ORACLE SQL

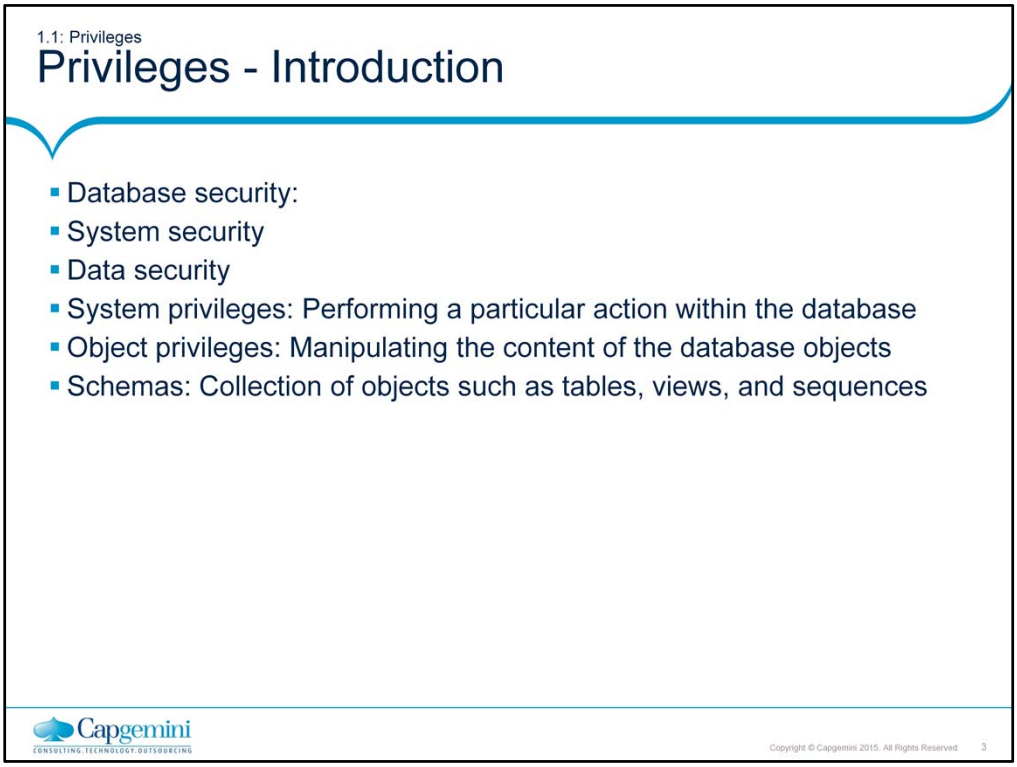
Lesson 01: Privileges, Multitable Inserts, External Tables

Lesson Objectives

- To understand the following topics:
  - Differentiate system privileges from object privileges
  - Grant privileges on tables
  - Grant roles
  - Distinguish between privileges and roles
  - Manipulating data by using subqueries
  - Specifying explicit default values in the INSERT and UPDATE statements
  - Using the following types of multitable INSERTs:
  - Unconditional INSERT
  - Pivoting INSERT
  - Conditional INSERT ALL
  - Conditional INSERT FIRST
  - Merging rows in a table
  - Tracking the changes to data over a period of time

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



Privileges

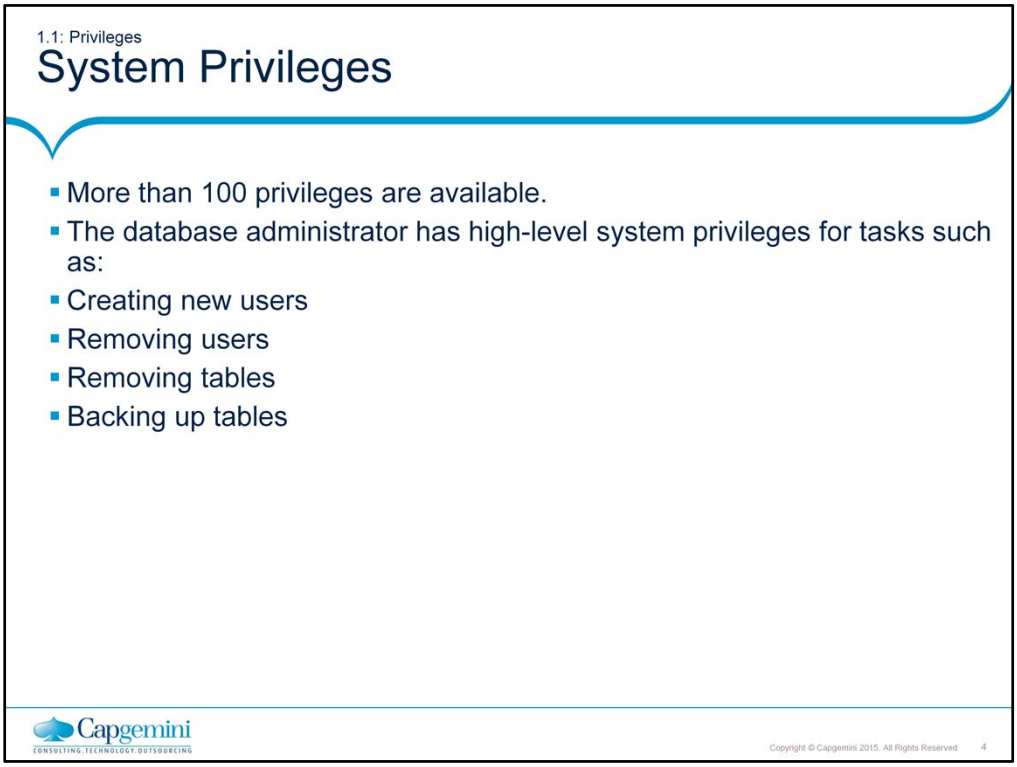
A privilege is the right to execute particular SQL statements. The database administrator (DBA) is a high-level user with the ability to create users and grant users access to the database and its objects. Users require system privileges to gain access to the database and object privileges to manipulate the content of the objects in the database. Users can also be given the privilege to grant additional privileges to other users or to roles, which are named groups of related privileges.

Schemas

A schema is a collection of objects such as tables, views, and sequences. The schema is owned by a database user and has the same name as that user.

A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type. An object privilege provides the user the ability to perform a particular action on a specific schema object.

For more information, see the Oracle Database 2 Day DBA 11g Release 2 (11.2) reference manual.

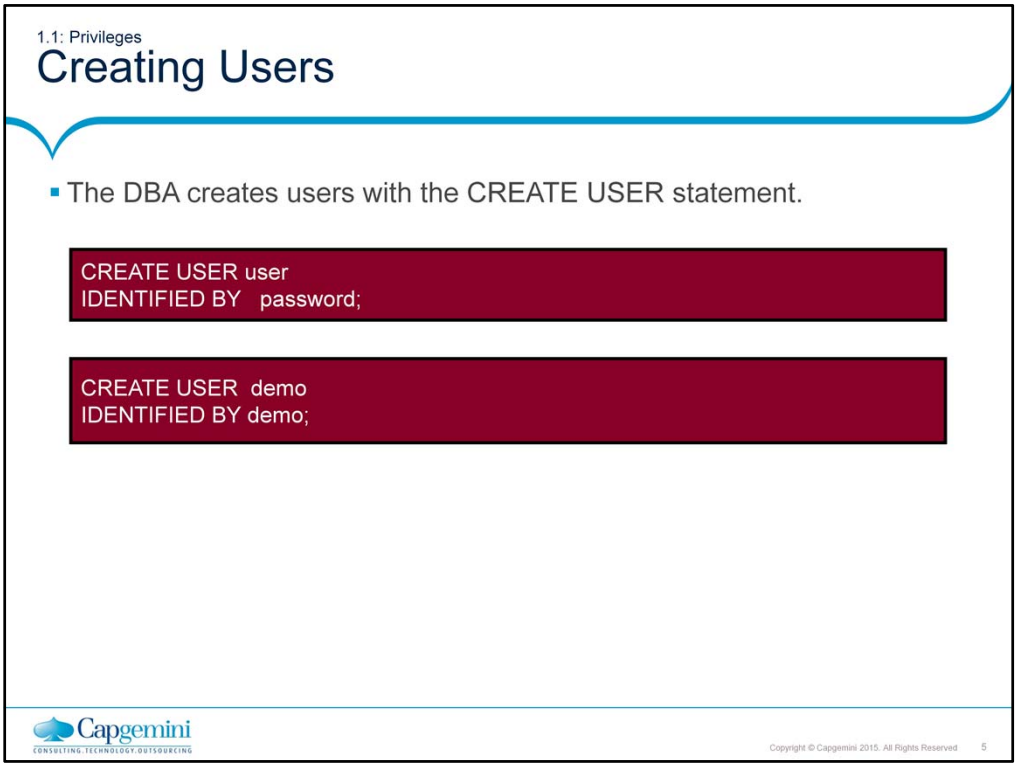


System Privileges

More than 100 distinct system privileges are available for users and roles. Typically, system privileges are provided by the database administrator (DBA).

Typical DBA Privileges

System Privilege	Operations Authorized
CREATE USER	Grantee can create other Oracle users.
DROP USER	Grantee can drop another user.
DROP ANY TABLE	Grantee can drop a table in any schema.
BACKUP ANY TABLE	Grantee can back up any table in any schema with the export utility.
SELECT ANY TABLE	Grantee can query tables, views, or materialized views in any schema.
CREATE ANY TABLE	Grantee can create tables in any schema.



Creating Users

The DBA creates the user by executing the CREATE USER statement. The user does not have any privileges at this point. The DBA can then grant privileges to that user. These privileges determine what the user can do at the database level.

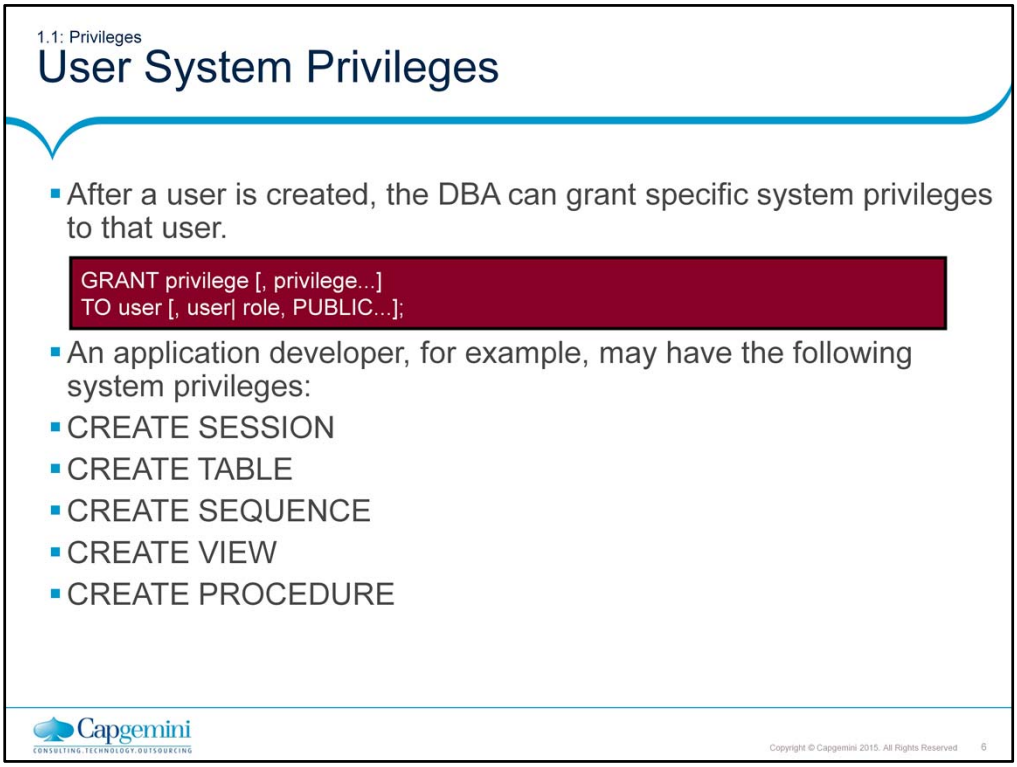
The slide gives the abridged syntax for creating a user.

In the syntax:

user	Is the name of the user to be created
Password	Specifies that the user must log in with this password

For more information, see the Oracle Database11g SQL Reference.

Note: Starting with Oracle Database 11g, passwords are case-sensitive.



Typical User Privileges

After the DBA creates a user, the DBA can assign privileges to that user.

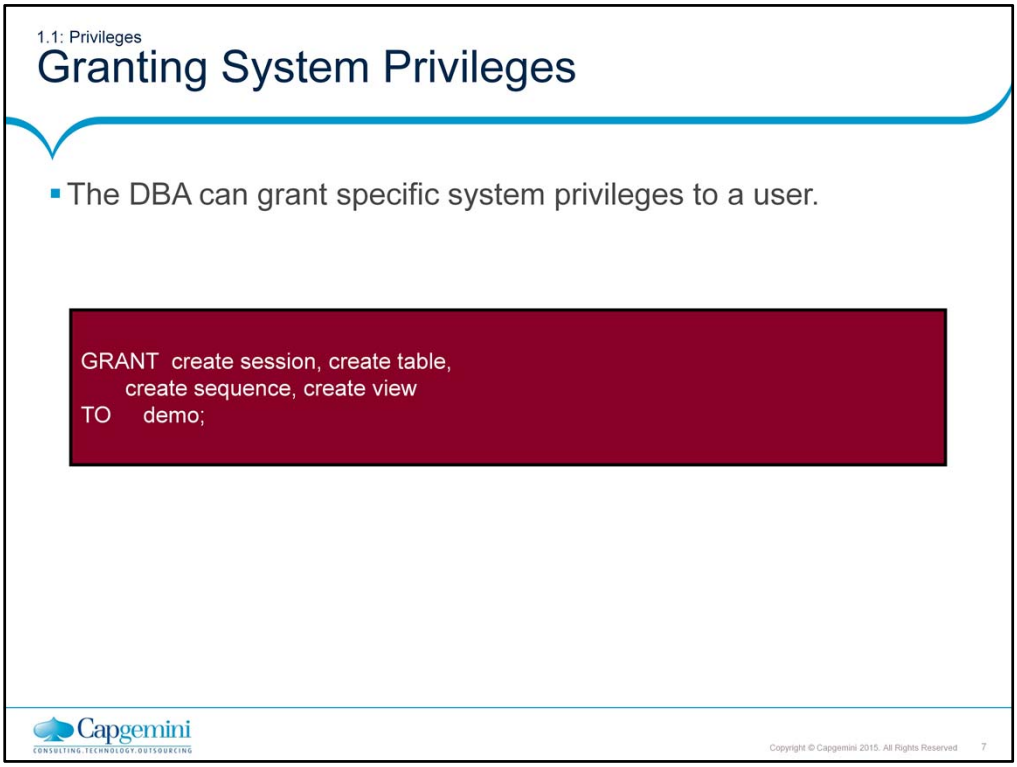
System Privilege	Operations Authorized
CREATE SESSION	Connect to the database.
CREATE TABLE	Create tables in the user's schema.
CREATE SEQUENCE	Create a sequence in the user's schema.
CREATE VIEW	Create a view in the user's schema.
CREATE PROCEDURE	Create a stored procedure, function, or package in the user's schema.

In the syntax:

privilege Is the system privilege to be granted

user|role|PUBLIC Is the name of the user, the name of the role, or PUBLIC (which designates that every user is granted the privilege)

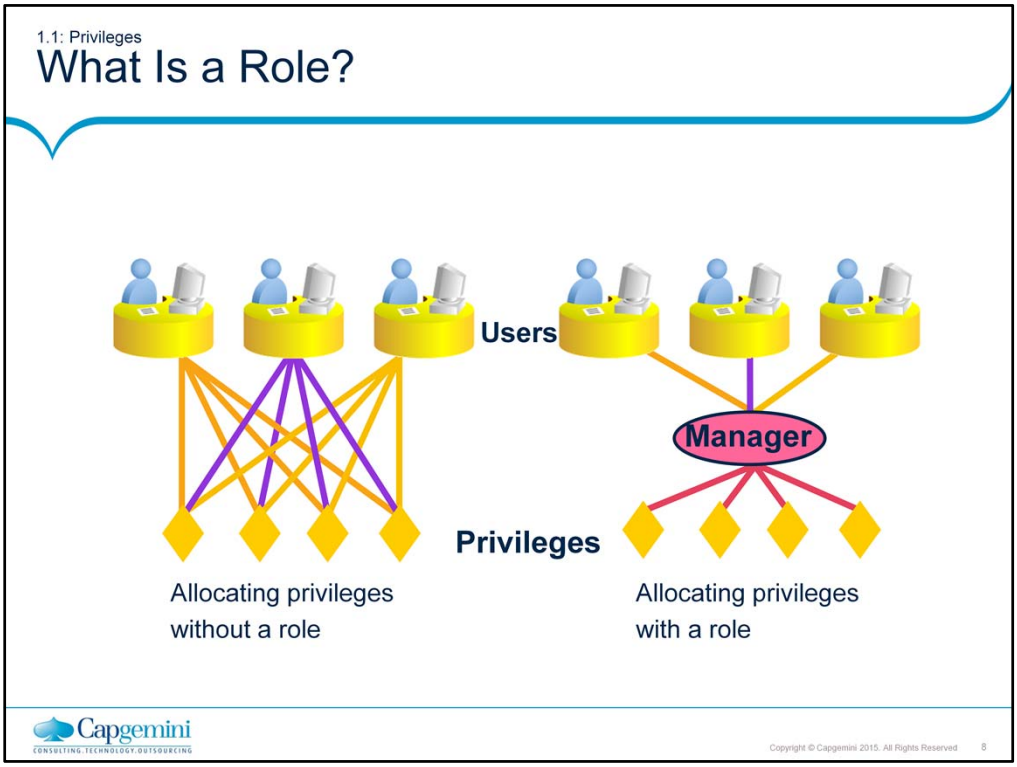
Note: Current system privileges can be found in the SESSION_PRIVS dictionary view. Data dictionary is a collection of tables and views created and maintained by the Oracle Server. They contain information about the database.



Granting System Privileges

The DBA uses the GRANT statement to allocate system privileges to the user. After the user has been granted the privileges, the user can immediately use those privileges.

In the example in the slide, the demo user has been assigned the privileges to create sessions, tables, sequences, and views.



What Is a Role?

A role is a named group of related privileges that can be granted to the user. This method makes it easier to revoke and maintain privileges. A user can have access to several roles, and several users can be assigned the same role. Roles are typically created for a database application.

Creating and Assigning a Role

First, the DBA must create the role. Then the DBA can assign privileges to the role and assign the role to users.

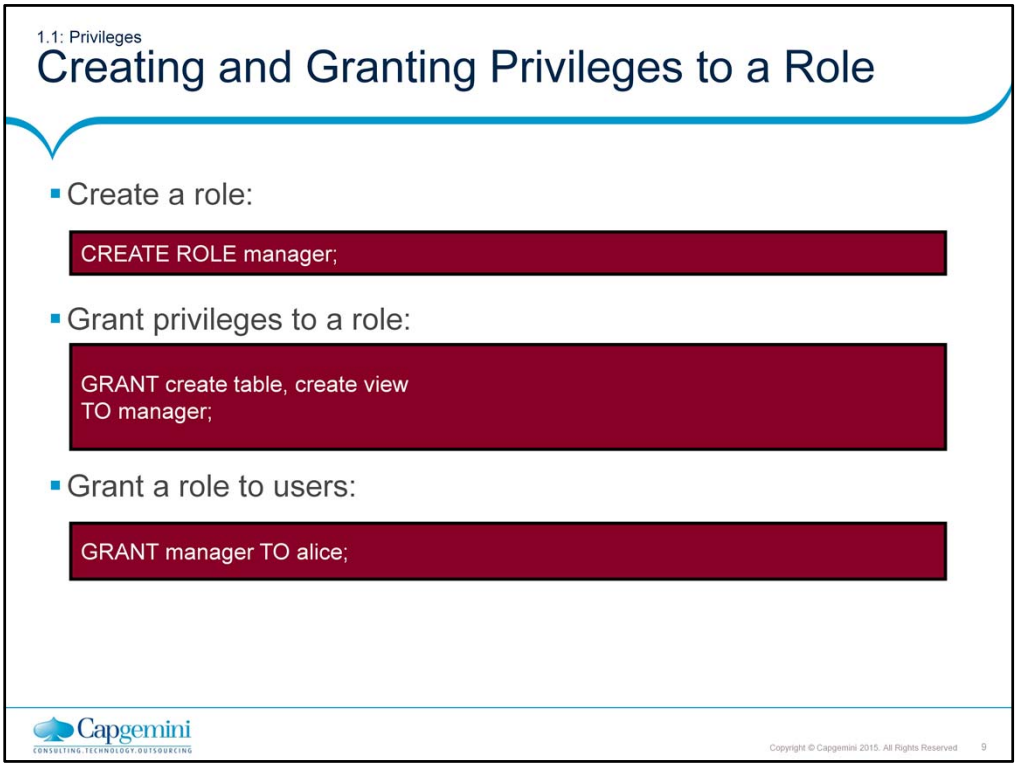
Syntax

```
CREATE   ROLE  role;
```

In the syntax:

role Is the name of the role to be created

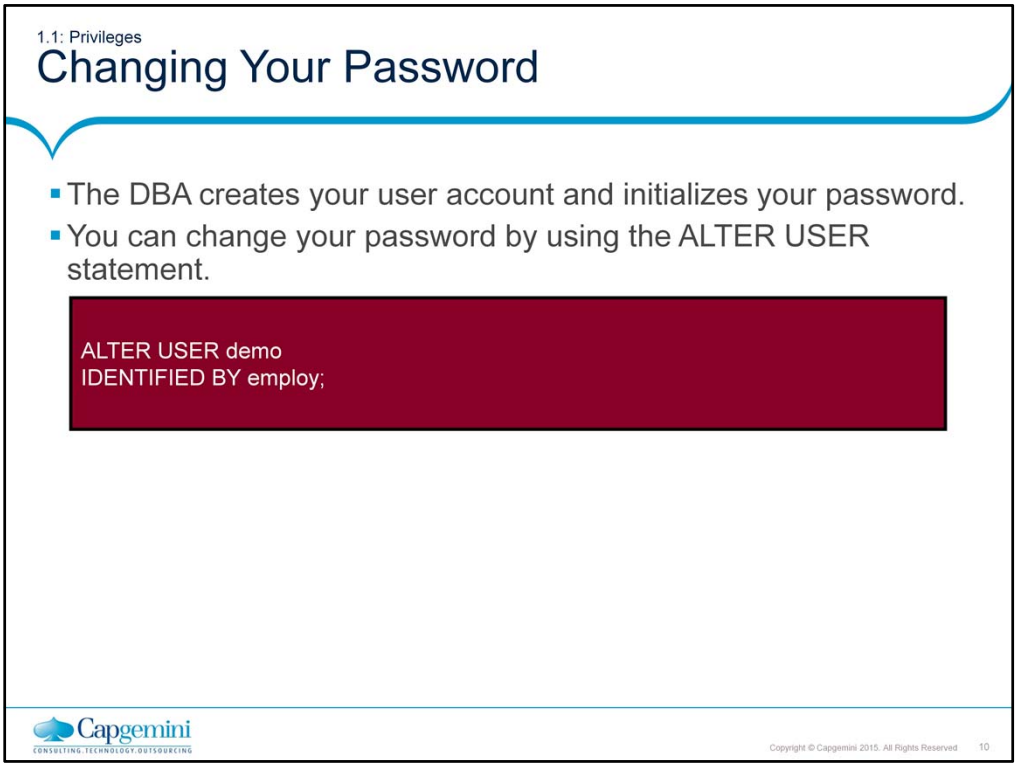
After the role is created, the DBA can use the GRANT statement to assign the role to users as well as assign privileges to the role. A role is not a schema object, therefore any user can add privileges to a role.



Creating a Role

The example in the slide creates a manager role and then enables the manager to create tables and views. It then grants user alice the role of a manager. Now alice can create tables and views.

If users have multiple roles granted to them, they receive all the privileges associated with all the roles.



Changing Your Password

The DBA creates an account and initializes a password for every user. You can change your password by using the ALTER USER statement.

The slide example shows that the demo user changes the password by using the ALTER USER statement.

Syntax

ALTER USER user IDENTIFIED BY password;

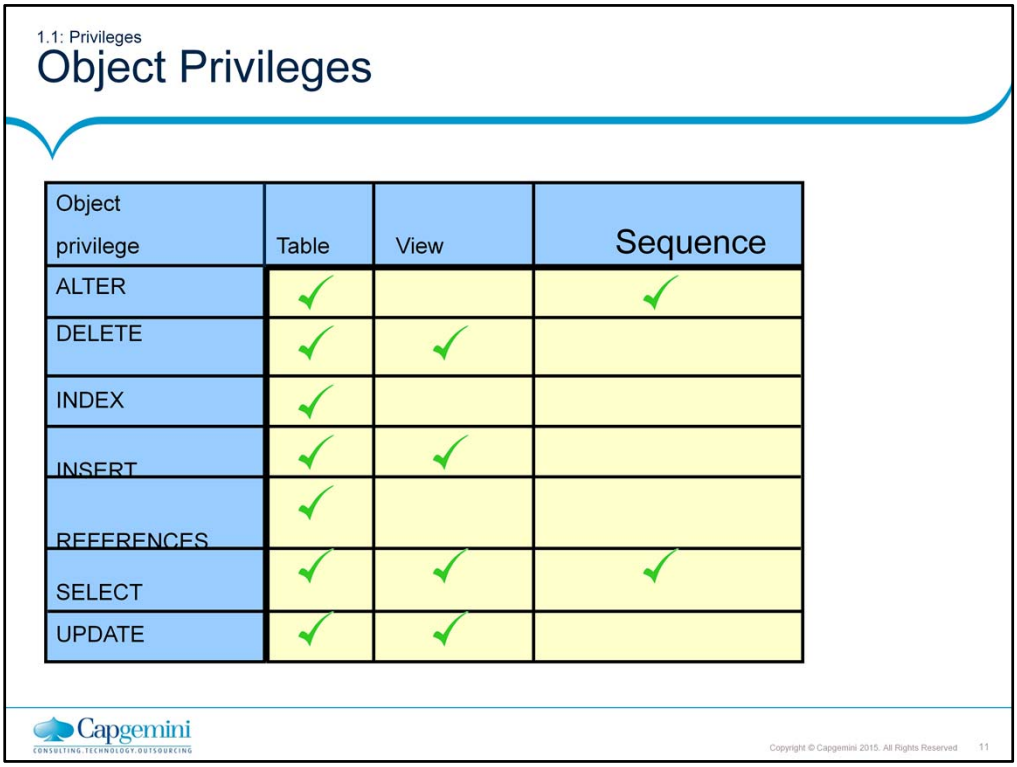
In the syntax:

user	Is the name of the user
password	Specifies the new password

Although this statement can be used to change your password, there are many other options. You must have the ALTER USER privilege to change any other option.

For more information, see the Oracle Database11g SQL Reference manual.

Note: SQL*Plus has a PASSWORD command (PASSW) that can be used to change the password of a user when the user is logged in. This command is not available in SQL Developer.

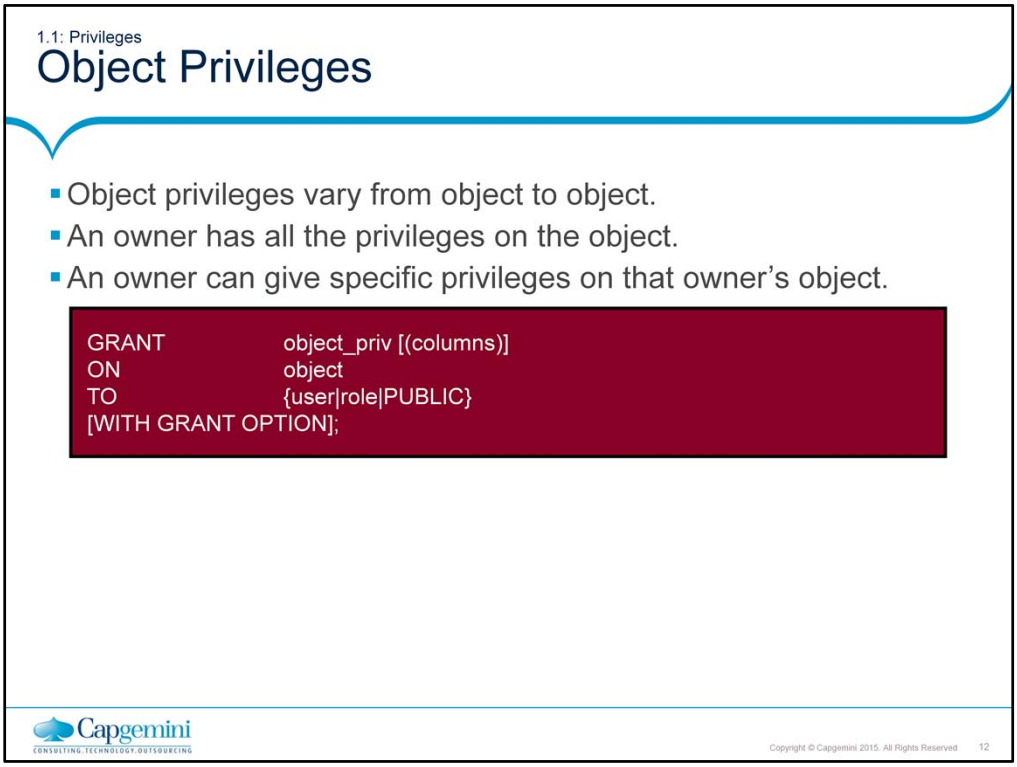


Object Privileges

An object privilege is a privilege or right to perform a particular action on a specific table, view, sequence, or procedure. Each object has a particular set of grantable privileges. The table in the slide lists the privileges for various objects. Note that the only privileges that apply to a sequence are SELECT and ALTER. UPDATE, REFERENCES, and INSERT can be restricted by specifying a subset of updatable columns.

A SELECT privilege can be restricted by creating a view with a subset of columns and granting the SELECT privilege only on the view. A privilege granted on a synonym is converted to a privilege on the base table referenced by the synonym.

Note: With the REFERENCES privilege, you can ensure that other users can create FOREIGN KEY constraints that reference your table.



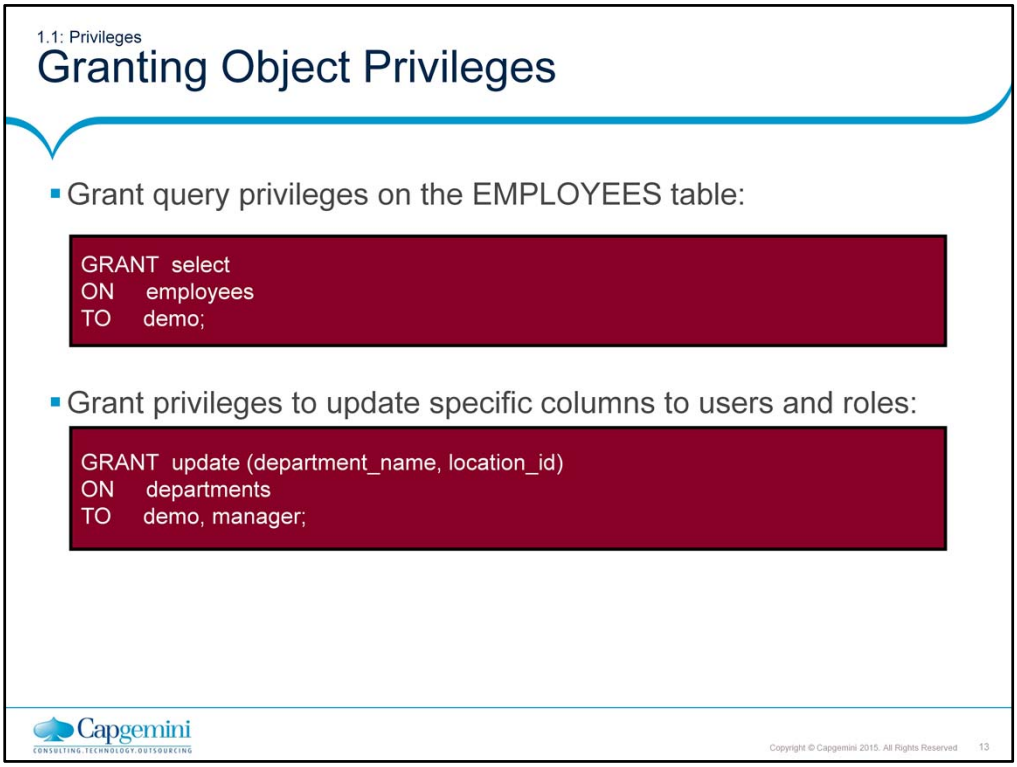
Granting Object Privileges

Different object privileges are available for different types of schema objects. A user automatically has all object privileges for schema objects contained in the user's schema. A user can grant any object privilege on any schema object that the user owns to any other user or role. If the grant includes WITH GRANT OPTION, the grantee can further grant the object privilege to other users; otherwise, the grantee can use the privilege but cannot grant it to other users.

In the syntax:

object_priv	Is an object privilege to be granted
ALL	Specifies all object privileges
columns	Specifies the column from a table or view on which privileges are granted
ON object	Is the object on which the privileges are granted
TO	Identifies to whom the privilege is granted
PUBLIC	Grants object privileges to all users
WITH GRANT OPTION	Enables the grantee to grant the object privileges to other users and roles

Note: In the syntax, schema is the same as the owner's name.



Guidelines

To grant privileges on an object, the object must be in your own schema, or you must have been granted the object privileges WITH GRANT OPTION.

An object owner can grant any object privilege on the object to any other user or role of the database.

The owner of an object automatically acquires all object privileges on that object.

The first example in the slide grants the demo user the privilege to query your EMPLOYEES table. The second example grants UPDATE privileges on specific columns in the DEPARTMENTS table to demo and to the manager role.

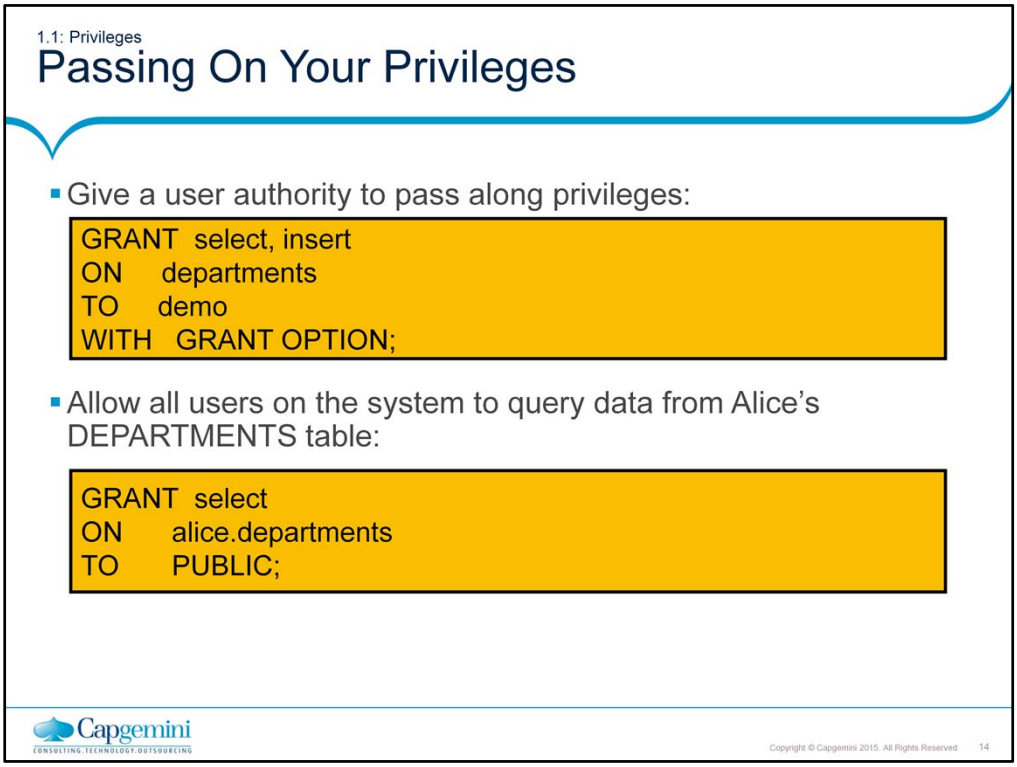
For example, if your schema is oraxx, and the demo user now wants to use a SELECT statement to obtain data from your EMPLOYEES table, the syntax he or she must use is:

```
SELECT  * FROM oraxx.employees;
```

Alternatively, the demo user can create a synonym for the table and issue a SELECT statement from the synonym:

```
CREATE SYNONYM emp FOR oraxx.employees;
SELECT * FROM emp;
```

Note: DBAs generally allocate system privileges; any user who owns an object can grant object privileges.



Passing On Your Privileges

WITH GRANT OPTION Keyword

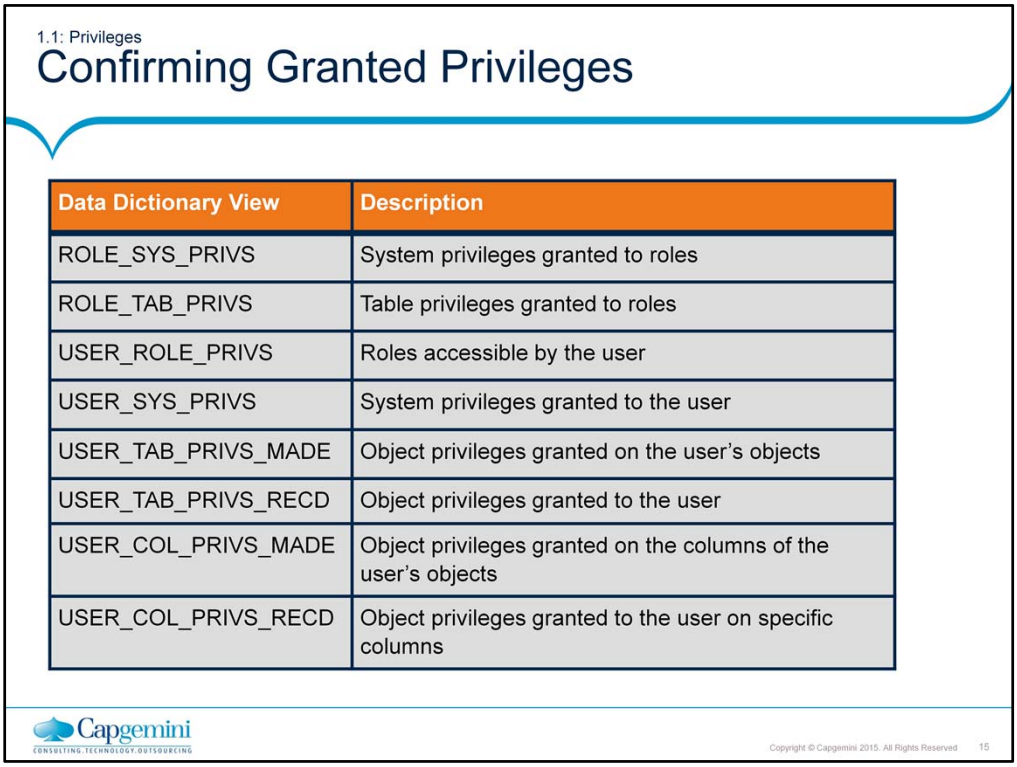
A privilege that is granted with the WITH GRANT OPTION clause can be passed on to other users and roles by the grantee. Object privileges granted with the WITH GRANT OPTION clause are revoked when the grantor's privilege is revoked.

The example in the slide gives the demo user access to your DEPARTMENTS table with the privileges to query the table and add rows to the table. The example also shows that user1 can give others these privileges.

PUBLIC Keyword

An owner of a table can grant access to all users by using the PUBLIC keyword.

The second example allows all users on the system to query data from Alice's DEPARTMENTS table.



Confirming Granted Privileges

If you attempt to perform an unauthorized operation, such as deleting a row from a table for which you do not have the DELETE privilege, the Oracle server does not permit the operation to take place.

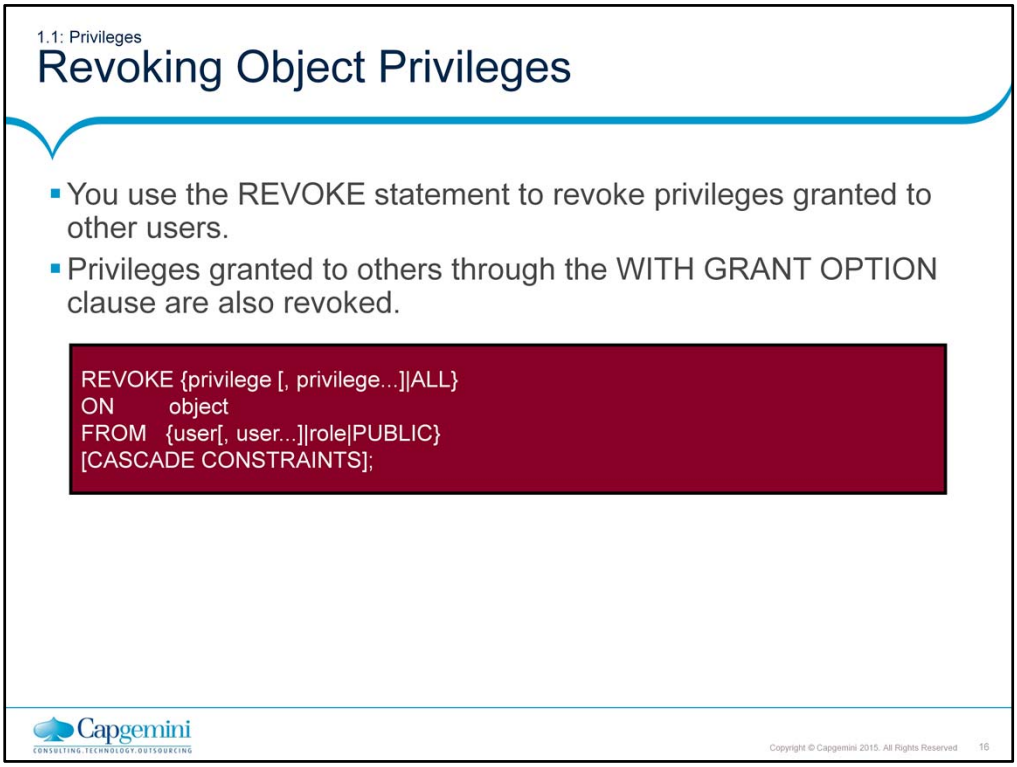
If you receive the Oracle server error message "Table or view does not exist," you have done either of the following:

- Named a table or view that does not exist
- Attempted to perform an operation on a table or view for which you do not have the appropriate privilege

The data dictionary is organized in tables and views and contains information about the database. You can access the data dictionary to view the privileges that you have. The table in the slide describes various data dictionary views.

You learn more about data dictionary views in the lesson titled "Managing Objects with Data Dictionary Views."

Note: The ALL_TAB_PRIVS_MADE dictionary view describes all the object grants made by the user or made on the objects owned by the user.



Revoking Object Privileges

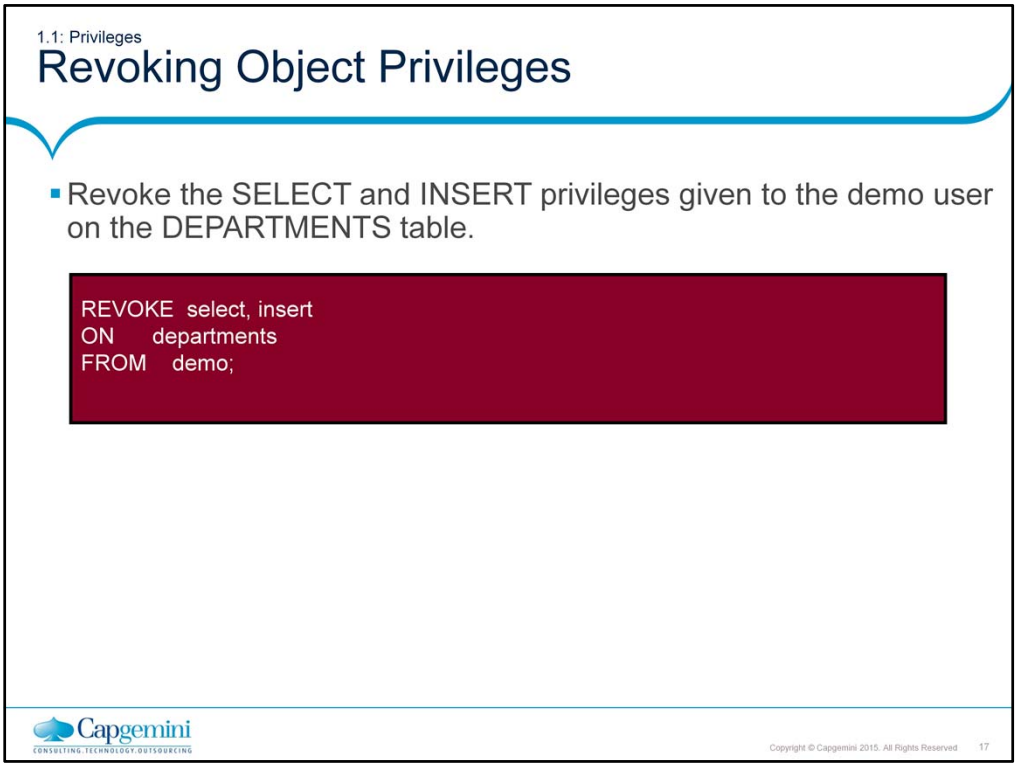
You can remove privileges granted to other users by using the REVOKE statement. When you use the REVOKE statement, the privileges that you specify are revoked from the users you name and from any other users to whom those privileges were granted by the revoked user.

In the syntax:

CASCADE Is required to remove any referential integrity constraints made to the CONSTRAINTS object by means of the REFERENCES privilege

For more information, see the Oracle Database11g SQL Reference.

Note: If a user were to leave the company and you revoke his or her privileges, you must regrant any privileges that this user may have granted to other users. If you drop the user account without revoking privileges from it, the system privileges granted by this user to other users are not affected by this action.

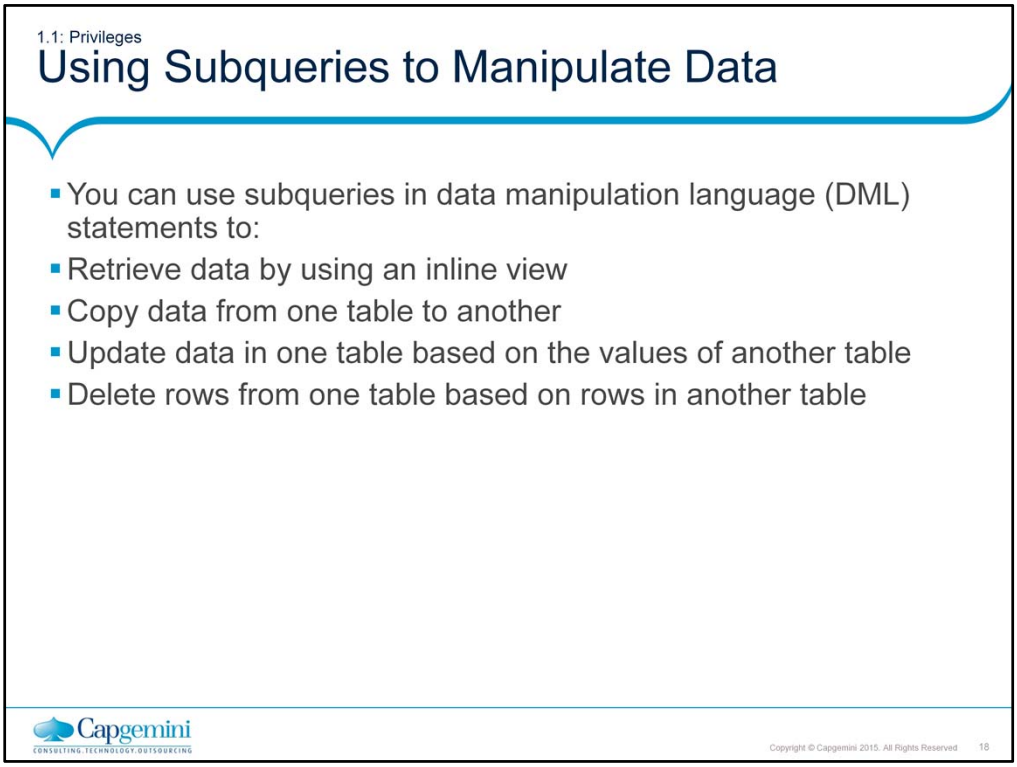


Revoking Object Privileges (continued)

The example in the slide revokes SELECT and INSERT privileges given to the demo user on the DEPARTMENTS table.

Note: If a user is granted a privilege with the WITH GRANT OPTION clause, that user can also grant the privilege with the WITH GRANT OPTION clause, so that a long chain of grantees is possible, but no circular grants (granting to a grant ancestor) are permitted. If the owner revokes a privilege from a user who granted the privilege to other users, the revoking cascades to all the privileges granted.

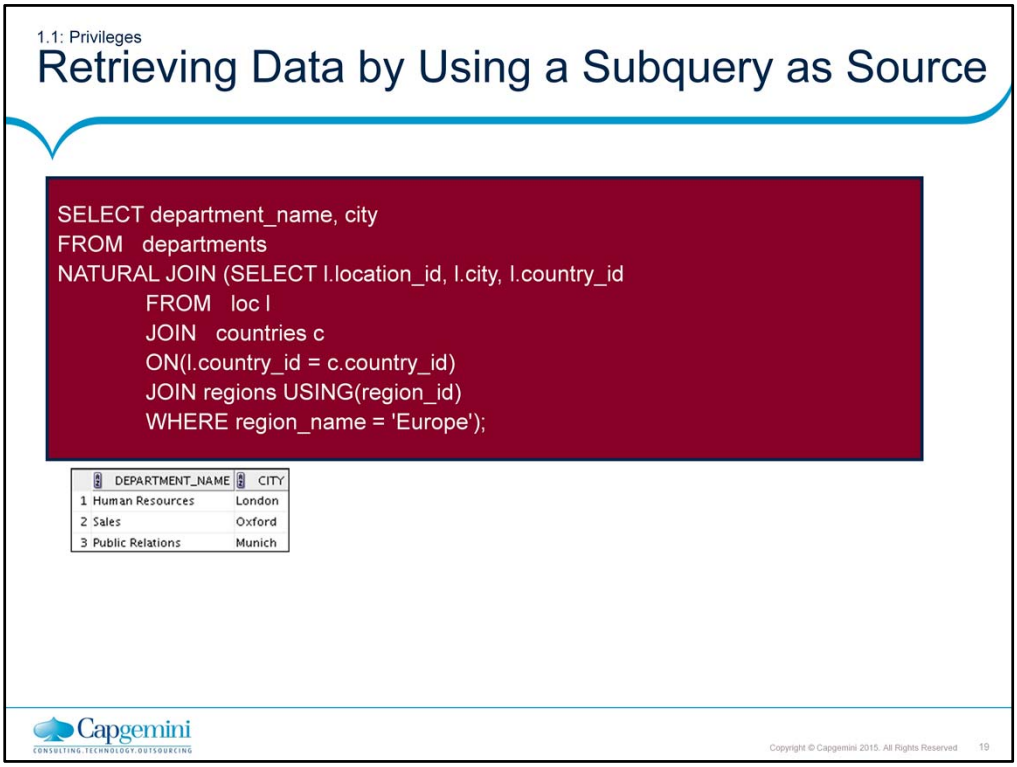
For example, if user A grants a SELECT privilege on a table to user B including the WITH GRANT OPTION clause, user B can grant to user C the SELECT privilege with the WITH GRANT OPTION clause as well, and user C can then grant to user D the SELECT privilege. If user A revokes privileges from user B, the privileges granted to users C and D are also revoked.



Using Subqueries to Manipulate Data

Subqueries can be used to retrieve data from a table that you can use as input to an INSERT into a different table. In this way, you can easily copy large volumes of data from one table to another with one single SELECT statement. Similarly, you can use subqueries to do mass updates and deletes by using them in the WHERE clause of the UPDATE and DELETE statements. You can also use subqueries in the FROM clause of a SELECT statement. This is called an inline view.

Note: You learned how to update and delete rows based on another table in the course titled Oracle Database 11g: SQL Fundamentals I.



Retrieving Data by Using a Subquery as Source

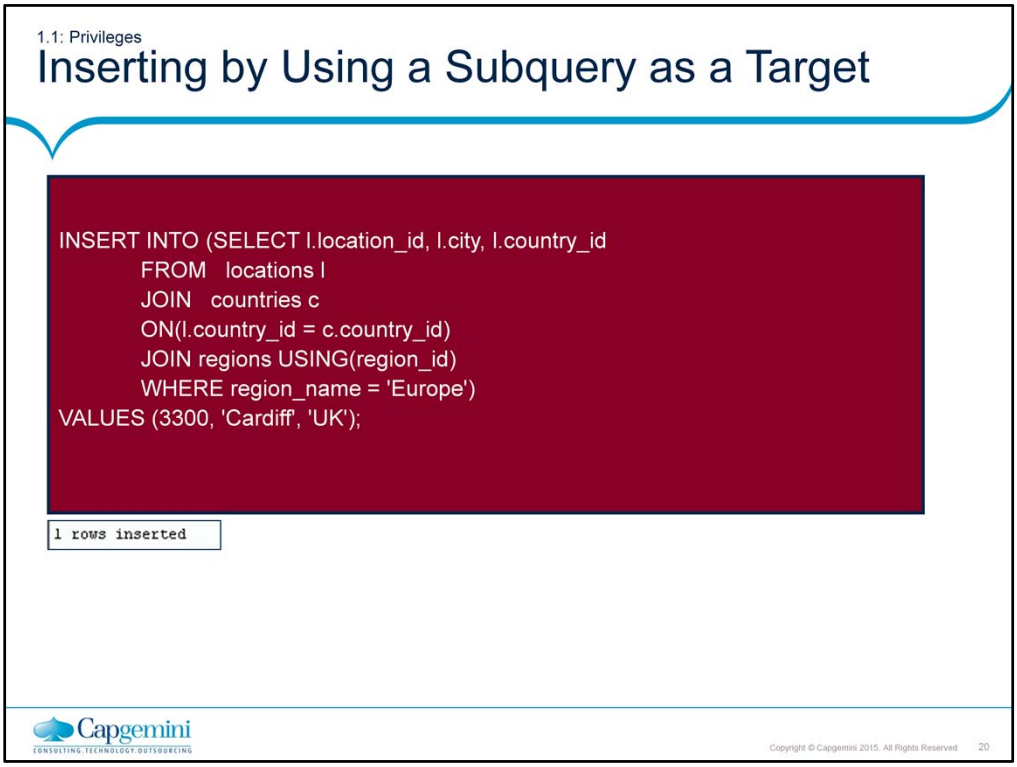
You can use a subquery in the FROM clause of a SELECT statement, which is very similar to how views are used. A subquery in the FROM clause of a SELECT statement is also called an inline view. A subquery in the FROM clause of a SELECT statement defines a data source for that particular SELECT statement, and only that SELECT statement. As with a database view, the SELECT statement in the subquery can be as simple or as complex as you like.

When a database view is created, the associated SELECT statement is stored in the data dictionary. In situations where you do not have the necessary privileges to create database views, or when you would like to test the suitability of a SELECT statement to become a view, you can use an inline view.

With inline views, you can have all the code needed to support the query in one place. This means that you can avoid the complexity of creating a separate database view. The example in the slide shows how to use an inline view to display the department name and the city in Europe. The subquery in the FROM clause fetches the location ID, city name, and the country by joining three different tables. The output of the inner query is considered as a table for the outer query. The inner query is similar to that of a database view but does not have any physical name.

For the example in the slide, the loc table is created by running the following statement:

```
CREATE TABLE loc AS SELECT * FROM
locations;
```



Inserting by Using a Subquery as a Target

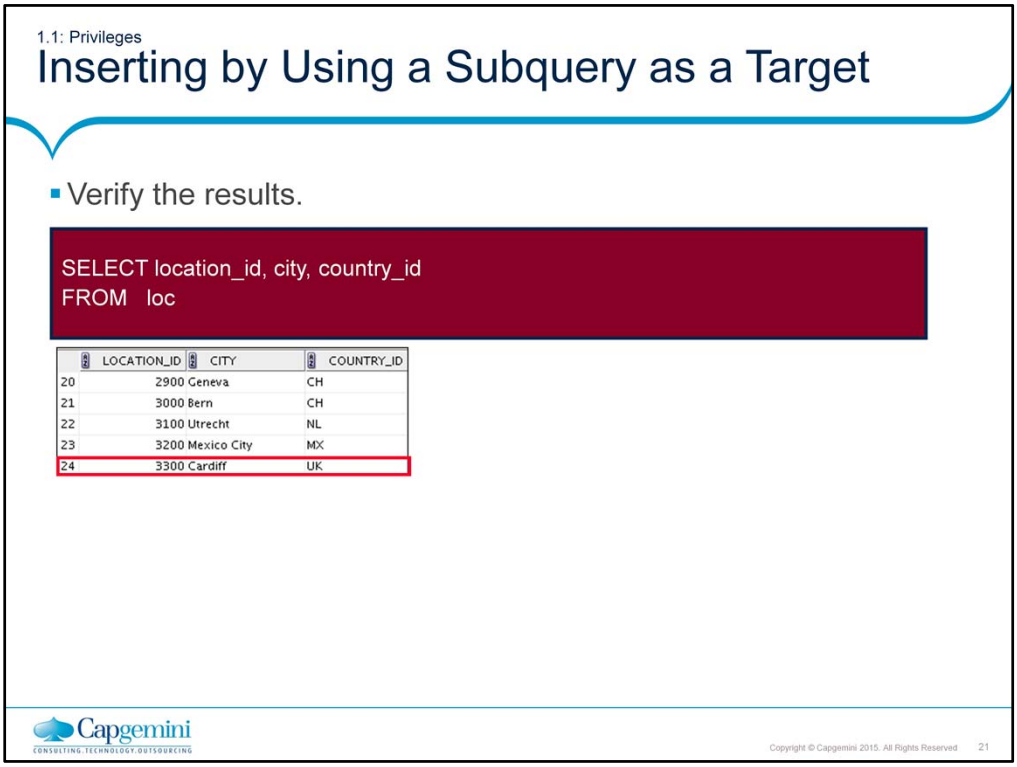
You can use a subquery in place of the table name in the INTO clause of the INSERT statement. The SELECT list of this subquery must have the same number of columns as the column list of the VALUES clause. Any rules on the columns of the base table must be followed in order for the INSERT statement to work successfully. For example, you cannot put in a duplicate location ID or leave out a value for a mandatory NOT NULL column.

This use of subqueries helps you avoid having to create a view just for performing an INSERT.

The example in the slide uses a subquery in the place of LOC to create a record for a new European city.

Note: You can also perform the INSERT operation on the EUROPEAN_CITIES view by using the following code:

```
INSERT INTO european_cities
VALUES (3300,'Cardiff','UK');
```



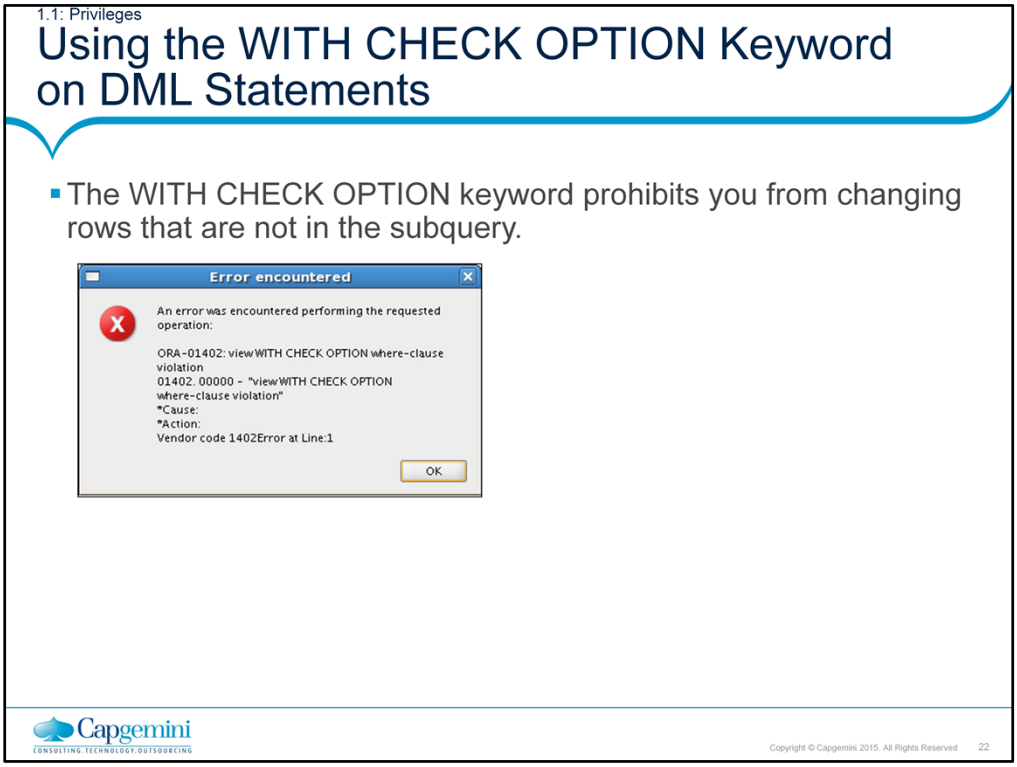
Inserting by Using a Subquery as a Target (continued)

The example in the slide shows that the insert via the inline view created a new record in the base table LOC.

The following example shows the results of the subquery that was used to identify the table for the INSERT statement.

```
SELECT l.location_id, l.city, l.country_id
FROM   loc l
JOIN   countries c
ON(l.country_id = c.country_id)
JOIN regions USING(region_id)
WHERE region_name = 'Europe'
```

	LOCATION_ID	CITY	COUNTRY_ID
6	2700	Munich	DE
7	2900	Geneva	CH
8	3000	Bern	CH
9	3100	Utrecht	NL
10	3300	Cardiff	UK



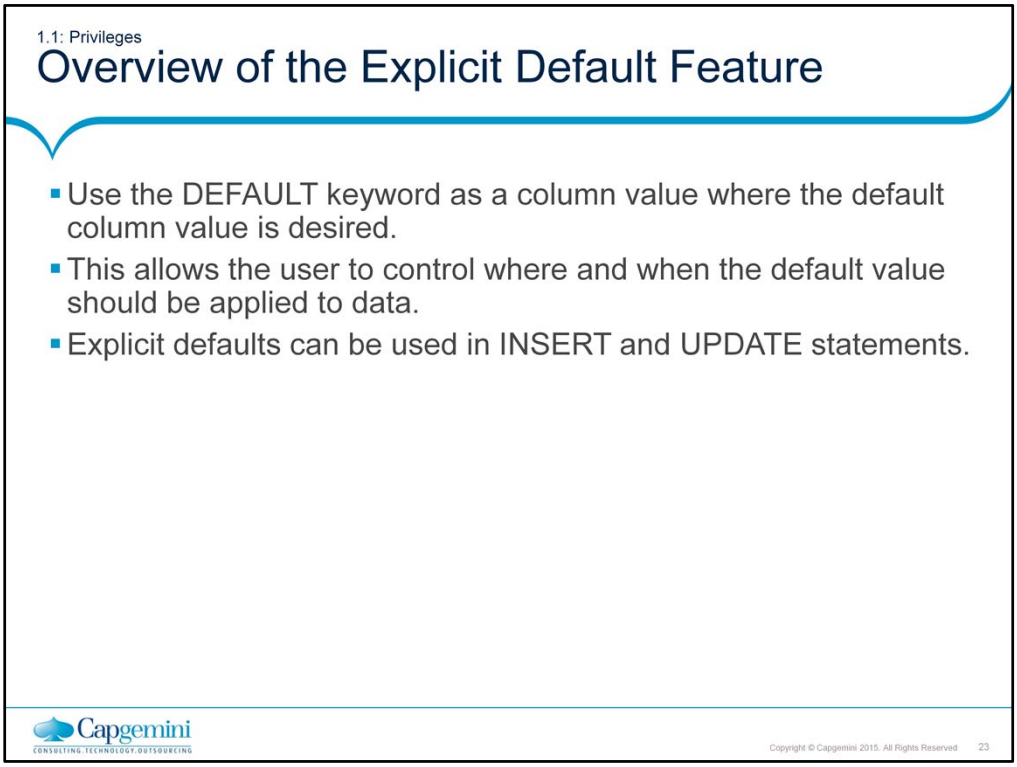
Using the WITH CHECK OPTION Keyword on DML Statements

Specify the WITH CHECK OPTION keyword to indicate that if the subquery is used in place of a table in an INSERT, UPDATE, or DELETE statement, no changes that produce rows that are not included in the subquery are permitted to that table.

The example in the slide shows how to use an inline view with WITH CHECK OPTION. The INSERT statement prevents the creation of records in the LOC table for a city that is not in Europe.

The following example executes successfully because of the changes in the VALUES list.

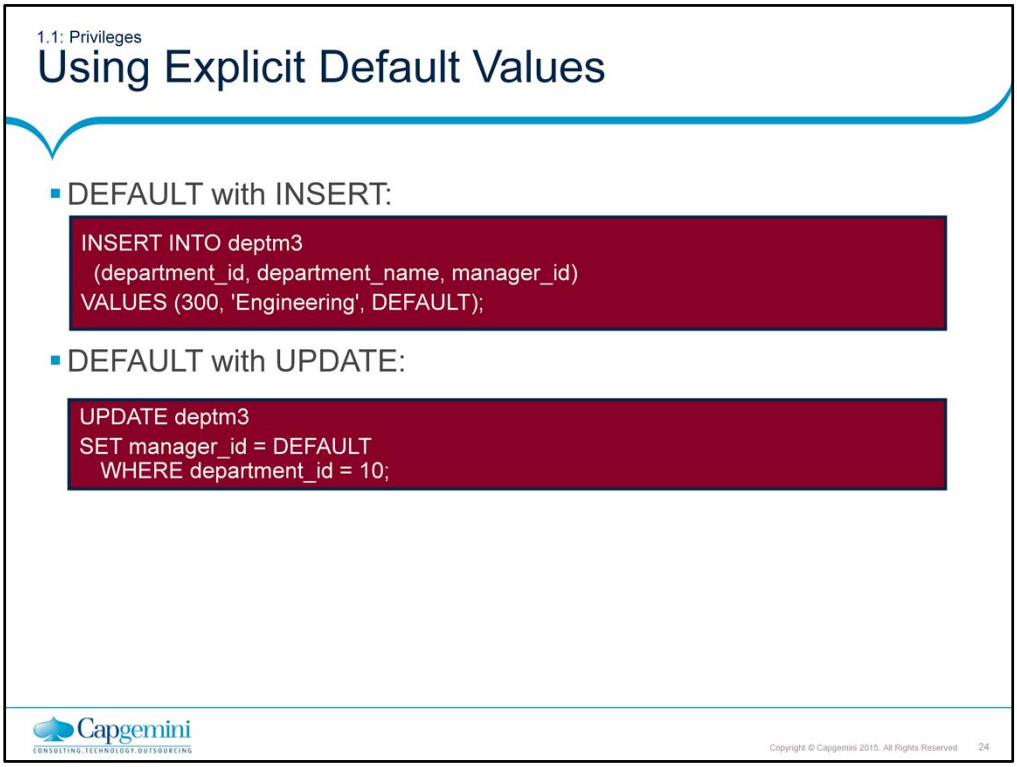
```
INSERT INTO (SELECT location_id, city,
country_id
        FROM   loc
        WHERE  country_id IN
        (SELECT country_id
         FROM countries
         NATURAL JOIN regions
         WHERE region_name = 'Europe')
         WITH CHECK OPTION)
VALUES (3500, 'Berlin', 'DE');
```



Explicit Defaults

The DEFAULT keyword can be used in INSERT and UPDATE statements to identify a default column value. If no default value exists, a null value is used.

The DEFAULT option saves you from having to hard code the default value in your programs or querying the dictionary to find it, as was done before this feature was introduced. Hard coding the default is a problem if the default changes, because the code consequently needs changing. Accessing the dictionary is not usually done in an application; therefore, this is a very important feature.



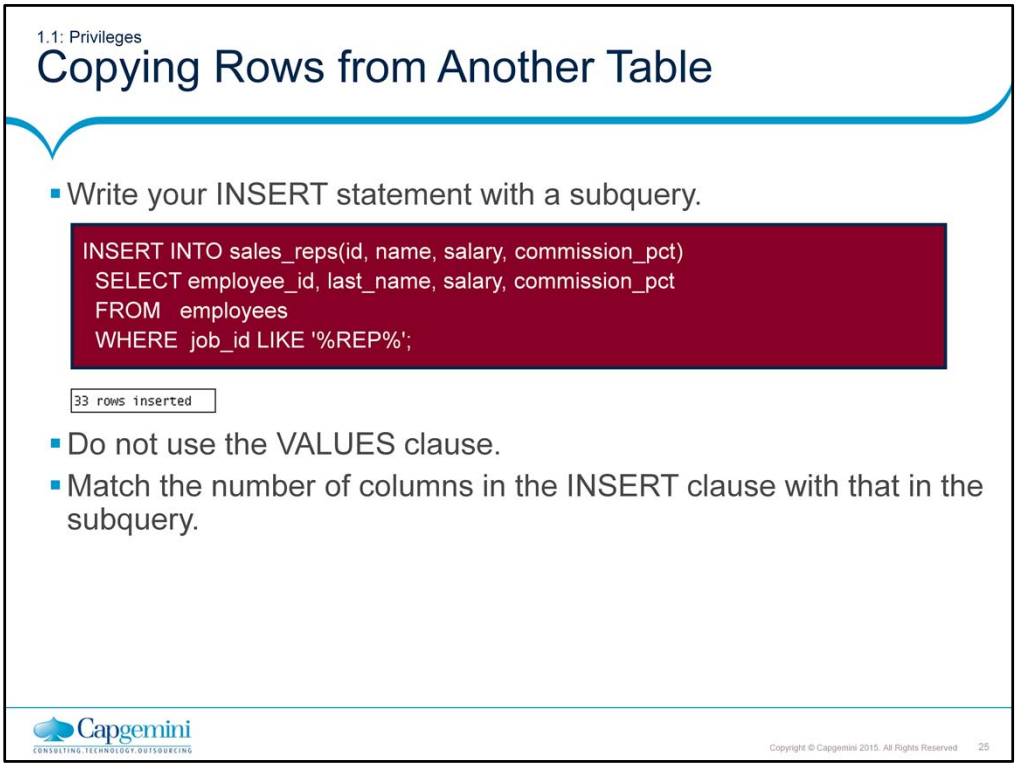
Using Explicit Default Values

Specify DEFAULT to set the column to the value previously specified as the default value for the column. If no default value for the corresponding column has been specified, the Oracle server sets the column to null.

In the first example in the slide, the INSERT statement uses a default value for the MANAGER_ID column. If there is no default value defined for the column, a null value is inserted instead.

The second example uses the UPDATE statement to set the MANAGER_ID column to a default value for department 10. If no default value is defined for the column, it changes the value to null.

Note: When creating a table, you can specify a default value for a column. This is discussed in SQL Fundamentals I.



Copying Rows from Another Table

You can use the INSERT statement to add rows to a table where the values are derived from existing tables. In place of the VALUES clause, you use a subquery.

Syntax

```
INSERT INTO table [ column (, column) ]
subquery;
```

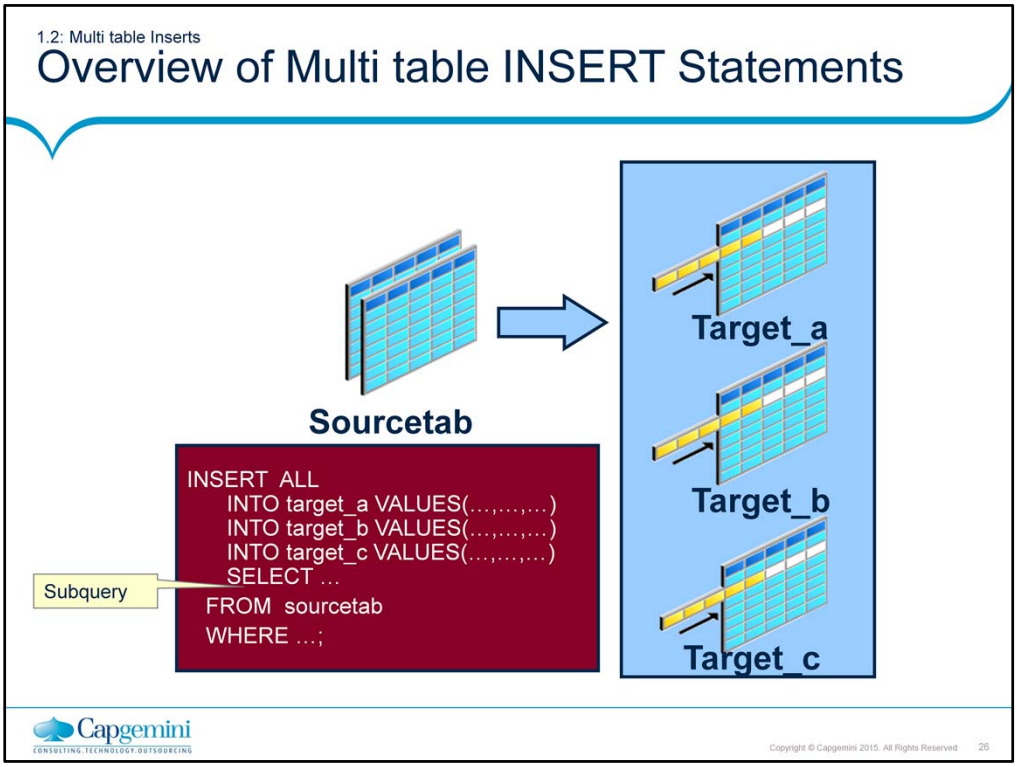
In the syntax:

table	Is the table name
column	Is the name of the column in the table to populate
subquery	Is the subquery that returns rows into the table

The number of columns and their data types in the column list of the INSERT clause must match the number of values and their data types in the subquery. To create a copy of the rows of a table, use SELECT * in the subquery.

```
INSERT INTO EMPL3
  SELECT *
  FROM   employees;
```

Note: You use the LOG ERRORS clause in your DML statement to enable the DML operation to complete regardless of errors. Oracle writes the details of the error message to an error-logging table that you have created. For more information, see Oracle Database 11g SQL Reference.



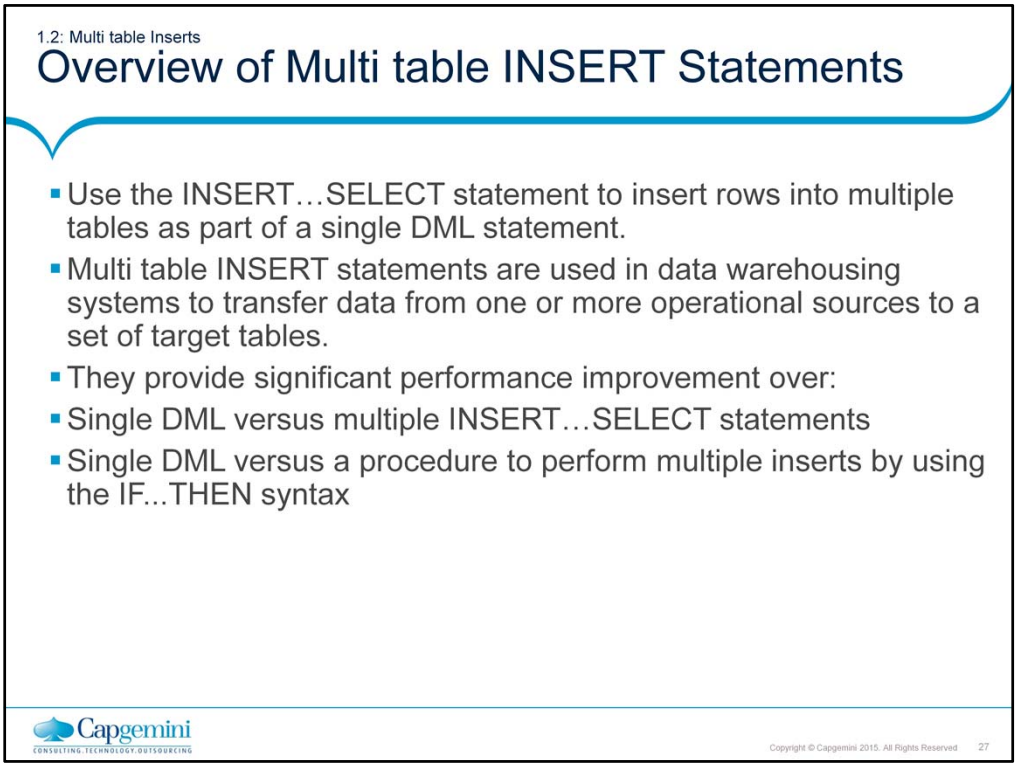
Overview of Multitable INSERT Statements

In a multitable INSERT statement, you insert computed rows derived from the rows returned from the evaluation of a subquery into one or more tables.

Multitable INSERT statements are useful in a data warehouse scenario. You need to load your data warehouse regularly so that it can serve its purpose of facilitating business analysis. To do this, data from one or more operational systems must be extracted and copied into the warehouse. The process of extracting data from the source system and bringing it into the data warehouse is commonly called ETL, which stands for extraction, transformation, and loading.

During extraction, the desired data must be identified and extracted from many different sources, such as database systems and applications. After extraction, the data must be physically transported to the target system or an intermediate system for further processing. Depending on the chosen means of transportation, some transformations can be done during this process. For example, a SQL statement that directly accesses a remote target through a gateway can concatenate two columns as part of the SELECT statement.

After data is loaded into the Oracle database, data transformations can be executed using SQL operations. A multitable INSERT statement is one of the techniques for implementing SQL data transformations.

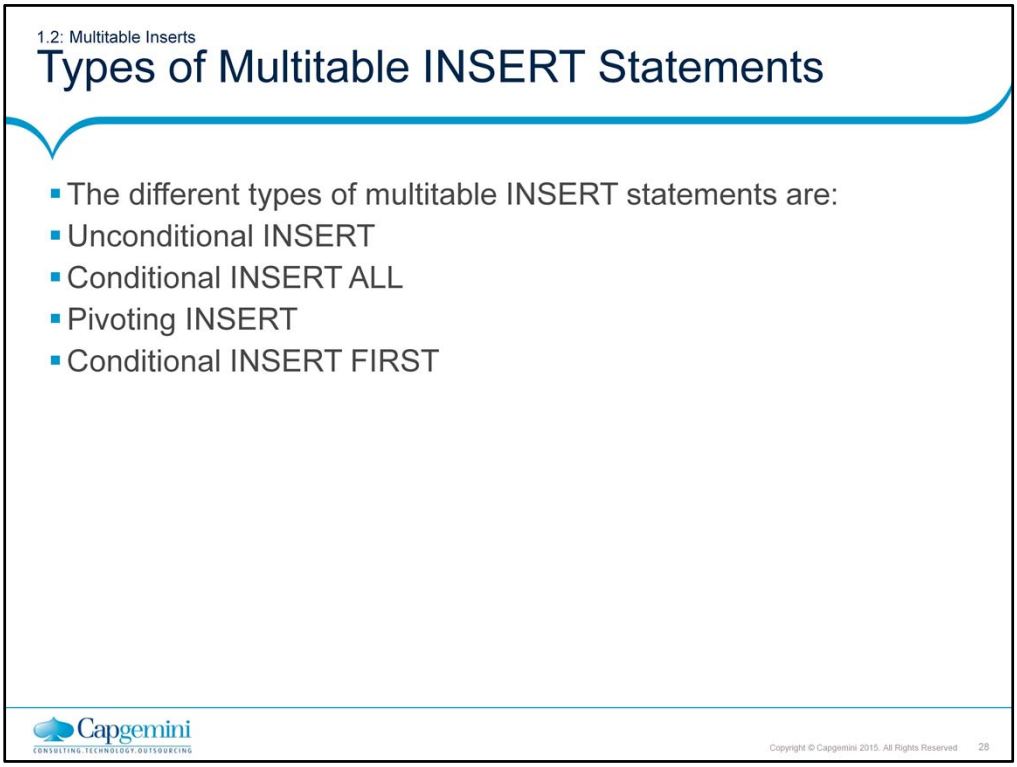


Overview of Multitable INSERT Statements (continued)

Multitable INSERT statements offer the benefits of the INSERT ... SELECT statement when multiple tables are involved as targets. Without multitable INSERT, you had to deal with n independent INSERT ... SELECT statements, thus processing the same source data n times and increasing the transformation workload n times.

As with the existing INSERT ... SELECT statement, the new statement can be parallelized and used with the direct-load mechanism for faster performance.

Each record from any input stream, such as a nonrelational database table, can now be converted into multiple records for a more relational database table environment. To alternatively implement this functionality, you were required to write multiple INSERT statements.



Types of Multitable INSERT Statements

You use different clauses to indicate the type of INSERT to be executed.

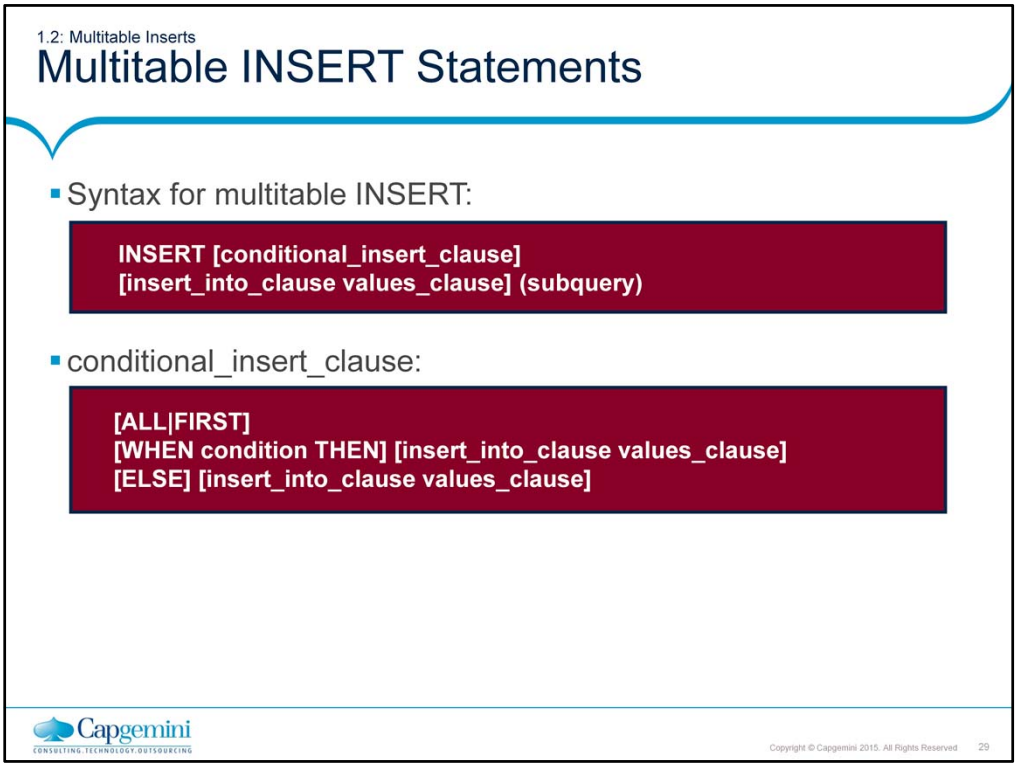
The types of multitable INSERT statements are:

Unconditional INSERT: For each row returned by the subquery, a row is inserted into each of the target tables.

Conditional INSERT ALL: For each row returned by the subquery, a row is inserted into each target table if the specified condition is met.

Pivoting INSERT: This is a special case of the unconditional INSERT ALL.

Conditional INSERT FIRST: For each row returned by the subquery, a row is inserted into the very first target table in which the condition is met.



Multitable INSERT Statements

The slide displays the generic format for multitable INSERT statements.

Unconditional INSERT: ALL into_clause

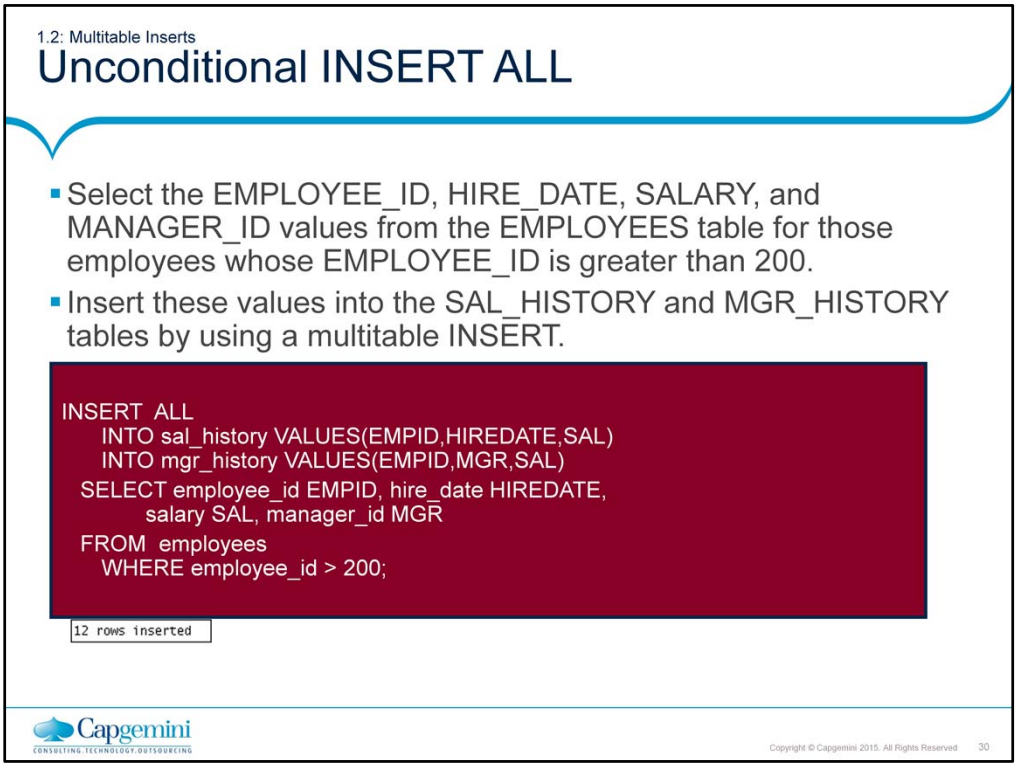
Specify ALL followed by multiple insert_into_clauses to perform an unconditional multitable INSERT. The Oracle server executes each insert_into_clause once for each row returned by the subquery.

Conditional INSERT: conditional_insert_clause

Specify the conditional_insert_clause to perform a conditional multitable INSERT. The Oracle server filters each insert_into_clause through the corresponding WHEN condition, which determines whether that insert_into_clause is executed. A single multitable INSERT statement can contain up to 127 WHEN clauses.

Conditional INSERT: ALL

If you specify ALL, the Oracle server evaluates each WHEN clause regardless of the results of the evaluation of any other WHEN clause. For each WHEN clause whose condition evaluates to true, the Oracle server executes the corresponding INTO clause list.

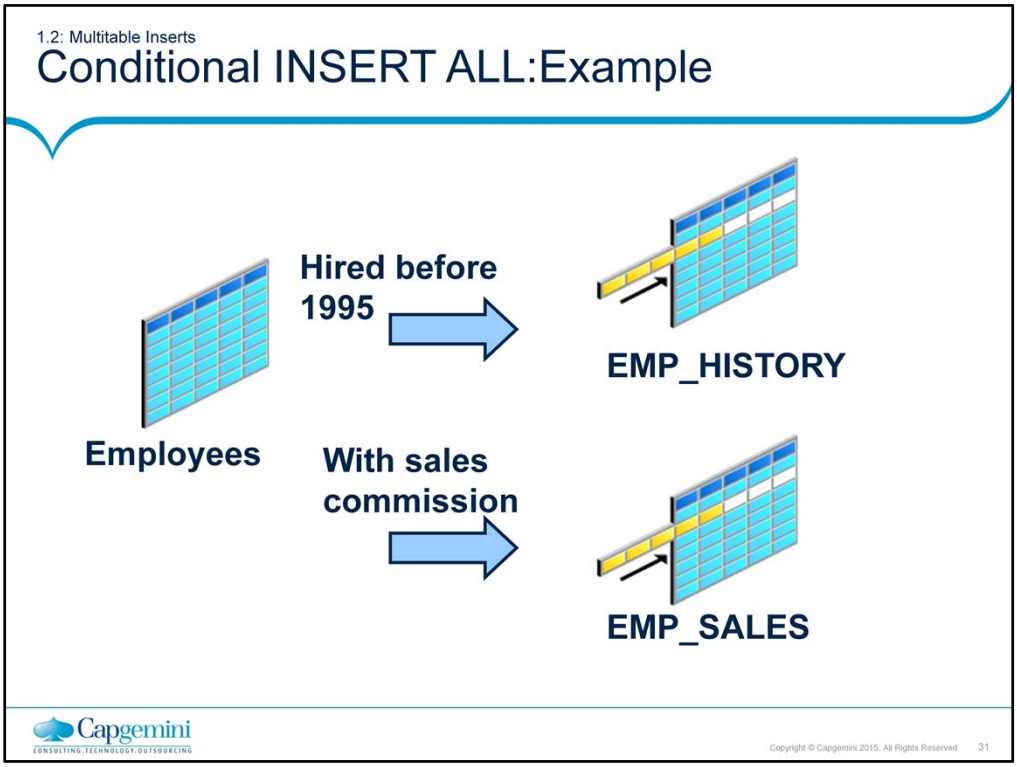


Unconditional INSERT ALL

The example in the slide inserts rows into both the SAL_HISTORY and the MGR_HISTORY tables.

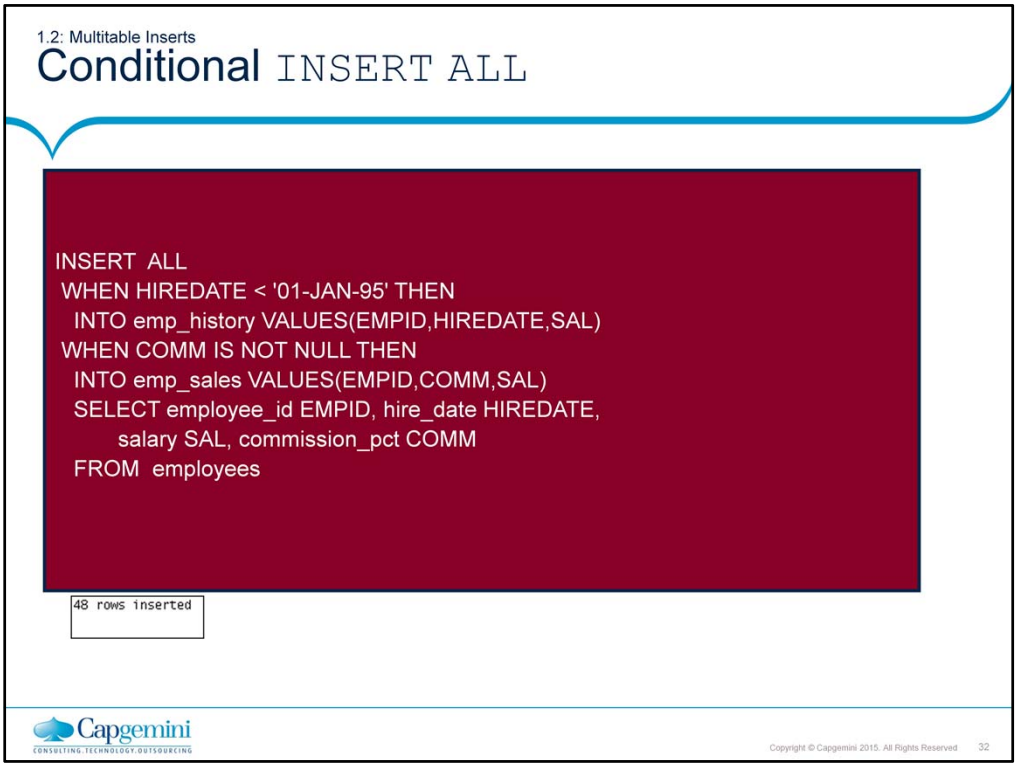
The SELECT statement retrieves the details of employee ID, hire date, salary, and manager ID of those employees whose employee ID is greater than 200 from the EMPLOYEES table. The details of the employee ID, hire date, and salary are inserted into the SAL_HISTORY table. The details of employee ID, manager ID, and salary are inserted into the MGR_HISTORY table.

This INSERT statement is referred to as an unconditional INSERT because no further restriction is applied to the rows that are retrieved by the SELECT statement. All the rows retrieved by the SELECT statement are inserted into the two tables: SAL_HISTORY and MGR_HISTORY. The VALUES clause in the INSERT statements specifies the columns from the SELECT statement that must be inserted into each of the tables. Each row returned by the SELECT statement results in two insertions: one for the SAL_HISTORY table and one for the MGR_HISTORY table.



Conditional INSERT ALL: Example

For all employees in the employees tables, if the employee was hired before 1995, insert that employee record into the employee history. If the employee earns a sales commission, insert the record information into the EMP_SALES table. The SQL statement is shown on the next page.



Conditional INSERT ALL

The example in the slide is similar to the example in the previous slide because it inserts rows into both the EMP_HISTORY and the EMP_SALES tables. The SELECT statement retrieves details such as employee ID, hire date, salary, and commission percentage for all employees from the EMPLOYEES table. Details such as employee ID, hire date, and salary are inserted into the EMP_HISTORY table. Details such as employee ID, commission percentage, and salary are inserted into the EMP_SALES table.

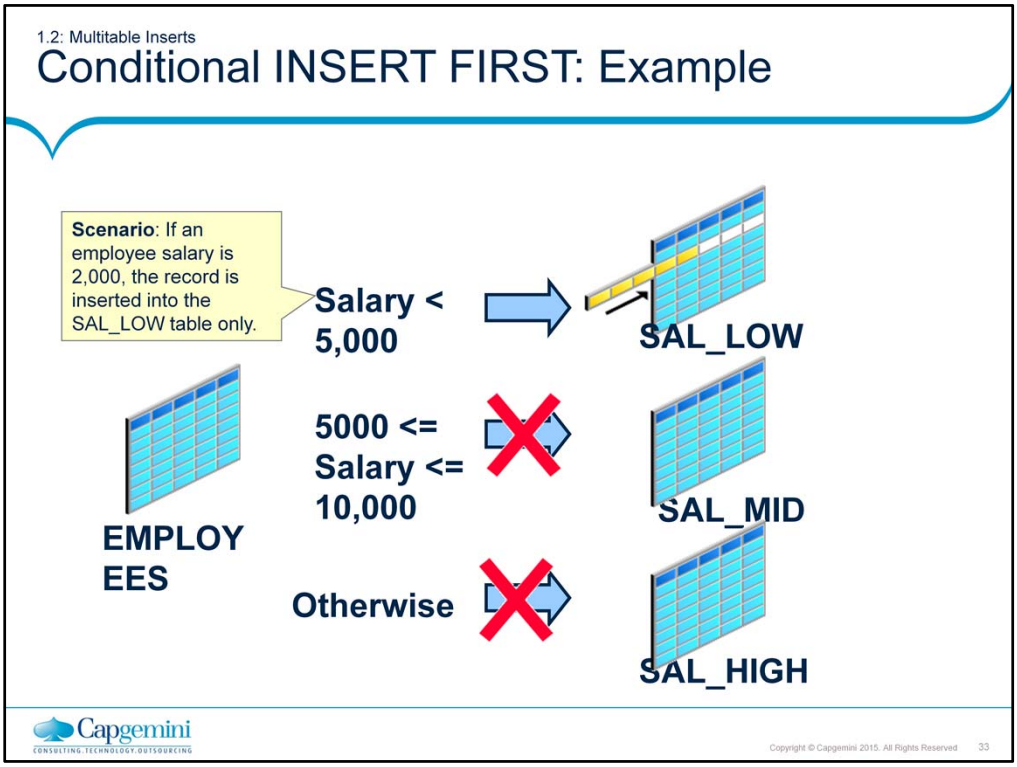
This INSERT statement is referred to as a conditional INSERT ALL because a further restriction is applied to the rows that are retrieved by the SELECT statement. From the rows that are retrieved by the SELECT statement, only those rows in which the hire date was prior to 1995 are inserted in the EMP_HISTORY table. Similarly, only those rows where the value of commission percentage is not null are inserted in the EMP_SALES table.

```
SELECT count(*) FROM emp_history;

SELECT count(*) FROM emp_sales;
```

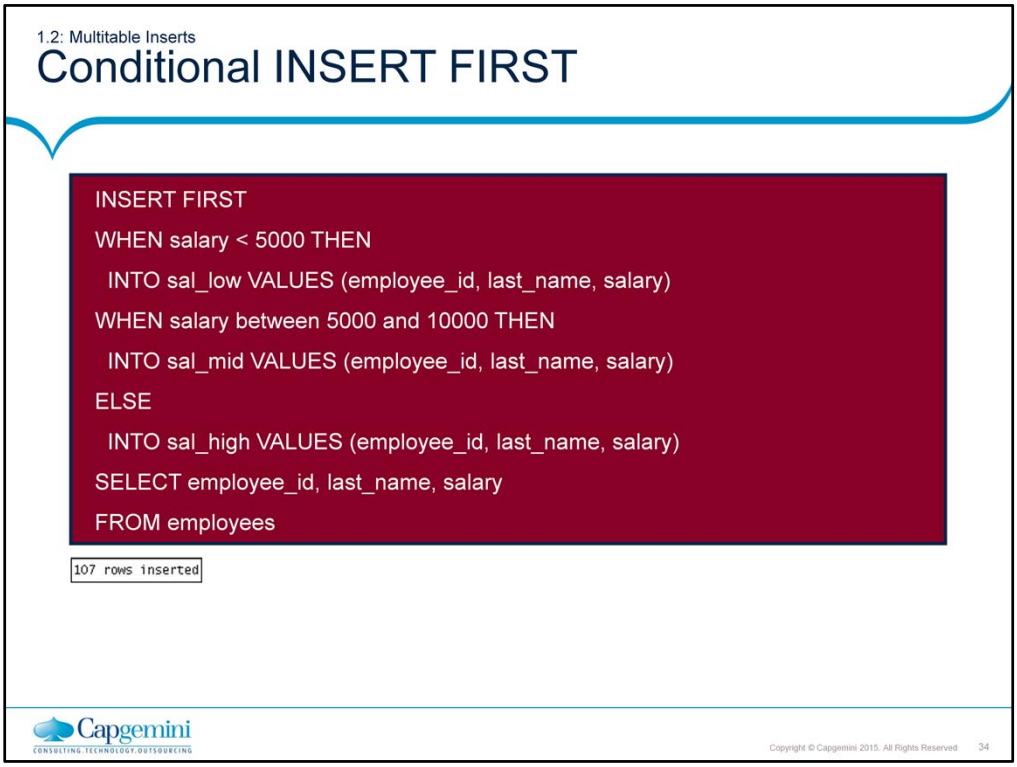
	COUNT(*)
1	13

	COUNT(*)
1	35



Conditional INSERT FIRST: Example

For all employees in the EMPLOYEES table, insert the employee information into the first target table that meets the condition. In the example, if an employee has a salary of 2,000, the record is inserted into the SAL_LOW table only. The SQL statement is shown on the next page.



Conditional INSERT FIRST

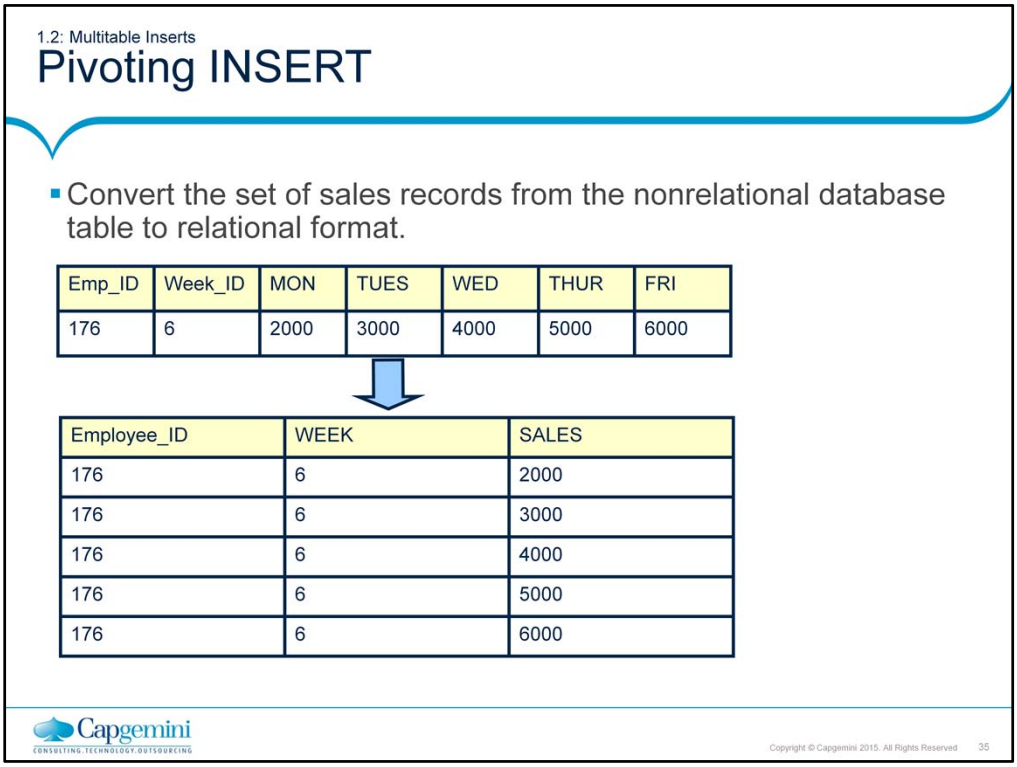
The SELECT statement retrieves details such as employee ID, last name, and salary for every employee in the EMPLOYEES table. For each employee record, it is inserted into the very first target table that meets the condition.

This INSERT statement is referred to as a conditional INSERT FIRST. The WHEN salary

< 5000 condition is evaluated first. If this first WHEN clause evaluates to true, the Oracle server executes the corresponding INTO clause and inserts the record into the SAL_LOW table. It skips subsequent WHEN clauses for this row.

If the row does not satisfy the first WHEN condition (WHEN salary < 5000), the next condition (WHEN salary between 5000 and 10000) is evaluated. If this condition evaluates to true, the record is inserted into the SAL_MID table, and the last condition is skipped.

If neither the first condition (WHEN salary < 5000) nor the second condition (WHEN salary between 5000 and 10000) is evaluated to true, the Oracle server executes the corresponding INTO clause for the ELSE clause.



Pivoting INSERT

Pivoting is an operation in which you must build a transformation such that each record from any input stream, such as a nonrelational database table, must be converted into multiple records for a more relational database table environment.

Suppose you receive a set of sales records from a nonrelational database table:

SALES_SOURCE_DATA, in the following format:

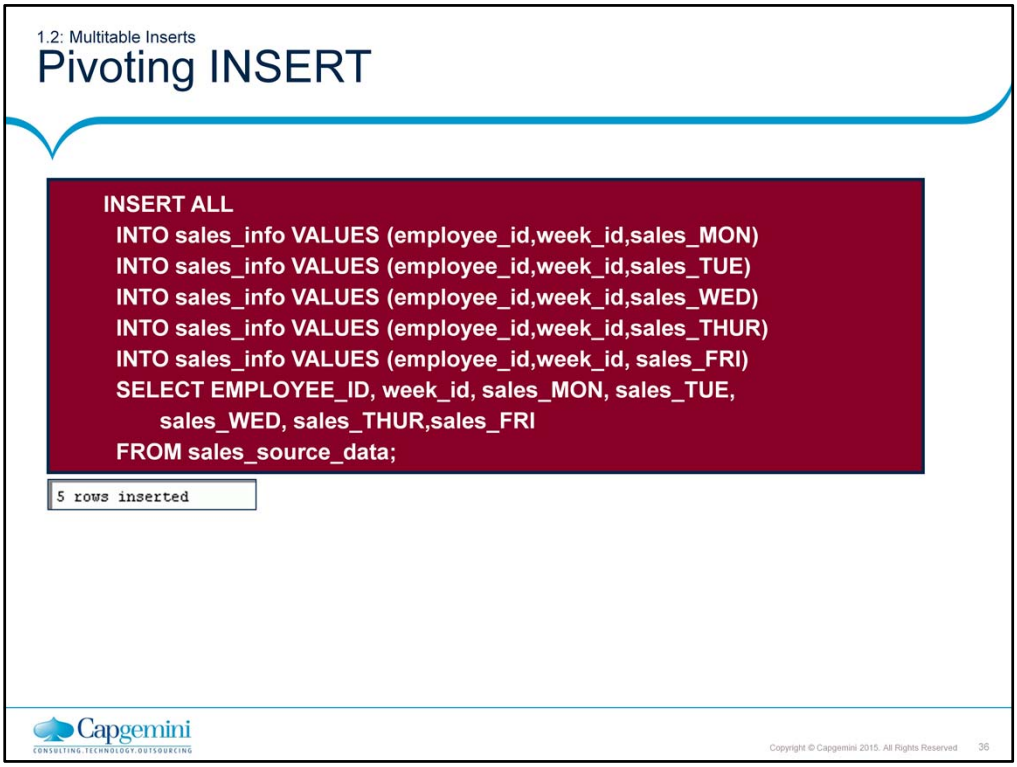
EMPLOYEE_ID, WEEK_ID, SALES_MON, SALES_TUE, SALES_WED, SALES_THUR, SALES_FRI

You want to store these records in the SALES_INFO table in a more typical relational format:

EMPLOYEE_ID, WEEK, SALES

To solve this problem, you must build a transformation such that each record from the original nonrelational database table, SALES_SOURCE_DATA, is converted into five records for the data warehouse's SALES_INFO table. This operation is commonly referred to as pivoting.

The solution to this problem is shown on the next page.

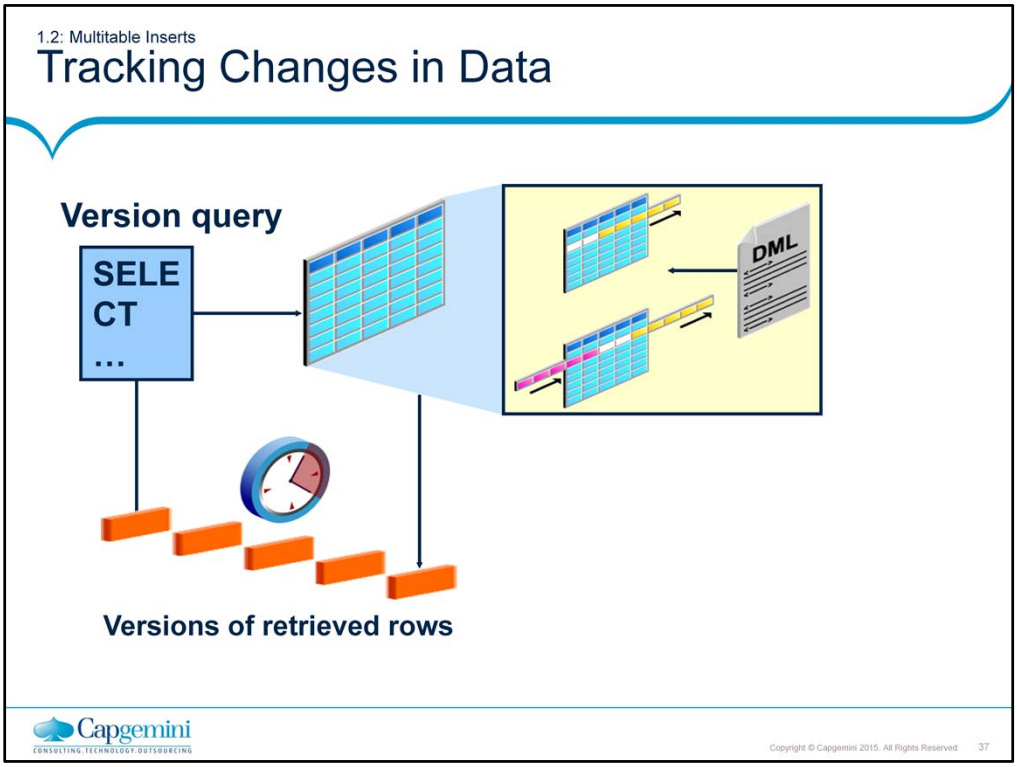


Pivoting INSERT (continued)

In the example in the slide, the sales data is received from the nonrelational database table SALES_SOURCE_DATA, which is the details of the sales performed by a sales representative on each day of a week, for a week with a particular week ID.

DESC SALES_SOURCE_DATA

```
Name                          Null     Type
----------------------------- -------- -------------
EMPLOYEE_ID                            NUMBER(6)
WEEK_ID                                NUMBER(2)
SALES_MON                              NUMBER(8,2)
SALES_TUE                              NUMBER(8,2)
SALES_WED                              NUMBER(8,2)
SALES_THUR                             NUMBER(8,2)
SALES_FRI                              NUMBER(8,2)
```



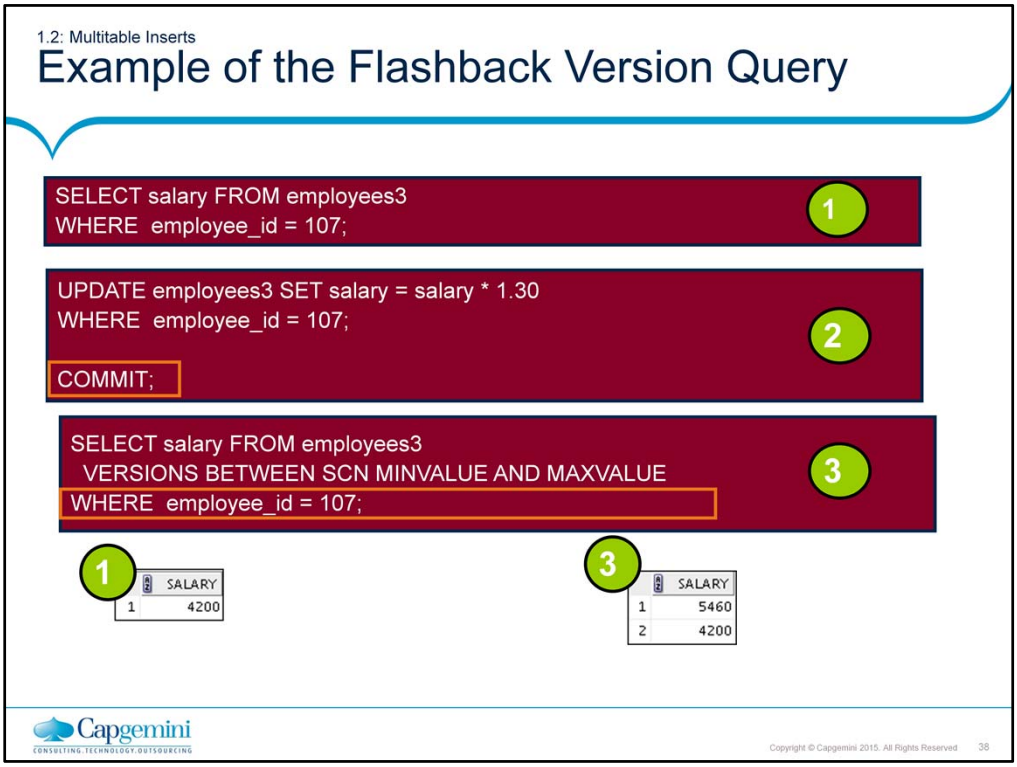
Tracking Changes in Data

You may discover that somehow data in a table has been inappropriately changed. To research this, you can use multiple flashback queries to view row data at specific points in time. More efficiently, you can use the Flashback Version Query feature to view all changes to a row over a period of time. This feature enables you to append a VERSIONS clause to a SELECT statement that specifies a system change number (SCN) or the time stamp range within which you want to view changes to row values. The query also can return associated metadata, such as the transaction responsible for the change.

Further, after you identify an erroneous transaction, you can use the Flashback Transaction Query feature to identify other changes that were done by the transaction. You then have the option of using the Flashback Table feature to restore the table to a state before the changes were made.

You can use a query on a table with a VERSIONS clause to produce all the versions of all the rows that exist or ever existed between the time the query was issued and the undo_retention seconds before the current time. undo_retention is an initialization parameter, which is an autotuned parameter. A query that includes a VERSIONS clause is referred to as a version query. The results of a version query behaves as though the WHERE clause were applied to the versions of the rows. The version query returns versions of the rows only across transactions.

System change number (SCN): The Oracle server assigns an SCN to identify the redo records for each committed transaction.



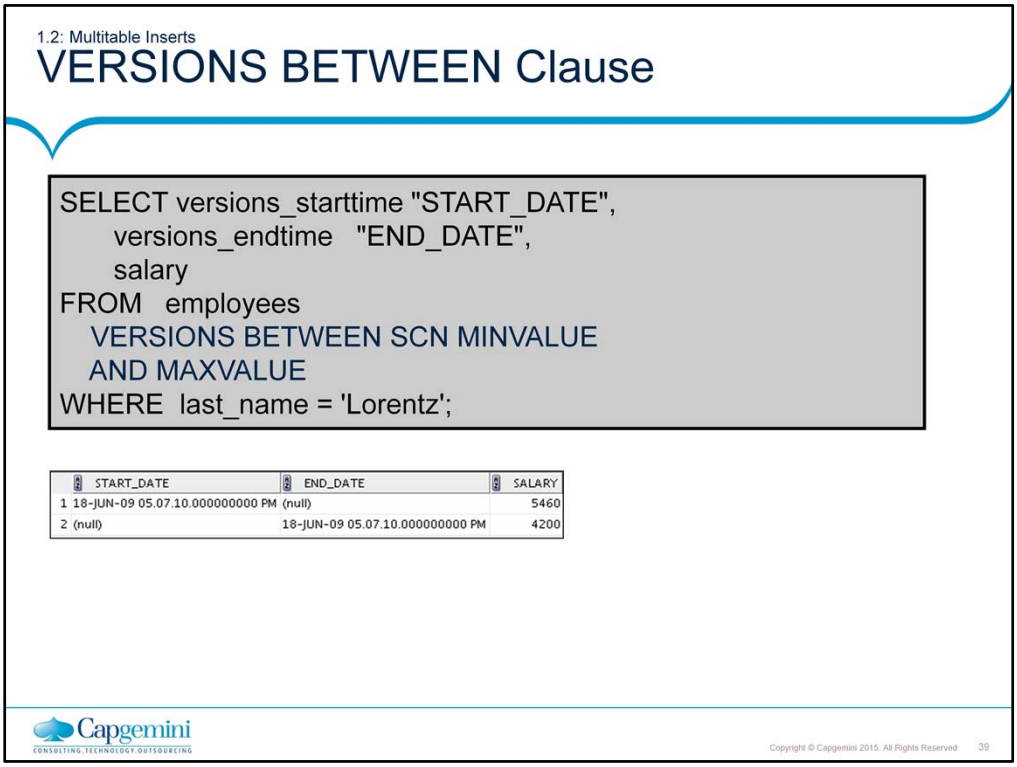
Example of the Flashback Version Query

In the example in the slide, the salary for employee 107 is retrieved (1). The salary for employee 107 is increased by 30 percent and this change is committed (2). The different versions of salary are displayed (3).

The VERSIONS clause does not change the plan of the query. For example, if you run a query on a table that uses the index access method, the same query on the same table with a VERSIONS clause continues to use the index access method. The versions of the rows returned by the version query are versions of the rows across transactions. The VERSIONS clause has no effect on the transactional behavior of a query. This means that a query on a table with a VERSIONS clause still inherits the query environment of the ongoing transaction.

The default VERSIONS clause can be specified as VERSIONS BETWEEN {SCN|TIMESTAMP} MINVALUE AND MAXVALUE.

The VERSIONS clause is a SQL extension only for queries. You can have DML and DDL operations that use a VERSIONS clause within subqueries. The row version query retrieves all the committed versions of the selected rows. Changes made by the current active transaction are not returned. The version query retrieves all incarnations of the rows. This essentially means that versions returned include deleted and subsequent reinserted versions of the rows.

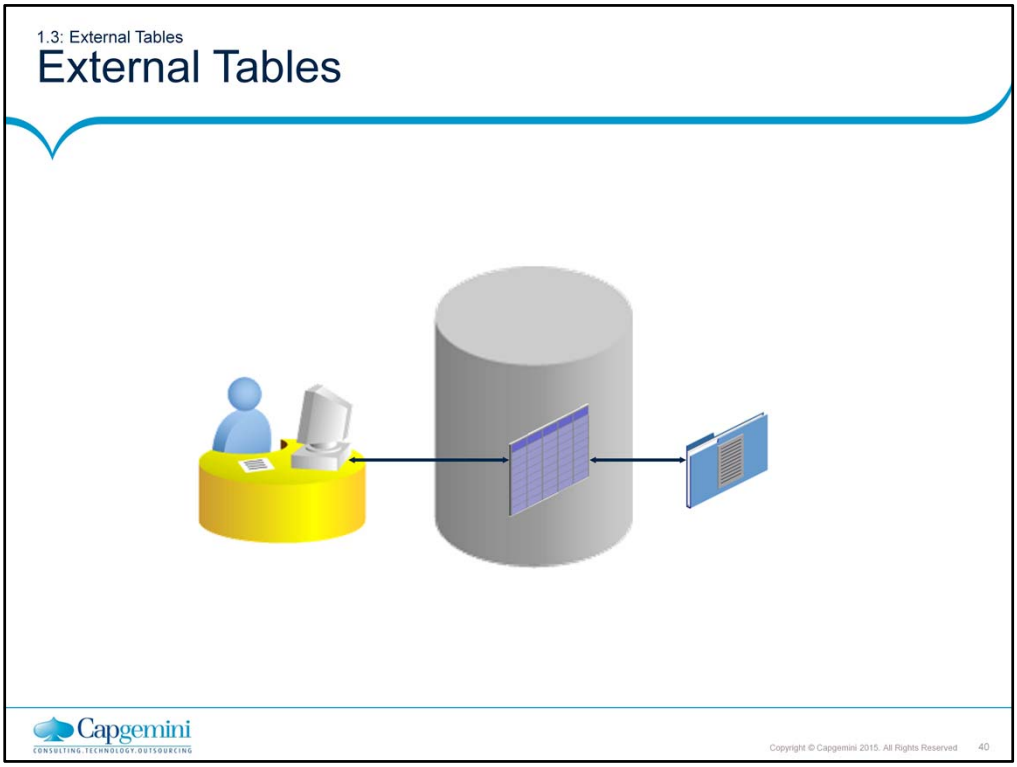


VERSIONS BETWEEN Clause

You can use the VERSIONS BETWEEN clause to retrieve all the versions of the rows that exist or have ever existed between the time the query was issued and a point back in time.

If the undo retention time is less than the lower bound time or the SCN of the BETWEEN clause, the query retrieves versions up to the undo retention time only. The time interval of the BETWEEN clause can be specified as an SCN interval or a wall-clock interval. This time interval is closed at both the lower and the upper bounds.

In the example, Lorentz's salary changes are retrieved. The NULL value for END_DATE for the first version indicates that this was the existing version at the time of the query. The NULL value for START_DATE for the last version indicates that this version was created at a time before the undo retention time.

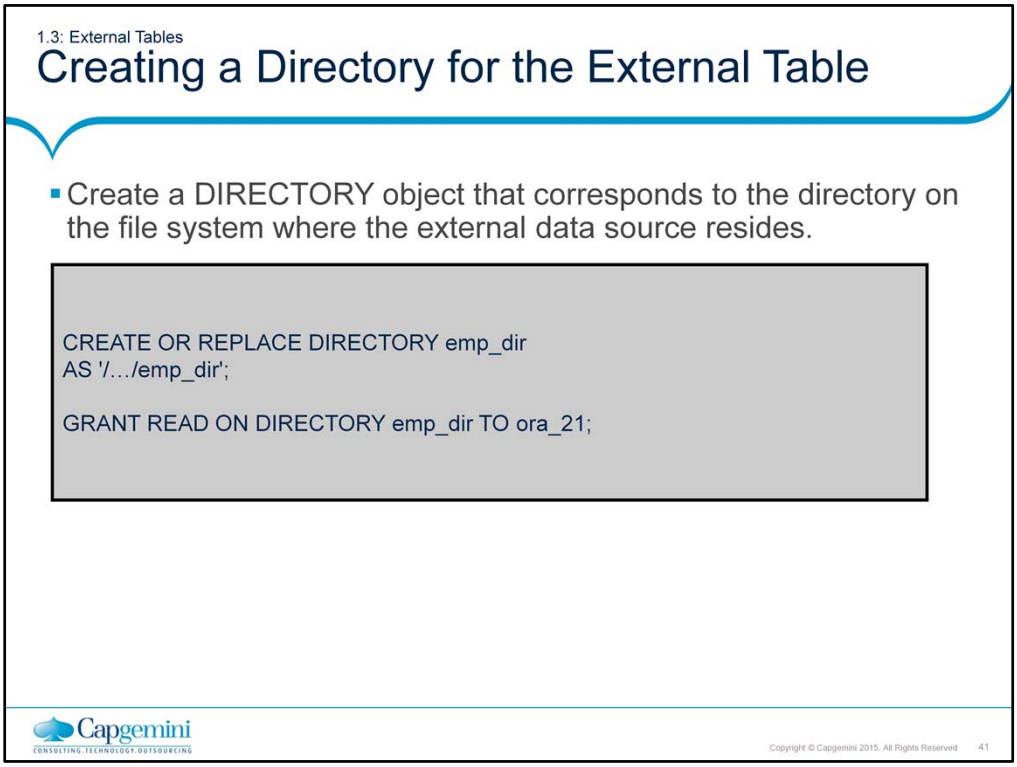


External Tables

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 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.



Example of Creating an External Table

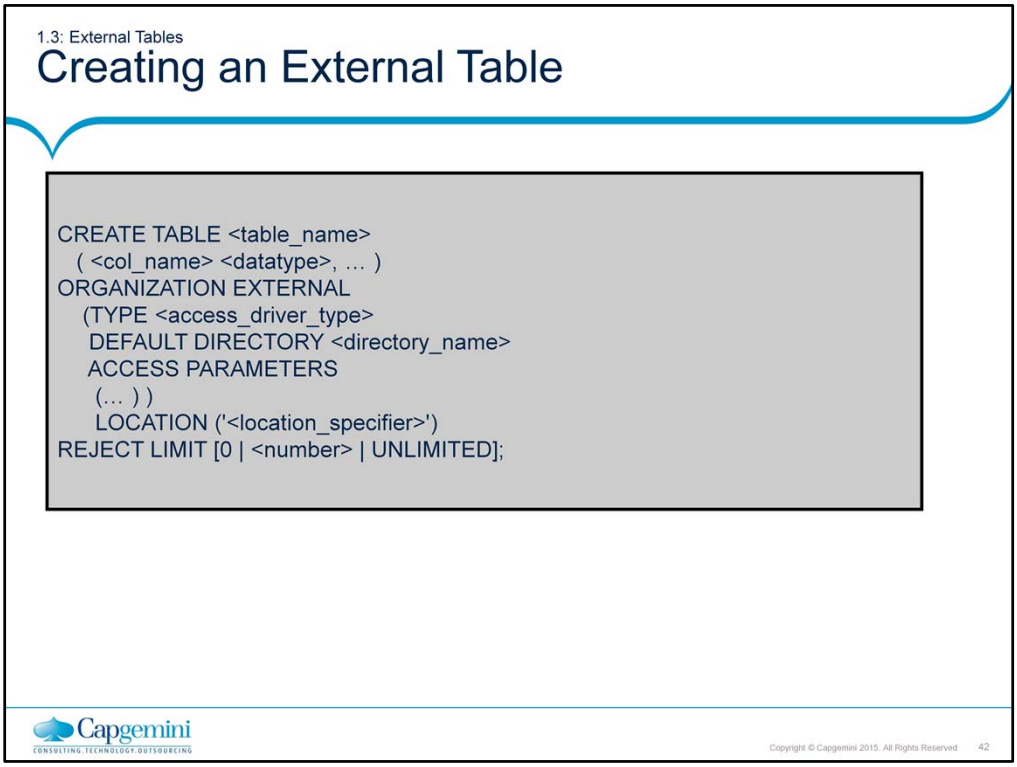
Use the CREATE DIRECTORY statement to create a directory object. A directory object specifies an alias for a directory on the server's file system where an external data source resides. You can use directory names when referring to an external data source, rather than hard code the operating system path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



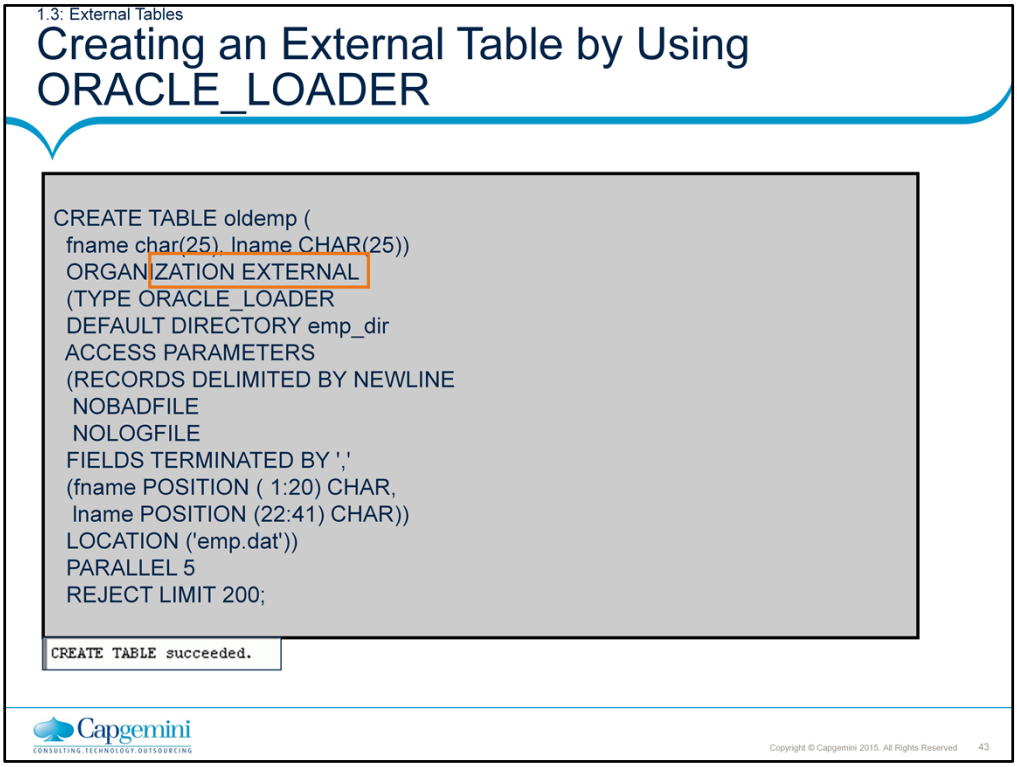
Creating an External Table

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.



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

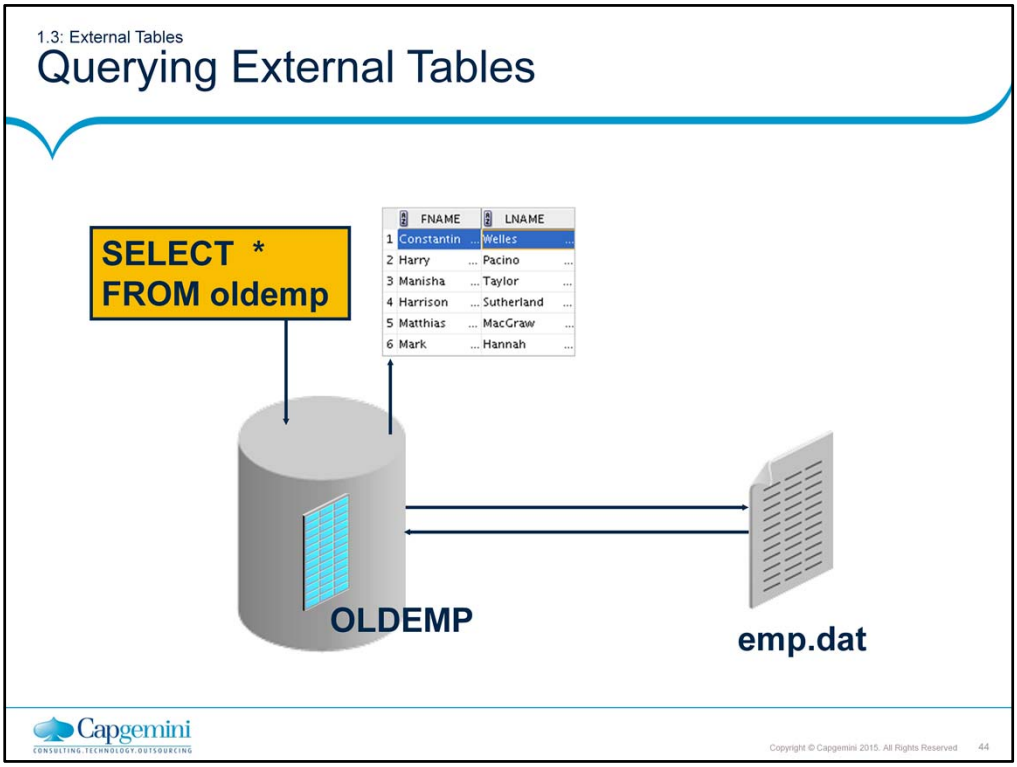
Records are delimited by new lines, and the fields are all terminated by a comma (,). The name of the file is /emp_dir/emp.dat.

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

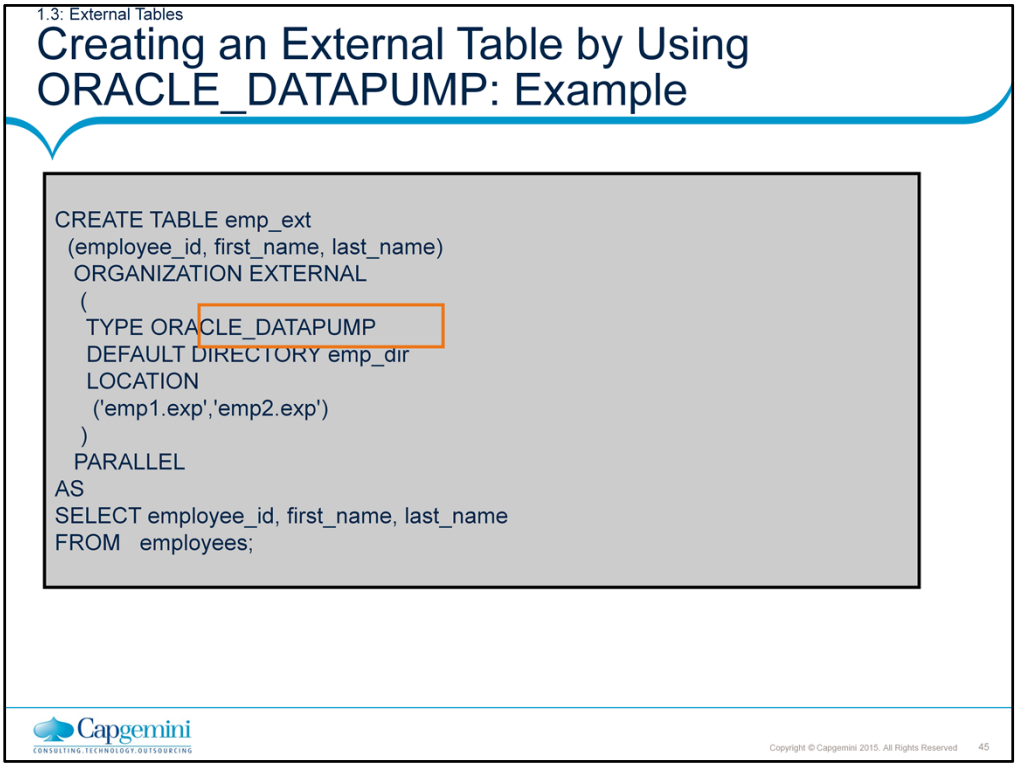


Querying External Tables

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.



Creating an External Table by Using ORACLE_DATAPUMP: Example

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.

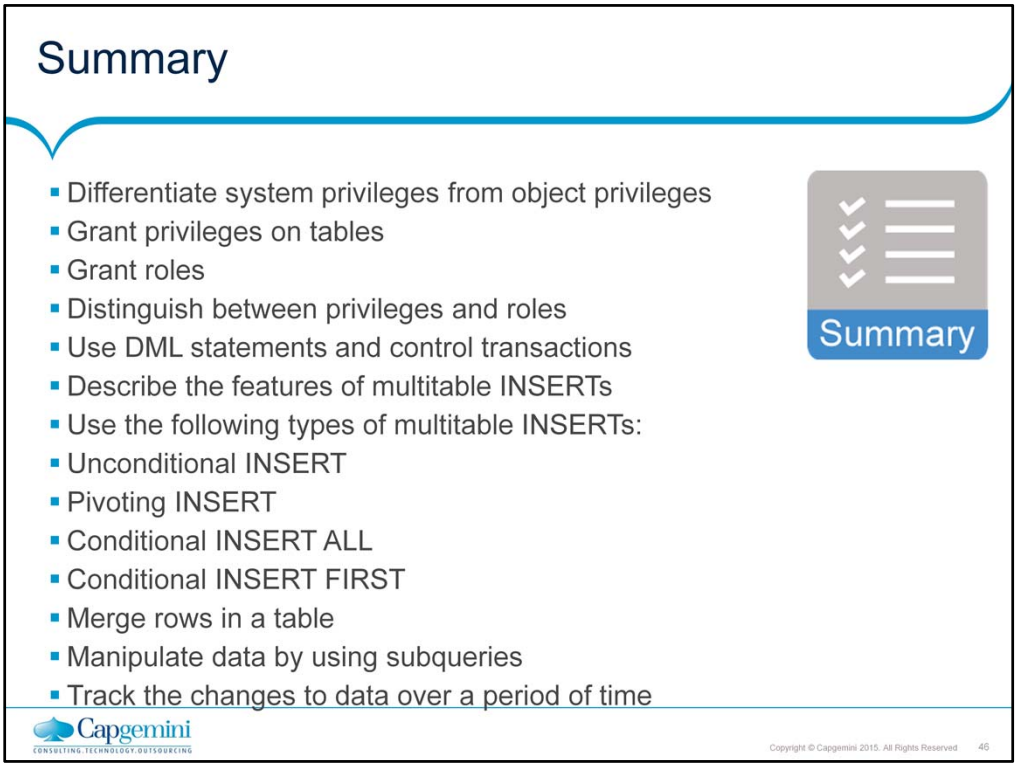
To populate data read from the EMPLOYEES table into an external table, you must perform the following steps:

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;



Add the notes here.

Review Question

- Question 1: You can use subqueries in DML statements to retrieve data by using an inline view. True/False
- Question 2: _____ statement is used to create users by DBA.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 47

Add the notes here.