                  DATE
 SALARY                                     NUMBER(8,2)
 COMMISSION_PCT                             NUMBER(2,2)

SQL> DESCRIBE hr.int_emp30
 Name                              Null?    Type
 --------------------------------- -------- --------------
 EMPLOYEE_ID                       NOT NULL NUMBER(6)
 FIRST_NAME                                 VARCHAR2(20)
 LAST_NAME                         NOT NULL VARCHAR2(25)
 EMAIL                             NOT NULL VARCHAR2(25)
 PHONE_NUMBER                               VARCHAR2(20)
 HIRE_DATE                         NOT NULL DATE
 JOB_ID                            NOT NULL VARCHAR2(10)
 SALARY                                     NUMBER(8,2)
 COMMISSION_PCT                             NUMBER(2,2)
 MANAGER_ID                                 NUMBER(6)
```

DEPARTMENT_ID	NUMBER(4)

5. Clean up and drop the tables:

```
SQL> DROP TABLE HR.EMP30;
SQL> DROP TABLE HR.INT_EMP30;
```

Practice 7-2 Solution: New Parameter File

1. Start up the instance with the supplied text parameter file.

```
SQL> STARTUP PFILE=$HOME/ADMIN/PFILE/init.ora
ORACLE instance started.

Total System Global Area   21790412 bytes
Fixed Size                   278220 bytes
Variable Size              16777216 bytes
Database Buffers            4194304 bytes
Redo Buffers                 540672 bytes
Database mounted.
Database opened.

SQL> SHOW PARAMETER spfile

NAME                 TYPE        VALUE
-------------------- ----------- -------------
spfile               string
```

2. Now create the SPFILE from the PFILE used to start up the instance.

```
SQL> create spfile from pfile='$HOME/ADMIN/PFILE/init.ora';
File created.
```

3. Shut down the instance and restart using the SPFILE. Again, examine the `SPFILE` parameter.

```
SQL> SHUTDOWN IMMEDIATE
Database closed.
Database dismounted.
ORACLE instance shut down.

SQL> STARTUP
ORACLE instance started.

Total System Global Area   21790412 bytes
Fixed Size                   278220 bytes
Variable Size              16777216 bytes
Database Buffers            4194304 bytes
Redo Buffers                 540672 bytes
Database mounted.
Database opened.

SQL> show parameter spfile

NAME                 TYPE        VALUE
-------------------- ----------- -----------------
spfile               string      ?/dbs/spfile@.ora
```

4. Examine the OPEN_CURSORS parameter and increase it with 50, but only change the SPFILE and add a comment of "Test Run 1." Check to see that the change has only happened in the SPFILE.

```
SQL> SHOW PARAMETER open_cursors

NAME                    TYPE        VALUE
----------------------- ----------- -----
open_cursors            integer     350

SQL> ALTER SYSTEM SET open_cursors=400
  2  COMMENT='Test Run 1' SCOPE=SPFILE;
System altered.

SQL> SHOW PARAMETER open_cursors

NAME                    TYPE        VALUE
----------------------- ----------- -----
open_cursors            integer     350

SQL> SELECT name,value,update_comment
  2  FROM   v$spparameter
  3  WHERE  name = 'open_cursors';

NAME               VALUE                UPDATE_COMMENT
------------------ -------------------- --------------
open_cursors       400                  Test Run 1
```

5. In preparation for removing the SPFILE create a PFILE from the current SPFILE. Then shut down the instance, remove the SPFILE and delete the SPFILE. Restart the instance using the newly created PFILE.

```
SQL> CREATE PFILE='$HOME/ADMIN/PFILE/mypfile.ora' FROM SPFILE;
File created.

SQL> SHUTDOWN IMMEDIATE
Database closed.
Database dismounted.
ORACLE instance shut down.

SQL> host rm $ORACLE_HOME/dbs/spfileed27.ora

SQL> STARTUP PFILE=$HOME/ADMIN/PFILE/mypfile.ora
ORACLE instance started.

Total System Global Area   21790412 bytes
Fixed Size                   278220 bytes
Variable Size              16777216 bytes
Database Buffers            4194304 bytes
Redo Buffers                 540672 bytes
Database mounted.
Database opened.
```

Practice 8-1 Solution: Creating an External Table

1. Connected as SYSTEM under SQL*Plus, create a copy of the OE.ORDER_ITEMS table (only the first ten rows) in the SYSTEM schema.

```
$ sqlplus system/manager
SQL*Plus: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> create table order_items as
  2  select * from oe.order_items where rownum < 11;
Table created.

SQL> select * from order_items;

  ORDER_ID LINE_ITEM_ID PRODUCT_ID UNIT_PRICE   QUANTITY
---------- ------------ ---------- ---------- ----------
      2355            1       2289         46        200
      2356            1       2264      199.1         38
      2357            1       2211        3.3        140
      2358            1       1781      226.6          9
      2359            1       2337      270.6          1
      2360            1       2058         23         29
      2361            1       2289         46        180
      2362            1       2289         48        200
      2363            1       2264      199.1          9
      2364            1       1910         14          6
10 rows selected.
```

2. Create an Oracle directory called EXT_TABLES pointing to your $HOME/STUDENT/LABS/ O/S directory (specify full path!). Then, create an external table called ORDER_ITEMS_EXT mapping to the previously created ORDER_ITEMS table. Use the two precreated files, order_items1.dat and order_items2.dat, stored in your $HOME/STUDENT/LABS directory as the LOCATION for the external table. Once done, verify that you can select from the created external table.

```
SQL> CREATE OR REPLACE DIRECTORY ext_tables
  2  AS '/databases/ed25/9iNFDBA/STUDENT/LABS/';
Directory created.

SQL> create table order_items_ext
  2     (order_id   number(12), line_item_id number(3)   ,
  3      product_id number(6) , unit_price   number(8,2) ,
  4      quantity   number(8)                            )
  5  organization external
  6     (type oracle_loader
  7      default directory ext_tables
  8      access parameters (records delimited by newline
  9                         fields terminated by ',')
 10      location ('order_items1.dat','order_items2.dat'));
Table created.

SQL> select * from order_items_ext;
```

```
  ORDER_ID LINE_ITEM_ID PRODUCT_ID UNIT_PRICE   QUANTITY
---------- ------------ ---------- ---------- ----------
      2355            1       2289         46        200
      2355            2       2264         50        100
      2356            1       2264        100         10
      2356            2       2264        110         20
      2356            3       2264        120         30
```

3. Use the two new dictionary views, DBA_EXTERNAL_TABLES and DBA_EXTERNAL_LOCATIONS, to verify the various characteristics of the newly created external table.

```
SQL> select owner,table_name,type_name,default_directory_name
  2  from   dba_external_tables;

OWNER                          TABLE_NAME
------------------------------ ------------------------------
TYPE_NAME                      DEFAULT_DIRECTORY_NAME
------------------------------ ------------------------------
SYSTEM                         ORDER_ITEMS_EXT
ORACLE_LOADER                  EXT_TABLES

SQL> select location from dba_external_locations;

LOCATION
----------------
order_items1.dat
order_items2.dat
```

4. If you don't know the exact syntax you should use to create an external table, you can use SQL*Loader as follows.
   Under the precreated $HOME/STUDENT/LABS directory, you should find the two data files used in the previous steps plus a SQL*Loader control file called order_items.ctl. View the contents of the SQL*Loader control file. As you can see, it describes the ORDER_ITEMS table structure.

```
$ cd $HOME/STUDENT/LABS
$ ls
order_items.ctl   order_items1.dat  order_items2.dat
$ more order_items.ctl
LOAD DATA
INFILE '$HOME/STUDENT/LABS/order_items1.dat'
INTO TABLE order_items
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(ORDER_ID,LINE_ITEM_ID,PRODUCT_ID,UNIT_PRICE,QUANTITY)
```

5. Execute the SQL*Loader utility under the SYSTEM account with the order_items.ctl control file as parameter. Use also the brand new external_table parameter with the GENERATE_ONLY value. This will generate the various external table commands inside the SQL*Loader log file.

```
$ sqlplus system/manager
SQL*Plus: Release 9.0.1.0.0 – Production
```

```
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> host sqlldr system/manager control= order_items.ctl –
>    log=order_items.log external_table=GENERATE_ONLY
SQL*Loader: Release 9.0.1.0.0 – Production
```

6. Look at the generated SQL*Loader log file.

```
SQL> host vi order_items.log
"order_items.log" 106 lines, 2975 characters
SQL*Loader: Release 9.0.1.0.0 – Production

Control File:   order_items.ctl
Data File:      /databases/ed25/DIR/order_items1.dat
Bad File:       order_items1.bad
Discard File:   none specified (Allow all discards)
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Continuation:   none specified
Path used:      External Table

Table ORDER_ITEMS, loaded from every logical record.
Insert option in effect for this table: INSERT

Column Name                     Position   Len Term Encl Datatype
------------------------------ ---------- ----- ---- ---- ---------
ORDER_ID                           FIRST     *   ,  O(") CHARACTER
LINE_ITEM_ID                        NEXT     *   ,  O(") CHARACTER
PRODUCT_ID                          NEXT     *   ,  O(") CHARACTER
UNIT_PRICE                          NEXT     *   ,  O(") CHARACTER
QUANTITY                            NEXT     *   ,  O(") CHARACTER

CREATE DIRECTORY statements needed for one or more data files
------------------------------------------------------------------------
CREATE DIRECTORY SYS_SQLLDR_XT_TMPDIR_00000 AS '/databases/ed25/DIR/'

CREATE TABLE statement for external table:
------------------------------------------------------------------------
CREATE TABLE SYS_SQLLDR_X_EXT
(
  ORDER_ID NUMBER(12),
  LINE_ITEM_ID NUMBER(3),
  PRODUCT_ID NUMBER(6),
  UNIT_PRICE NUMBER(8,2),
  QUANTITY NUMBER(8)
)
ORGANIZATION external
(
  TYPE oracle_loader
  DEFAULT DIRECTORY SYS_SQLLDR_XT_TMPDIR_00000
  ACCESS PARAMETERS
  (
    RECORDS DELIMITED BY NEWLINE
    BADFILE 'order_items1.bad'
    FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"' LDRTRIM
```

```
    (
      ORDER_ID CHAR(255)
        TERMINATED BY "," OPTIONALLY ENCLOSED BY '"',
      LINE_ITEM_ID CHAR(255)
        TERMINATED BY "," OPTIONALLY ENCLOSED BY '"',
      PRODUCT_ID CHAR(255)
        TERMINATED BY "," OPTIONALLY ENCLOSED BY '"',
      UNIT_PRICE CHAR(255)
        TERMINATED BY "," OPTIONALLY ENCLOSED BY '"',
      QUANTITY CHAR(255)
        TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
    )
  )
  location
  (
    'order_items1.dat'
  )
)REJECT LIMIT UNLIMITED

INSERT statements used to load internal tables:
-----------------------------------------------
INSERT /*+ append */ INTO ORDER_ITEMS
(
  ORDER_ID,
  LINE_ITEM_ID,
  PRODUCT_ID,
  UNIT_PRICE,
  QUANTITY
)
SELECT
  ORDER_ID,
  LINE_ITEM_ID,
  PRODUCT_ID,
  UNIT_PRICE,
  QUANTITY
FROM SYS_SQLLDR_X_EXT

statements to cleanup objects created by previous statements:
-------------------------------------------------------------
DROP TABLE SYS_SQLLDR_X_EXT
DROP DIRECTORY SYS_SQLLDR_XT_TMPDIR_00000

Run began on Wed Jul 04 07:15:57 2001
:q
```

7. Connected as SYSTEM, drop the tables ORDER_ITEMS_EXT, ORDER_ITEMS and then drop directory EXT_TABLES.

```
SQL> connect system/manager
Connected.

SQL> drop table order_items_ext;
Table dropped.

SQL> drop table order_items;
Table dropped.
```

```
SQL> drop directory ext_tables;
Directory dropped.
```

Practice 8-2 Solution: Creating a List-Partitioned Table

1. Connected as SYSTEM under SQL*Plus, create the list partitioned table SALES_LIST using the lab_08_01.sql script.

```
$ sqlplus system/manager
SQL*Plus: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> connect system/manager
Connected.

SQL> CREATE TABLE sales_list
  2  ( salesman_id NUMBER(5)
  3  , salesman_name VARCHAR2(30)
  4  , sales_state VARCHAR2(20)
  5  , sales_amount NUMBER(10)
  6  , sales_date DATE
  7  )
  8  PARTITION BY LIST(sales_state)
  9  ( PARTITION sales_west    VALUES ('a','b')
 10  , PARTITION sales_east    VALUES ('e','f')
 11  , PARTITION sales_central VALUES ('c','d')
 12  );
Table created.
```

2. Use the DBA_PART_TABLES and DBA_TAB_PARTITIONS dictionary views to identify the SALES_LIST table characteristics.

```
SQL> col tab_name    format a10
SQL> col part_type   format a10
SQL> col part_count  format 99999
SQL> col key_count   format 99999
SQL> col def_init    format a10
SQL> col def_next    format a10
SQL> col def_pctincr format a10

SQL> select table_name             tab_name,
  2         partitioning_type      part_type,
  3         partition_count        part_count,
  4         partitioning_key_count key_count,
  5         def_initial_extent     def_init,
  6         def_next_extent        def_next,
  7         def_pct_increase       def_pctincr
  8  from   user_part_tables
  9  where  table_name = 'SALES_LIST';

TAB_NAME   PART_TYPE PART_COUNT KEY_COUNT DEF_INIT DEF_NEXT DEF_PCTINC
---------- --------- ---------- --------- -------- -------- ----------
SALES_LIST LIST               3         1 DEFAULT  DEFAULT  DEFAULT

SQL> col table_name      format a12
SQL> col partition_name  format a20
SQL> col tablespace_name format a20
SQL> col high_value      format a19
```

```
SQL> select   table_name, partition_name, high_value, tablespace_name
  2  from     user_tab_partitions
  3  where    table_name='SALES_LIST'
  4  order by table_name,partition_name;

TABLE_NAME   PARTITION_NAME     HIGH_VALUE        TABLESPACE_NAME
------------ ------------------ ----------------- ---------------
SALES_LIST   SALES_CENTRAL      'c', 'd'          SYSTEM
SALES_LIST   SALES_EAST         'e', 'f'          SYSTEM
SALES_LIST   SALES_WEST         'a', 'b'          SYSTEM
```

3. Reconstitute the complete `CREATE TABLE SALES_LIST` command using only the data dictionary.

```
SQL> set long   4000
SQL> set pagesize 50

SQL> SELECT dbms_metadata.get_ddl('TABLE', 'SALES_LIST') from dual;

DBMS_METADATA.GET_DDL('TABLE','SALES_LIST')
--------------------------------------------------------------------------------
  CREATE TABLE "SYSTEM"."SALES_LIST"
   (    "SALESMAN_ID" NUMBER(5,0),
        "SALESMAN_NAME" VARCHAR2(30),
        "SALES_STATE" VARCHAR2(20),
        "SALES_AMOUNT" NUMBER(10,0),
        "SALES_DATE" DATE
   )  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 STORAGE( BUFFER_POOL DEFAULT)
  TABLESPACE "SYSTEM"
  PARTITION BY LIST ("SALES_STATE")
 (PARTITION "SALES_WEST"  VALUES ('a','b')
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
  STORAGE(INITIAL 12288 NEXT 12288 MINEXTENTS 1 MAXEXTENTS 249
  PCTINCREASE 50
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM"
,
 PARTITION "SALES_EAST"  VALUES ('e','f')
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
  STORAGE(INITIAL 12288 NEXT 12288 MINEXTENTS 1 MAXEXTENTS 249
  PCTINCREASE 50
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM"
,
 PARTITION "SALES_CENTRAL"  VALUES ('c','d')
  PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
  STORAGE(INITIAL 12288 NEXT 12288 MINEXTENTS 1 MAXEXTENTS 249
  PCTINCREASE 50
  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM"
)
```

4. Execute the `lab_08_02.sql` script in order to insert some rows into the `SALES_LIST` table. What happens and why?

Answer: The first three rows are correctly inserted into the table whereas the last one is rejected because the inserted partition key 'g' does not map to any partition.

```
SQL> insert into sales_list values
  2  (10, 'JFV', 'a', 100, '05-JAN-2001');
1 row created.

SQL> insert into sales_list values
  2  (21, 'MH', 'f', 150, '15-JAN-2001');
1 row created.

SQL> insert into sales_list values
  2  (32, 'Clement', 'c', 250, '20-JAN-2001');
1 row created.

SQL> commit;
Commit complete.

SQL> insert into sales_list values
  2  (44, 'Lois', 'g', 130, '28-JAN-2001');
insert into sales_list values
            *
ERROR at line 1:
ORA-14400: inserted partition key does not map to any partition
```

5. Analyze table SALES_LIST with the COMPUTE STATISTICS option and then create the PLAN_TABLE into the SYSTEM account. Once done, explain the following statements (you should use the explain.sql script to print the result of each explain plan command):

```
SQL> select * from sales_list;
SQL> select * from sales_list where sales_state = 'a';
SQL> select * from sales_list where sales_state in ('a','e');
SQL> select * from sales_list where sales_state = 'a'
  2                                or sales_state = 'd';
SQL> select * from sales_list where sales_state > 'e';
SQL> select * from sales_list where sales_state < 'e';
```

For each of the above statements, give your interpretation of the generated execution plans.

Answer: The Oracle server also uses partition pruning with a list-partitioned table such as SALES_LIST. The explain plan interpretation is the same as with previous partitioning methods.
The last one is probably the most difficult one to understand. The plan shows a PARTITION LIST(ALL) operation, which seems to imply a full scan of the complete table. In fact, a full scan is not done because only two partitions (the first one and the last one) verify the predicate (sales_state < 'e'). Thus, the Oracle server scans those two partitions only.

```
SQL> analyze table sales_list compute statistics;
Table analyzed.

SQL> @$ORACLE_HOME/rdbms/admin/utlxplan.sql
Table created.
```

```
SQL> explain plan for
  2  select * from sales_list;
Explained.

SQL> @./LABS/explain

RESULT
----------------------------------------------------------
0 SELECT STATEMENT  Optimizer=CHOOSE Card=3
1   PARTITION LIST(ALL)[1 --> 3]
2     TABLE ACCESS(FULL) OF 'SALES_LIST'[1 --> 3]  Card=3

SQL> truncate table plan_table;
Table truncated.

SQL> explain plan for
  2  select * from sales_list where sales_state='a';
Explained.

SQL> @./LABS/explain

RESULT
----------------------------------------------------------
0 SELECT STATEMENT  Optimizer=CHOOSE Card=1
1   TABLE ACCESS(FULL) OF 'SALES_LIST'[1 --> 1]  Card=1

SQL> truncate table plan_table;
Table truncated.

SQL> explain plan for
  2  select * from sales_list where sales_state in ('a','e');
Explained.

SQL> @./LABS/explain

RESULT
----------------------------------------------------------------------
0 SELECT STATEMENT  Optimizer=CHOOSE Card=2
1   PARTITION LIST(INLIST)[KEY(INLIST) --> KEY(INLIST)]
2     TABLE ACCESS(FULL) OF 'SALES_LIST'[KEY(INLIST) --> KEY(INLIST)]
                                                          Card=2

SQL> truncate table plan_table;
Table truncated.

SQL> explain plan for
  2  select * from sales_list where sales_state = 'a' or sales_state='d';
Explained.

SQL> @./LABS/explain

RESULT
----------------------------------------------------------------------
0 SELECT STATEMENT  Optimizer=CHOOSE Card=2
1   PARTITION LIST(INLIST)[KEY(INLIST) --> KEY(INLIST)]
2     TABLE ACCESS(FULL) OF 'SALES_LIST'[KEY(INLIST) --> KEY(INLIST)]
                                                          Card=2
```

```
SQL> truncate table plan_table;
Table truncated.

SQL> explain plan for
  2  select * from sales_list where sales_state > 'e';
Explained.

SQL> @./LABS/explain

RESULT
----------------------------------------------------------------
0 SELECT STATEMENT  Optimizer=CHOOSE Card=1
1   TABLE ACCESS(FULL) OF 'SALES_LIST'[2 --> 2]  Card=1

SQL> truncate table plan_table;
Table truncated.

SQL> explain plan for
  2  select * from sales_list where sales_state < 'e';
Explained.

SQL> @./LABS/explain

RESULT
----------------------------------------------------------------
0 SELECT STATEMENT  Optimizer=CHOOSE Card=3
1   PARTITION LIST(ALL)[LAST --> LAST]
2     TABLE ACCESS(FULL) OF 'SALES_LIST'[LAST --> LAST]  Card=3
```

6. How would you prove that the Oracle server really performed partition pruning in the above six cases?

   Answer: Execute the above six SELECT statements with AUTOTRACE on. For each statement, this gives you the number of db block gets executed by Oracle. Depending on the statement, you should see the differences. This way, you can see that the last statement only impacts two partitions out of three.

```
SQL> set autotrace on statistics
SQL> select * from sales_list;

SALESMAN_ID SALESMAN SALES_STATE          SALES_AMOUNT SALES_DAT
----------- -------- -------------------- ------------ ---------
         10 JFV      a                             100 05-JAN-01
         21 MH       f                             150 15-JAN-01
         32 Clement  c                             250 20-JAN-01

Statistics
----------------------------------------------------------------
          0  recursive calls
          6  db block gets
          3  consistent gets
          0  physical reads
          0  redo size
        806  bytes sent via SQL*Net to client
```

```
        519  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          3  rows processed

SQL> select * from sales_list where sales_state = 'a';

SALESMAN_ID SALESMAN SALES_STATE          SALES_AMOUNT SALES_DAT
----------- -------- -------------------- ------------ ---------
         10 JFV      a                             100 05-JAN-01

Statistics
----------------------------------------------------------
          0  recursive calls
          2  db block gets
          1  consistent gets
          0  physical reads
          0  redo size
        657  bytes sent via SQL*Net to client
        519  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed

SQL> select * from sales_list where sales_state in ('a','e');

SALESMAN_ID SALESMAN SALES_STATE          SALES_AMOUNT SALES_DAT
----------- -------- -------------------- ------------ ---------
         10 JFV      a                             100 05-JAN-01

Statistics
----------------------------------------------------------
          0  recursive calls
          4  db block gets
          2  consistent gets
          0  physical reads
          0  redo size
        657  bytes sent via SQL*Net to client
        519  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          2  sorts (memory)
          0  sorts (disk)
          1  rows processed

SQL> select * from sales_list where sales_state = 'a'
  2                              or sales_state = 'd';

SALESMAN_ID SALESMAN SALES_STATE          SALES_AMOUNT SALES_DAT
----------- -------- -------------------- ------------ ---------
         10 JFV      a                             100 05-JAN-01

Statistics
----------------------------------------------------------
          0  recursive calls
          4  db block gets
```

```
          2  consistent gets
          0  physical reads
          0  redo size
        657  bytes sent via SQL*Net to client
        519  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          2  sorts (memory)
          0  sorts (disk)
          1  rows processed

SQL> select * from sales_list where sales_state > 'e';

SALESMAN_ID SALESMAN SALES_STATE          SALES_AMOUNT SALES_DAT
----------- -------- -------------------- ------------ ---------
         21 MH       f                             150 15-JAN-01

Statistics
----------------------------------------------------------
          0  recursive calls
          2  db block gets
          1  consistent gets
          0  physical reads
          0  redo size
        657  bytes sent via SQL*Net to client
        519  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed

SQL> select * from sales_list where sales_state < 'e';

SALESMAN_ID SALESMAN SALES_STATE          SALES_AMOUNT SALES_DAT
----------- -------- -------------------- ------------ ---------
         10 JFV      a                             100 05-JAN-01
         32 Clement  c                             250 20-JAN-01

Statistics
----------------------------------------------------------
          0  recursive calls
          4  db block gets
          2  consistent gets
          0  physical reads
          0  redo size
        734  bytes sent via SQL*Net to client
        519  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
          2  rows processed
```

7. Connected as SYSTEM, and drop the `sales_list` table.

```
SQL> connect system/manager
Connected.
```

```
SQL> drop table sales_list;
Table dropped.
```

Practice 9-1 Solution: Using Automatic Free Space Management Segments

1. Connected as SYSTEM under SQL*Plus, create a new tablespace called BITMAP_SEGS with the following characteristics:
   - One 5 MB data file
   - Locally managed with uniform extents sizes of 100 KB
   - Containing only automatic free space management segments

   Then, query the DBA_TABLESPACES view in order to identify the different characteristics of each tablespace.

```
$ sqlplus system/manager
SQL*Plus: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> create tablespace bitmap_segs
  2  datafile '$HOME/ORADATA/u04/bitmap_segs01.dbf' size 5M
  3  extent management local uniform size 100K
  4  segment space management auto;
Tablespace created.

SQL> col tablespace_name format a12

SQL> select tablespace_name, block_size, initial_extent,
  2         contents, extent_management, allocation_type,
  3         segment_space_management
  4  from   dba_tablespaces;

TS_NAME BLOCK_SIZE INITIAL_EXTENT CONTENTS  EXTENT_MAN ALLOCATIO SEGMEN
------- ---------- -------------- --------- ---------- --------- ------
SYSTEM        4096          12288 PERMANENT DICTIONARY USER      MANUAL
UNDO1         4096          65536 UNDO      LOCAL      SYSTEM    MANUAL
INDX          4096          65536 PERMANENT LOCAL      SYSTEM    MANUAL
TEMP          4096        1048576 TEMPORARY LOCAL      UNIFORM   MANUAL
TOOLS         4096          65536 PERMANENT LOCAL      SYSTEM    MANUAL
USERS         4096          65536 PERMANENT LOCAL      SYSTEM    MANUAL
DRSYS         4096          20480 PERMANENT DICTIONARY USER      MANUAL
SAMPLE        4096          20480 PERMANENT DICTIONARY USER      MANUAL
CWMLITE       4096          20480 PERMANENT DICTIONARY USER      MANUAL
BITMAP_SEGS   4096         102400 PERMANENT LOCAL      UNIFORM   AUTO

10 rows selected.
```

2. Drop the already existing TEST table and create a new table called TEST in the BITMAP_SEGS tablespace. This table should only have one NUMBER column called C, and only one extent. Verify the size of the newly created extent.

```
SQL> drop table test;

SQL> create table test(c number)
  2  storage (minextents 1)
  3  tablespace bitmap_segs;
Table created.

SQL> select extent_id,block_id,blocks
```

```
  2  from dba_extents where segment_name='TEST';

 EXTENT_ID   BLOCK_ID     BLOCKS
---------- ---------- ----------
         0         17         25
```

3. Execute the `lab_09_01.sql` script in order to insert rows into the TEST table. Once done, determine the number of rows stored in each block of the table.

```
SQL> begin
  2    for b in 1..21 loop
  3       for i in 1..324 loop
  4          insert into test values(i);
  5       end loop;
  6    end loop;
  7  end;
  8  /
PL/SQL procedure successfully completed.

SQL> commit;
Commit complete.

SQL> select   DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid), count(*)
  2  from     test
  3  group by DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid);

DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)   COUNT(*)
------------------------------------ ----------
                                  21        324
                                  22        324
                                  23        324
                                  24        324
                                  25        324
                                  26        324
                                  27        324
                                  28        324
                                  29        324
                                  30        324
                                  31        324
                                  32        324
                                  33        324
                                  34        324
                                  35        324
                                  36        324
                                  37        324
                                  38        324
                                  39        324
                                  40        324
                                  41        324
21 rows selected.
```

4. Delete all rows of the TEST table having C < 70. Then, commit your changes. Once done, determine the number of rows stored in each block of the table. What do you observe?

Answer: The delete operation deletes almost 30% of the rows stored originally in each data block of the TEST table. Nevertheless, the number of blocks containing rows is still the same as in step 3.

```
SQL> delete test where c < 70;
1449 rows deleted.

SQL> commit;
Commit complete.

SQL> select   DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid), count(*)
  2  from     test
  3  group by DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid);

DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)   COUNT(*)
------------------------------------ ----------
                                  21        255
                                  22        255
                                  23        255
                                  24        255
                                  25        255
                                  26        255
                                  27        255
                                  28        255
                                  29        255
                                  30        255
                                  31        255
                                  32        255
                                  33        255
                                  34        255
                                  35        255
                                  36        255
                                  37        255
                                  38        255
                                  39        255
                                  40        255
                                  41        255
21 rows selected.
```

5. Although this step is optional, it is recommended that you do it in order to be sure of the lab's result. Connect as SYSDBA and fix the data blocks bimap status of the test table by using the SEGMENT_FIX_STATUS procedure of the DBMS_REPAIR package. Connect again as SYSTEM and execute the lab_09_02.sql script in order to insert back rows into the TEST table. Once done, determine and note the number of rows stored in each block of the table. What do you observe?

   Answer: You are now back to almost the same situation as in step 3. That is, the number of blocks containing rows is the same as in step 3.

```
SQL> connect / as sysdba;
Connected.

SQL> execute dbms_repair.segment_fix_status('SYSTEM','TEST');
PL/SQL procedure successfully completed.
```

```
SQL> connect system/manager
Connected.

SQL> begin
  2    for b in 1..21 loop
  3       for i in 1..65 loop
  4          insert into test values(i);
  5       end loop;
  6    end loop;
  7  end;
  8  /
PL/SQL procedure successfully completed.

SQL> commit;
Commit complete.

SQL> select DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid), count(*)
  2  from test
  3  group by DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid);

DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)   COUNT(*)
------------------------------------ ----------
                                  21        324
                                  22        324
                                  23        324
                                  24        324
                                  25        324
                                  26        324
                                  27        324
                                  28        324
                                  29        324
                                  30        324
                                  31        324
                                  32        324
                                  33        255
                                  34        324
                                  35        324
                                  36        324
                                  37        324
                                  38        324
                                  39        324
                                  40        324
                                  41        309
21 rows selected.
```

6. Create a new tablespace called USERS2 with only one 5 MB data file and with manual segment space management set. Create a new table called TEST1 with the same structure as the TEST table but stored in the USERS2 tablespace. Then edit the `lab_09_01.sql` script and change all occurrence of the word “TEST” to the word “TEST1.” Save the edited script and execute it. Once done, determine the number of rows stored in each block of the table. What do you observe?

   Answer: The situation is almost the same as in step 3. That is, the number of blocks containing data is 21 and the number of rows per block is 325. The difference between the

number of rows per block is due to the fact that the USERS2 tablespace is a normal one, not defined for handling automatic free space management segments.

```
SQL> create tablespace users2
  2  datafile '$HOME/ORADATA/u04/users2_01.dbf' size 5M;
Tablespace created.

SQL> create table test1(c number)
  2  storage (minextents 1)
  3  tablespace users2;
Table created.

SQL> begin
  2    for b in 1..21 loop
  3       for i in 1..324 loop
  4          insert into test1 values(i);
  5       end loop;
  6    end loop;
  7  end;
  8  /
PL/SQL procedure successfully completed.

SQL> commit;
Commit complete.

SQL> select   DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid), count(*)
  2  from     test1
  3  group by DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid);

DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)   COUNT(*)
------------------------------------ ----------
                                  34        325
                                  35        325
                                  36        325
                                  37        325
                                  38        325
                                  39        325
                                  40        325
                                  41        325
                                  42        325
                                  43        325
                                  44        325
                                  45        325
                                  46        325
                                  47        325
                                  48        325
                                  49        325
                                  50        325
                                  51        325
                                  52        325
                                  53        325
                                  54        304
21 rows selected.
```

7. Repeat step 4 but this time for the TEST1 table instead of the TEST table. What do you observe?

Answer: Same observation as in step 4.

```
SQL> delete test1 where c<70;
1449 rows deleted.

SQL> commit;
Commit complete.

SQL> select   DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid), count(*)
  2  from     test1
  3  group by DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid);

DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)   COUNT(*)
------------------------------------ ----------
                                  34        255
                                  35        255
                                  36        255
                                  37        255
                                  38        255
                                  39        255
                                  40        255
                                  41        255
                                  42        255
                                  43        255
                                  44        255
                                  45        255
                                  46        255
                                  47        255
                                  48        255
                                  49        255
                                  50        255
                                  51        255
                                  52        255
                                  53        255
                                  54        255
21 rows selected.
```

8. Edit the `lab_09_02.sql` script and change all occurrences of the word "TEST" to the word "TEST1." Save the edited script and execute it. Once done, determine the number of rows stored in each block of the table. What is your conclusion?

   Answer: The number of blocks grew from 21 to 25. This is due to the fact that the `TEST1` table is managed by free lists. When it was defined, `TEST1` inherited a `PCTUSED` value of 40%, and due to the previous delete operation on the `TEST1` table, the newly inserted rows are not reinserted into the previous blocks, which was the case for the `TEST` table which is managed with bitmaps. Thus, because the `PCTUSED` is not reached after the delete, the insert operation is using new blocks to store the data. This is a waste of space inside the segment.

```
SQL> begin
  2    for b in 1..21 loop
  3       for i in 1..65 loop
  4          insert into test1 values(i);
```

```
  5       end loop;
  6    end loop;
  7  end;
  8  /
PL/SQL procedure successfully completed.

SQL> commit;
Commit complete.

SQL> select   DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid), count(*)
  2  from     test1
  3  group by DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid);

DBMS_ROWID.ROWID_BLOCK_NUMBER(ROWID)   COUNT(*)
------------------------------------ ----------
                                  34        255
                                  35        255
                                  36        255
                                  37        255
                                  38        255
                                  39        255
                                  40        255
                                  41        255
                                  42        255
                                  43        255
                                  44        255
                                  45        255
                                  46        255
                                  47        255
                                  48        255
                                  49        255
                                  50        255
                                  51        255
                                  52        255
                                  53        255
                                  54        325
                                  55        325
                                  56        325
                                  57        325
                                  58        320
25 rows selected.
```

9. Connected as SYSTEM, drop the BITMAP_SEGS and USERS2 tablespaces. Remove also their corresponding data files.

```
SQL> connect system/manager
Connected.

SQL> drop tablespace bitmap_segs including contents and datafiles;
Tablespace dropped.

SQL> drop tablespace users2 including contents and datafiles;
Tablespace dropped.
```

Practice 10-1 Solution: Monitoring Indexes

1. The goal of this lab is to explain how to monitor index usage.
   First of all connect as user HR under SQL*Plus.

```
$ sqlplus hr/hr
SQL*Plus: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production
```

2. Create the PLAN_TABLE table using the `utlxplan.sql` script located under `$ORACLE_HOME/rdbms/admin`.

```
SQL> @$ORACLE_HOME/rdbms/admin/utlxplan
Table created.
```

3. Create the EMP table as a simple select from the EMPLOYEES table.

```
SQL> create table emp as select * from employees;
Table created.
```

4. Create the IEMAIL index on the EMAIL column of the EMP table.

```
SQL> create index iemail on emp(email);
Index created.
```

5. Use the V$OBJECT_USAGE view in order to determine if there are indexes currently monitored in the HR account.

```
SQL> select * from v$object_usage;
no rows selected
```

6. Monitor IEMAIL index usage.

```
SQL> alter index iemail monitoring usage;
Index altered.
```

7. Verify that the IEMAIL index is being monitored.

```
SQL> select * from v$object_usage;

INDEX_NAME           TABLE_NAME           MON USE
-------------------- -------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL               EMP                  YES NO
07/04/2001 06:27:29
```

8. Connected as SYSDBA, try to verify that IEMAIL is being monitored. What is your conclusion?

Answer: V$OBJECT_USAGE shows you only monitored indexes pertaining to the connected account from where the query was executed.

```
SQL> connect / as sysdba;
Connected.

SQL> select * from v$object_usage;
no rows selected
```

9. Connect again under the HR account and delete all rows of the EMP table. Do not commit or roll back your changes.

```
SQL> connect hr/hr
Connected.

SQL> delete emp;
107 rows deleted.
```

10. Verify if the index was considered as being used by the previous command. Then roll back your changes and verify again the usage status of the IEMAIL index. What is your conclusion?

    Answer: The monitoring usage functionality is not concerned by DML statements executed against the monitored indexes.

```
SQL> select * from v$object_usage;

INDEX_NAME          TABLE_NAME          MON USE
------------------- ------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL              EMP                 YES NO
07/04/2001 06:27:29

SQL> rollback;
Rollback complete.

SQL> select * from v$object_usage;

INDEX_NAME          TABLE_NAME          MON USE
------------------- ------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL              EMP                 YES NO
07/04/2001 06:27:29
```

11. Execute a select statement against the EMP table that uses the IEMAIL index. Then verify the usage status of the IEMAIL index. What is your conclusion?

    Answer: The monitoring usage functionality takes into account queries executed against monitored indexes.

```
SQL> select /*+ INDEX(EMP IEMAIL) */ count(*) from emp;

  COUNT(*)
----------
       107

SQL> select * from v$object_usage;

INDEX_NAME          TABLE_NAME          MON USE
------------------- ------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL              EMP                 YES YES
07/04/2001 06:27:29
```

12. Verify that the contents of the V$OBJECT_USAGE view persist across an instance crash.

```
SQL> connect / as sysdba;
Connected.

SQL> shutdown abort;
ORACLE instance shut down.

SQL> startup pfile=$HOME/ADMIN/PFILE/initjfv.ora;
ORACLE instance started.

Total System Global Area   25991956 bytes
Fixed Size                   285460 bytes
Variable Size              20971520 bytes
Database Buffers            4194304 bytes
Redo Buffers                 540672 bytes
Database mounted.
Database opened.

SQL> connect hr/hr
Connected.

SQL> select * from v$object_usage;

INDEX_NAME          TABLE_NAME          MON USE
------------------- ------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL              EMP                 YES YES
07/04/2001 06:27:29
```

13. Drop the EMP table and analyze the impact on the monitored index. What is the consequence?

    Answer: Because you dropped the EMP table, you implicitly dropped the IEMAIL index. The corresponding entry in V$OBJECT_USAGE view is also deleted.

```
SQL> drop table emp;
Table dropped.
```

```
SQL> select * from v$object_usage;
no rows selected
```

14. Re-create the same objects as in previous steps and activate monitoring usage again on the IEMAIL index.

```
SQL> create table emp as select * from employees;
Table created.

SQL> create index iemail on emp(email);
Index created.

SQL> select * from v$object_usage;
no rows selected

SQL> alter index iemail monitoring usage;
Index altered.

SQL> select * from v$object_usage;

INDEX_NAME          TABLE_NAME          MON USE
------------------- ------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL              EMP                 YES NO
07/04/2001 06:33:19
```

15. Use the EXPLAIN PLAN command to analyze the query you executed in step 11. Again analyze monitoring usage on the IEMAIL index. What is your conclusion?

    Answer: The USED flag in the V$OBJECT_USAGE view is positioned to YES at queries parse time.

```
SQL> explain plan for
  2  select /*+ INDEX(EMP IEMAIL) */ count(*) from emp;
Explained.

SQL> select operation,options,object_name from plan_table;

OPERATION         OPTIONS    OBJECT_NAME
----------------- ---------- ------------------------------
SELECT STATEMENT
SORT              AGGREGATE
INDEX             FULL SCAN  IEMAIL

SQL> select * from v$object_usage;

INDEX_NAME          TABLE_NAME          MON USE
------------------- ------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL              EMP                 YES YES
07/04/2001 06:33:19
```

16. Turn off monitoring usage on the IEMAIL index and look at the V$OBJECT_USAGE view. What is your conclusion?

    Answer: Based on the results of step 13, a monitoring entry in the V$OBJECT_USAGE view is only deleted after the drop of the corresponding index. Turning off monitoring just updates the MONITORING flag.

```
SQL> alter index iemail nomonitoring usage;
Index altered.

SQL> select * from v$object_usage;

INDEX_NAME          TABLE_NAME          MON USE
------------------- ------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL              EMP                 NO  YES
07/04/2001 06:33:19 07/04/2001 06:34:30
```

17. Drop the IEMAIL index and verify that the corresponding entry in the V$OBJECT_USAGE view has been deleted.

```
SQL> drop index iemail;
Index dropped.

SQL> select * from v$object_usage;
no rows selected
```

18. Re-create the IEMAIL index and activate its monitoring usage again. Then reexecute the query from step 11.

```
SQL> create index iemail on emp(email);
Index created.

SQL> alter index iemail monitoring usage;
Index altered.

SQL> select * from v$object_usage;

INDEX_NAME          TABLE_NAME          MON USE
------------------- ------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL              EMP                 YES NO
07/04/2001 06:39:03

SQL> select /*+ INDEX(EMP IEMAIL) */ count(*) from emp;

  COUNT(*)
----------
       107

SQL> select * from v$object_usage;
```

```
INDEX_NAME           TABLE_NAME           MON USE
-------------------- -------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL               EMP                  YES YES
07/04/2001 06:39:03
```

19. Turn off monitoring usage on the IEMAIL index, then activate it again. What are the consequences on the V$OBJECT_USAGE view?

    In the previous step, the MONITORING and USED columns were set to YES. Turning monitoring off updates *only* the MONITORING flag. Activating monitoring again resets both columns.

```
SQL> alter index iemail nomonitoring usage;
Index altered.

SQL> select * from v$object_usage;

INDEX_NAME           TABLE_NAME           MON USE
-------------------- -------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL               EMP                  NO  YES
07/04/2001 06:39:03 07/04/2001 06:39:50

SQL> alter index iemail monitoring usage;
Index altered.

SQL> select * from v$object_usage;

INDEX_NAME           TABLE_NAME           MON USE
-------------------- -------------------- --- ---
START_MONITORING    END_MONITORING
------------------- -------------------
IEMAIL               EMP                  YES NO
07/04/2001 06:40:14
```

20. Connected as HR, drop the EMP table and the PLAN_TABLE table.

```
SQL> connect hr/hr
Connected.

SQL> drop table emp;
Table dropped.

SQL> drop table plan_table;
Table dropped.
```

Practice 10-2 Solution: Cursor Sharing Enhancements

1. The goal of this lab is to explain the usage of the new cursor sharing optimization. Create the PLAN_TABLE under the SYSTEM account. Then, create a simple table T under the SYSTEM user in the USERS tablespace. It is enough to create a table with only one column (C) of NUMBER data type. Once done, insert the same value (1111111111111111) many times into this table. Once done, commit your modification (in order to insert the data you should use the lab_09_01.sql script located in your LABS directory). The idea is to generate data skew in the table.

```
SQL> connect system/manager
Connected.

SQL> @$ORACLE_HOME/rdbms/admin/utlxplan.sql
Table created.

SQL> create table t(c number) tablespace users;
Table created.

SQL> insert into t values(1111111111111111);
1 row created.

SQL> commit;
Commit complete.

SQL> insert into t select * from t;
1 row created.

SQL> / (    2 rows created.)
SQL> / (    4 rows created.)
SQL> / (    8 rows created.)
SQL> / (   16 rows created.)
SQL> / (   32 rows created.)
SQL> / (   64 rows created.)
SQL> / (  128 rows created.)
SQL> / (  256 rows created.)
SQL> / (  512 rows created.)
SQL> / ( 1024 rows created.)
SQL> / ( 2048 rows created.)
SQL> / ( 4096 rows created.)
SQL> / ( 8192 rows created.)
SQL> / (16384 rows created.)
SQL> / (32768 rows created.)

SQL> commit;
Commit complete.
```

2. In order to generate data skew in the table, insert a last row with a completely different value (222222222222222) from the previous ones. Commit this modification also. You are now left with an unbalanced repartition of the rows inside the T table.

```
SQL> insert into t values(222222222222222);
1 row created.
```

```
SQL> commit;
Commit complete.
```

3. Create a simple index called IT on the C column of the T table in the USERS tablespace. Then, gather statistics for table T with the dbms_stats.gather_table_stats procedure.

```
SQL> create index it on t(c) tablespace users;
Index created.

SQL> execute dbms_stats.gather_table_stats(USER, 'T', cascade => true);
PL/SQL procedure successfully completed.
```

4. Explain the following query with the two different values residing inside the T table:
   SQL> select count(*) from t where c = ...;
   What is your conclusion?

   Answer: It appears that the Oracle server uses the same access paths for both values. This is the expected behavior as no histograms were generated so far.

```
SQL> explain plan for
  2  select count(*) from t where c = 222222222222222;
Explained.

SQL> select operation,options from plan_table;

OPERATION          OPTIONS
------------------ --------------------------------
SELECT STATEMENT
SORT               AGGREGATE
TABLE ACCESS       FULL

SQL> truncate table plan_table;
Table truncated.

SQL> explain plan for
  2  select count(*) from t where c = 1111111111111111;
Explained.

SQL> select operation,options from plan_table;

OPERATION          OPTIONS
------------------ ---------
SELECT STATEMENT
SORT               AGGREGATE
TABLE ACCESS       FULL
```

5. Change the value of the CURSOR_SHARING initialization parameter for your session so that it uses the new possible value: SIMILAR. Once done, flush the contents of the shared pool.

```
SQL> alter session set cursor_sharing = similar;
```

```
Session altered.

SQL> alter system flush shared_pool;
System altered.
```

6. Now, execute the same select statement as in step 4 with the two different values.

```
SQL> select count(*) from t where c = 222222222222222;

  COUNT(*)
----------
         1

SQL> select count(*) from t where c = 1111111111111111;

  COUNT(*)
----------
     65536
```

7. Identify in the shared pool the corresponding SQL texts generated for the execution of the two statements above. What happened and why?

   Answer: Using the SIMILAR value for the CURSOR_SHARING parameter instructs the Oracle server to share the two cursors if the generated optimizer plans are guaranteed to be similar for both queries. As you have seen before, the Oracle server uses the same optimizer plan for both queries, and thus it is safe to share the same cursor for both statements. The sharing is possible because the Oracle server replaced the two different literals with the same system-generated bind variable.

```
SQL> select sql_text
  2  from   v$sql
  2  where  sql_text like '%count(*) from t%';

SQL_TEXT
----------------------------------------
select count(*) from t where c=:"SYS_B_0"
```

8. In order for the optimizer to possibly find data skew in the T table, gather table statistics again and also generate a histogram with 75 buckets for column C. After that, flush the shared pool again.

```
SQL> execute dbms_stats.gather_table_stats(USER, 'T', cascade => true, -
>    method_opt => 'for all columns size 75');
PL/SQL procedure successfully completed.

SQL> alter system flush shared_pool;
System altered.
```

9. Repeat steps 6 and 7. What is your conclusion?

Answer: Now that you have generated histogram statistics, you can clearly see that the Oracle server uses two different child cursors in order to execute the two different queries. The Oracle server is now able to identify data skew in table T, and thus generates two different optimizer plans. This prevents cursor sharing. Note that nevertheless the Oracle server replaced both literals with system-generated bind variables. This is because you used SIMILAR as opposed to EXACT for the CURSOR_SHARING parameter.

```
SQL> select count(*) from t where c = 222222222222222;

  COUNT(*)
----------
         1

SQL> select count(*) from t where c = 1111111111111111;

  COUNT(*)
----------
     65536

SQL> select sql_text,literal_hash_value
  2  from   v$sql
  3  where  sql_text like '%count(*) from t%';

SQL_TEXT                                    LITERAL_HASH_VALUE
------------------------------------------- ------------------
select count(*) from t where c=:"SYS_B_0"          2806080606
select count(*) from t where c=:"SYS_B_0"          3912347465
```

10. How would you prove that two different optimizer plans were used by the Oracle server to execute the previous two queries?

Answer: Using the new V$SQL_PLAN view as shown below.

```
SQL> col object_name format a5
SQL> col operation   format a16
SQL> col options     format a15

SQL> select address,hash_value,child_number,
  2         operation,options,object_name
  3  from   v$sql_plan
  4  where (address,hash_value) in
  5        (select address,hash_value
  6         from   v$sql
  7         where  sql_text like '%count(*) from t%');

                    CHILD_
ADDRESS  HASH_VALUE NUMBER OPERATION        OPTIONS      OBJECT_NAME
-------- ---------- ------ ---------------- ------------ -----------
80AFB7D4 1195701597      1 SELECT STATEMENT
80AFB7D4 1195701597      1 SORT             AGGREGATE
80AFB7D4 1195701597      1 TABLE ACCESS     FULL         T
80AFB7D4 1195701597      0 SELECT STATEMENT
80AFB7D4 1195701597      0 SORT             AGGREGATE
80AFB7D4 1195701597      0 INDEX            RANGE SCAN   IT
```

```
6 rows selected.
```

11. Connect as SYSTEM, and drop table T.

```
SQL> connect system/manager
Connected.

SQL> drop table t;
Table dropped.
```

Practice 13-1 Solution: Create OMF and Non-OMF Files in the Same Database

1. Under your HOME directory, create two new subdirectories called OMF1 and OMF2 respectively. Then connect as SYSDBA under SQL*Plus.

```
$ mkdir OMF1
$ mkdir OMF2
$ sqlplus /nolog
SQL*Plus: Release 9.0.1.0.0 - Production

SQL> connect / as sysdba;
Connected.
```

2. Try to create tablespace OMF1 with the following command:

   ```
   SQL> CREATE TABLESPACE omf1 DATAFILE SIZE 10M AUTOEXTEND OFF;
   ```

   What happens and why?

   Answer: It is not possible to use this OMF syntax as the DB_CREATE_FILE_DEST initialization parameter is NULL.

```
SQL> show parameter db_create_file_dest

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------
db_create_file_dest                  string

SQL> CREATE TABLESPACE omf1 DATAFILE SIZE 10M AUTOEXTEND OFF;
CREATE TABLESPACE omf1 DATAFILE SIZE 10M AUTOEXTEND OFF
                                 *
ERROR at line 1:
ORA-02236: invalid file name
```

3. How would you modify your environment to allow the previous command to create OMF files in the $HOME/OMF1 directory?
   Create tablespace OMF1 and verify that an OMF file was created in the $HOME/OMF1 directory.

   Answer: In order to allow an OMF specification to work, you must at least modify the DB_CREATE_FILE_DEST parameter. To answer the question, you must assign it the '$HOME/OMF1/' value.

```
SQL> alter session set db_create_file_dest='$HOME/OMF1/';
Session altered.

SQL> CREATE TABLESPACE omf1 DATAFILE SIZE 10M AUTOEXTEND OFF;
Tablespace created.

SQL> host ls ./OMF1
```

```
ora_omf1_xcp6rg84.dbf
```

4. Now, change the DB_CREATE_FILE_DEST value to point to $HOME/OMF2 directory. Then drop tablespace OMF1. What happens?

   Answer: Because OMF1 was defined as containing OMF files, the Oracle server drops the created OMF files. This does not depend on the OMF directory specified by the DB_CREATE_FILE_DEST parameter.

```
SQL> alter session set db_create_file_dest='$HOME/OMF2/';
Session altered.

SQL> drop tablespace omf1;
Tablespace dropped.

SQL> host ls ./OMF1

```

5. How would you verify all this?

   Answer: Look at the alert.log file for this instance.

```
SQL> host vi $HOME/ADMIN/BDUMP/alert*
"./dump/alert_ed25.log" 888 lines, 33868 characters
Mon Jul 02 09:19:18 2001
ORA-1119 signalled during: CREATE TABLESPACE omf1 DATAFILE SIZE 10M
AUTOEXTE...
Fri Jul 06 01:44:46 2001
CREATE TABLESPACE omf1 DATAFILE SIZE 10M AUTOEXTEND OFF
Fri Jul 06 01:44:47 2001
Created Oracle managed file /databases/ed25/OMF1/ora_omf1_xcp6rg84.dbf
Completed: CREATE TABLESPACE omf1 DATAFILE SIZE 10M AUTOEXTE
Fri Jul 06 01:45:45 2001
drop tablespace omf1
Fri Jul 06 01:45:45 2001
Deleted Oracle managed file /databases/ed25/OMF1/ora_omf_xcp6rg84.dbf
Completed: drop tablespace omf1
```

6. Create the same OMF1 tablespace again and verify that the Oracle server creates the corresponding OMF file in the $HOME/OMF2 directory.

```
SQL> CREATE TABLESPACE omf1 DATAFILE SIZE 10M AUTOEXTEND OFF;
Tablespace created.

SQL> host ls ./OMF2
ora_omf1_xcp6xrvr.dbf
```

7. In order to demonstrate how you can migrate an OMF file to a non-OMF file:

   - Take the OMF1 tablespace offline
   - Make an O/S copy of the created OMF file in the same directory. Call it omf1.dbf

- Use the appropriate ALTER TABLESPACE command in order to rename the OMF data file to the newly created file
- Bring the OMF1 tablespace back online

What happens and why?

Answer: When you rename an OMF data file, the Oracle server automatically deletes the old OMF copy. This is important to realize, especially if you intended to use the old version as a backup.

```
SQL> alter tablespace omf1 offline;
Tablespace altered.

SQL> host cp ./OMF2/ora_omf1_xcp6xrvr.dbf ./OMF2/omf1.dbf

SQL> host ls ./OMF2
ora_omf1_xcp6xrvr.dbf  omf1.dbf

SQL> alter tablespace omf1 rename datafile
  2     '$HOME/OMF2/ora_omf1_xcp6xrvr.dbf'
  3  to '$HOME/OMF2/omf1.dbf';
Tablespace altered.

SQL> host ls ./OMF2
omf1.dbf

SQL> alter tablespace omf1 online;
Tablespace altered.
```

8. How would you drop tablespace OMF1, including its data file, at the O/S level?

   Answer: By using the AND DATAFILES clause of the DROP TABLESPACE command. This is because tablespace OMF1 contains at least one non-OMF data file.

```
SQL> DROP TABLESPACE omf1 INCLUDING CONTENTS AND DATAFILES
  2  CASCADE CONSTRAINTS;
Tablespace dropped.

SQL> host ls ./OMF2
```

Practice 13-2 Solution: Using the Default Temporary Tablespace Assignment

1. Switch the database default temporary tablespace to SYSTEM. How would you determine the current default temporary tablespace used in your database?

   Answer: By looking at the PROPERTY_VALUE column of the DATABASE_PROPERTIES view for the property name 'DEFAULT_TEMP_TABLESPACE'.

```
SQL> alter database default temporary tablespace system;
Database altered.

SQL> SELECT PROPERTY_VALUE
  2  FROM   DATABASE_PROPERTIES
  3  WHERE  PROPERTY_NAME= 'DEFAULT_TEMP_TABLESPACE';

PROPERTY_VALUE
--------------
SYSTEM
```

2. Create a user without assigning a temporary tablespace. Then, verify that this user was automatically assigned the previous default temporary tablespace.

```
SQL> create user jfv identified by jfv
  2  default tablespace sample;
User created.

SQL> select temporary_tablespace from dba_users where username='JFV';

TEMPORARY_TABLESPACE
--------------------
SYSTEM
```

3. Now, change the default temporary tablespace for your database to the TEMP tablespace and verify that the previously created user inherits the new database default temporary tablespace.

```
SQL> alter database default temporary tablespace temp;
Database altered.

SQL> select temporary_tablespace from dba_users where username='JFV';

TEMPORARY_TABLESPACE
--------------------
TEMP
```

4. Create a tablespace called DEF1. Make sure that this tablespace is dictionary managed with a file size of 10 MB. Once done, make it the database default temporary tablespace. What happens and why?

   Answer: Because the DEF1 tablespace is not defined as TEMPORARY, it is not possible to use it as the database default temporary tablespace.

```
SQL> CREATE TABLESPACE def1
  2  DATAFILE '$HOME/ORADATA/u04/def1.dbf' SIZE 10M
  3  EXTENT MANAGEMENT DICTIONARY;
Tablespace created.

SQL> alter database default temporary tablespace def1;
alter database default temporary tablespace def1
*
ERROR at line 1:
ORA-12902: default temporary tablespace must be SYSTEM
           or of TEMPORARY type
```

5. How would you modify this situation in order to make the DEF1 tablespace the database default temporary tablespace?

```
SQL> alter tablespace def1 temporary;
Tablespace altered.

SQL> alter database default temporary tablespace def1;
Database altered.
```

6. Once tablespace DEF1 is the database default temporary tablespace, how would you drop tablespace DEF1?

```
SQL> alter tablespace def1 offline;
alter tablespace def1 offline
*
ERROR at line 1:
ORA-12905: default temporary tablespace cannot be brought OFFLINE

SQL> drop tablespace def1;
drop tablespace def1
*
ERROR at line 1:
ORA-12906: cannot drop default temporary tablespace

SQL> alter database default temporary tablespace temp;
Database altered.

SQL> drop tablespace def1;
Tablespace dropped.
```

Practice 14-1 Solution: Maintaining Automatic Undo Management Tablespaces

1. Connected as SYSTEM under SQL*Plus, execute the lab_14_01.sql script in order to create some tables into the SYSTEM account. It is assumed that the USERS tablespace exists and has at least 1 MB free space. After executing the script, change the retention period of your instance to one hour.

```
$ sqlplus system/manager
SQL*Plus: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> drop table t;
SQL> drop table t1;
SQL> drop table t2;
SQL> drop table t3;

SQL> create table t(c varchar2(50)) tablespace users;
Table created.

SQL> insert into t
  2  values('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
1 row created.

SQL> commit;
Commit complete.

SQL> insert into t select * from t;
1 row created.

SQL> /  ( 2 rows created.)
SQL> /  ( 4 rows created.)
SQL> /  ( 8 rows created.)
SQL> /  (16 rows created.)
SQL> /  (32 rows created.)
SQL> /  (64 rows created.)

SQL> commit;
Commit complete.

SQL> create table t1 tablespace users as select * from t;
Table created.

SQL> create table t2 tablespace users as select * from t;
Table created.

SQL> alter system set undo_retention=3600;
System altered.
```

2. Create a new undo tablespace called UNDO02 containing only one 60 KB data file. What happens and why?

   Answer: 60 KB seems to be too small for the Oracle server to create this undo tablespace. Remember that an undo tablespace is managed with bitmaps and that at least one undo segment containing two extents must be created.

```
SQL> create undo tablespace undo02
  2  datafile '$HOME/ORADATA/u04/undo02_01.dbf' size 60K;
create undo tablespace undo02
*
ERROR at line 1:
ORA-03214: File Size specified is smaller than minimum required
```

3. Repeat the previous step but this time with a data file size of 80 KB instead of 60 KB.

```
SQL> create undo tablespace undo02
  2  datafile '$HOME/ORADATA/u04/undo02_01.dbf' size 80K;
Tablespace created.
```

4. Determine how many undo segments are currently created. Also, determine their current status. Then, change the active undo tablespace to be the UNDO02 tablesapce. Once done, try to delete the SYSTEM.T table. What happens and why?

   Answer: The Oracle server was able to create the 80 KB undo tablespace. Nevertheless, no undo segment was created in this undo tablespace. This means that the UNDO02 tablespace is still too small to create an undo segment. This is verified when trying to delete the SYSTEM.T table. Because it is impossible to create an undo segment in the undo tablespace UNDO02, the Oracle server tries to use the SYSTEM rollback segment which results in an error message as table SYSTEM.T does not reside in the SYSTEM tablespace.

```
SQL> select tablespace_name,segment_name,status
  2  from   dba_rollback_segs;

TABLESPACE_NAME                SEGMENT_NAME    STATUS
------------------------------ --------------- -------
SYSTEM                         SYSTEM          ONLINE
UNDO1                          _SYSSMU1$       ONLINE
UNDO1                          _SYSSMU2$       ONLINE
UNDO1                          _SYSSMU3$       ONLINE
UNDO1                          _SYSSMU4$       ONLINE
UNDO1                          _SYSSMU5$       ONLINE
UNDO1                          _SYSSMU6$       ONLINE
UNDO1                          _SYSSMU7$       ONLINE
UNDO1                          _SYSSMU8$       ONLINE

9 rows selected.

SQL> alter system set undo_tablespace=undo02;
System altered.

SQL> delete t;
delete t
       *
ERROR at line 1:
ORA-01552: cannot use system rollback segment
           for non-system tablespace 'USERS'
```

5. Now switch back to the UNDO1 tablespace as the active undo tablespace. Then drop and re-create the UNDO02 tablespace but this time with a 400 KB data file. Once done, determine the number of undo segments and their statuses. What is your conclusion?

   Answer: This time, the Oracle server is able to create two additional undo segments in the UNDO02 tablespace. Nevertheless, these two undo segments cannot be used because the active undo tablespace is UNDO1. That is why the two newly created undo segments show an OFFLINE status.

```
SQL> alter system set undo_tablespace=UNDO1;
System altered.

SQL> drop tablespace undo02 including contents and datafiles;
Tablespace dropped.

SQL> create undo tablespace undo02
  2  datafile '$HOME/ORADATA/u04/undo02_01.dbf ' size 400K;
Tablespace created.

SQL> select tablespace_name,segment_name,status
  2  from dba_rollback_segs;

TABLESPACE_NAME                SEGMENT_NAME    STATUS
------------------------------ --------------- -------
SYSTEM                         SYSTEM          ONLINE
UNDO1                          _SYSSMU1$       ONLINE
UNDO1                          _SYSSMU2$       ONLINE
UNDO1                          _SYSSMU3$       ONLINE
UNDO1                          _SYSSMU4$       ONLINE
UNDO1                          _SYSSMU5$       ONLINE
UNDO1                          _SYSSMU6$       ONLINE
UNDO1                          _SYSSMU7$       ONLINE
UNDO1                          _SYSSMU8$       ONLINE
UNDO02                         _SYSSMU9$       OFFLINE
UNDO02                         _SYSSMU10$      OFFLINE

11 rows selected.
```

6. Change the active undo tablespace to be the UNDO02 tablespace. Then, look at the undo segments statuses. What happened?

   Answer: By switching the undo tablespace to UNDO02, all undo segments residing in the UNDO1 tablespace were offline. Whereas the ones residing in the UNDO02 tablespace were online. Note the SYSTEM rollback segment is always ONLINE and cannot be offline.

```
SQL> alter system set undo_tablespace=undo02;
System altered.

SQL> select tablespace_name,segment_name,status
  2  from dba_rollback_segs;

TABLESPACE_NAME                SEGMENT_NAME    STATUS
------------------------------ --------------- -------
```

```
SYSTEM                         SYSTEM          ONLINE
UNDO1                          _SYSSMU1$       OFFLINE
UNDO1                          _SYSSMU2$       OFFLINE
UNDO1                          _SYSSMU3$       OFFLINE
UNDO1                          _SYSSMU4$       OFFLINE
UNDO1                          _SYSSMU5$       OFFLINE
UNDO1                          _SYSSMU6$       OFFLINE
UNDO1                          _SYSSMU7$       OFFLINE
UNDO1                          _SYSSMU8$       OFFLINE
UNDO02                         _SYSSMU9$       ONLINE
UNDO02                         _SYSSMU10$      ONLINE

11 rows selected.
```

7. Execute the `lab_14_02.sql` script in order to delete rows as well as creating and populating one additional table in the `SYSTEM` schema. This is done to get assigned to a transaction table in the `UNDO02` tablespace. Using `V$TRANSACTION`, check to which transaction table you were assigned. Once done, commit your changes.

```
SQL> delete t;
128 rows deleted.

SQL> delete t2;
128 rows deleted.

SQL> create table t3(c varchar2(50)) tablespace users;
Table created.

SQL> insert into t3 values
  2  ('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
1 row created.

SQL> insert into t3 select * from t3;
1 row created.

SQL> / (   2 rows created.)
SQL> / (   4 rows created.)
SQL> / (   8 rows created.)
SQL> / (  16 rows created.)
SQL> / (  32 rows created.)
SQL> / (  64 rows created.)
SQL> / ( 128 rows created.)
SQL> / ( 256 rows created.)
SQL> / ( 512 rows created.)
SQL> / (1024 rows created.)
SQL> / (2048 rows created.)
SQL> / (4096 rows created.)
SQL> / (8192 rows created.)

SQL> /
insert into t3 select * from t3
*
ERROR at line 1:
ORA-30036: unable to extend segment by 16 in undo tablespace 'UNDO02'
```

```
SQL> select xidusn from v$transaction;

    XIDUSN
----------
        10

SQL> commit;
Commit complete.
```

8. Determine the number of extents used by each undo segment in both undo tablespaces (UNDO02, and UNDO1). What do you observe?

   Answer: It appears that the previous transaction (step 7) was assigned to one particular undo segment (_SYSSMU10$) in the active undo tablespace UNDO02. Also, all undo segments in tablespace UNDO02 have two extents only, except for _SYSSMU10$ which now contains three extents.

```
SQL> col segment_name format a15

SQL> select   segment_name,tablespace_name,sum(blocks),count(*)
  2  from     dba_extents
  3  where    tablespace_name in ('UNDO02','UNDO1')
  4  group by segment_name, tablespace_name;

SEGMENT_NAME    TABLESPACE_NAME SUM(BLOCKS) COUNT(*)
--------------- --------------- ----------- --------
_SYSSMU1$       UNDO1                    47        3
_SYSSMU10$      UNDO02                   47        3
_SYSSMU2$       UNDO1                    31        2
_SYSSMU3$       UNDO1                    31        2
_SYSSMU4$       UNDO1                    31        2
_SYSSMU5$       UNDO1                    31        2
_SYSSMU6$       UNDO1                    31        2
_SYSSMU7$       UNDO1                    47        3
_SYSSMU8$       UNDO1                    31        2
_SYSSMU9$       UNDO02                   31        2

10 rows selected.
```

9. Delete table T1 and find out which transaction table was assigned to this new transaction. Once done, try to delete table T3. What happens and why?

   Answer: The active undo tablespace is still UNTO02. This tablespace is quite small. Although it is possible to delete the small table T1, Oracle fails to delete the bigger table T3. You may have noticed that each new transaction is assigned to a different undo segment.

```
SQL> delete t1;
128 rows deleted.

SQL> select xidusn from v$transaction;

    XIDUSN
----------
```

```
         9

SQL> delete t3;
delete t3
       *
ERROR at line 1:
ORA-30036: unable to extend segment by 16 in undo tablespace 'UNDO02'
```

10. Reexecute the query you used in step 8. What is your conclusion?

Answer: You executed another transaction in the second undo segment (_SYSSMU09$). What is remarkable here, compared to previous releases, is that the Oracle server was able to reclaim one extent from undo segment _SYSSMU10$. Although the transaction failed to allocate another extent in the UNDO02 tablespace, this proves that the Oracle server first tried to reuse unused extents in other undo segments. This new behavior potentially reduces the chance of getting errors during DML executions.

```
SQL> select   segment_name,tablespace_name,sum(blocks),count(*)
  2  from     dba_extents
  3  where    tablespace_name in ('UNDO02','UNDO1')
  4  group by segment_name, tablespace_name;

SEGMENT_NAME    TABLESPACE_NAME SUM(BLOCKS) COUNT(*)
--------------- --------------- ----------- --------
_SYSSMU1$       UNDO1                    47        3
_SYSSMU10$      UNDO02                   31        2
_SYSSMU2$       UNDO1                    31        2
_SYSSMU3$       UNDO1                    31        2
_SYSSMU4$       UNDO1                    31        2
_SYSSMU5$       UNDO1                    31        2
_SYSSMU6$       UNDO1                    31        2
_SYSSMU7$       UNDO1                    47        3
_SYSSMU8$       UNDO1                    31        2
_SYSSMU9$       UNDO02                   47        3

10 rows selected.
```

11. Commit your modifications. Then, check the number of rows currently stored inside table T3. Once done, insert one new row inside table T3 and do not commit your modification.

```
SQL> commit;
Commit complete.

SQL> select count(*) from t3;

  COUNT(*)
----------
     16384

SQL> insert into t3 values('bbbbb');
1 row created.

SQL> select count(*) from t3;
```

```
  COUNT(*)
----------
     16385
```

12. From a second session connected as SYSTEM, switch the active undo tablespace to UNDO1, and then look at all undo segment statuses. What is your conclusion?

    Answer: Because _SYSSMU9$ still has an active transaction (the one from the first session), this undo segment is still online while _SYSSMU10$ was offline by the switch operation. Note also that all the other undo segments residing in the UNDO1 tablespace are now online.

```
$ sqlplus system/manager
SQL*Plus: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

SQL> alter system set undo_tablespace=UNDO1;
System altered.

SQL> col segment_name format a15

SQL> select tablespace_name,segment_name,status
  2  from   dba_rollback_segs;

TABLESPACE_NAME      SEGMENT_NAME    STATUS
-------------------- --------------- -------
SYSTEM               SYSTEM          ONLINE
UNDO1                _SYSSMU1$       ONLINE
UNDO1                _SYSSMU2$       ONLINE
UNDO1                _SYSSMU3$       ONLINE
UNDO1                _SYSSMU4$       ONLINE
UNDO1                _SYSSMU5$       ONLINE
UNDO1                _SYSSMU6$       ONLINE
UNDO1                _SYSSMU7$       ONLINE
UNDO1                _SYSSMU8$       ONLINE
UNDO02               _SYSSMU9$       OFFLINE
UNDO02               _SYSSMU10$      ONLINE

11 rows selected.
```

13. From the first session, commit your changes.

```
SQL> commit;
Commit complete.
```

14. Still in the first session, check the undo segments status many times during five minutes. Once done, drop tablespace UNDO02 and its datafile. What are your conclusions?

    Answer: One undo segment in the UNDO02 tablespace is still online initially; after a while, it is automatically taken OFFLINE.

```
SQL> select tablespace_name,segment_name,status
  2  from dba_rollback_segs;
```

```
TABLESPACE_NAME      SEGMENT_NAME    STATUS
-------------------- --------------- -------
SYSTEM               SYSTEM          ONLINE
UNDO1                _SYSSMU1$       ONLINE
UNDO1                _SYSSMU2$       ONLINE
UNDO1                _SYSSMU3$       ONLINE
UNDO1                _SYSSMU4$       ONLINE
UNDO1                _SYSSMU5$       ONLINE
UNDO1                _SYSSMU6$       ONLINE
UNDO1                _SYSSMU7$       ONLINE
UNDO1                _SYSSMU8$       ONLINE
UNDO02               _SYSSMU9$       OFFLINE
UNDO02               _SYSSMU10$      ONLINE

11 rows selected.

SQL> /

TABLESPACE_NAME      SEGMENT_NAME    STATUS
-------------------- --------------- -------
SYSTEM               SYSTEM          ONLINE
UNDO1                _SYSSMU1$       ONLINE
UNDO1                _SYSSMU2$       ONLINE
UNDO1                _SYSSMU3$       ONLINE
UNDO1                _SYSSMU4$       ONLINE
UNDO1                _SYSSMU5$       ONLINE
UNDO1                _SYSSMU6$       ONLINE
UNDO1                _SYSSMU7$       ONLINE
UNDO1                _SYSSMU8$       ONLINE
UNDO02               _SYSSMU9$       OFFLINE
UNDO02               _SYSSMU10$      OFFLINE

11 rows selected.

SQL> drop tablespace undo02 including contents and datafiles;
Tablespace dropped.
```

15. Connected as SYSTEM, drop the tables T, T1, T2, and T3.

```
SQL> drop table t;
Table dropped.

SQL> drop table t1;
Table dropped.

SQL> drop table t2;
Table dropped.

SQL> drop table t3;
Table dropped.
```

Practice 14-2 Solution: Transporting a Tablespace with Nondefault Block Size

1. Connected as SYSDBA under SQL*Plus, determine the default block size used for this database and also the various parameters used to configure the buffer cache of this instance.

```
$ sqlplus /nolog
SQL*Plus: Release 9.0.1.0.0 – Production

SQL> connect / as sysdba;
Connected.

SQL> show parameter db_block_size

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
db_block_size                        integer     4096

SQL> show parameter cache

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
db_16k_cache_size                    big integer 0
db_2k_cache_size                     big integer 0
db_32k_cache_size                    big integer 0
db_4k_cache_size                     big integer 0
db_8k_cache_size                     big integer 0
db_cache_advice                      string      OFF
db_cache_size                        big integer 4194304
db_keep_cache_size                   big integer 0
db_recycle_cache_size                big integer 0
object_cache_max_size_percent        integer     10
object_cache_optimal_size            integer     102400
session_cached_cursors               integer     0
```

2. Drop tablespace TEST1, including its data files, if it already exists. Determine what is inside the $HOME/STUDENT/LABS directory. Then, copy the test1.dbf file into the $HOME/ORADATA/u04 directory. Also, drop the SYSTEM.T table.

```
SQL> drop tablespace test1 including contents and datafiles
  2  cascade constraints;
Tablespace dropped.

SQL> host ls -l $HOME/STUDENT/LABS
total …
…
-rw-r--r--   1 ed25     dba         4096 Jul 03 12:18 plug.dmp
-rw-r-----   1 ed25     dba       212992 Jul 03 12:18 test1.dbf

SQL> host cp $HOME/STUDENT/LABS/test1.dbf $HOME/ORADATA/u04

SQL> drop table SYSTEM.T;
Table dropped.
```

3. Try to plug the TEST1 tablespace using the plug.dmp export dump file. What happens and why?

   Answer: The Oracle server is not able to plug this tablespace into the instance as no 8 KB buffer cache is currently configured.

```
SQL> host imp userid=\'/ as sysdba\' file=$HOME/STUDENT/LABS/plug.dmp -
> transport_tablespace=y datafiles=$HOME/ORADATA/u04/test1.dbf
Import: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

Export file created by EXPORT:V09.00.01 via conventional path
About to import transportable tablespace(s) metadata...
import done in US7ASCII character set and AL16UTF16 NCHAR character set
import server uses WE8ISO8859P1 character set (possible charset
conversion)
. importing SYS's objects into SYS
IMP-00017: following statement failed with ORACLE error 29339:
 "BEGIN
sys.dbms_plugts.beginImpTablespace('TEST1',9,'SYS',1,0,8192,14,2057"
 "12,1,249,5,5,0,50,1,0,0,3707708581,1,0,205636,NULL,0,0,NULL,NULL);
END;"
IMP-00003: ORACLE error 29339 encountered
ORA-29339: tablespace block size 8192 does not match configured block
sizes
ORA-06512: at "SYS.DBMS_PLUGTS", line 1378
ORA-06512: at line 1
IMP-00000: Import terminated unsuccessfully
```

4. How would you get around this problem?

   Answer: Shut down the instance and start it up with an 8 KB buffer cache configured. That is, edit your initialization parameter file and add the following line: db_8k_cache_size=4194304. Once done, start up the instance again.

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

SQL> startup pfile=$HOME/ADMIN/PFILE/init8K.ora
ORACLE instance started.

Total System Global Area   25991956 bytes
Fixed Size                   285460 bytes
Variable Size              16777216 bytes
Database Buffers            8388608 bytes
Redo Buffers                 540672 bytes
Database mounted.
Database opened.

SQL> show parameter cache

NAME                                 TYPE        VALUE
```

```
---------------------------------- ----------- -----------
db_16k_cache_size                  big integer 0
db_2k_cache_size                   big integer 0
db_32k_cache_size                  big integer 0
db_4k_cache_size                   big integer 0
db_8k_cache_size                   big integer 4194304
db_cache_advice                    string      OFF
db_cache_size                      big integer 4194304
db_keep_cache_size                 big integer 0
db_recycle_cache_size              big integer 0
object_cache_max_size_percent      integer     10
object_cache_optimal_size          integer     102400
session_cached_cursors             integer     0
```

5. After fixing the problem, plug in tablespace TEST1 using the same procedure as in step 3.

```
SQL> host imp userid=\'/ as sysdba\' file=$HOME/STUDENT/LABS/plug.dmp -
> transport_tablespace=y datafiles=$HOME/ORADATA/u04/test1.dbf
Import: Release 9.0.1.0.0 – Production
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.0 – Production

Export file created by EXPORT:V09.00.01 via conventional path
About to import transportable tablespace(s) metadata...
import done in US7ASCII character set and AL16UTF16 NCHAR character set
import server uses WE8ISO8859P1 character set (possible charset
conversion)
. importing SYS's objects into SYS
. importing SYSTEM's objects into SYSTEM
. . importing table                            "T"
Import terminated successfully without warnings.
```

6. Verify that you can access table SYSTEM.T and look at the DBA_TABLESPACES view to see the characteristics of the plugged tablespace TEST1.

```
SQL> select count(*) from system.t;

  COUNT(*)
----------
       512

SQL> select tablespace_name,block_size from dba_tablespaces;

TABLESPACE_NAME           BLOCK_SIZE
------------------------- ----------
SYSTEM                          4096
RBS                             4096
INDX                            4096
TEMP                            4096
TOOLS                           4096
USERS                           4096
DRSYS                           4096
SAMPLE                          4096
CWMLITE                         4096
TEST1                           8192
```

```
10 rows selected.
```

7. Connected as SYSDBA, drop tablespace TEST1 including its data file.

```
SQL> drop tablespace test1 including contents and datafiles
  2  cascade constraints;
Tablespace dropped.
```

Practice 17-1 Solution: ANSI/ISO SQL:1999

1. With the natural join, you do not need to specify any join predicate or join columns. What happens if you apply the natural join to two tables that do not have any common column names?
   Think about this first, then connect to the HR schema, and try the following statement:

```
SQL> select *
  2  from   regions
  3         natural join
  4         jobs;

REGION_ID REGION_NAME JOB_ID  JOB_TITLE  MIN_SALARY MAX_SALARY
--------- ----------- ------- ---------- ---------- ----------
        1 Europe      AD_PRES President       20000      40000
        1 Europe      AD_VP   ...
      ... ...         ...

76 rows selected.
```

   Answer: You get the Cartesian product of those two tables.

2. What is the difference between the cross join and the natural join, if two tables do not have any column names in common? Change NATURAL into CROSS in the previous example and compare the results.

   Answer: There is no difference.

3. Write a six-table join by using the USING and ON syntax, to show the last name, department name, city, and region name for all employees reporting to Steven King.

```
SQL> select e.last_name
  2  ,      d.department_name
  3  ,      l.city
  4  ,      r.region_name
  5  from   employees   e JOIN
  6         departments d USING (department_id) JOIN
  7         locations   l USING (location_id)   JOIN
  8         countries   c USING (country_id)    JOIN
  9         regions     r USING (region_id)     JOIN
 10         employees   m ON    (e.manager_id = m.employee_id)
 11  where  m.first_name = 'Steven'
 12  and    m.last_name  = 'King';

LAST_NAME      DEPARTMENT_NAME CITY                REGION_NAME
-------------- --------------- ------------------- -----------
Kochhar        Executive       Seattle             Americas
De Haan        Executive       Seattle             Americas
Raphaely       Purchasing      Seattle             Americas
Weiss          Shipping        South San Francisco  Americas
Fripp          Shipping        South San Francisco  Americas
Kaufling       Shipping        South San Francisco  Americas
Vollman        Shipping        South San Francisco  Americas
```

```
Mourgos         Shipping        South San Francisco  Americas
Russell         Sales           Oxford               Europe
Partners        Sales           Oxford               Europe
Errazuriz       Sales           Oxford               Europe
Cambrault       Sales           Oxford               Europe
Zlotkey         Sales           Oxford               Europe
Hartstein       Marketing       Toronto              Americas

14 rows selected.
```

4. Look at the example of the WIDTH_BUCKET function (on page 17-25):

```
SQL> select last_name, salary
  2  ,      WIDTH_BUCKET(salary,3000,13000,5)
  3  from   employees;
```

Rewrite this query as a searched CASE statement.

```
SQL> select last_name, salary
  2  ,     (case when salary <  3000 then 0
  3              when salary <  5000 then 1
  4              when salary <  7000 then 2
  5              when salary <  9000 then 3
  6              when salary < 11000 then 4
  7              when salary < 13000 then 5
  8                                  else 6
  9         end) as bucket
 10  from   employees;

LAST_NAME            SALARY BUCKET
-------------------- ------ ------
King                  24000      6
Kochhar               17000      6
De Haan               17000      6
Hunold                 9000      4
Ernst                  6000      2
Austin                 4800      1
...

107 rows selected.
```

5. Write a query to retrieve the average salary, excluding any employees that have a salary of 2500; try to find a solution without a WHERE clause, using the NULLIF function.

```
SQL> select avg(nullif(salary,2500))
  2  from   employees;

AVG(NULLIF(SALARY,2500))
------------------------
              6697.0297

1 row selected.
```

6. Write a single query, using scalar subqueries, to return the current system date if you have more employees than jobs.

```
SQL> select sysdate from dual
  2  where (select count(*) from employees) >
  3        (select count(*) from jobs);

SYSDATE
-----------
03-JUL-2001

1 row selected.
```

Practice 17-2 Solution: Other SQL Enhancements

1. Look at the JOB_HISTORY table, and check the primary key constraint and the associated index. Drop the primary key constraint, and make sure that the associated index is not dropped. Restore the original situation by creating the primary key constraint.

   For the remaining steps of this practice, you need two SQL*Plus sessions connected to the HR schema (referred to as session A and session B respecively).

2. a. Make sure that you don't have an index on the department_id column of the EMPLOYEES table.

```
SQL> select   index_name
  2  from     user_ind_columns
  3  where    table_name  = 'EMPLOYEES'
  4  and      column_name = 'DEPARTMENT_ID';

INDEX_NAME
-----------------
EMP_DEPARTMENT_IX

1 row selected.

SQL> drop index EMP_DEPARTMENT_IX;
Index dropped.
```

   b. From session A, delete any department from the DEPARTMENTS table without issuing a commit; to avoid foreign key violations, choose a department without employees. Save the delete statement, because you will need it again in the next exercise.

```
SQL> delete from departments      -- In session A
  2  where  department_id = 250;
1 row deleted.

SQL> save delete_parent
```

   c. From the same session A, query v$locked_object (joined with the user_objects view to display the object name) to see which locks are held by your session. In Oracle8*i* you would see at least an additional shared lock on the EMPLOYEES table, because you do not have an index on the foreign key column. Save the query for the next exercise.

```
SQL> select object_name           -- In session A
  2  ,      decode(locked_mode
  3               ,0,'None'
  4               ,1,'NULL'
  5               ,2,'SS'
  6               ,3,'SX'
  7               ,4,'S'
  8               ,5,'SSX'
  9               ,6,'X'
 10                 ,null) as lmode
 11  from   v$locked_object natural join
```

```
 12         user_objects
 13  where  oracle_username = USER;

OBJECT_NAME                      LMODE
-------------------------------- -----
DEPARTMENTS                      SX

1 row selected.

SQL> save user_locks
```

d. From session B, try to update any employee; as you see, this is possible. Save this update statement in a script file too.

```
SQL> update employees              -- In session B
  2  set    salary = salary + 1
  3  where  employees_id = 102;
1 row updated.

SQL> save update_child
```

e. Roll back the changes you made in both sessions (but stay connected.)

```
SQL> rollback;                     -- In sessions A and B
Rollback complete.
```

3. To show that the shared lock is needed, you repeat the previous exercise in the opposite order. You can use the three SQL statements you saved during the previous exercise.

a. From session B, update any employee; do not commit.

```
SQL> @update_child                 -- In session B
1 row updated.
```

b. From session B, query the data dictionary for locks; note that you have an exclusive row lock on the EMPLOYEES table.

```
SQL> @user_locks                   -- In session B

OBJECT_NAME LMODE
----------- -----
EMPLOYEES   SX

1 row selected.
```

c. From session A, try to delete the same department again.

```
SQL> @delete_parent                -- In session A
```

Now you are unable to delete any department, because the exclusive row lock in session B causes the shared lock request from session A to wait.

d. Roll back the changes you made in both sessions (but stay connected.)

```
SQL> rollback;                    -- In sessions A and B
Rollback complete.
```

4. From session A, update the salary of any employee without issuing a commit; then, try to select the same row for update from the other session. First try the default FOR UPDATE clause; then, try to specify a wait time of five seconds.

```
SQL> @update_child                -- In session A
1 row updated.
```

```
SQL> select * from empoyees       -- In session B
  2  where employee_id = 102
  3  for update;
```

```
SQL> select * from empoyees       -- In session B
  2  where employee_id = 102
  3  for update wait 5;
```

Roll back the changes you made.

```
SQL> rollback;                    -- In session A
Rollback complete.
```

Practice 18-1 Solution: Globalization Support

1. Log on as user HR. Create a table TIMES, with the following four columns:
   - Column TS, data type TIMESTAMP
   - Column TSZ, data type TIMESTAMP WITH TIME ZONE
   - Column TLZ, data type TIMESTAMP WITH LOCAL TIME ZONE
   - Column OLDTIME, data type DATE

```
SQL> CONNECT hr/hr
Connected.

SQL> CREATE TABLE times (
  2    ts      TIMESTAMP
  3  , tsz     TIMESTAMP WITH TIME ZONE
  4  , tlz     TIMESTAMP WITH LOCAL TIME ZONE
  5  , oldtime DATE );
Table created.
```

2. Check your database timezone. Check and possibly adjust your session timezone to be 'Europe/London.'

```
SQL> SELECT DBTIMEZONE, SESSIONTIMEZONE
  2  FROM   DUAL;

DBTIMEZONE SESSIONTIMEZONE
---------- ---------------
-08:00     -01:00

SQL> ALTER SESSION SET TIME_ZONE='Europe/London';
Session altered.

SQL> SELECT DBTIMEZONE, SESSIONTIMEZONE FROM DUAL;

DBTIMEZONE SESSIONTIMEZONE
---------- ---------------
-08:00     Europe/London
```

3. Determine the hour offset for this region.

```
SQL> SELECT TZ_OFFSET('Europe/London')
  2  FROM   DUAL;

TZ_OFFSET
---------
+01:00

(Result may vary, depending on Daylight Saving Time)
```

4. Insert a row that populates all columns in the table. For the time component use one quarter of a second after 10 a.m. Specify the CET timezone where appropiate.

```
SQL> INSERT INTO times VALUES
     ( '21-MAY-01 10:00:00.25 AM'
```

```
     , '21-MAY-01 10:00:00.25 AM CET'
     , '21-MAY-01 10:00:00.25 AM'
     , '21-MAY-01' );
```

5. Alter your session timezone to be 'America/LosAngeles' and display the values. Explain what you see.

```
SQL> ALTER SESSION SET TIME_ZONE='America/Los_Angeles';
Session altered.

SQL> SELECT * FROM times;
TS                                TSZ
--------------------------------- ------------------------------
TLZ                               OLDTIME
--------------------------------- ---------
21-MAY-01 10.00.00.250000 AM      21-MAY-01 10.00.00.250000 AM CET
21-MAY-01 02.00.00.250000 AM      21-MAY-01
```

Answer: The TLZ column value has changed, because you are seeing the time from a (simulated) different client.

6. Drop the TIMES table.

```
SQL> DROP TABLE times;
Table dropped.
```

Practice 19-1 Solution: Workspaces

1. Connected as SYSDBA under SQL*Plus, create a user called WM_DEVELOPER.

```
$ sqlplus /nolog
SQL*Plus: Release 9.0.1.0.0 – Production

SQL> connect / as sysdba;
Connected.

SQL> CREATE USER wm_developer IDENTIFIED BY wm_developer;
User created.
```

2. Grant CONNECT, RESOURCE roles to WM_DEVELOPER. Also directly grant the CREATE TABLE privilege to WM_DEVELOPER.

```
SQL> GRANT connect, resource to wm_developer;
Grant succeeded.

SQL> GRANT create table to wm_developer;
Grant succeeded.
```

3. Grant the WM-specific privileges (with grant_option = YES) to WM_DEVELOPER. Specifically:
ACCESS_ANY_WORKSPACE, MERGE_ANY_WORKSPACE, CREATE_ANY_WORKSPACE, REMOVE_ANY_WORKSPACE, and ROLLBACK_ANY_WORKSPACE.

```
SQL> EXECUTE DBMS_WM.GrantSystemPriv -
>    ( 'ACCESS_ANY_WORKSPACE,-
>       MERGE_ANY_WORKSPACE, CREATE_ANY_WORKSPACE,-
>       REMOVE_ANY_WORKSPACE, ROLLBACK_ANY_WORKSPACE' -
>    , 'wm_developer'
>    , 'YES');
PL/SQL procedure successfully completed.
```

4. Connected as WM_DEVELOPER, create a table for the annual marketing budget for several cola (soft drink) markets in a given geography (such as a city or a state). Each row will contain budget data for a specific cola.

   Note: This table does not reflect recommended database design. For example, a manager ID should be used, not a name. It is deliberately oversimplified for purposes of illustration. Budget is in Millions. In order to help you create this table, simply execute the lab_19_01.sql script located in your LABS directory.

```
SQL> CONNECT wm_developer/wm_developer
Connected.

SQL> CREATE TABLE cola_marketing_budget (
  2    mkt_id   NUMBER PRIMARY KEY,
  3    mkt_name VARCHAR2(32),
  4    manager  VARCHAR2(32),
  5    budget   NUMBER
```

```
  6  );
Table created.
```

5. Version-enable the table. Specify the `hist` option of `VIEW_WO_OVERWRITE` so that the `COLA_MARKETING_BUDGET_HIST` view contains complete history information. Once done, populate the table with some rows using the `lab_19_02.sql` script locate in your `LABS` directory.

```
SQL> EXECUTE DBMS_WM.EnableVersioning –
>    ('cola_marketing_budget', 'VIEW_WO_OVERWRITE');
PL/SQL procedure successfully completed.

SQL> INSERT INTO cola_marketing_budget VALUES
  2  (1, 'cola_a', 'Alvarez', 2.0);
1 row created.

SQL> INSERT INTO cola_marketing_budget VALUES
  2  (2, 'cola_b', 'Baker', 1.5);
1 row created.

SQL> INSERT INTO cola_marketing_budget VALUES
  2  (3, 'cola_c', 'Chen', 1.5);
1 row created.

SQL> INSERT INTO cola_marketing_budget VALUES
  2  (4, 'cola_d', 'Davis', 3.5);
1 row created.

SQL> COMMIT;
Commit complete.
```

6. Now create workspaces for the following scenario: a major marketing focus in the `cola_b` area. Managers and budget amounts for each market can change, but the total marketing budget cannot grow. The `B_focus_1` scenario features a manager with more expensive plans (which means more money taken from other areas' budgets). The `B_focus_2` scenario features a manager with less expensive plans (which means less money taken from other areas' budgets). Two workspaces (`B_focus_1` and `B_focus_2`) are created as child workspaces of the `LIVE` database workspace.

```
SQL> EXECUTE DBMS_WM.CreateWorkspace ('B_focus_1');
PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_WM.CreateWorkspace ('B_focus_2');
PL/SQL procedure successfully completed.
```

7. Enter the `B_focus_1` workspace and change the `cola_b` manager to Beasley and raise the `cola_b` budget amount by 1.5 to bring it to 3.0. Reduce all other area budget amounts by 0.5 to stay within the overall budget. In order to make the previous changes, you can use the `lab_19_03.sql` script.

```
SQL> EXECUTE DBMS_WM.GotoWorkspace ('B_focus_1');
PL/SQL procedure successfully completed.
```

```
SQL> UPDATE cola_marketing_budget
  2  SET    manager  = 'Beasley'
  3  ,      budget   = 3
  4  WHERE  mkt_name = 'cola_b';
1 row updated.

SQL> UPDATE cola_marketing_budget
  2  SET    budget   = budget – 0.5
  2  WHERE  mkt_name <> 'cola_b';
3 rows updated.

SQL> COMMIT;
Commit complete.
```

8. Enter the B_focus_2 workspace and change the cola_b manager to Burton and raise the cola_b budget amount by 0.5 to bring it to 2.0. Reduce only the cola_d amount by 0.5 to stay within the overall budget. You can use the lab_19_04.sql script in order to make the changes. Once done, select every column from the COLA_MARKETING_BUDGET table.

```
SQL> EXECUTE DBMS_WM.GotoWorkspace ('B_focus_2');
PL/SQL procedure successfully completed.

SQL> UPDATE cola_marketing_budget
  2  SET    manager  = 'Burton'
  3  ,      budget   = budget + 0.5
  4  WHERE  mkt_name = 'cola_b';
1 row updated.

SQL> UPDATE cola_marketing_budget
  2  SET    budget   = budget – 0.5
  3  WHERE  mkt_name = 'cola_d';
1 row updated.

SQL> COMMIT;
Commit complete.

SQL> SELECT * FROM cola_marketing_budget;

MKT_ID MKT_NAME MANAGER  BUDGET
------ -------- -------- ------
     1 cola_a   Alvarez       2
     3 cola_c   Chen        1.5
     2 cola_b   Burton        2
     4 cola_d   Davis         3
```

9. Assume that you have decided to adopt the scenario of the B_focus_2 workspace using that workspace's current values; Go to the LIVE workspace, and remove the B_focus_1 workspace. Once done, apply changes in the second workspace to the LIVE database workspace. Note that the workspace is removed by default after MergeWorkspace. Once done, select every column from the COLA_MARKETING_BUDGET table.

```
SQL> EXECUTE DBMS_WM.GotoWorkspace ('LIVE');
PL/SQL procedure successfully completed.
```

```
SQL> EXECUTE DBMS_WM.RemoveWorkspace ('B_focus_1');
PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_WM.MergeWorkspace ('B_focus_2');
PL/SQL procedure successfully completed.

SQL> SELECT * FROM cola_marketing_budget;

MKT_ID MKT_NAME MANAGER  BUDGET
------ -------- -------- ------
     1 cola_a   Alvarez       2
     3 cola_c   Chen         1.5
     2 cola_b   Burton         2
     4 cola_d   Davis          3
```

10. Disable versioning on the table because you are finished testing scenarios. Also, users with version-enabled tables cannot be dropped, in case you want to drop the WM_DEVELOPER user. Set the force parameter to TRUE if you want to force the disabling even if changes were made in a non-LIVE workspace. Also, remove the B_focus_2 workspace.

```
SQL> EXECUTE DBMS_WM.DisableVersioning('cola_marketing_budget', TRUE);
PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_WM.RemoveWorkspace ('B_focus_2');
PL/SQL procedure successfully completed.
```

11. Connected as SYSDBA, drop user WM_DEVELOPER.

```
SQL> connect / as sysdba
Connected.

SQL> drop user wm_developer cascade;
User dropped.
```