

# 2

## Managing Schema Objects

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## Objectives

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

- Add constraints
- Create indexes
- Create indexes using the `CREATE TABLE` statement
- Creating function-based indexes
- Drop columns and set column `UNUSED`
- Perform `FLASHBACK` operations
- Create and use external tables

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Objectives

This lesson contains information about creating indexes and constraints, and altering existing objects. You also learn about external tables, and the provision to name the index at the time of creating a primary key constraint.

## ALTER TABLE Statement

Use the ALTER TABLE statement to:

- Add a new column
- Modify an existing column
- Define a default value for the new column
- Drop a column

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### ALTER TABLE Statement

After you create a table, you may need to change the table structure because you omitted a column, your column definition needs to be changed, or you need to remove columns. You can do this by using the ALTER TABLE statement.

## ALTER TABLE Statement

Use the ALTER TABLE statement to add, modify, or drop columns:

```
ALTER TABLE table
ADD          (column datatype [DEFAULT expr]
             [, column datatype]...);
```

```
ALTER TABLE table
MODIFY       (column datatype [DEFAULT expr]
             [, column datatype]...);
```

```
ALTER TABLE table
DROP         (column);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### ALTER TABLE Statement (continued)

You can add columns to a table, modify columns, and drop columns from a table by using the ALTER TABLE statement.

In the syntax:

<i>table</i>	Is the name of the table
ADD   MODIFY   DROP	Is the type of modification
<i>column</i>	Is the name of the new column
<i>datatype</i>	Is the data type and length of the new column
DEFAULT <i>expr</i>	Specifies the default value for a new column

## Adding a Column

- You use the ADD clause to add columns:

```
ALTER TABLE dept80
ADD      (job_id VARCHAR2(9));
ALTER TABLE succeeded.
```

- The new column becomes the last column:

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE	JOB_ID
1	149	Zlotkey	126000	29-JAN-00	(null)
2	174	Abel	132000	11-MAY-96	(null)
3	176	Taylor	103200	24-MAR-98	(null)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Guidelines for Adding a Column

- You can add or modify columns.
- You cannot specify where the column is to appear. The new column becomes the last column.

The example in the slide adds a column named `JOB_ID` to the `DEPT80` table. The `JOB_ID` column becomes the last column in the table.

**Note:** If a table already contains rows when a column is added, then the new column is initially null for all the rows. You cannot add a mandatory `NOT NULL` column to a table that contains data in the other columns. You can only add a `NOT NULL` column to an empty table.

## Modifying a Column

- You can change a column's data type, size, and default value.

```
ALTER TABLE dept80
MODIFY      (last_name VARCHAR2(30));
ALTER TABLE succeeded.
```

- A change to the default value affects only subsequent insertions to the table.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Modifying a Column

You can modify a column definition by using the ALTER TABLE statement with the MODIFY clause. Column modification can include changes to a column's data type, size, and default value.

#### Guidelines

- You can increase the width or precision of a numeric column.
- You can increase the width of numeric or character columns.
- You can decrease the width of a column if:
  - The column contains only null values
  - The table has no rows
  - The decrease in column width is not less than the existing values in that column
- You can change the data type if the column contains only null values. The exception to this is CHAR-to-VARCHAR2 conversions, which can be done with data in the columns.
- You can convert a CHAR column to the VARCHAR2 data type or convert a VARCHAR2 column to the CHAR data type only if the column contains null values or if you do not change the size.
- A change to the default value of a column affects only subsequent insertions to the table.

## Dropping a Column

Use the DROP COLUMN clause to drop columns you no longer need from the table:

```
ALTER TABLE dept80
DROP COLUMN job_id;
ALTER TABLE succeeded.
```

	EMPLOYEE_ID	LAST_NAME	ANNSAL	HIRE_DATE
1	149	Zlotkey	126000	29-JAN-00
2	174	Abel	132000	11-MAY-96
3	176	Taylor	103200	24-MAR-98

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Dropping a Column

You can drop a column from a table by using the ALTER TABLE statement with the DROP COLUMN clause.

#### Guidelines

- The column may or may not contain data.
- Using the ALTER TABLE statement, only one column can be dropped at a time.
- The table must have at least one column remaining in it after it is altered.
- After a column is dropped, it cannot be recovered.
- A column cannot be dropped if it is part of a constraint or part of an index key unless the cascade option is added.
- Dropping a column can take a while if the column has a large number of values. In this case, it may be better to set it to be unused and drop it when there are fewer users on the system to avoid extended locks.

**Note:** Certain columns can never be dropped such as columns that form part of the partitioning key of a partitioned table or columns that form part of the primary key of an index-organized table.

## SET UNUSED Option

- You use the SET UNUSED option to mark one or more columns as unused.
- You use the DROP UNUSED COLUMNS option to remove the columns that are marked as unused.

```
ALTER TABLE <table_name>
SET UNUSED(<column_name>);
OR
ALTER TABLE <table_name>
SET UNUSED COLUMN <column_name>;

ALTER TABLE <table_name>
DROP UNUSED COLUMNS;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### SET UNUSED Option

The SET UNUSED option marks one or more columns as unused so that they can be dropped when the demand on system resources is lower. Specifying this clause does not actually remove the target columns from each row in the table (that is, it does not restore the disk space used by these columns). Therefore, the response time is faster than if you executed the DROP clause. Unused columns are treated as if they were dropped, even though their column data remains in the table's rows. After a column has been marked as unused, you have no access to that column. A SELECT \* query will not retrieve data from unused columns. In addition, the names and types of columns marked unused will not be displayed during a DESCRIBE statement, and you can add to the table a new column with the same name as an unused column. SET UNUSED information is stored in the USER\_UNUSED\_COL\_TABS dictionary view.

**Note:** The guidelines for setting a column to be UNUSED are similar to those of dropping a column.



**DROP UNUSED COLUMNS Option**

DROP UNUSED COLUMNS removes from the table all columns currently marked as unused. You can use this statement when you want to reclaim the extra disk space from unused columns in the table. If the table contains no unused columns, the statement returns with no errors.

```
ALTER TABLE dept80
SET UNUSED (last_name);
ALTER TABLE succeeded.
```

```
ALTER TABLE dept80
DROP UNUSED COLUMNS;
ALTER TABLE succeeded.
```

## Adding a Constraint Syntax

Use the ALTER TABLE statement to:

- Add or drop a constraint, but not modify its structure
- Enable or disable constraints
- Add a NOT NULL constraint by using the MODIFY clause

```
ALTER TABLE <table_name>
ADD [CONSTRAINT <constraint_name>]
type (<column_name>);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Adding a Constraint

You can add a constraint for existing tables by using the ALTER TABLE statement with the ADD clause.

In the syntax:

<i>table</i>	Is the name of the table
<i>constraint</i>	Is the name of the constraint
<i>type</i>	Is the constraint type
<i>column</i>	Is the name of the column affected by the constraint

The constraint name syntax is optional, although recommended. If you do not name your constraints, the system generates constraint names.

#### Guidelines

- You can add, drop, enable, or disable a constraint, but you cannot modify its structure.
- You can add a NOT NULL constraint to an existing column by using the MODIFY clause of the ALTER TABLE statement.

**Note:** You can define a NOT NULL column only if the table is empty or if the column has a value for every row.

## Adding a Constraint

Add a FOREIGN KEY constraint to the EMP2 table indicating that a manager must already exist as a valid employee in the EMP2 table.

```
ALTER TABLE emp2  
modify employee_id Primary Key;  
ALTER TABLE succeeded.
```

```
ALTER TABLE emp2  
ADD CONSTRAINT emp_mgr_fk  
FOREIGN KEY(manager_id)  
REFERENCES emp2(employee_id);  
ALTER TABLE succeeded.
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Adding a Constraint (continued)

The first example in the slide modifies the EMP2 table to add a PRIMARY KEY constraint on the EMPLOYEE\_ID column. Note that because no constraint name is provided, the constraint is automatically named by the Oracle server. The second example in the slide creates a FOREIGN KEY constraint on the EMP2 table. The constraint ensures that a manager exists as a valid employee in the EMP2 table.

## ON DELETE CASCADE

Delete child rows when a parent key is deleted:

```
ALTER TABLE Emp2 ADD CONSTRAINT emp_dt_fk  
FOREIGN KEY (Department_id)  
REFERENCES departments ON DELETE CASCADE);  
ALTER TABLE succeeded.
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### ON DELETE CASCADE

The ON DELETE CASCADE action allows parent key data that is referenced from the child table to be deleted, but not updated. When data in the parent key is deleted, all rows in the child table that depend on the deleted parent key values are also deleted. To specify this referential action, include the ON DELETE CASCADE option in the definition of the FOREIGN KEY constraint.

## Deferring Constraints

Constraints can have the following attributes:

- DEFERRABLE or NOT DEFERRABLE
- INITIALLY DEFERRED or INITIALLY IMMEDIATE

```
ALTER TABLE dept2
ADD CONSTRAINT dept2_id_pk
PRIMARY KEY (department_id)
DEFERRABLE INITIALLY DEFERRED
```

Deferring constraint on creation

```
SET CONSTRAINTS dept2_id_pk IMMEDIATE
```

Changing a specific constraint attribute

```
ALTER SESSION
SET CONSTRAINTS= IMMEDIATE
```

Changing all constraints for a session

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Deferring Constraints

You can defer checking constraints for validity until the end of the transaction. A constraint is deferred if the system checks that it is satisfied only on commit. If a deferred constraint is violated, then commit causes the transaction to roll back. If a constraint is immediate (not deferred), then it is checked at the end of each statement. If it is violated, the statement is rolled back immediately. If a constraint causes an action (for example, DELETE CASCADE), that action is always taken as part of the statement that caused it, whether the constraint is deferred or immediate. Use the SET CONSTRAINTS statement to specify, for a particular transaction, whether a deferrable constraint is checked following each DML statement or when the transaction is committed. To create deferrable constraints, you must create a nonunique index for that constraint.

You can define constraints as either deferrable or not deferrable, and either initially deferred or initially immediate. These attributes can be different for each constraint.

**Usage scenario:** Company policy dictates that department number 40 should be changed to 45. Changing the DEPARTMENT\_ID column affects employees assigned to this department. Therefore, you make the primary key and foreign keys deferrable and initially deferred. You update both department and employee information, and at the time of commit, all rows are validated.

## Dropping a Constraint

- Remove the manager constraint from the EMP2 table:

```
ALTER TABLE emp2
DROP CONSTRAINT emp_mgr_fk;
ALTER TABLE succeeded.
```

- Remove the PRIMARY KEY constraint on the DEPT2 table and drop the associated FOREIGN KEY constraint on the EMP2.DEPARTMENT\_ID column:

```
ALTER TABLE dept2
DROP PRIMARY KEY CASCADE;
ALTER TABLE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Dropping a Constraint

To drop a constraint, you can identify the constraint name from the USER\_CONSTRAINTS and USER\_CONS\_COLUMNS data dictionary views. Then use the ALTER TABLE statement with the DROP clause. The CASCADE option of the DROP clause causes any dependent constraints also to be dropped.

#### Syntax

```
ALTER TABLE table
DROP PRIMARY KEY | UNIQUE (column) |
CONSTRAINT constraint [CASCADE];
```

In the syntax:

<i>table</i>	Is the name of the table
<i>column</i>	Is the name of the column affected by the constraint
<i>constraint</i>	Is the name of the constraint

When you drop an integrity constraint, that constraint is no longer enforced by the Oracle server and is no longer available in the data dictionary.

## Disabling Constraints

- Execute the `DISABLE` clause of the `ALTER TABLE` statement to deactivate an integrity constraint.
- Apply the `CASCADE` option to disable dependent integrity constraints.

```
ALTER TABLE emp2
DISABLE CONSTRAINT emp_dt_fk;
ALTER TABLE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Disabling a Constraint

You can disable a constraint without dropping it or re-creating it by using the `ALTER TABLE` statement with the `DISABLE` clause.

#### Syntax

```
ALTER TABLE table
DISABLE CONSTRAINT constraint [CASCADE];
```

In the syntax:

*table*                      Is the name of the table  
*constraint*                Is the name of the constraint

#### Guidelines

- You can use the `DISABLE` clause in both the `CREATE TABLE` statement and the `ALTER TABLE` statement.
- The `CASCADE` clause disables dependent integrity constraints.
- Disabling a unique or primary key constraint removes the unique index.

## Enabling Constraints

- Activate an integrity constraint currently disabled in the table definition by using the `ENABLE` clause.

```
ALTER TABLE      emp2
ENABLE CONSTRAINT emp_dt_fk;
ALTER TABLE succeeded.
```

- A `UNIQUE` index is automatically created if you enable a `UNIQUE` key or a `PRIMARY KEY` constraint.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Enabling a Constraint

You can enable a constraint without dropping it or re-creating it by using the `ALTER TABLE` statement with the `ENABLE` clause.

#### Syntax

```
ALTER      TABLE      table
ENABLE     CONSTRAINT constraint;
```

In the syntax:

*table*                      Is the name of the table  
*constraint*                Is the name of the constraint

#### Guidelines

- If you enable a constraint, that constraint applies to all the data in the table. All the data in the table must comply with the constraint.
- If you enable a `UNIQUE` key or a `PRIMARY KEY` constraint, a `UNIQUE` or `PRIMARY KEY` index is created automatically. If an index already exists, then it can be used by these keys.
- You can use the `ENABLE` clause in both the `CREATE TABLE` statement and the `ALTER TABLE` statement.



## Enabling a Constraint (continued)

### Guidelines (continued)

- Enabling a primary key constraint that was disabled with the CASCADE option does not enable any foreign keys that are dependent on the primary key.
- To enable a UNIQUE or PRIMARY KEY constraint, you must have the privileges necessary to create an index on the table.

## Cascading Constraints

- The `CASCADE CONSTRAINTS` clause is used along with the `DROP COLUMN` clause.
- The `CASCADE CONSTRAINTS` clause drops all referential integrity constraints that refer to the primary and unique keys defined on the dropped columns.
- The `CASCADE CONSTRAINTS` clause also drops all multicolumn constraints defined on the dropped columns.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Cascading Constraints

This statement illustrates the usage of the `CASCADE CONSTRAINTS` clause. Assume that the `TEST1` table is created as follows:

```
CREATE TABLE test1 (
  pk NUMBER PRIMARY KEY,
  fk NUMBER,
  col1 NUMBER,
  col2 NUMBER,
  CONSTRAINT fk_constraint FOREIGN KEY (fk) REFERENCES test1,
  CONSTRAINT ck1 CHECK (pk > 0 and col1 > 0),
  CONSTRAINT ck2 CHECK (col2 > 0));
```

An error is returned for the following statements:

```
ALTER TABLE test1 DROP (pk);    —pk is a parent key.
ALTER TABLE test1 DROP (col1);  —col1 is referenced by the multicolumn
                                constraint, ck1.
```

## Cascading Constraints

Example:

```
ALTER TABLE emp2
DROP COLUMN employee_id CASCADE CONSTRAINTS;
ALTER TABLE succeeded.
```

```
ALTER TABLE test1
DROP (pk, fk, coll);
ALTER TABLE succeeded.
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Cascading Constraints (continued)

Submitting the following statement drops the EMPLOYEE\_ID column, the primary key constraint, and any foreign key constraints referencing the primary key constraint for the EMP2 table:

```
ALTER TABLE emp2 DROP COLUMN employee_id CASCADE CONSTRAINTS;
```

If all columns referenced by the constraints defined on the dropped columns are also dropped, then CASCADE CONSTRAINTS is not required. For example, assuming that no other referential constraints from other tables refer to the PK column, it is valid to submit the following statement without the CASCADE CONSTRAINTS clause for the TEST1 table created on the previous page:

```
ALTER TABLE test1 DROP (pk, fk, coll);
```

## Overview of Indexes

Indexes are created:

- Automatically
  - PRIMARY KEY creation
  - UNIQUE KEY creation
- Manually
  - CREATE INDEX statement
  - CREATE TABLE statement

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Overview of Indexes

Two types of indexes can be created. One type is a unique index. The Oracle server automatically creates a unique index when you define a column or group of columns in a table to have a PRIMARY KEY or a UNIQUE key constraint. The name of the index is the name given to the constraint.

The other type of index is a nonunique index, which a user can create. For example, you can create an index for a FOREIGN KEY column to be used in joins to improve retrieval speed.

You can create an index on one or more columns by issuing the CREATE INDEX statement.

For more information, see *Oracle Database 10g SQL Reference*.

**Note:** You can manually create a unique index, but it is recommended that you create a unique constraint, which implicitly creates a unique index.

## CREATE INDEX with the CREATE TABLE Statement

```
CREATE TABLE NEW_EMP
(employee_id NUMBER(6)
PRIMARY KEY USING INDEX
(CREATE INDEX emp_id_idx ON
NEW_EMP(employee_id)),
first_name VARCHAR2(20),
last_name VARCHAR2(25));
CREATE TABLE succeeded.
```

```
SELECT INDEX_NAME, TABLE_NAME
FROM USER_INDEXES
WHERE TABLE_NAME = 'NEW_EMP';
```

	INDEX_NAME	TABLE_NAME
1	EMP_ID_IDX	NEW_EMP

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### CREATE INDEX with the CREATE TABLE Statement

In the example in the slide, the CREATE INDEX clause is used with the CREATE TABLE statement to create a primary key index explicitly. You can name your indexes at the time of primary key creation to be different from the name of the PRIMARY KEY constraint. The following example illustrates the database behavior if the index is not explicitly named:

```
CREATE TABLE EMP_UNNAMED_INDEX
(employee_id NUMBER(6) PRIMARY KEY ,
first_name VARCHAR2(20),
last_name VARCHAR2(25));
CREATE TABLE succeeded.
SELECT INDEX_NAME, TABLE_NAME
FROM USER_INDEXES
WHERE TABLE_NAME = 'EMP_UNNAMED_INDEX';
```

	INDEX_NAME	TABLE_NAME
1	SYS_C006083	EMP_UNNAMED_INDEX

**CREATE INDEX with the CREATE TABLE Statement (continued)**

Observe that the Oracle server gives a generic name to the index that is created for the PRIMARY KEY column.

You can also use an existing index for your PRIMARY KEY column—for example, when you are expecting a large data load and want to speed up the operation. You may want to disable the constraints while performing the load and then enable them, in which case having a unique index on the primary key will still cause the data to be verified during the load. So you can first create a nonunique index on the column designated as PRIMARY KEY, and then create the PRIMARY KEY column and specify that it should use the existing index. The following examples illustrate this process:

**Step 1: Create the table:**

```
CREATE TABLE NEW_EMP2
(employee_id NUMBER(6)
first_name  VARCHAR2(20),
last_name   VARCHAR2(25)
);
```

**Step 2: Create the index:**

```
CREATE INDEX emp_id_idx2 ON
new_emp2(employee_id);
```

**Step 3: Create the Primary Key:**

```
ALTER TABLE new_emp2 ADD PRIMARY KEY (employee_id) USING INDEX
emp_id_idx2;
```

## Function-Based Indexes

- A function-based index is based on expressions.
- The index expression is built from table columns, constants, SQL functions, and user-defined functions.

```
CREATE INDEX upper_dept_name_idx
ON dept2(UPPER(department_name));
```

**CREATE INDEX succeeded.**

```
SELECT *
FROM   dept2
WHERE  UPPER(department_name) = 'SALES';
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Function-Based Indexes

Function-based indexes defined with the `UPPER(column_name)` or `LOWER(column_name)` keywords allow non-case-sensitive searches. For example, the following index:

```
CREATE INDEX upper_last_name_idx ON emp2 (UPPER(last_name));
```

facilitates processing queries such as:

```
SELECT * FROM emp2 WHERE UPPER(last_name) = 'KING';
```

The Oracle server uses the index only when that particular function is used in a query. For example, the following statement may use the index, but without the `WHERE` clause, the Oracle server may perform a full table scan:

```
SELECT *
FROM   employees
WHERE  UPPER(last_name) IS NOT NULL
ORDER BY UPPER(last_name);
```

**Note:** The `QUERY_REWRITE_ENABLED` initialization parameter must be set to `TRUE` for a function-based index to be used.

The Oracle server treats indexes with columns marked `DESC` as function-based indexes. The columns marked `DESC` are sorted in descending order.

## Removing an Index

- Remove an index from the data dictionary by using the `DROP INDEX` command:

```
DROP INDEX index;
```

- Remove the `UPPER_DEPT_NAME_IDX` index from the data dictionary:

```
DROP INDEX upper_dept_name_idx;  
DROP INDEX succeeded.
```

- To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege:

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Removing an Index

You cannot modify indexes. To change an index, you must drop it and then re-create it. Remove an index definition from the data dictionary by issuing the `DROP INDEX` statement. To drop an index, you must be the owner of the index or have the `DROP ANY INDEX` privilege.

In the syntax:

*index*                      Is the name of the index

**Note:** If you drop a table, indexes and constraints are automatically dropped, but views and sequences remain.



## DROP TABLE ... PURGE

```
DROP TABLE dept80 PURGE;
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### DROP TABLE ... PURGE

Oracle Database 10g introduces a new feature for dropping tables. When you drop a table, the database does not immediately release the space associated with the table. Rather, the database renames the table and places it in a recycle bin, where it can later be recovered with the `FLASHBACK TABLE` statement if you find that you dropped the table in error. If you want to immediately release the space associated with the table at the time you issue the `DROP TABLE` statement, then include the `PURGE` clause as shown in the statement in the slide.

Specify `PURGE` only if you want to drop the table and release the space associated with it in a single step. If you specify `PURGE`, then the database does not place the table and its dependent objects into the recycle bin.

Using this clause is equivalent to first dropping the table and then purging it from the recycle bin. This clause saves you one step in the process. It also provides enhanced security if you want to prevent sensitive material from appearing in the recycle bin.

**Note:** You cannot roll back a `DROP TABLE` statement with the `PURGE` clause, and you cannot recover the table if you drop it with the `PURGE` clause. This feature was not available in earlier releases.

## FLASHBACK TABLE Statement

- Repair tool for accidental table modifications
  - Restores a table to an earlier point in time
  - Benefits: Ease of use, availability, fast execution
  - Performed in place
- Syntax:

```
FLASHBACK TABLE[schema.]table[,
[ schema.]table ]...
TO { TIMESTAMP | SCN } expr
[ { ENABLE | DISABLE } TRIGGERS ];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

## FLASHBACK TABLE Statement

### Self-Service Repair Facility

Oracle Database 10g provides a new SQL DDL command, FLASHBACK TABLE, to restore the state of a table to an earlier point in time in case it is inadvertently deleted or modified. The FLASHBACK TABLE command is a self-service repair tool to restore data in a table along with associated attributes such as indexes or views. This is done while the database is online by rolling back only the subsequent changes to the given table. Compared to traditional recovery mechanisms, this feature offers significant benefits such as ease of use, availability, and faster restoration. It also takes the burden off the DBA to find and restore application-specific properties. The flashback table feature does not address physical corruption caused because of a bad disk.

### Syntax

You can invoke a FLASHBACK TABLE operation on one or more tables, even on tables in different schemas. You specify the point in time to which you want to revert by providing a valid time stamp. By default, database triggers are disabled for all tables involved. You can override this default behavior by specifying the ENABLE TRIGGERS clause.

**Note:** For more information about recycle bin and flashback semantics, refer to *Oracle Database Administrator's Guide 10g Release 2 (10.2)*.

## FLASHBACK TABLE Statement

```
DROP TABLE emp2;
DROP TABLE succeeded.
```

```
SELECT original_name, operation, droptime
FROM recyclebin;
```

	ORIGINAL_NAME	OPERATION	DROPTIME
1	EMP2	DROP	2008-11-13:08:44:39

```
FLASHBACK TABLE emp2 TO BEFORE DROP;
FLASHBACK TABLE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### FLASHBACK TABLE Statement (continued)

#### Syntax and Examples

The example restores the EMP2 table to a state before a DROP statement.

The recycle bin is actually a data dictionary table containing information about dropped objects. Dropped tables and any associated objects, such as indexes, constraints, nested tables, and so on, are not removed and still occupy space. They continue to count against user space quotas, until specifically purged from the recycle bin or the unlikely situation where they must be purged by the database because of tablespace space constraints.

Each user can be thought of as an owner of a recycle bin because, unless a user has the SYSDBA privilege, the only objects that the user has access to in the recycle bin are those that the user owns. A user can view his or her objects in the recycle bin by using the following statement:

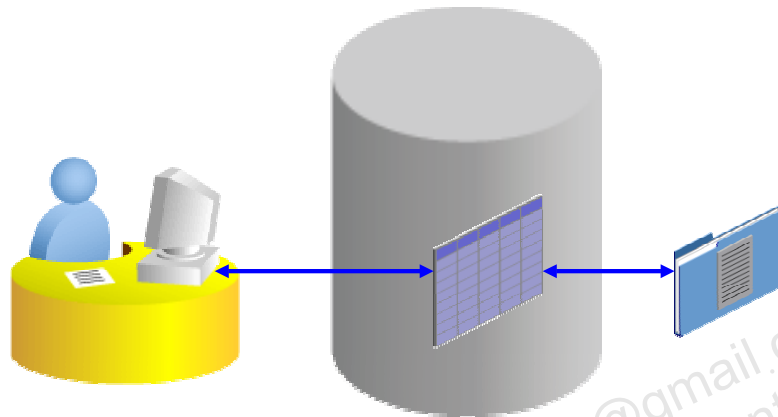
```
SELECT * FROM RECYCLEBIN;
```

When you drop a user, any objects belonging to that user are not placed in the recycle bin and any objects in the recycle bin are purged.

You can purge the recycle bin with the following statement:

```
PURGE RECYCLEBIN;
```

## External Tables



### External Tables

An external table is a read-only table whose metadata is stored in the database but whose data is stored outside the database. This external table definition can be thought of as a view that is used for running any SQL query against external data without requiring that the external data first be loaded into the database. The external table data can be queried and joined directly and in parallel without requiring that the external data first be loaded in the database. You can use SQL, PL/SQL, and Java to query the data in an external table.

The main difference between external tables and regular tables is that externally organized tables are read-only. No data manipulation language (DML) operations are possible, and no indexes can be created on them. However, you can create an external table, and thus unload data, by using the `CREATE TABLE AS SELECT` command.

The Oracle server provides two major access drivers for external tables. One, the loader access driver (or `ORACLE_LOADER`), is used for reading of data from external files whose format can be interpreted by the `SQL*Loader` utility. Note that not all `SQL*Loader` functionality is supported with external tables.

**External Tables (continued)**

The ORACLE\_DATAPUMP access driver can be used to both import and export data using a platform-independent format. The ORACLE\_DATAPUMP access driver writes rows from a SELECT statement to be loaded into an external table as part of a CREATE TABLE . . . ORGANIZATION EXTERNAL . . . AS SELECT statement. You can then use SELECT to read data out of that data file. You can also create an external table definition on another system and use that data file. This allows data to be moved between Oracle databases.

## Creating a Directory for the External Table

Create a `DIRECTORY` object that corresponds to the directory on the file system where the external data source resides.

```
CREATE OR REPLACE DIRECTORY emp_dir
AS '/.../emp_dir';

GRANT READ ON DIRECTORY emp_dir TO hr;
```

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Example of Creating an External Table

Use the `CREATE DIRECTORY` statement to create a directory object. A directory object specifies an alias for a directory on the server's file system where an external data source resides. You can use directory names when referring to an external data source, rather than hard code the operating system path name, for greater file management flexibility.

You must have `CREATE ANY DIRECTORY` system privileges to create directories. When you create a directory, you are automatically granted the `READ` and `WRITE` object privileges and can grant `READ` and `WRITE` privileges to other users and roles. The DBA can also grant these privileges to other users and roles.

A user needs `READ` privileges for all directories used in external tables to be accessed and `WRITE` privileges for the log, bad, and discard file locations being used.

In addition, a `WRITE` privilege is necessary when the external table framework is being used to unload data.

Oracle also provides the `ORACLE_DATAPUMP` type, with which you can unload data (that is, read data from a table in the database and insert it into an external table) and then reload it into an Oracle Database. This is a one-time operation that can be done when the table is created. After the creation and initial population is done, you cannot update, insert, or delete any rows.

**Example of Creating an External Table (continued)****Syntax**

```
CREATE [OR REPLACE] DIRECTORY AS 'path_name' ;
```

In the syntax:

**OR REPLACE** Specify OR REPLACE to re-create the directory database object if it already exists. You can use this clause to change the definition of an existing directory without dropping, re-creating, and regranting database object privileges previously granted on the directory. Users who were previously granted privileges on a redefined directory can continue to access the directory without requiring that the privileges be regranted.

**directory** Specify the name of the directory object to be created. The maximum length of the directory name is 30 bytes. You cannot qualify a directory object with a schema name.

**'path\_name'** Specify the full path name of the operating system directory to be accessed. The path name is case sensitive.

The syntax for using the ORACLE\_DATAPUMP access driver is as follows:

```
CREATE TABLE extract_emps
  ORGANIZATION EXTERNAL ( TYPE ORACLE_DATAPUMP
                           DEFAULT DIRECTORY ...
                           ACCESS PARAMETERS ( ... )
                           LOCATION ( ... )
                           PARALLEL 4
                           REJECT LIMIT UNLIMITED
  AS
  SELECT * FROM ...;
```

## Creating an External Table

```
CREATE TABLE <table_name>
  ( <col_name> <datatype>, ... )
ORGANIZATION EXTERNAL
  (TYPE <access_driver_type>
   DEFAULT DIRECTORY <directory_name>
   ACCESS PARAMETERS
     (... ) )
   LOCATION ('<location_specifier>')
REJECT LIMIT [0 | <number> | UNLIMITED];
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Creating an External Table

You create external tables using the `ORGANIZATION EXTERNAL` clause of the `CREATE TABLE` statement. You are not, in fact, creating a table. Rather, you are creating metadata in the data dictionary that you can use to access external data. You use the `ORGANIZATION` clause to specify the order in which the data rows of the table are stored. By specifying `EXTERNAL` in the `ORGANIZATION` clause, you indicate that the table is a read-only table located outside the database. Note that the external files must already exist outside the database.

`TYPE <access_driver_type>` indicates the access driver of the external table. The access driver is the API that interprets the external data for the database. If you do not specify `TYPE`, Oracle uses the default access driver, `ORACLE_LOADER`. The other option is `ORACLE_DATAPUMP`.

You use the `DEFAULT DIRECTORY` clause to specify one or more Oracle Database directory objects that correspond to directories on the file system where the external data sources may reside.

The optional `ACCESS PARAMETERS` clause enables you to assign values to the parameters of the specific access driver for this external table.



**Creating an External Table (continued)**

Use the `LOCATION` clause to specify one external locator for each external data source. Usually, the `<location_specifier>` is a file, but it need not be.

The `REJECT LIMIT` clause enables you to specify how many conversion errors can occur during a query of the external data before an Oracle error is returned and the query is aborted. The default value is 0.

## Creating an External Table by Using ORACLE\_LOADER

```
CREATE TABLE oldemp (
  fname char(25), lname CHAR(25))
  ORGANIZATION EXTERNAL
  (TYPE ORACLE_LOADER
  DEFAULT DIRECTORY emp_dir
  ACCESS PARAMETERS
  (RECORDS DELIMITED BY NEWLINE
  NOBADFILE
  NOLOGFILE
  FIELDS TERMINATED BY ', '
  (fname POSITION ( 1:20) CHAR,
  lname POSITION (22:41) CHAR))
  LOCATION ('emp.dat'))
  PARALLEL 5
  REJECT LIMIT 200;
CREATE TABLE succeeded.
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Example of Creating an External Table by Using the ORACLE\_LOADER Access Driver

Assume that there is a flat file that has records in the following format:

```
10,jones,11-Dec-1934
20,smith,12-Jun-1972
```

Records are delimited by new lines, and the fields are all terminated by a comma ( , ). The name of the file is /emp\_dir/emp.dat.

To convert this file as the data source for an external table, whose metadata will reside in the database, you must perform the following steps:

1. Create a directory object, emp\_dir, as follows:  
CREATE DIRECTORY emp\_dir AS '/emp\_dir' ;
2. Run the CREATE TABLE command shown in the slide.

The example in the slide illustrates the table specification to create an external table for the file:  
/emp\_dir/emp.dat

## Example of Creating an External Table by Using the ORACLE\_LOADER Access Driver (continued)

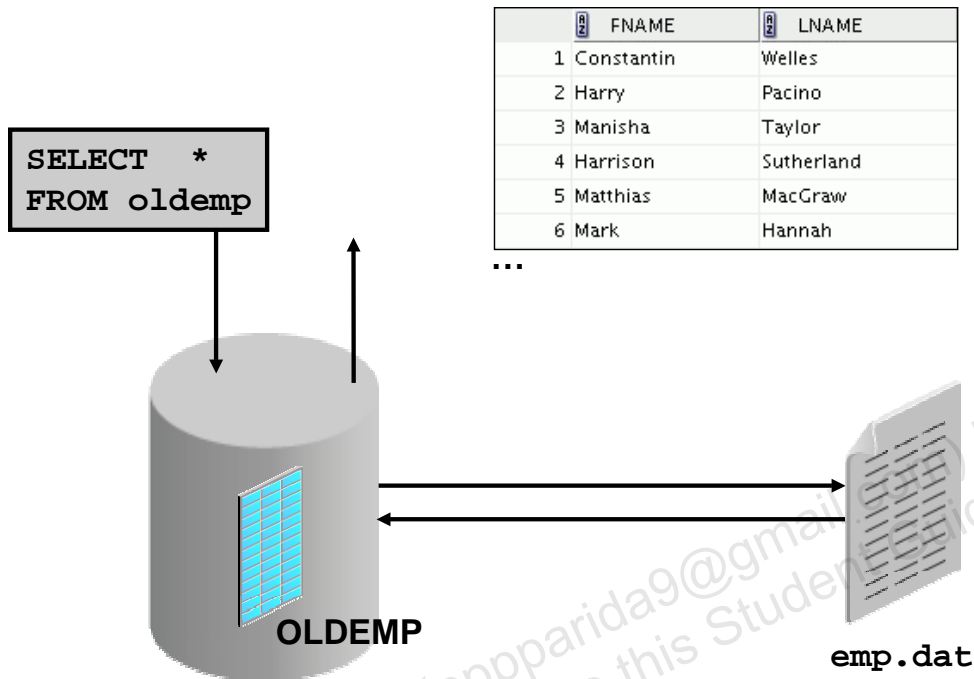
In the example, the TYPE specification is given only to illustrate its use. ORACLE\_LOADER is the default access driver if not specified. The ACCESS PARAMETERS option provides values to parameters of the specific access driver, which are interpreted by the access driver, not by the Oracle server.

The PARALLEL clause enables five parallel execution servers to simultaneously scan the external data sources (files) when executing the INSERT INTO TABLE statement. For example, if PARALLEL=5 were specified, then more than one parallel execution server can be working on a data source. Because external tables can be very large, for performance reasons, it is advisable to specify the PARALLEL clause, or a parallel hint for the query.

The REJECT LIMIT clause specifies that if more than 200 conversion errors occur during a query of the external data, then the query be aborted and an error be returned. These conversion errors can arise when the access driver tries to transform the data in the data file to match the external table definition.

After the CREATE TABLE command executes successfully, the OLDEMP external table can be described and queried like a relational table.

## Querying External Tables



ORACLE

Copyright © 2009, Oracle. All rights reserved.

### Querying External Tables

An external table does not describe any data that is stored in the database. It does not describe how data is stored in the external source. Instead, it describes how the external table layer must present the data to the server. It is the responsibility of the access driver and the external table layer to do the necessary transformations required on the data in the data file so that it matches the external table definition.

When the database server accesses data in an external source, it calls the appropriate access driver to get the data from an external source in a form that the database server expects.

It is important to remember that the description of the data in the data source is separate from the definition of the external table. The source file can contain more or fewer fields than there are columns in the table. Also, the data types for fields in the data source can be different from the columns in the table. The access driver takes care of ensuring that the data from the data source is processed so that it matches the definition of the external table.

## Summary

In this lesson, you should have learned how to:

- Add constraints
- Create indexes
- Create a primary key constraint using an index
- Create indexes using the `CREATE TABLE` statement
- Create function-based indexes
- Drop columns and set column `UNUSED`
- Perform `FLASHBACK` operations
- Create and use external tables

**ORACLE**

Copyright © 2009, Oracle. All rights reserved.

### Summary

Alter tables to add or modify columns or constraints. Create indexes and function-based indexes using the `CREATE INDEX` statement. Drop unused columns. Use `FLASHBACK` mechanics to restore tables. Use the `external_table` clause to create an external table, which is a read-only table whose metadata is stored in the database but whose data is stored outside the database. Use external tables to query data without first loading it into the database. Name your `PRIMARY KEY` column indexes as you create the table with the `CREATE TABLE` statement.

## Practice 2: Overview

This practice covers the following topics:

- Altering tables
- Adding columns
- Dropping columns
- Creating indexes
- Creating external tables

The Oracle logo, consisting of the word "ORACLE" in a bold, sans-serif font, is positioned on the right side of a red horizontal bar.

Copyright © 2009, Oracle. All rights reserved.

### Practice 2: Overview

In this practice, you use the `ALTER TABLE` command to modify columns and add constraints. You use the `CREATE INDEX` command to create indexes when creating a table, along with the `CREATE TABLE` command. You create external tables. You drop columns and use the `FLASHBACK` operation.

**Practice 2**

1. Create the DEPT2 table based on the following table instance chart. Place the syntax in a script called lab\_02\_01.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

<b>Column Name</b>	ID	NAME
<b>Key Type</b>		
<b>Nulls/Unique</b>		
<b>FK Table</b>		
<b>FK Column</b>		
<b>Data type</b>	NUMBER	VARCHAR2
<b>Length</b>	7	25

Name	Null	Type
-----	-----	-----
ID		NUMBER(7)
NAME		VARCHAR2(25)

2. Populate the DEPT2 table with data from the DEPARTMENTS table. Include only the columns that you need.
3. Create the EMP2 table based on the following table instance chart. Place the syntax in a script called lab\_02\_03.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

<b>Column Name</b>	ID	LAST_NAME	FIRST_NAME	DEPT_ID
<b>Key Type</b>				
<b>Nulls/Unique</b>				
<b>FK Table</b>				
<b>FK Column</b>				
<b>Data type</b>	NUMBER	VARCHAR2	VARCHAR2	NUMBER
<b>Length</b>	7	25	25	7

Name	Null	Type
-----	-----	-----
ID		NUMBER(7)
LAST_NAME		VARCHAR2(25)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)

**Practice 2 (continued)**

4. Modify the EMP2 table to allow for longer employee last names. Confirm your modification.

Name	Null	Type
-----	-----	-----
ID		NUMBER(7)
LAST_NAME		VARCHAR2(50)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)

5. Confirm that both the DEPT2 and EMP2 tables are stored in the data dictionary.  
(**Hint:** USER\_TABLES)

	TABLE_NAME
1	DEPT2
2	EMP2

6. Create the EMPLOYEES2 table based on the structure of the EMPLOYEES table. Include only the EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, SALARY, and DEPARTMENT\_ID columns. Name the columns in your new table ID, FIRST\_NAME, LAST\_NAME, SALARY, and DEPT\_ID, respectively. Confirm that the table is created.

Name	Null	Type
-----	-----	-----
ID		NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
DEPT_ID		NUMBER(4)

7. Drop the EMP2 table.  
8. Query the Recycle Bin to see whether the table is present.

	ORIGINAL_NAME	OPERATION	DROPTIME
1	EMP2	DROP	2008-11-14:01:03:45

9. Undrop the EMP2 table.

Name	Null	Type
-----	-----	-----
ID		NUMBER(7)
LAST_NAME		VARCHAR2(50)
FIRST_NAME		VARCHAR2(25)
DEPT_ID		NUMBER(7)

10. Drop the FIRST\_NAME column from the EMPLOYEES2 table. Confirm your modification by checking the description of the table.

Name	Null	Type
-----	-----	-----
ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)
DEPT_ID		NUMBER(4)



**Practice 2 (continued)**

11. In the EMPLOYEES2 table, mark the DEPT\_ID column as UNUSED. Confirm your modification by checking the description of the table.

Name	Null	Type
-----	-----	-----
ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)

12. Drop all the UNUSED columns from the EMPLOYEES2 table. Confirm your modification by checking the description of the table.

Name	Null	Type
-----	-----	-----
ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
SALARY		NUMBER(8,2)

13. Add a table-level PRIMARY KEY constraint to the EMP2 table on the ID column. The constraint should be named at creation. Name the constraint my\_emp\_id\_pk.
14. Create a PRIMARY KEY constraint to the DEPT2 table using the ID column. The constraint should be named at creation. Name the constraint my\_dept\_id\_pk.
15. Add a foreign key reference on the EMP2 table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my\_emp\_dept\_id\_fk.
16. Confirm that the constraints were added by querying the USER\_CONSTRAINTS view. Note the types and names of the constraints.

	CONSTRAINT_NAME	CONSTRAINT_TYPE
1	MY_DEPT_ID_PK	P
2	MY_EMP_ID_PK	P
3	MY_EMP_DEPT_ID_FK	R

17. Display the object names and types from the USER\_OBJECTS data dictionary view for the EMP2 and DEPT2 tables. Note that the new tables are created and a new index is created.

	OBJECT_NAME	OBJECT_TYPE
1	DEPT2	TABLE
2	DEPT_ID_PK	INDEX
3	DEPT_LOCATION_IX	INDEX
4	EMP2	TABLE
5	EMPLOYEES	TABLE
6	EMPLOYEES2	TABLE
7	EMPLOYEES_SEQ	SEQUENCE
8	EMP_DEPARTMENT_IX	INDEX

...

Practice 2 (continued)

If you have time, complete the following exercise:

- 18. Modify the EMP2 table. Add a COMMISSION column of the NUMBER data type, precision 2, scale 2. Add a constraint to the COMMISSION column that ensures that a commission value is greater than zero.
- 19. Drop the EMP2 and DEPT2 tables so that they cannot be restored. Check the Recycle Bin.

<div><div></div><div>2</div></div> ORIGINAL_NAME	<div><div></div><div>2</div></div> OPERATION	<div><div></div><div>2</div></div> DROPTIME
--	--	---

0 rows selected.

- 20. Create the DEPT\_NAMED\_INDEX table based on the following table instance chart. Name the index for the PRIMARY KEY column as DEPT\_PK\_IDX. Confirm that the table is created.

Column Name	Deptno	Dname
Primary Key	Yes	
Data Type	Number	VARCHAR2
Length	4	30

Name	Null	Type
-----	-----	-----
DEPTNO	NOT NULL	NUMBER(4)
DNAME		VARCHAR2(30)