5

Managing Schema Objects

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Manage constraints
- Create and use temporary tables
- Create and use external tables

This lesson contains information about 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.

Lesson Agenda

- Managing constraints:
  - Adding and dropping a constraint
  - Enabling and disabling a constraint
  - Deferring constraints
- Creating and using temporary tables
- Creating and using external tables

ORACLE

Adding a Constraint Syntax

Use the ALTER TABLE statement to:

- Add or drop a constraint, but not to 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>);
```

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

In the syntax:

table	Is the name of the table
constraint	Is the name of the constraint
type	Is the constraint type
column	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;
```

```
table EMP2 altered.
```

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

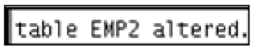
```
table EMP2 altered.
```

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.

Dropping a Constraint

- The `drop_constraint_clause` enables you to drop an integrity constraint from a database.
- Remove the manager constraint from the `EMP2` table:

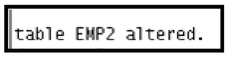
```
ALTER TABLE emp2
DROP CONSTRAINT emp_mgr_fk;
```



```
table EMP2 altered.
```

- Remove the `PRIMARY KEY` constraint on the `DEPT2` table and drop the associated `FOREIGN KEY` constraint on the `EMP2.DEPARTMENT_ID` column:

```
ALTER TABLE emp2
DROP PRIMARY KEY CASCADE;
```



```
table EMP2 altered.
```

The `drop_constraint_clause` enables you to drop an integrity constraint from a database.

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:

table	Is the name of the table
column	Is the name of the column affected by the constraint
constraint	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.

Dropping a `CONSTRAINT ONLINE`

You can specify the `ONLINE` keyword to indicate that DML operations on the table are allowed while dropping the constraint.

```
ALTER TABLE myemp2
DROP CONSTRAINT emp_name_pk ONLINE;
```

You can also drop a constraint using an `ONLINE` keyword.

```
ALTER TABLE myemp2
DROP CONSTRAINT emp_id_pk ONLINE;
```

Use the `ALTER TABLE` statement with the `DROP` clause. The `ONLINE` option of the `DROP` clause indicates that DML operations on the table are allowed while dropping the constraint.

ON DELETE Clause

- Use the `ON DELETE CASCADE` clause to delete child rows when a parent key is deleted:

```
ALTER TABLE dept2 ADD CONSTRAINT dept_lc_fk
FOREIGN KEY (location_id)
REFERENCES locations(location_id) ON DELETE CASCADE;
```

```
table DEPT2 altered.
```

- Use the `ON DELETE SET NULL` clause to set the child rows value to null when a parent key is deleted:

```
ALTER TABLE emp2 ADD CONSTRAINT emp_dt_fk
FOREIGN KEY (Department_id)
REFERENCES departments(department_id) ON DELETE SET NULL;
```

```
table EMP2 altered.
```

ON DELETE

By using the `ON DELETE` clause, you can determine how Oracle Database handles referential integrity if you remove a referenced primary or unique key value.

`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 the 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.

`ON DELETE SET NULL`

When data in the parent key is deleted, the `ON DELETE SET NULL` action causes all the rows in the child table that depend on the deleted parent key value to be converted to null.

If you omit this clause, Oracle does not allow you to delete referenced key values in the parent table that have dependent rows in the child table.

Cascading Constraints

- The `CASCADE CONSTRAINTS` clause:
  - Is used along with the `DROP COLUMN` clause
  - Drops all referential integrity constraints that refer to the `PRIMARY` and `UNIQUE` keys defined on the dropped columns
  - Drops all multicolumn constraints defined on the dropped columns

ORACLE

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

```
CREATE TABLE test1 (
  col1_pk NUMBER PRIMARY KEY,
  col2_fk NUMBER,
  col1 NUMBER,
  col2 NUMBER,
  CONSTRAINT fk_constraint FOREIGN KEY (col2_fk) REFERENCES
    test1,
  CONSTRAINT ck1 CHECK (col1_pk > 0 and col1 > 0),
  CONSTRAINT ck2 CHECK (col2_fk > 0));
```

An error is returned for the following statements:

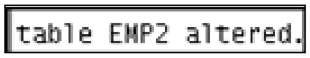
`ALTER TABLE test1 DROP (col1_pk);` —`col1_pk` is a parent key.

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

Cascading Constraints

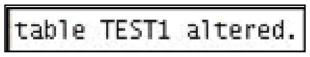
Example:

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



```
table EMP2 altered.
```

```
ALTER TABLE test1
DROP (col1_pk, col2_fk, col1);
```



```
table TEST1 altered.
```

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, CASCADE CONSTRAINTS is not required. For example, assuming that no other referential constraints from other tables refer to the COL1_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 (col1_pk, col2_fk, col1);
```

- 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.

Renaming Table Columns and Constraints

- Use the RENAME TABLE clause of the ALTER TABLE statement to rename tables.

a

```
ALTER TABLE marketing RENAME to new_marketing;
```

- Use the RENAME COLUMN clause of the ALTER TABLE statement to rename table columns.

b

```
ALTER TABLE new_marketing RENAME COLUMN team_id
TO id;
```

- Use the RENAME CONSTRAINT clause of the ALTER TABLE statement to rename any existing constraint for a table.

c

```
ALTER TABLE new_marketing RENAME CONSTRAINT mktg_pk
TO new_mktg_pk;
```

RENAME TABLE clause allows you to rename an existing table in any schema (except the schema *SYS*).To rename a table, you must either be the database owner or the table owner.

When you rename a table column, the new name must not conflict with the name of any existing column in the table. You cannot use any other clauses in conjunction with the RENAME COLUMN clause.

The slide examples use the marketing table with the PRIMARY KEY mktg_pk defined on the id column.

```
CREATE TABLE marketing (team_id  NUMBER(10),
                        target   VARCHAR2(50),
CONSTRAINT mktg_pk PRIMARY KEY(team_id));
```

Example **a** shows that the marketing table is renamed new_marketing. Example **b** shows that the id column of the new_marketing table is renamed mktg_id and example **c** shows that mktg_pk is renamed new_mktg_pk.

When you rename any existing constraint for a table, the new name must not conflict with any of your existing constraint names. You can use the RENAME CONSTRAINT clause to rename system-generated constraint names.

Disabling Constraints

- Execute the `DISABLE` clause of the `ALTER TABLE` statement to deactivate an integrity constraint.
- Apply the `CASCADE` option to disable the primary key and it will disable all dependent `FOREIGN KEY` constraints automatically as well.

```
ALTER TABLE emp2
DISABLE CONSTRAINTS emp_dt_pk;
```

```
table EMP2 altered.
```

```
ALTER TABLE dept2
DISABLE primary key CASCADE;
```

```
table DEPT2 altered.
```

You can disable a constraint, without dropping it or re-creating it, by using the `ALTER TABLE` statement with the `DISABLE` clause. You can also disable the primary key or unique key using the `CASCADE` option.

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 that is currently disabled in the table definition by using the `ENABLE` clause.

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

```
table EMP2 altered.
```

- A `UNIQUE` index is automatically created if you enable a `UNIQUE` key or a `PRIMARY KEY` 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, it can be used by these keys.
- You can use the `ENABLE` clause in both the `CREATE TABLE` statement and the `ALTER TABLE` statement.

Constraint States

An integrity constraint defined on a table can be in one of the following states:

- `ENABLE VALIDATE`
- `ENABLE NOVALIDATE`
- `DISABLE VALIDATE`
- `DISABLE NOVALIDATE`

```
ALTER TABLE dept2
ENABLE NOVALIDATE PRIMARY KEY;
```

```
table DEPT2 altered.
```

You can enable or disable integrity constraints at the table level using the `CREATE TABLE` or `ALTER TABLE` statement. You can also set constraints to `VALIDATE` or `NOVALIDATE`, in any combination with `ENABLE` or `DISABLE`, where:

- `ENABLE` ensures that all incoming data conforms to the constraint
- `DISABLE` allows incoming data, regardless of whether it conforms to the constraint
- `VALIDATE` ensures that existing data conforms to the constraint
- `NOVALIDATE` means that some existing data may not conform to the constraint

`ENABLE VALIDATE` is the same as `ENABLE`. The constraint is checked and is guaranteed to hold for all rows. `ENABLE NOVALIDATE` means that the constraint is checked, but it does not have to be true for all rows. This allows existing rows to violate the constraint, while ensuring that all new or modified rows are valid. In an `ALTER TABLE` statement, `ENABLE NOVALIDATE` resumes constraint checking on disabled constraints without first validating all data in the table. `DISABLE NOVALIDATE` is the same as `DISABLE`. The constraint is not checked and is not necessarily true. `DISABLE VALIDATE` disables the constraint, drops the index on the constraint, and disallows any modification of the constrained columns.

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

You can defer checking constraints for validity until the end of the transaction. A constraint is deferred if the system does not check whether the constraint is satisfied, until a COMMIT statement is submitted. If a deferred constraint is violated, the database returns an error and the transaction is not committed and it is rolled back. If a constraint is immediate (not deferred), 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 data manipulation language (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 (default), and either initially deferred or INITIALLY IMMEDIATE (default). 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 the rows are validated.

Difference Between `INITIALLY DEFERRED` and `INITIALLY IMMEDIATE`

`INITIALLY DEFERRED`	Waits until the transaction ends to check the constraint
`INITIALLY IMMEDIATE`	Checks the constraint at the end of the statement execution

```
CREATE TABLE emp_new_sal (salary NUMBER
        CONSTRAINT sal_ck
        CHECK (salary > 100)
        DEFERRABLE INITIALLY IMMEDIATE,
        bonus NUMBER
        CONSTRAINT bonus_ck
        CHECK (bonus > 0 )
        DEFERRABLE INITIALLY DEFERRED );
```

```
table EMP_NEW_SAL created.
```

A constraint that is defined as deferrable can be specified as either `INITIALLY DEFERRED` or `INITIALLY IMMEDIATE`. The `INITIALLY IMMEDIATE` clause is the default.

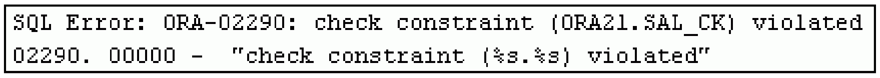
In the slide example:

- The `sal_ck` constraint is created as `DEFERRABLE INITIALLY IMMEDIATE`
- The `bonus_ck` constraint is created as `DEFERRABLE INITIALLY DEFERRED`

After creating the `emp_new_sal` table, as shown in the slide, you attempt to insert values into the table and observe the results. When both the `sal_ck` and `bonus_ck` constraints are satisfied, the rows are inserted without an error.

Example 1: Insert a row that violates `sal_ck`. In the `CREATE TABLE` statement, `sal_ck` is specified as an initially immediate constraint. This means that the constraint is verified immediately after the `INSERT` statement and you observe an error.

```
INSERT INTO emp_new_sal VALUES(90,5);
```



```
SQL Error: ORA-02290: check constraint (ORA21.SAL_CK) violated
02290. 00000 -  "check constraint (%s.%s) violated"
```

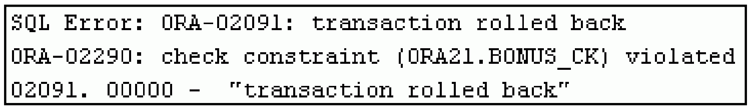
Example 2: Insert a row that violates `bonus_ck`. In the `CREATE TABLE` statement, `bonus_ck` is specified as deferrable and also initially deferred. Therefore, the constraint is not verified until you `COMMIT` or set the constraint state back to immediate.

```
INSERT INTO emp_new_sal VALUES(110, -1);
```

```
1 rows inserted
```

The row insertion is successful. But you observe an error when you commit the transaction.

```
COMMIT;
```



```
SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (ORA21.BONUS_CK) violated
02091. 00000 -  "transaction rolled back"
```

The commit failed due to constraint violation. Therefore, at this point, the transaction is rolled back by the database.

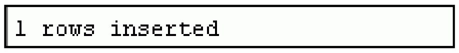
Example 3: Set the DEFERRED status to all constraints that can be deferred. Note that you can also set the DEFERRED status to a single constraint if required.

```
SET CONSTRAINTS ALL DEFERRED;
```

```
constraints ALL succeeded.
```

Now, if you attempt to insert a row that violates the sal_ck constraint, the statement is executed successfully.

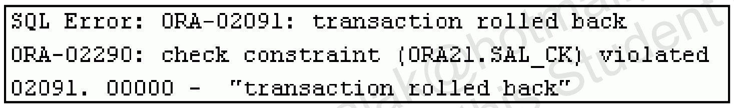
```
INSERT INTO emp_new_sal VALUES(90,5);
```



```
1 rows inserted
```

However, you observe an error when you commit the transaction. The transaction fails and is rolled back. This is because both the constraints are checked upon COMMIT.

```
COMMIT;
```



```
SQL Error: ORA-02091: transaction rolled back
ORA-02290: check constraint (ORA21.SAL_CK) violated
02091. 00000 -  "transaction rolled back"
```

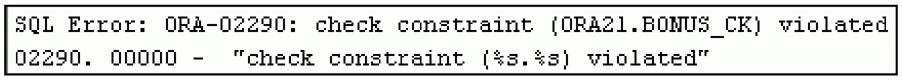
Example 4: Set the IMMEDIATE status to both the constraints that were set as DEFERRED in the previous example.

```
SET CONSTRAINTS ALL IMMEDIATE;
```

```
constraints ALL succeeded.
```

You observe an error if you attempt to insert a row that violates either sal_ck or bonus_ck.

```
INSERT INTO emp_new_sal VALUES(110, -1);
```



```
SQL Error: ORA-02290: check constraint (ORA21.BONUS_CK) violated
02290. 00000 -  "check constraint (%s.%s) violated"
```

Note: If you create a table without specifying constraint deferability, the constraint is checked immediately at the end of each statement. For example, with the CREATE TABLE statement of the newemp_details table, if you do not specify the newemp_det_pk constraint deferability, the constraint is checked immediately.

```
CREATE TABLE newemp_details(emp_id NUMBER, emp_name
VARCHAR2(20),
CONSTRAINT newemp_det_pk PRIMARY KEY(emp_id));
```

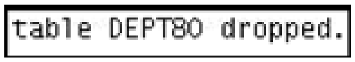
When you attempt to defer the newemp_det_pk constraint that is not deferrable, you observe the following error:

```
SET CONSTRAINT newemp_det_pk DEFERRED;
```

```
SQL Error: ORA-02447: cannot defer a constraint that is not deferrable
```

DROP TABLE … PURGE

```
DROP TABLE emp_new_sal PURGE;
```



```
table DEPT80 dropped.
```

Oracle Database provides a 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, include the `PURGE` clause as shown in the statement in the slide.

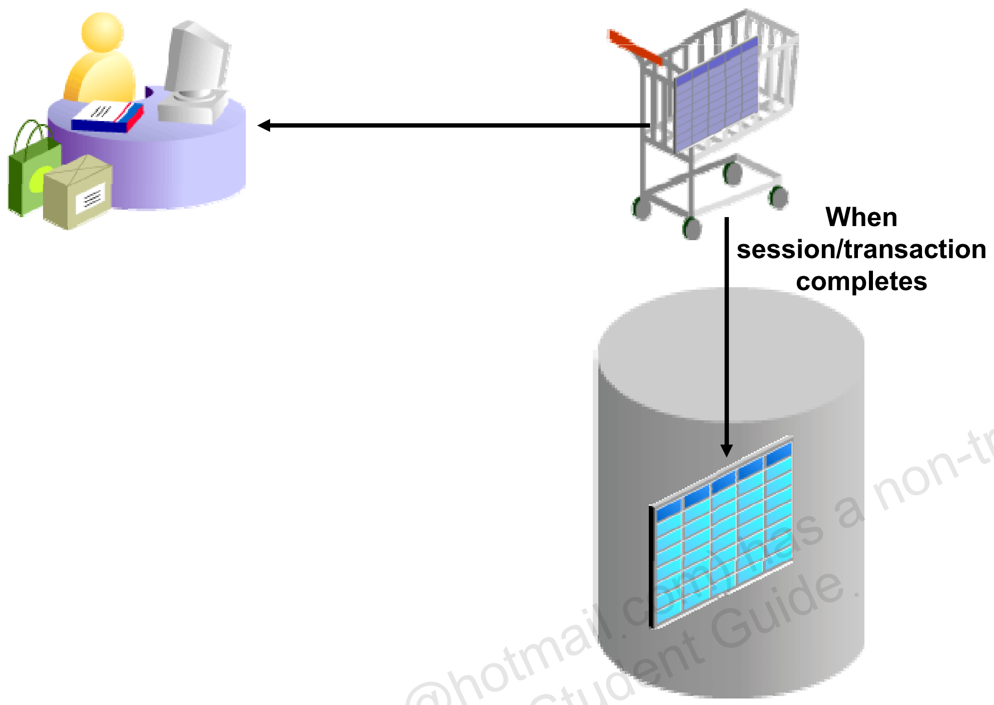
Specify `PURGE` only if you want to drop the table and release the space associated with it in a single step. If you specify `PURGE`, the database does not place the table and its dependent objects into the recycle bin.

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.

Lesson Agenda

- Managing constraints:
  - Adding and dropping a constraint
  - Enabling and disabling a constraint
  - Deferring constraints
- Creating and using temporary tables
- Creating and using external tables

Temporary Tables



A temporary table is a table that holds data that exists only for the duration of a transaction or session. Data in a temporary table is private to the session, which means that each session can see and modify only its own data.

Temporary tables are useful in applications where a result set must be buffered. For example, a shopping cart in an online application can be a temporary table. Each item is represented by a row in the temporary table. While you are shopping in an online store, you can keep on adding or removing items from your cart. During the session, this cart data is private. After you finalize your shopping and make the payments, the application moves the row for the chosen cart to a permanent table. At the end of the session, the data in the temporary table is automatically dropped.

Because temporary tables are statically defined, you can create indexes for them. Indexes created on temporary tables are also temporary. The data in the index has the same session or transaction scope as the data in the temporary table. You can also create a view or trigger on a temporary table.

Creating a Temporary Table

```
CREATE GLOBAL TEMPORARY TABLE cart(n NUMBER,d DATE)
ON COMMIT DELETE ROWS;
```
1

```
CREATE GLOBAL TEMPORARY TABLE today_sales
ON COMMIT PRESERVE ROWS AS
     SELECT * FROM orders
             WHERE order_date = SYSDATE;
```
2

To create a temporary table, you can use the following command:

```
CREATE GLOBAL TEMPORARY TABLE tablename
ON COMMIT [PRESERVE | DELETE] ROWS
```

By associating one of the following settings with the `ON COMMIT` clause, you can decide whether the data in the temporary table is transaction-specific (default) or session-specific.

1. `DELETE ROWS`: As shown in example 1 in the slide, the `DELETE ROWS` setting creates a temporary table that is transaction-specific. A session becomes bound to the temporary table with a transaction's first insert into the table. The binding goes away at the end of the transaction. The database truncates the table (delete all rows) after each commit.
2. `PRESERVE ROWS`: As shown in example 2 in the slide, the `PRESERVE ROWS` setting creates a temporary table that is session-specific. Each sales representative session can store its own sales data for the day in the table. When a salesperson performs first insert on the `today_sales` table, his or her session gets bound to the `today_sales` table. This binding goes away at the end of the session or by issuing a `TRUNCATE` of the table in the session. The database truncates the table when you terminate the session.

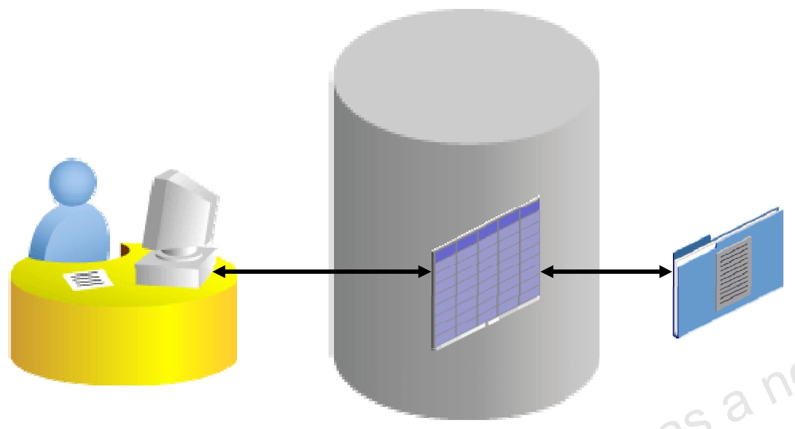
When you create a temporary table in an Oracle database, you create a static table definition. Like permanent tables, temporary tables are defined in the data dictionary. However, temporary tables and their indexes do not automatically allocate a segment when created. Instead, temporary segments are allocated when data is first inserted. Until data is loaded in a session, the table appears empty.

Lesson Agenda

- Managing constraints:
  - Adding and dropping a constraint
  - Enabling and disabling a constraint
  - Deferring constraints
- Creating and using temporary tables
- Creating and using external tables

ORACLE

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 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. The `ORACLE_DATAPUMP` access driver can be used to both import and export data by 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 ora_21;
```

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.

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.

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];
```

You create external tables by 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 application programming interface (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.

Use the LOCATION clause to specify one external locator for each external data source. Usually, *<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.

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 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
  FIELDS(fname POSITION ( 1:20) CHAR,
  lname POSITION (22:41) CHAR))
 LOCATION ('emp.dat'));
```

```
table OLDEMP created.
```

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. 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.

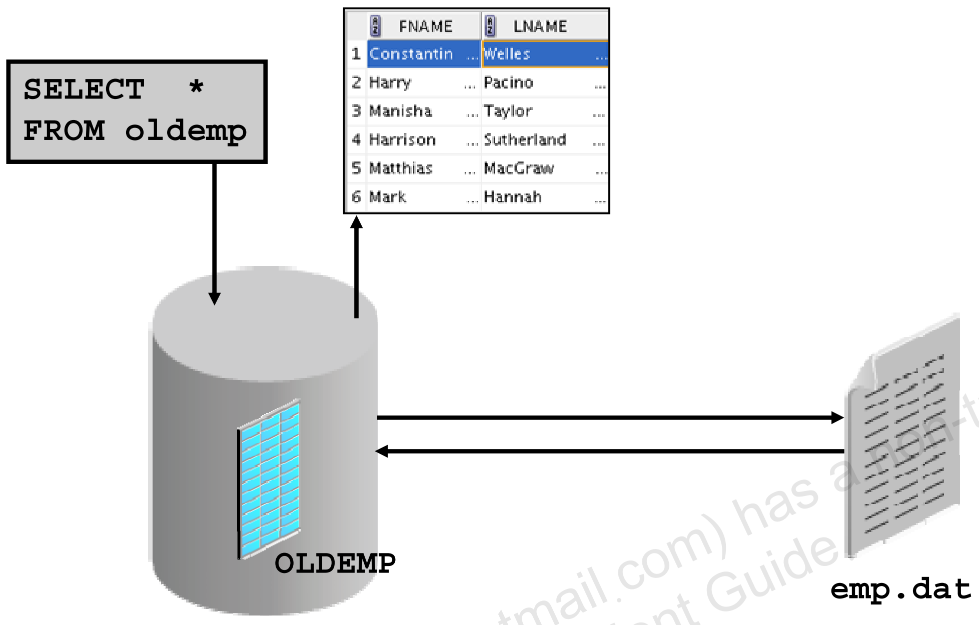
The example in the slide illustrates the table specification to create an external table for the file:

```
/emp_dir/emp.dat
```

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.

After the CREATE TABLE command executes successfully, the OLDEMP external table can be described and queried in the same way as a relational table.

Querying External Tables



	FNAME	LNAME
1	Constantin ...	Welles ...
2	Harry ...	Pacino ...
3	Manisha ...	Taylor ...
4	Harrison ...	Sutherland ...
5	Matthias ...	MacGraw ...
6	Mark ...	Hannah ...

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.

Creating an External Table by Using ORACLE_DATAPUMP: Example

```
CREATE TABLE emp_ext
   (employee_id, first_name, last_name)
    ORGANIZATION EXTERNAL
      (
       TYPE ORACLE_DATAPUMP
       DEFAULT DIRECTORY emp_dir
       LOCATION
         ('emp1.exp','emp2.exp')
      )
    PARALLEL
AS
SELECT employee_id, first_name, last_name
FROM   employees;
```

```
table EMP_EXT created.
```

You can perform the unload and reload operations with external tables by using the ORACLE_DATAPUMP access driver.

Note: In the context of external tables, loading data refers to the act of data being read from an external table and loaded into a table in the database. Unloading data refers to the act of reading data from a table and inserting it into an external table.

The example in the slide illustrates the table specification to create an external table by using the ORACLE_DATAPUMP access driver. Data is then populated into the two files: emp1.exp and emp2.exp.

To populate data read from the EMPLOYEES table into an external table, 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.

Note: The emp_dir directory is the same as created in the previous example of using ORACLE_LOADER.

You can query the external table by executing the following code:

```
SELECT * FROM emp_ext;
```

Quiz

A `FOREIGN KEY` constraint enforces the following action: When the data in the parent key is deleted, all the rows in the child table that depend on the deleted parent key values are also deleted.

a. True

b. False

Answer: b

Summary

In this lesson, you should have learned how to:

- Manage constraints
- Create and use temporary tables
- Create and use external tables

In this lesson, you learned how to perform the following tasks for schema object management:

- Alter tables to add or modify columns or constraints.
- Use the `ORGANIZATION EXTERNAL` clause of the `CREATE TABLE` statement to create an external table. An external table 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.

Practice 5: Overview

This practice covers the following topics:

- Adding and dropping constraints
- Deferring constraints
- Creating external tables

In this practice, you use the `ALTER TABLE` command to add, drop and defer constraints. You create external tables.