Oracle Database 11*g*: SQL Fundamentals I

Volume II • Student Guide

ORACLE®

Authors

Salome Clement
Brian Pottle
Puja Singh

**Technical Contributors
and Reviewers**

Anjulaponni Azhagulekshmi
Clair Bennett
Zarko Cesljas
Yanti Chang
Gerlinde Frenzen
Steve Friedberg
Joel Goodman
Nancy Greenberg
Pedro Neves
Surya Rekha
Helen Robertson
Lauran Serhal
Tulika Srivastava

Editors

Aju Kumar
Arijit Ghosh

Graphic Designer

Rajiv Chandrabhanu

Publishers

Pavithran Adka
Veena Narasimhan

Contents

6 Displaying Data from Multiple Tables Using Joins

11 Creating Other Schema Objects

9

Manipulating Data

Objectives

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

- Describe each data manipulation language (DML) statement
- Insert rows into a table
- Update rows in a table
- Delete rows from a table
- Control transactions

Oracle University and NETEC, S.A. de C.V. use only.

Objective

In this lesson, you learn how to use the data manipulation language (DML) statements to insert rows into a table, update existing rows in a table, and delete existing rows from a table. You also learn how to control transactions with the COMMIT, SAVEPOINT, and ROLLBACK statements.

Lesson Agenda

- Adding new rows in a table
  - INSERT statement
- Changing data in a table
  - UPDATE statement
- Removing rows from a table:
  - DELETE statement
  - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

Data Manipulation Language

- A DML statement is executed when you:
  - Add new rows to a table
  - Modify existing rows in a table
  - Remove existing rows from a table
- A *transaction* consists of a collection of DML statements that form a logical unit of work.

Data Manipulation Language

Data manipulation language (DML) is a core part of SQL. When you want to add, update, or delete data in the database, you execute a DML statement. A collection of DML statements that form a logical unit of work is called a *transaction*.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decreasing the savings account, increasing the checking account, and recording the transaction in the transaction journal. The Oracle server must guarantee that all the three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing, the other statements of the transaction must be undone.

Note
- Most of the DML statements in this lesson assume that no constraints on the table are violated. Constraints are discussed later in this course.
- In SQL Developer, click the Run Script icon or press [F5] to run the DML statements. The feedback messages will be shown on the Script Output tabbed page.

Adding a New Row to a Table

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
	70 Public Relations		100	1700

New row

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10 Administration		200	1700
2	20 Marketing		201	1800
3	50 Shipping		124	1500
4	60 IT		103	1400
5	80 Sales		149	2500
6	90 Executive		100	1700
7	110 Accounting		205	1700
8	190 Contracting		(null)	1700

Insert new row into the DEPARTMENTS table.

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	70 Public Relations		100	1700
2	10 Administration		200	1700
3	20 Marketing		201	1800
4	50 Shipping		124	1500
5	60 IT		103	1400
6	80 Sales		149	2500
7	90 Executive		100	1700
8	110 Accounting		205	1700
9	190 Contracting		(null)	1700

ORACLE

Adding a New Row to a Table

The graphic in the slide illustrates the addition of a new department to the DEPARTMENTS table.

INSERT Statement Syntax

- Add new rows to a table by using the INSERT statement:

```
INSERT INTO    table [(column [, column...])]
VALUES         (value [, value...]);
```

- With this syntax, only one row is inserted at a time.

INSERT Statement Syntax

You can add new rows to a table by issuing the INSERT statement.

In the syntax:

table	Is the name of the table
column	Is the name of the column in the table to populate
value	Is the corresponding value for the column

Note: This statement with the VALUES clause adds only one row at a time to a table.

Inserting New Rows

- Insert a new row containing values for each column.
- List values in the default order of the columns in the table.
- Optionally, list the columns in the `INSERT` clause.

```
INSERT INTO departments(department_id,
        department_name, manager_id, location_id)
VALUES (70, 'Public Relations', 100, 1700);
```
`1 rows inserted`

- Enclose character and date values within single quotation marks.

Inserting New Rows

Because you can insert a new row that contains values for each column, the column list is not required in the `INSERT` clause. However, if you do not use the column list, the values must be listed according to the default order of the columns in the table, and a value must be provided for each column.

```
DESCRIBE departments
```

Name	Null	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

For clarity, use the column list in the `INSERT` clause.

Enclose character and date values within single quotation marks; however, it is not recommended that you enclose numeric values within single quotation marks.

Inserting Rows with Null Values

- Implicit method: Omit the column from the column list.

```
INSERT INTO   departments (department_id,
                           department_name)
VALUES        (30, 'Purchasing');
1 rows inserted
```

- Explicit method: Specify the NULL keyword in the VALUES clause.

```
INSERT INTO   departments
VALUES        (100, 'Finance', NULL, NULL);
1 rows inserted
```

Inserting Rows with Null Values

Method	Description
Implicit	Omit the column from the column list.
Explicit	Specify the NULL keyword in the VALUES list; specify the empty string (' ') in the VALUES list for character strings and dates.

Be sure that you can use null values in the targeted column by verifying the Null status with the DESCRIBE command.

The Oracle server automatically enforces all data types, data ranges, and data integrity constraints. Any column that is not listed explicitly obtains a null value in the new row.

Common errors that can occur during user input are checked in the following order:
- Mandatory value missing for a NOT NULL column
- Duplicate value violating any unique or primary key constraint
- Any value violating a CHECK constraint
- Referential integrity maintained for foreign key constraint
- Data type mismatches or values too wide to fit in column

Note: Use of the column list is recommended because it makes the INSERT statement more readable and reliable, or less prone to mistakes.

Inserting Special Values

The `SYSDATE` function records the current date and time.

```
INSERT INTO employees (employee_id,
              first_name, last_name,
              email, phone_number,
              hire_date, job_id, salary,
              commission_pct, manager_id,
              department_id)
VALUES        (113,
              'Louis', 'Popp',
              'LPOPP', '515.124.4567',
              SYSDATE, 'AC_ACCOUNT', 6900,
              NULL, 205, 110);
```
```
1 rows inserted
```

Inserting Special Values

You can use functions to enter special values in your table.

The slide example records information for employee Popp in the `EMPLOYEES` table. It supplies the current date and time in the `HIRE_DATE` column. It uses the `SYSDATE` function that returns the current date and time of the database server. You may also use the `CURRENT_DATE` function to get the current date in the session time zone. You can also use the `USER` function when inserting rows in a table. The `USER` function records the current username.

Confirming Additions to the Table

```
SELECT employee_id, last_name, job_id, hire_date, commission_pct
FROM   employees
WHERE  employee_id = 113;
```

	EMPLOYEE_ID	LAST_NAME	JOB_ID	HIRE_DATE	COMMISSION_PCT
1	113	Popp	AC_ACCOUNT	10-JUL-09	(null)

Inserting Specific Date and Time Values

- Add a new employee.

```
INSERT INTO employees
VALUES     (114,
            'Den', 'Raphealy',
            'DRAPHEAL', '515.127.4561',
            TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
            'SA_REP', 11000, 0.2, 100, 60);
1 rows inserted
```

- Verify your addition.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT
1	114 Den		Raphealy	DRAPHEAL	515.127.4561	03-FEB-99	SA_REP	11000	0.2

ORACLE

Inserting Specific Date and Time Values

The DD-MON-RR format is generally used to insert a date value. With the RR format, the system provides the correct century automatically.

You may also supply the date value in the DD-MON-YYYY format. This is recommended because it clearly specifies the century and does not depend on the internal RR format logic of specifying the correct century.

If a date must be entered in a format other than the default format (for example, with another century or a specific time), you must use the TO_DATE function.

The example in the slide records information for employee Raphealy in the EMPLOYEES table. It sets the HIRE_DATE column to be February 3, 1999.

Creating a Script

- Use the & substitution in a SQL statement to prompt for values.
- & is a placeholder for the variable value.

```
INSERT INTO departments
          (department_id, department_name, location_id)
VALUES    (&department_id, '&department_name',&location);
```

Enter Substitution Variable

DEPARTMENT_ID:

40

OK

Enter Substitution Variable

DEPARTMENT_NAME:

Human Resources

OK

Enter Substitution Variable

LOCATION:

2500

OK   Cancel
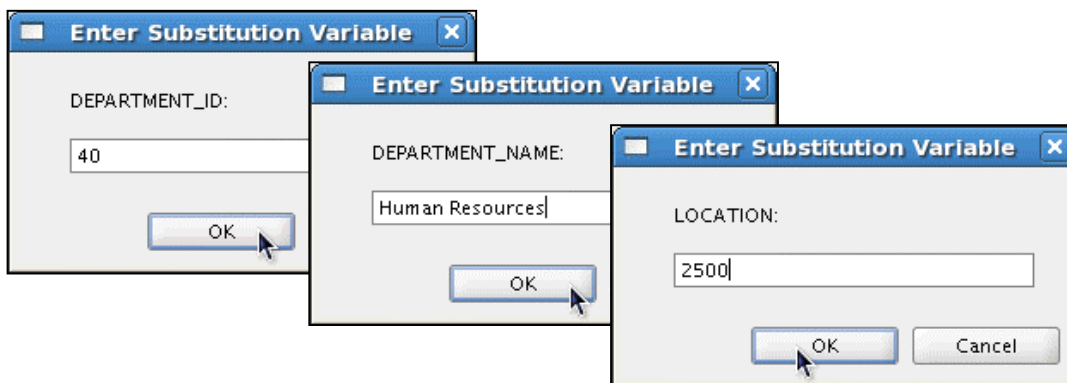
ORACLE

Creating a Script

You can save commands with substitution variables to a file and execute the commands in the file. The example in the slide records information for a department in the DEPARTMENTS table.

Run the script file and you are prompted for input for each of the ampersand (&) substitution variables. After entering a value for the substitution variable, click the OK button. The values that you input are then substituted into the statement. This enables you to run the same script file over and over, but supply a different set of values each time you run it.

Copying Rows
from Another Table

- Write your `INSERT` statement with a subquery:

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
  SELECT employee_id, last_name, salary, commission_pct
  FROM   employees
  WHERE  job_id LIKE '%REP%';

4 rows inserted
```

- Do not use the `VALUES` clause.
- Match the number of columns in the `INSERT` clause to those in the subquery.
- Inserts all the rows returned by the subquery in the table, `sales_reps`.

ORACLE

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 the example in the slide, for the `INSERT INTO` statement to work, you must have already created the `sales_reps` table using the `CREATE TABLE` statement. `CREATE TABLE` is discussed in the lesson titled "Using DDL Statements to Create and Manage Tables."

In place of the `VALUES` clause, you use a subquery.

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

In the syntax:

table	Is the name of the table
column	Is the name of the column in the table to populate
subquery	Is the subquery that returns rows to 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. Zero or more rows are added depending on the number of rows returned by the subquery. To create a copy of the rows of a table, use `SELECT *` in the subquery:

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

Lesson Agenda

- Adding new rows in a table
  - INSERT statement
- **Changing data in a table**
  - **UPDATE statement**
- Removing rows from a table:
  - DELETE statement
  - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

Oracle University and NETEC, S.A. de C.V. use only.

Changing Data in a Table

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	60
104	Bruce	Ernst	6000	103	(null)	60
107	Diana	Lorentz	4200	103	(null)	60
124	Kevin	Mourgos	5800	100	(null)	50

Update rows in the EMPLOYEES table:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	80
104	Bruce	Ernst	6000	103	(null)	80
107	Diana	Lorentz	4200	103	(null)	80
124	Kevin	Mourgos	5800	100	(null)	50

Changing Data in a Table

The slide illustrates changing the department number for employees in department 60 to department 80.

UPDATE Statement Syntax

- Modify existing values in a table with the UPDATE statement:

```
UPDATE        table
SET           column = value [, column = value, ...]
[WHERE        condition];
```

- Update more than one row at a time (if required).

UPDATE Statement Syntax

You can modify the existing values in a table by using the UPDATE statement.

In the syntax:

table	Is the name of the table
column	Is the name of the column in the table to populate
value	Is the corresponding value or subquery for the column
condition	Identifies the rows to be updated and is composed of column names, expressions, constants, subqueries, and comparison operators

Confirm the update operation by querying the table to display the updated rows.

For more information, see the section on "UPDATE" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Note: In general, use the primary key column in the WHERE clause to identify a single row for update. Using other columns can unexpectedly cause several rows to be updated. For example, identifying a single row in the EMPLOYEES table by name is dangerous, because more than one employee may have the same name.

Updating Rows in a Table

- Values for a specific row or rows are modified if you specify the WHERE clause:

```
UPDATE  employees
SET     department_id = 50
WHERE   employee_id = 113;
```
1 rows updated

- Values for all the rows in the table are modified if you omit the WHERE clause:

```
UPDATE    copy_emp
SET       department_id = 110;
```
22 rows updated

- Specify SET *column_name*= NULL to update a column value to NULL.

Updating Rows in a Table

The UPDATE statement modifies the values of a specific row or rows if the WHERE clause is specified. The example in the slide shows the transfer of employee 113 (Popp) to department 50.

If you omit the WHERE clause, values for all the rows in the table are modified. Examine the updated rows in the COPY_EMP table.

```
SELECT last_name, department_id
FROM   copy_emp;
```

	LAST_NAME	DEPARTMENT_ID
1	Whalen	110
2	Hartstein	110
3	Fay	110

. . .

For example, an employee who was an SA_REP has now changed his job to an IT_PROG. Therefore, his JOB_ID needs to be updated and the commission field needs to be set to NULL.

```
UPDATE employees
SET job_id = 'IT_PROG', commission_pct = NULL
WHERE employee_id = 114;
```

Note: The COPY_EMP table has the same data as the EMPLOYEES table.

Oracle University and NETEC, S.A. de C.V. use only.

Updating Two Columns with a Subquery

Update employee 113's job and salary to match those of employee 205.

```
UPDATE    employees
SET       job_id  = (SELECT  job_id
                     FROM    employees
                     WHERE   employee_id = 205),
          salary  = (SELECT  salary
                     FROM    employees
                     WHERE   employee_id = 205)
WHERE     employee_id    =   113;
1 rows updated
```

ORACLE

Updating Two Columns with a Subquery

You can update multiple columns in the SET clause of an UPDATE statement by writing multiple subqueries. The syntax is as follows:

```
UPDATE table
SET     column =
                     (SELECT     column
                      FROM table
                      WHERE condition)
        [ ,
         column  =
                     (SELECT     column
                      FROM table
                      WHERE condition)]
  [WHERE   condition ]   ;
```

The example in the slide can also be written as follows:

```
 UPDATE employees
 SET (job_id, salary)  = (SELECT  job_id, salary
                    FROM      employees
                    WHERE   employee_id = 205)
 WHERE     employee_id    =   113;
```

Updating Rows Based on Another Table

Use the subqueries in the `UPDATE` statements to update row values in a table based on values from another table:

```
UPDATE   copy_emp
SET      department_id  =  (SELECT department_id
                              FROM employees
                              WHERE employee id = 100)
WHERE    job_id         =  (SELECT job_id
                              FROM employees
                              WHERE employee_id = 200);
1 rows updated
```

Updating Rows Based on Another Table

You can use the subqueries in the `UPDATE` statements to update values in a table. The example in the slide updates the `COPY_EMP` table based on the values from the `EMPLOYEES` table. It changes the department number of all employees with employee 200's job ID to employee 100's current department number.

Lesson Agenda

- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- **Removing rows from a table:**
  - **`DELETE` statement**
  - **`TRUNCATE` statement**
- Database transactions control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- `FOR UPDATE` clause in a `SELECT` statement

Removing a Row from a Table

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

Delete a row from the `DEPARTMENTS` table:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700

ORACLE

Removing a Row from a Table

The Contracting department has been removed from the DEPARTMENTS table (assuming no constraints on the DEPARTMENTS table are violated), as shown by the graphic in the slide.

DELETE Statement

You can remove existing rows from a table by using the DELETE statement:

```
DELETE [FROM]    table
[WHERE           condition];
```

Oracle University and NETEC, S.A. de C.V. use only.

DELETE Statement Syntax

You can remove existing rows from a table by using the DELETE statement.

In the syntax:

table	Is the name of the table
condition	Identifies the rows to be deleted, and is composed of column names, expressions, constants, subqueries, and comparison operators

Note: If no rows are deleted, the message "0 rows deleted" is returned (on the Script Output tab in SQL Developer)

For more information, see the section on "DELETE" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Deleting Rows from a Table

- Specific rows are deleted if you specify the WHERE clause:

```
DELETE FROM departments
WHERE   department_name = 'Finance';
```
`1 rows deleted`

- All rows in the table are deleted if you omit the WHERE clause:

```
DELETE FROM   copy_emp;
```
`22 rows deleted`

Deleting Rows from a Table

You can delete specific rows by specifying the WHERE clause in the DELETE statement. The first example in the slide deletes the Accounting department from the DEPARTMENTS table. You can confirm the delete operation by displaying the deleted rows using the SELECT statement.

```
SELECT  *
FROM    departments
WHERE   department_name = 'Finance';
```
`0 rows selected`

However, if you omit the WHERE clause, all rows in the table are deleted. The second example in the slide deletes all rows from the COPY_EMP table, because no WHERE clause was specified.

Example:

Remove rows identified in the WHERE clause.

```
DELETE FROM  employees WHERE employee_id = 114;
```
`1 rows deleted`

```
DELETE FROM  departments WHERE department_id IN (30, 40);
```
`2 rows deleted`

Deleting Rows Based
on Another Table

Use the subqueries in the DELETE statements to remove rows
from a table based on values from another table:

```
DELETE FROM employees
WHERE   department_id =
                    (SELECT department_id
                     FROM   departments
                     WHERE  department_name
                            LIKE '%Public%');
1 rows deleted
```

ORACLE

Deleting Rows Based on Another Table

You can use the subqueries to delete rows from a table based on values from another table. The
example in the slide deletes all the employees in a department, where the department name contains
the string Public.

The subquery searches the DEPARTMENTS table to find the department number based on the
department name containing the string Public. The subquery then feeds the department number to
the main query, which deletes rows of data from the EMPLOYEES table based on this department
number.

TRUNCATE **Statement**

- Removes all rows from a table, leaving the table empty and the table structure intact
- Is a data definition language (DDL) statement rather than a DML statement; cannot easily be undone
- Syntax:

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE copy_emp;
```

TRUNCATE **Statement**

A more efficient method of emptying a table is by using the TRUNCATE statement.

You can use the TRUNCATE statement to quickly remove all rows from a table or cluster. Removing rows with the TRUNCATE statement is faster than removing them with the DELETE statement for the following reasons:

- The TRUNCATE statement is a data definition language (DDL) statement and generates no rollback information. Rollback information is covered later in this lesson.
- Truncating a table does not fire the delete triggers of the table.

If the table is the parent of a referential integrity constraint, you cannot truncate the table. You need to disable the constraint before issuing the TRUNCATE statement. Disabling constraints is covered in the lesson titled "Using DDL Statements to Create and Manage Tables."

Lesson Agenda

- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- **Database transactions control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`**
- Read consistency
- `FOR UPDATE` clause in a `SELECT` statement

Database Transactions

A database transaction consists of one of the following:

- DML statements that constitute one consistent change to the data
- One DDL statement
- One data control language (DCL) statement

Database Transactions

The Oracle server ensures data consistency based on transactions. Transactions give you more flexibility and control when changing data, and they ensure data consistency in the event of user process failure or system failure.

Transactions consist of DML statements that constitute one consistent change to the data. For example, a transfer of funds between two accounts should include the debit in one account and the credit to another account of the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

Transaction Types

Type	Description
Data manipulation language (DML)	Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work
Data definition language (DDL)	Consists of only one DDL statement
Data control language (DCL)	Consists of only one DCL statement

Database Transactions: Start and End

- Begin when the first DML SQL statement is executed.
- End with one of the following events:
  - A `COMMIT` or `ROLLBACK` statement is issued.
  - A DDL or DCL statement executes (automatic commit).
  - The user exits SQL Developer or SQL*Plus.
  - The system crashes.

ORACLE

Database Transaction: Start and End

When does a database transaction start and end?

A transaction begins when the first DML statement is encountered and ends when one of the following occurs:

- A `COMMIT` or `ROLLBACK` statement is issued.
- A DDL statement, such as `CREATE`, is issued.
- A DCL statement is issued.
- The user exits SQL Developer or SQL*Plus.
- A machine fails or the system crashes.

After one transaction ends, the next executable SQL statement automatically starts the next transaction.

A DDL statement or a DCL statement is automatically committed and, therefore, implicitly ends a transaction.

Advantages of COMMIT and ROLLBACK Statements

With COMMIT and ROLLBACK statements, you can:

- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically-related operations

ORACLE

Advantages of COMMIT and ROLLBACK Statements

With the COMMIT and ROLLBACK statements, you have control over making changes to the data permanent.

Explicit Transaction Control Statements
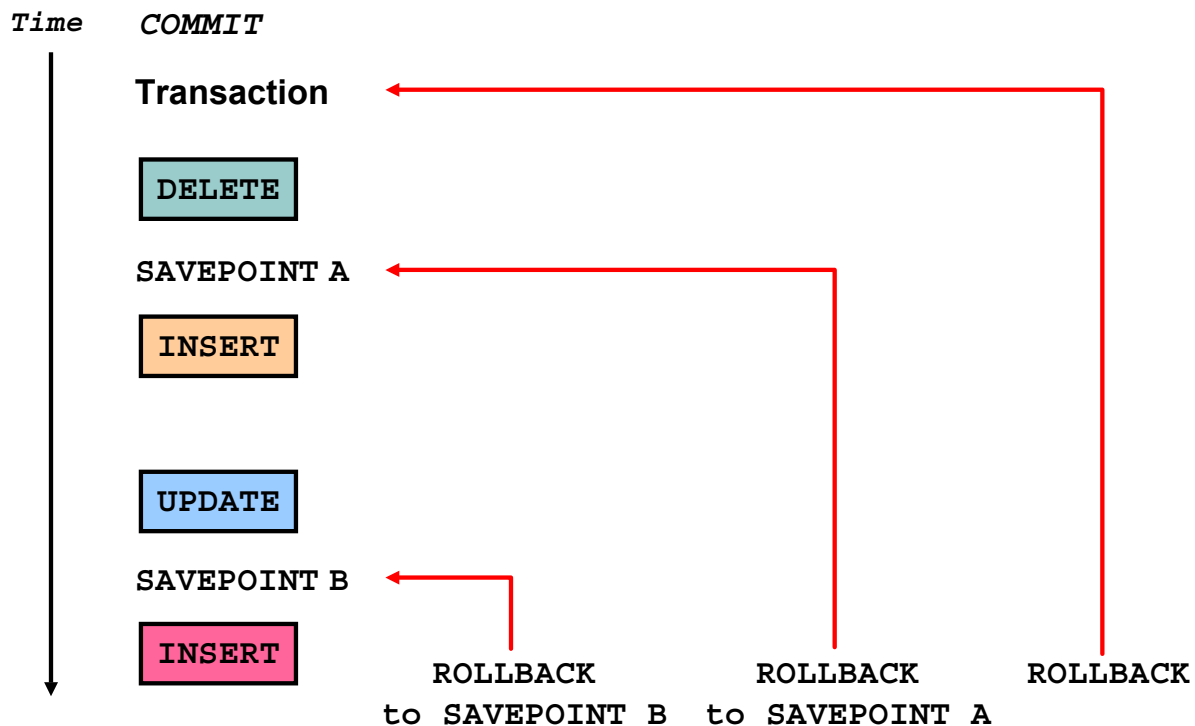
Time   *COMMIT*

Transaction

```
DELETE
```

SAVEPOINT A

```
INSERT
```

```
UPDATE
```

SAVEPOINT B

```
INSERT
```

ROLLBACK   **ROLLBACK**   **ROLLBACK**
to SAVEPOINT B   **to SAVEPOINT A**

Explicit Transaction Control Statements

You can control the logic of transactions by using the COMMIT, SAVEPOINT, and ROLLBACK statements.

Statement	Description
COMMIT	COMMIT ends the current transaction by making all pending data changes permanent.
SAVEPOINT *name*	SAVEPOINT *name* marks a savepoint within the current transaction.
ROLLBACK	ROLLBACK ends the current transaction by discarding all pending data changes.
ROLLBACK TO *SAVEPOINT name*	ROLLBACK TO SAVEPOINT rolls back the current transaction to the specified savepoint, thereby discarding any changes and/or savepoints that were created after the savepoint to which you are rolling back. If you omit the TO SAVEPOINT clause, the ROLLBACK statement rolls back the entire transaction. Because savepoints are logical, there is no way to list the savepoints that you have created.

Note: You cannot COMMIT to a SAVEPOINT. SAVEPOINT is not ANSI-standard SQL.

Rolling Back Changes to a Marker

- Create a marker in the current transaction by using the SAVEPOINT statement.
- Roll back to that marker by using the ROLLBACK TO SAVEPOINT statement.

```
UPDATE...
SAVEPOINT update_done;
SAVEPOINT update_done succeeded.
INSERT...
ROLLBACK TO update_done;
ROLLBACK TO succeeded.
```

Rolling Back Changes to a Marker

You can create a marker in the current transaction by using the SAVEPOINT statement, which divides the transaction into smaller sections. You can then discard pending changes up to that marker by using the ROLLBACK TO SAVEPOINT statement.

Note that if you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.

Implicit Transaction Processing

- An automatic commit occurs in the following circumstances:
  - A DDL statement issued
  - A DCL statement issued
  - Normal exit from SQL Developer or SQL*Plus, without explicitly issuing COMMIT or ROLLBACK statements
- An automatic rollback occurs when there is an abnormal termination of SQL Developer or SQL*Plus or a system failure.

ORACLE

Implicit Transaction Processing

Status	Circumstances
Automatic commit	DDL statement or DCL statement issued SQL Developer or SQL*Plus exited normally, without explicitly issuing COMMIT or ROLLBACK commands
Automatic rollback	Abnormal termination of SQL Developer or SQL*Plus or system failure

Note: In SQL*Plus, the AUTOCOMMIT command can be toggled ON or OFF. If set to ON, each individual DML statement is committed as soon as it is executed. You cannot roll back the changes. If set to OFF, the COMMIT statement can still be issued explicitly. Also, the COMMIT statement is issued when a DDL statement is issued or when you exit SQL*Plus. The SET AUTOCOMMIT ON/OFF command is skipped in SQL Developer. DML is committed on a normal exit from SQL Developer only if you have the Autocommit preference enabled. To enable Autocommit, perform the following:

- In the Tools menu, select Preferences. In the Preferences dialog box, expand Database and select Worksheet Parameters.
- In the right pane, select the "Autocommit in SQL Worksheet" option. Click OK.

Implicit Transaction Processing (continued)

System Failures

When a transaction is interrupted by a system failure, the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to the data and returns the tables to the state at the time of the last commit. In this way, the Oracle server protects the integrity of the tables.

In SQL Developer, a normal exit from the session is accomplished by selecting Exit from the File menu. In SQL*Plus, a normal exit is accomplished by entering the EXIT command at the prompt. Closing the window is interpreted as an abnormal exit.

State of the Data Before COMMIT or ROLLBACK

- The previous state of the data can be recovered.
- The current user can review the results of the DML operations by using the SELECT statement.
- Other users *cannot* view the results of the DML statements issued by the current user.
- The affected rows are *locked*; other users cannot change the data in the affected rows.

State of the Data Before COMMIT or ROLLBACK

Every data change made during the transaction is temporary until the transaction is committed.

The state of the data before COMMIT or ROLLBACK statements are issued can be described as follows:

- Data manipulation operations primarily affect the database buffer; therefore, the previous state of the data can be recovered.
- The current user can review the results of the data manipulation operations by querying the tables.
- Other users cannot view the results of the data manipulation operations made by the current user. The Oracle server institutes read consistency to ensure that each user sees data as it existed at the last commit.
- The affected rows are locked; other users cannot change the data in the affected rows.

State of the Data After `COMMIT`

- Data changes are saved in the database.
- The previous state of the data is overwritten.
- All users can view the results.
- Locks on the affected rows are released; those rows are available for other users to manipulate.
- All savepoints are erased.

State of the Data After `COMMIT`

Make all pending changes permanent by using the `COMMIT` statement. Here is what happens after a `COMMIT` statement:

- Data changes are written to the database.
- The previous state of the data is no longer available with normal SQL queries.
- All users can view the results of the transaction.
- The locks on the affected rows are released; the rows are now available for other users to perform new data changes.
- All savepoints are erased.

Committing Data

- Make the changes:

```
DELETE FROM employees
WHERE   employee_id = 99999;
```
`1 rows deleted`

```
INSERT INTO departments
VALUES (290, 'Corporate Tax', NULL, 1700);
```
`1 rows inserted`

- Commit the changes:

```
COMMIT;
```
`COMMIT succeeded.`

ORACLE

Committing Data

In the example in the slide, a row is deleted from the EMPLOYEES table and a new row is inserted into the DEPARTMENTS table. The changes are saved by issuing the COMMIT statement.

Example:

Remove departments 290 and 300 in the DEPARTMENTS table and update a row in the EMPLOYEES table. Save the data change.

```
DELETE FROM departments
WHERE   department_id IN (290, 300);

UPDATE  employees
  SET   department_id = 80
  WHERE employee_id = 206;

COMMIT;
```

State of the Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement:

- Data changes are undone.
- Previous state of the data is restored.
- Locks on the affected rows are released.

```
DELETE FROM copy_emp;
ROLLBACK ;
```

State of the Data After ROLLBACK

Discard all pending changes by using the ROLLBACK statement, which results in the following:
- Data changes are undone.
- The previous state of the data is restored.
- Locks on the affected rows are released.

State of the Data After ROLLBACK: Example

```
DELETE FROM test;
25,000 rows deleted.

ROLLBACK;
Rollback complete.

DELETE FROM test WHERE  id = 100;
1 row deleted.

SELECT * FROM   test WHERE  id = 100;
No rows selected.

COMMIT;
Commit complete.
```

State of the Data After ROLLBACK: Example

While attempting to remove a record from the TEST table, you may accidentally empty the table.
However, you can correct the mistake, reissue a proper statement, and make the data change
permanent.

Statement-Level Rollback

- If a single DML statement fails during execution, only that statement is rolled back.
- The Oracle server implements an implicit savepoint.
- All other changes are retained.
- The user should terminate transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.

Statement-Level Rollback

A part of a transaction can be discarded through an implicit rollback if a statement execution error is detected. If a single DML statement fails during execution of a transaction, its effect is undone by a statement-level rollback, but the changes made by the previous DML statements in the transaction are not discarded. They can be committed or rolled back explicitly by the user.

The Oracle server issues an implicit commit before and after any DDL statement. So, even if your DDL statement does not execute successfully, you cannot roll back the previous statement because the server issued a commit.

Terminate your transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.

Lesson Agenda

- Adding new rows in a table
  - `INSERT` statement
- Changing data in a table
  - `UPDATE` statement
- Removing rows from a table:
  - `DELETE` statement
  - `TRUNCATE` statement
- Database transactions control using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`
- Read consistency
- `FOR UPDATE` clause in a `SELECT` statement

Read Consistency

- Read consistency guarantees a consistent view of the data at all times.
- Changes made by one user do not conflict with the changes made by another user.
- Read consistency ensures that, on the same data:
  - Readers do not wait for writers
  - Writers do not wait for readers
  - Writers wait for writers

Read Consistency

Database users access the database in two ways:
- Read operations (SELECT statement)
- Write operations (INSERT, UPDATE, DELETE statements)

You need read consistency so that the following occur:
- The database reader and writer are ensured a consistent view of the data.
- Readers do not view data that is in the process of being changed.
- Writers are ensured that the changes to the database are done in a consistent manner.
- Changes made by one writer do not disrupt or conflict with the changes being made by another writer.

The purpose of read consistency is to ensure that each user sees data as it existed at the last commit, before a DML operation started.

Note: The same user can log in to different sessions. Each session maintains read consistency in the manner described above, even if they are the same users.

Implementing Read Consistency
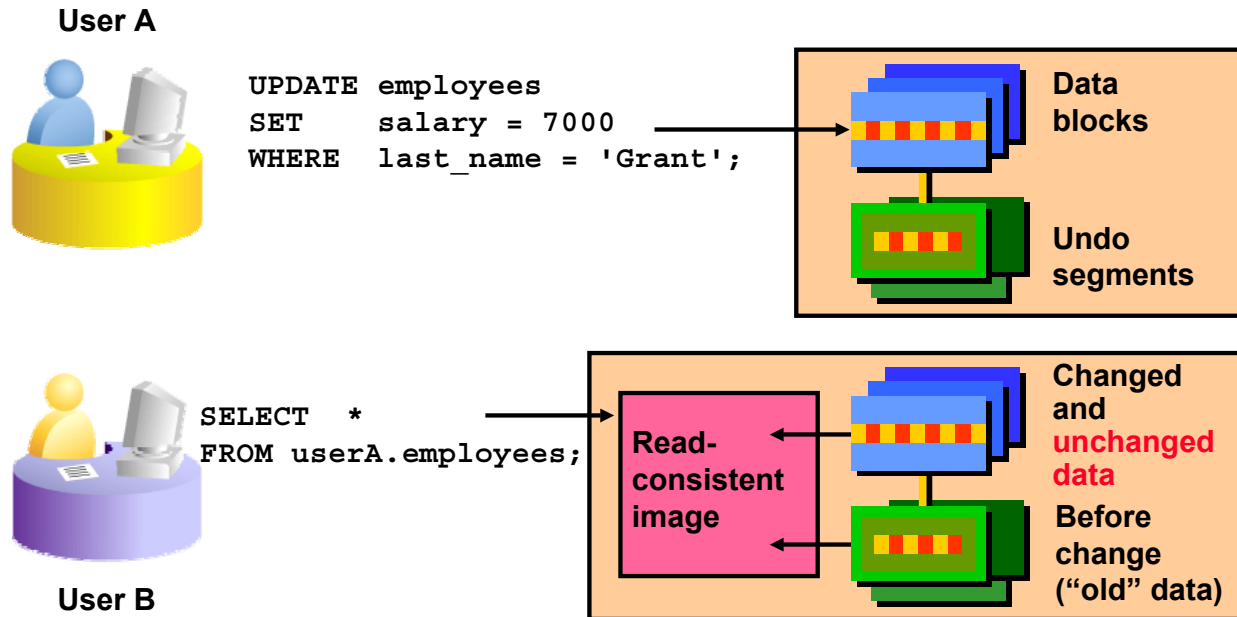
User A

```
UPDATE employees
SET    salary = 7000
WHERE  last_name = 'Grant';
```

Data blocks

Undo segments

User B

```
SELECT  *
FROM userA.employees;
```

Read-consistent image

Changed and unchanged data

Before change ("old" data)

Implementing Read Consistency

Read consistency is an automatic implementation. It keeps a partial copy of the database in the undo segments. The read-consistent image is constructed from the committed data in the table and the old data that is being changed and is not yet committed from the undo segment.

When an insert, update, or delete operation is made on the database, the Oracle server takes a copy of the data before it is changed and writes it to an *undo segment*.

All readers, except the one who issued the change, see the database as it existed before the changes started; they view the undo segment's "snapshot" of the data.

Before the changes are committed to the database, only the user who is modifying the data sees the database with the alterations. Everyone else sees the snapshot in the undo segment. This guarantees that readers of the data read consistent data that is not currently undergoing change.

When a DML statement is committed, the change made to the database becomes visible to anyone issuing a `SELECT` statement *after* the commit is done. The space occupied by the *old* data in the undo segment file is freed for reuse.

If the transaction is rolled back, the changes are undone:
- The original, older version of the data in the undo segment is written back to the table.
- All users see the database as it existed before the transaction began.

Lesson Agenda

- Adding new rows in a table
  - INSERT statement
- Changing data in a table
  - UPDATE statement
- Removing rows from a table:
  - DELETE statement
  - TRUNCATE statement
- Database transactions control using COMMIT, ROLLBACK, and SAVEPOINT
- Read consistency
- FOR UPDATE clause in a SELECT statement

ORACLE

FOR UPDATE Clause in a SELECT Statement

- Locks the rows in the `EMPLOYEES` table where `job_id` is `SA_REP`.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE
ORDER BY employee_id;
```

- Lock is released only when you issue a `ROLLBACK` or a `COMMIT`.
- If the `SELECT` statement attempts to lock a row that is locked by another user, the database waits until the row is available, and then returns the results of the `SELECT` statement.

FOR UPDATE Clause in a SELECT Statement

When you issue a `SELECT` statement against the database to query some records, no locks are placed on the selected rows. In general, this is required because the number of records locked at any given time is (by default) kept to the absolute minimum: only those records that have been changed but not yet committed are locked. Even then, others will be able to read those records as they appeared before the change (the "before image" of the data). There are times, however, when you may want to lock a set of records even before you change them in your program. Oracle offers the FOR UPDATE clause of the SELECT statement to perform this locking.

When you issue a `SELECT...FOR UPDATE` statement, the relational database management system (RDBMS) automatically obtains exclusive row-level locks on all the rows identified by the SELECT statement, thereby holding the records "for your changes only." No one else will be able to change any of these records until you perform a ROLLBACK or a COMMIT.

You can append the optional keyword NOWAIT to the FOR UPDATE clause to tell the Oracle server not to wait if the table has been locked by another user. In this case, control will be returned immediately to your program or to your SQL Developer environment so that you can perform other work, or simply wait for a period of time before trying again. Without the NOWAIT clause, your process will block until the table is available, when the locks are released by the other user through the issue of a COMMIT or a ROLLBACK command.

FOR UPDATE Clause: Examples

- You can use the `FOR UPDATE` clause in a `SELECT` statement against multiple tables.

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK'
AND location_id = 1500
FOR UPDATE
ORDER BY e.employee_id;
```

- Rows from both the `EMPLOYEES` and `DEPARTMENTS` tables are locked.
- Use `FOR UPDATE OF column_name` to qualify the column you intend to change, then only the rows from that specific table are locked.

Oracle University and NETEC, S.A. de C.V. use only.

FOR UPDATE Clause: Examples

In the example in the slide, the statement locks rows in the `EMPLOYEES` table with `JOB_ID` set to `ST_CLERK` and `LOCATION_ID` set to 1500, and locks rows in the `DEPARTMENTS` table with departments in `LOCATION_ID` set as 1500.

You can use the `FOR UPDATE OF` *column_name* to qualify the column that you intend to change. The `OF` list of the `FOR UPDATE` clause does not restrict you to changing only those columns of the selected rows. Locks are still placed on all rows; if you simply state `FOR UPDATE` in the query and do not include one or more columns after the `OF` keyword, the database will lock all identified rows across all the tables listed in the `FROM` clause.

The following statement locks only those rows in the `EMPLOYEES` table with `ST_CLERK` located in `LOCATION_ID` 1500. No rows are locked in the `DEPARTMENTS` table:

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK' AND location_id = 1500
FOR UPDATE OF e.salary
ORDER BY e.employee_id;
```

FOR UPDATE Clause: Examples (continued)

In the following example, the database is instructed to wait for five seconds for the row to become available, and then return control to you.

```
SELECT employee_id, salary, commission_pct, job_id
FROM employees
WHERE job_id = 'SA_REP'
FOR UPDATE WAIT 5
ORDER BY employee_id;
```

Quiz

The following statements produce the same results:

```
DELETE FROM copy_emp;
```

```
TRUNCATE TABLE copy_emp;
```

1. True
2. False

Answer: 2

Summary

In this lesson, you should have learned how to use the following statements:

Function	Description
INSERT	Adds a new row to the table
UPDATE	Modifies existing rows in the table
DELETE	Removes existing rows from the table
TRUNCATE	Removes all rows from a table
COMMIT	Makes all pending changes permanent
SAVEPOINT	Is used to roll back to the savepoint marker
ROLLBACK	Discards all pending data changes
FOR UPDATE clause in SELECT	Locks rows identified by the SELECT query

Summary

In this lesson, you should have learned how to manipulate data in the Oracle database by using the INSERT, UPDATE, DELETE, and TRUNCATE statements, as well as how to control data changes by using the COMMIT, SAVEPOINT, and ROLLBACK statements. You also learned how to use the FOR UPDATE clause of the SELECT statement to lock rows for your changes only.

Remember that the Oracle server guarantees a consistent view of data at all times.

Practice 9: Overview

This practice covers the following topics:

- Inserting rows into the tables
- Updating and deleting rows in the table
- Controlling transactions

Practice 9: Overview

In this practice, you add rows to the MY_EMPLOYEE table, update and delete data from the table, and control your transactions. You run a script to create the MY_EMPLOYEE table.

10

Using DDL Statements
to Create and Manage Tables

Objectives

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

- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation
- Describe how schema objects work

Oracle University and NETEC, S.A. de C.V. use only.

Objectives

In this lesson, you are introduced to the data definition language (DDL) statements. You learn the basics of how to create simple tables, alter them, and remove them. The data types available in DDL are shown and schema concepts are introduced. Constraints are discussed in this lesson. Exception messages that are generated from violating constraints during DML operations are shown and explained.

Lesson Agenda

- **Database objects**
  - Naming rules
- `CREATE TABLE` statement:
  - Access another user's tables
  - `DEFAULT` option
- Data types
- Overview of constraints: `NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`, `CHECK` constraints
- Creating a table using a subquery
- `ALTER TABLE`
  - Read-only tables
- `DROP TABLE` statement

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative name to an object

ORACLE

Database Objects

The Oracle Database can contain multiple data structures. Each structure should be outlined in the database design so that it can be created during the build stage of database development.

- **Table:** Stores data
- **View:** Subset of data from one or more tables
- **Sequence:** Generates numeric values
- **Index:** Improves the performance of some queries
- **Synonym:** Gives alternative name to an object

Oracle Table Structures

- Tables can be created at any time, even when users are using the database.
- You do not need to specify the size of a table. The size is ultimately defined by the amount of space allocated to the database as a whole. It is important, however, to estimate how much space a table will use over time.
- Table structure can be modified online.

Note: More database objects are available, but are not covered in this course.

Naming Rules

Table names and column names must:

- Begin with a letter
- Be 1–30 characters long
- Contain only A–Z, a–z, 0–9, _, $, and #
- Not duplicate the name of another object owned by the same user
- Not be an Oracle server–reserved word

Naming Rules

You name database tables and columns according to the standard rules for naming any Oracle database object:

- Table names and column names must begin with a letter and be 1–30 characters long.
- Names must contain only the characters A–Z, a–z, 0–9, _ (underscore), $, and # (legal characters, but their use is discouraged).
- Names must not duplicate the name of another object owned by the same Oracle server user.
- Names must not be an Oracle server–reserved word.
  - You may also use quoted identifiers to represent the name of an object. A quoted identifier begins and ends with double quotation marks (""). If you name a schema object using a quoted identifier, you must use the double quotation marks whenever you refer to that object. Quoted identifiers can be reserved words, although this is not recommended.

Naming Guidelines

Use descriptive names for tables and other database objects.

Note: Names are not case-sensitive. For example, EMPLOYEES is treated to be the same name as eMPloyees or eMpLOYEES. However, quoted identifiers are case-sensitive.

For more information, see the section on *Schema Object Names and Qualifiers* in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1).*

Lesson Agenda

- Database objects
  - Naming rules
- **CREATE TABLE** statement:
  - Access another user's tables
  - **DEFAULT** option
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- Creating a table using a subquery
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

CREATE TABLE Statement

- You must have:
  - The CREATE TABLE privilege
  - A storage area

```
CREATE TABLE [schema.]table
        (column datatype [DEFAULT expr][, ...]);
```

- You specify:
  - The table name
  - The column name, column data type, and column size
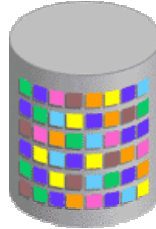
CREATE TABLE Statement

You create tables to store data by executing the SQL CREATE TABLE statement. This statement is one of the DDL statements that are a subset of the SQL statements used to create, modify, or remove Oracle Database structures. These statements have an immediate effect on the database and they also record information in the data dictionary.

To create a table, a user must have the CREATE TABLE privilege and a storage area in which to create objects. The database administrator (DBA) uses data control language (DCL) statements to grant privileges to users.

In the syntax:

schema	Is the same as the owner's name
table	Is the name of the table
DEFAULT expr	Specifies a default value if a value is omitted in the INSERT statement
column	Is the name of the column
datatype	Is the column's data type and length

Referencing Another User's Tables

- Tables belonging to other users are not in the user's schema.
- You should use the owner's name as a prefix to those tables.

USERA

USERB

```
SELECT *
FROM userB.employees;
```

```
SELECT *
FROM userA.employees;
```
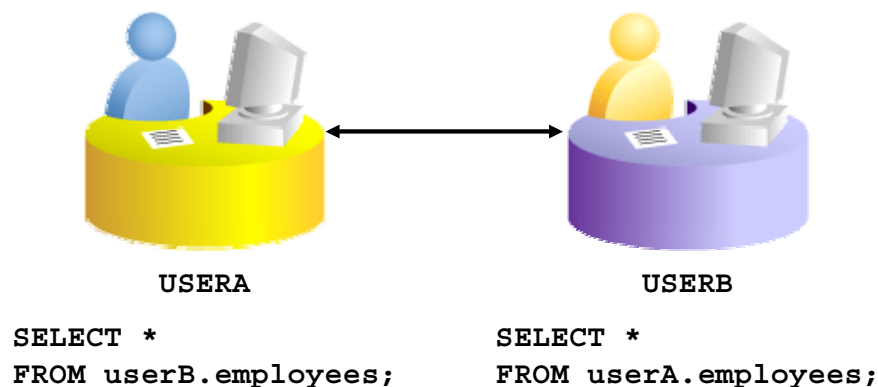
Referencing Another User's Tables

A schema is a collection of logical structures of data or *schema objects*. A schema is owned by a database user and has the same name as that user. Each user owns a single schema.

Schema objects can be created and manipulated with SQL and include tables, views, synonyms, sequences, stored procedures, indexes, clusters, and database links.

If a table does not belong to the user, the owner's name must be prefixed to the table. For example, if there are schemas named USERA and USERB, and both have an EMPLOYEES table, then if USERA wants to access the EMPLOYEES table that belongs to USERB, USERA must prefix the table name with the schema name:

```
SELECT *
FROM   userb.employees;
```

If USERB wants to access the EMPLOYEES table that is owned by USERA, USERB must prefix the table name with the schema name:

```
SELECT *
FROM   usera.employees;
```

DEFAULT Option

- Specify a default value for a column during an insert.

```
... hire_date DATE DEFAULT SYSDATE, ...
```

- Literal values, expressions, or SQL functions are legal values.
- Another column's name or a pseudocolumn are illegal values.
- The default data type must match the column data type.

```
CREATE TABLE hire_dates
        (id              NUMBER(8),
         hire_date DATE DEFAULT SYSDATE);
CREATE TABLE succeeded.
```

ORACLE

DEFAULT Option

When you define a table, you can specify that a column should be given a default value by using the DEFAULT option. This option prevents null values from entering the columns when a row is inserted without a value for the column. The default value can be a literal, an expression, or a SQL function (such as SYSDATE or USER), but the value cannot be the name of another column or a pseudocolumn (such as NEXTVAL or CURRVAL). The default expression must match the data type of the column.

Consider the following examples:

```
  INSERT INTO hire_dates values(45, NULL);
```
The above statement will insert the null value rather than the default value.

```
  INSERT INTO hire_dates(id) values(35);
```
The above statement will insert SYSDATE for the HIRE_DATE column.

Note: In SQL Developer, click the Run Script icon or press [F5] to run the DDL statements. The feedback messages will be shown on the Script Output tabbed page.

Creating Tables

- Create the table:

```
CREATE TABLE dept
      (deptno       NUMBER(2),
       dname        VARCHAR2(14),
       loc          VARCHAR2(13),
       create_date DATE DEFAULT SYSDATE);
```
CREATE TABLE succeeded.

- Confirm table creation:

```
DESCRIBE dept
```

```
DESCRIBE dept
Name                            Null     Type
------------------------------- -------- ----------------
DEPTNO                                   NUMBER(2)
DNAME                                    VARCHAR2(14)
LOC                                      VARCHAR2(13)
CREATE_DATE                              DATE

4 rows selected
```

ORACLE

Creating Tables

The example in the slide creates the DEPT table with four columns: DEPTNO, DNAME, LOC, and CREATE_DATE. The CREATE_DATE column has a default value. If a value is not provided for an INSERT statement, the system date is automatically inserted.

To confirm that the table was created, run the DESCRIBE command.

Because creating a table is a DDL statement, an automatic commit takes place when this statement is executed.

Note: You can view the list of tables you own by querying the data dictionary. For example:
```
select table_name from user_tables
```
Using data dictionary views, you can also find information about other database objects such as views, indexes, and so on. You will learn about data dictionaries in detail in the *Oracle Database 11g: SQL Fundaments II* course.

Lesson Agenda

- Database objects
  - Naming rules
- `CREATE TABLE` statement:
  - Access another user's tables
  - `DEFAULT` option
- **Data types**
- Overview of constraints: `NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`, `CHECK` constraints
- Creating a table using a subquery
- `ALTER TABLE`
  - Read-only tables
- `DROP TABLE` statement

Data Types

Data Type	Description
VARCHAR2(*size*)	Variable-length character data
CHAR(*size*)	Fixed-length character data
NUMBER(*p,s*)	Variable-length numeric data
DATE	Date and time values
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)
RAW and LONG RAW	Raw binary data
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

Data Types

When you identify a column for a table, you need to provide a data type for the column. There are several data types available:

Data Type	Description
VARCHAR2(*size*)	Variable-length character data (A maximum *size* must be specified: minimum *size* is 1; maximum *size* is 4,000.)
CHAR [(*size*)]	Fixed-length character data of length *size* bytes (Default and minimum *size* is 1; maximum *size* is 2,000.)
NUMBER [(*p,s*)]	Number having precision *p* and scale *s* (Precision is the total number of decimal digits and scale is the number of digits to the right of the decimal point; precision can range from 1 to 38, and scale can range from –84 to 127.)
DATE	Date and time values to the nearest second between January 1, 4712 B.C., and December 31, 9999 A.D.
LONG	Variable-length character data (up to 2 GB)
CLOB	Character data (up to 4 GB)

Data Types (continued)

Data Type	Description
RAW(*size*)	Raw binary data of length *size* (A maximum *size* must be specified: maximum *size* is 2,000.)
LONG RAW	Raw binary data of variable length (up to 2 GB)
BLOB	Binary data (up to 4 GB)
BFILE	Binary data stored in an external file (up to 4 GB)
ROWID	A base-64 number system representing the unique address of a row in its table

Guidelines
- A LONG column is not copied when a table is created using a subquery.
- A LONG column cannot be included in a GROUP BY or an ORDER BY clause.
- Only one LONG column can be used per table.
- No constraints can be defined on a LONG column.
- You might want to use a CLOB column rather than a LONG column.

Datetime Data Types

You can use several datetime data types:

Data Type	Description
`TIMESTAMP`	Date with fractional seconds
`INTERVAL YEAR TO MONTH`	Stored as an interval of years and months
`INTERVAL DAY TO SECOND`	Stored as an interval of days, hours, minutes, and seconds

Datetime Data Types

Data Type	Description
`TIMESTAMP`	Enables storage of time as a date with fractional seconds. It stores the year, month, day, hour, minute, and the second value of the `DATE` data type as well as the fractional seconds value There are several variations of this data type such as `WITH TIMEZONE`, `WITH LOCALTIMEZONE`.
`INTERVAL YEAR TO MONTH`	Enables storage of time as an interval of years and months. Used to represent the difference between two datetime values in which the only significant portions are the year and month
`INTERVAL DAY TO SECOND`	Enables storage of time as an interval of days, hours, minutes, and seconds. Used to represent the precise difference between two datetime values

Note: These datetime data types are available with Oracle9*i* and later releases. The datetime data types are discussed in detail in the lesson titled "Managing Data in Different Time Zones" in the *Oracle Database 11g: SQL Fundamentals II* course.

Also, for more information about the datetime data types, see the sections on "`TIMESTAMP` Datatype," "`INTERVAL YEAR TO MONTH` Datatype," and "`INTERVAL DAY TO SECOND` Datatype" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Lesson Agenda

- Database objects
  - Naming rules
- CREATE TABLE statement:
  - Access another user's tables
  - DEFAULT option
- Data types
- **Overview of constraints:** NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK **constraints**
- Creating a table using a subquery
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

Including Constraints

- Constraints enforce rules at the table level.
- Constraints prevent the deletion of a table if there are dependencies.
- The following constraint types are valid:
  - NOT NULL
  - UNIQUE
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK

ORACLE

Constraints

The Oracle server uses constraints to prevent invalid data entry into tables.

You can use constraints to do the following:
- Enforce rules on the data in a table whenever a row is inserted, updated, or deleted from that table. The constraint must be satisfied for the operation to succeed.
- Prevent the deletion of a table if there are dependencies from other tables.
- Provide rules for Oracle tools, such as Oracle Developer.

Data Integrity Constraints

Constraint	Description
NOT NULL	Specifies that the column cannot contain a null value
UNIQUE	Specifies a column or combination of columns whose values must be unique for all rows in the table
PRIMARY KEY	Uniquely identifies each row of the table
FOREIGN KEY	Establishes and enforces a referential integrity between the column and a column of the referenced table such that values in one table match values in another table.
CHECK	Specifies a condition that must be true

Constraint Guidelines

- You can name a constraint, or the Oracle server generates a name by using the `SYS_Cn` format.
- Create a constraint at either of the following times:
  – At the same time as the creation of the table
  – After the creation of the table
- Define a constraint at the column or table level.
- View a constraint in the data dictionary.

Constraint Guidelines

All constraints are stored in the data dictionary. Constraints are easy to reference if you give them a meaningful name. Constraint names must follow the standard object-naming rules, except that the name cannot be the same as another object owned by the same user. If you do not name your constraint, the Oracle server generates a name with the format `SYS_Cn`, where *n* is an integer so that the constraint name is unique.

Constraints can be defined at the time of table creation or after the creation of the table. You can define a constraint at the column or table level. Functionally, a table-level constraint is the same as a column-level constraint.

For more information, see the section on "Constraints" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Defining Constraints

- Syntax:

```
CREATE TABLE [schema.]table
     (column datatype [DEFAULT expr]
     [column_constraint],
     ...
     [table_constraint][,...]);
```

- Column-level constraint syntax:

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Table-level constraint syntax:

```
column,...
  [CONSTRAINT constraint_name] constraint_type
  (column, ...),
```

Defining Constraints

The slide gives the syntax for defining constraints when creating a table. You can create constraints at either the column level or table level. Constraints defined at the column level are included when the column is defined. Table-level constraints are defined at the end of the table definition and must refer to the column or columns on which the constraint pertains in a set of parentheses. It is mainly the syntax that differentiates the two; otherwise, functionally, a column-level constraint is the same as a table-level constraint.

NOT NULL constraints must be defined at the column level.

Constraints that apply to more than one column must be defined at the table level.

In the syntax:

schema   Is the same as the owner's name
table   Is the name of the table
DEFAULT expr   Specifies a default value to be used if a value is omitted in the
    INSERT statement
column   Is the name of the column
datatype   Is the column's data type and length
column_constraint Is an integrity constraint as part of the column definition
table_constraint   Is an integrity constraint as part of the table definition

Defining Constraints

- Example of a column-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6)
    CONSTRAINT emp_emp_id_pk PRIMARY KEY,
  first_name   VARCHAR2(20),
  ...);
```
1

- Example of a table-level constraint:

```
CREATE TABLE employees(
  employee_id  NUMBER(6),
  first_name   VARCHAR2(20),
  ...
  job_id       VARCHAR2(10) NOT NULL,
  CONSTRAINT emp_emp_id_pk
    PRIMARY KEY (EMPLOYEE_ID));
```
2

Defining Constraints (continued)

Constraints are usually created at the same time as the table. Constraints can be added to a table after its creation and also be temporarily disabled.

Both examples in the slide create a primary key constraint on the EMPLOYEE_ID column of the EMPLOYEES table.

1. The first example uses the column-level syntax to define the constraint.
2. The second example uses the table-level syntax to define the constraint.

More details about the primary key constraint are provided later in this lesson.

NOT NULL Constraint

Ensures that null values are not permitted for the column:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	24000	(null)	90	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	17000	(null)	90	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	17000	(null)	90	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	Hunold	9000	(null)	60	AHUNOLD	590.423.4567	03-JAN-90
104	Bruce	Ernst	6000	(null)	60	BERNST	590.423.4568	21-MAY-91
107	Diana	Lorentz	4200	(null)	60	DLORENTZ	590.423.5567	07-FEB-99
124	Kevin	Mourgos	5800	(null)	50	KMOURGOS	650.123.5234	16-NOV-99
141	Trenna	Rajs	3500	(null)	50	TRAJS	650.121.8009	17-OCT-95
142	Curtis	Davies	3100	(null)	50	CDAVIES	650.121.2994	29-JAN-97
143	Randall	Matos	2600	(null)	50	RMATOS	650.121.2874	15-MAR-98
144	Peter	Vargas	2500	(null)	50	PVARGAS	650.121.2004	09-JUL-98
149	Eleni	Zlotkey	10500	0.2	80	EZLOTKEY	011.44.1344.429018	29-JAN-00
174	Ellen	Abel	11000	0.3	80	EABEL	011.44.1644.429267	11-MAY-96
176	Jonathon	Taylor	8600	0.2	80	JTAYLOR	011.44.1644.429265	24-MAR-98
178	Kimberely	Grant	7000	0.15	(null)	KGRANT	011.44.1644.429263	24-MAY-99
200	Jennifer	Whalen	4400	(null)	10	JWHALEN	515.123.4444	17-SEP-87
201	Michael	Hartstein	13000	(null)	20	MHARTSTE	515.123.5555	17-FEB-96
202	Pat	Fay	6000	(null)	20	PFAY	603.123.6666	17-AUG-97
205	Shelley	Higgins	12000	(null)	110	SHIGGINS	515.123.8080	07-JUN-94
206	William	Gietz	8300	(null)	110	WGIETZ	515.123.8181	07-JUN-94

**NOT NULL constraint
(Primary Key enforces
NOT NULL constraint.)**

NOT NULL constraint

**Absence of NOT NULL constraint
(Any row can contain a null
value for this column.)**

ORACLE

Copyright © 2009, Oracle. All rights reserved.

NOT NULL Constraint

The NOT NULL constraint ensures that the column contains no null values. Columns without the NOT NULL constraint can contain null values by default. NOT NULL constraints must be defined at the column level. In the EMPLOYEES table, the EMPLOYEE_ID column inherits a NOT NULL constraint as it is defined as a primary key. Otherwise, the LAST_NAME, EMAIL, HIRE_DATE, and JOB_ID columns have the NOT NULL constraint enforced on them.

Note: Primary key constraint is discussed in detail later in this lesson.

Oracle University and NETEC, S.A. de C.V. use only.

Development Program (WDP) eKit materials are provided for WDP in-class use only. Copying eKit materials is strictly prohibited and is in violation of Oracle copyright. All WDP students must receive an eKit watermarked with their name and email. Contact OracleWDP_ww@oracle.com if you have not received your personalized eKit.

Oracle Database 11g: SQL Fundamentals I 10 - 20

UNIQUE Constraint

UNIQUE constraint

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	EMAIL
1	100	King	SKING
2	101	Kochhar	NKOCHHAR
3	102	De Haan	LDEHAAN
4	103	Hunold	AHUNOLD
5	104	Ernst	BERNST
6	107	Lorentz	DLORENTZ

...

INSERT INTO

| 208 | SMITH | JSMITH | ← **Allowed** |
| 209 | SMITH | JSMITH | ← **Not allowed: already exists** |

`UNIQUE` **Constraint**

A `UNIQUE` key integrity constraint requires that every value in a column or a set of columns (key) be unique—that is, no two rows of a table can have duplicate values in a specified column or a set of columns. The column (or set of columns) included in the definition of the `UNIQUE` key constraint is called the *unique key*. If the `UNIQUE` constraint comprises more than one column, that group of columns is called a *composite unique key*.

`UNIQUE` constraints enable the input of nulls unless you also define `NOT NULL` constraints for the same columns. In fact, any number of rows can include nulls for columns without the `NOT NULL` constraints because nulls are not considered equal to anything. A null in a column (or in all columns of a composite `UNIQUE` key) always satisfies a `UNIQUE` constraint.

Note: Because of the search mechanism for the `UNIQUE` constraints on more than one column, you cannot have identical values in the non-null columns of a partially null composite `UNIQUE` key constraint.

UNIQUE Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    CONSTRAINT emp_email_uk UNIQUE(email));
```

UNIQUE Constraint (continued)

UNIQUE constraints can be defined at the column level or table level. You define the constraint at the table level when you want to create a composite unique key. A composite key is defined when there is not a single attribute that can uniquely identify a row. In that case, you can have a unique key that is composed of two or more columns, the combined value of which is always unique and can identify rows.

The example in the slide applies the UNIQUE constraint to the EMAIL column of the EMPLOYEES table. The name of the constraint is EMP_EMAIL_UK.

Note: The Oracle server enforces the UNIQUE constraint by implicitly creating a unique index on the unique key column or columns.

PRIMARY KEY Constraint

PRIMARY KEY

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

Not allowed
(null value)

INSERT INTO

(null)	Public Accounting	124	2500
50	Finance	124	1500

Not allowed
(50 already exists)
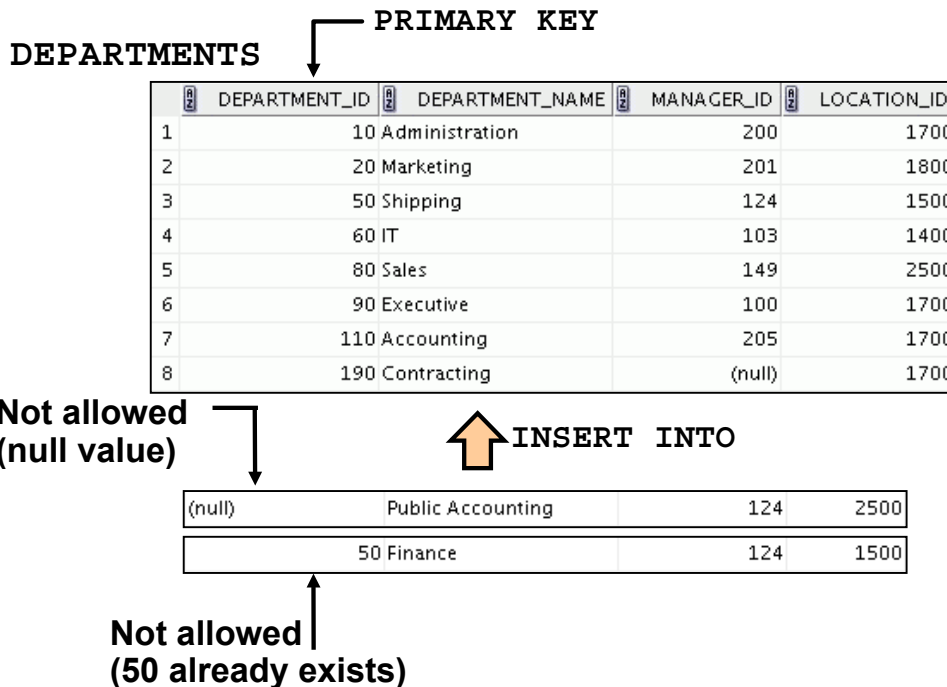
PRIMARY KEY Constraint

A PRIMARY KEY constraint creates a primary key for the table. Only one primary key can be created for each table. The PRIMARY KEY constraint is a column or a set of columns that uniquely identifies each row in a table. This constraint enforces the uniqueness of the column or column combination and ensures that no column that is part of the primary key can contain a null value.

Note: Because uniqueness is part of the primary key constraint definition, the Oracle server enforces the uniqueness by implicitly creating a unique index on the primary key column or columns.

FOREIGN KEY Constraint

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500

PRIMARY KEY

...

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
5	206	Gietz	110

FOREIGN KEY

...

INSERT INTO

200	Ford	9

Not allowed (9 does not exist)

200	Ford	60

Allowed
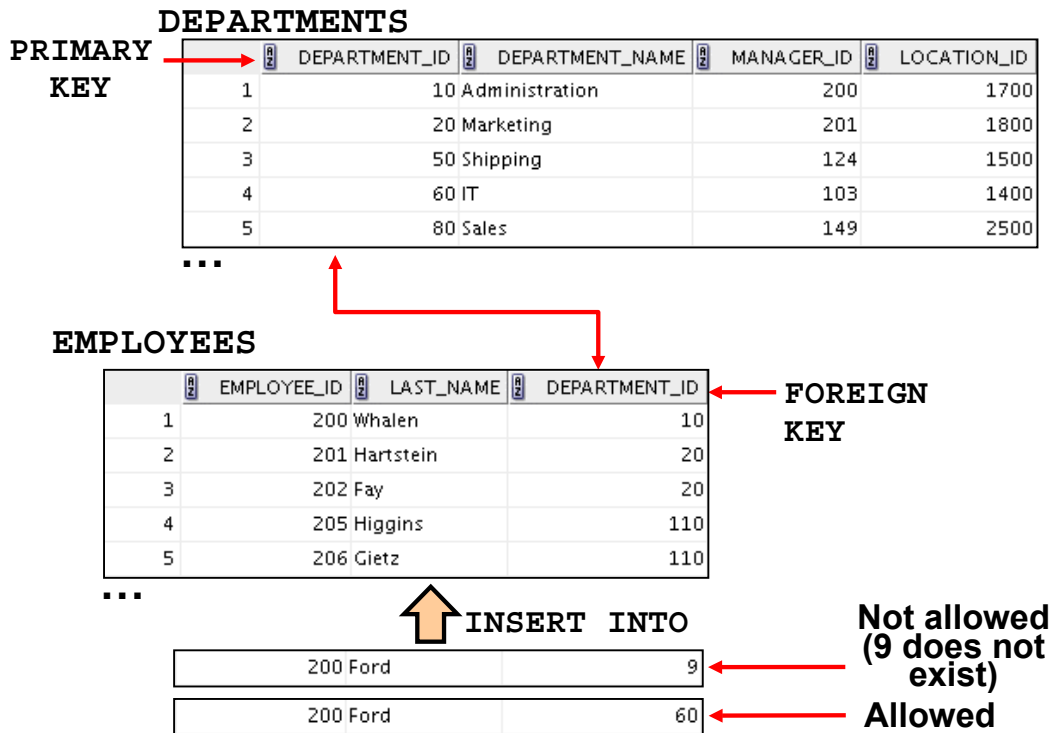
FOREIGN KEY Constraint

The FOREIGN KEY (or referential integrity) constraint designates a column or a combination of columns as a foreign key and establishes a relationship with a primary key or a unique key in the same table or a different table.

In the example in the slide, DEPARTMENT_ID has been defined as the foreign key in the EMPLOYEES table (dependent or child table); it references the DEPARTMENT_ID column of the DEPARTMENTS table (the referenced or parent table).

Guidelines
- A foreign key value must match an existing value in the parent table or be NULL.
- Foreign keys are based on data values and are purely logical, rather than physical, pointers.

FOREIGN KEY Constraint

Defined at either the table level or the column level:

```
CREATE TABLE employees(
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
...
    department_id    NUMBER(4),
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
      REFERENCES departments(department_id),
    CONSTRAINT emp_email_uk UNIQUE(email));
```

FOREIGN KEY Constraint (continued)

FOREIGN KEY constraints can be defined at the column or table constraint level. A composite foreign key must be created by using the table-level definition.

The example in the slide defines a FOREIGN KEY constraint on the DEPARTMENT_ID column of the EMPLOYEES table, using table-level syntax. The name of the constraint is EMP_DEPT_FK.

The foreign key can also be defined at the column level, provided that the constraint is based on a single column. The syntax differs in that the keywords FOREIGN KEY do not appear. For example:

```
CREATE TABLE employees
(...
department_id NUMBER(4) CONSTRAINT emp_deptid_fk
REFERENCES departments(department_id),
...
)
```

FOREIGN KEY Constraint: Keywords

- `FOREIGN KEY`: Defines the column in the child table at the table-constraint level
- `REFERENCES`: Identifies the table and column in the parent table
- `ON DELETE CASCADE`: Deletes the dependent rows in the child table when a row in the parent table is deleted
- `ON DELETE SET NULL`: Converts dependent foreign key values to null

FOREIGN KEY Constraint: Keywords

The foreign key is defined in the child table and the table containing the referenced column is the parent table. The foreign key is defined using a combination of the following keywords:

- `FOREIGN KEY` is used to define the column in the child table at the table-constraint level.
- `REFERENCES` identifies the table and the column in the parent table.
- `ON DELETE CASCADE` indicates that when a row in the parent table is deleted, the dependent rows in the child table are also deleted.
- `ON DELETE SET NULL` indicates that when a row in the parent table is deleted, the foreign key values are set to null.

The default behavior is called the *restrict rule,* which disallows the update or deletion of referenced data.

Without the `ON DELETE CASCADE` or the `ON DELETE SET NULL` options, the row in the parent table cannot be deleted if it is referenced in the child table.

CHECK Constraint

- Defines a condition that each row must satisfy
- The following expressions are not allowed:
  - References to `CURRVAL`, `NEXTVAL`, `LEVEL`, and `ROWNUM` pseudocolumns
  - Calls to `SYSDATE`, `UID`, `USER`, and `USERENV` functions
  - Queries that refer to other values in other rows

```
..., salary   NUMBER(2)
     CONSTRAINT emp_salary_min
           CHECK (salary > 0),...
```

CHECK Constraint

The `CHECK` constraint defines a condition that each row must satisfy. The condition can use the same constructs as the query conditions, with the following exceptions:

- References to the `CURRVAL`, `NEXTVAL`, `LEVEL`, and `ROWNUM` pseudocolumns
- Calls to `SYSDATE`, `UID`, `USER`, and `USERENV` functions
- Queries that refer to other values in other rows

A single column can have multiple `CHECK` constraints that refer to the column in its definition. There is no limit to the number of `CHECK` constraints that you can define on a column.

`CHECK` constraints can be defined at the column level or table level.

```
CREATE TABLE employees
   (...
    salary NUMBER(8,2) CONSTRAINT emp_salary_min
                       CHECK (salary > 0),
    ...
```

```
CREATE TABLE employees
    ( employee_id     NUMBER(6)
        CONSTRAINT      emp_employee_id   PRIMARY KEY
    , first_name      VARCHAR2(20)
    , last_name       VARCHAR2(25)
        CONSTRAINT      emp_last_name_nn  NOT NULL
    , email           VARCHAR2(25)
        CONSTRAINT      emp_email_nn      NOT NULL
        CONSTRAINT      emp_email_uk      UNIQUE
    , phone_number    VARCHAR2(20)
    , hire_date       DATE
        CONSTRAINT      emp_hire_date_nn  NOT NULL
    , job_id          VARCHAR2(10)
        CONSTRAINT      emp_job_nn        NOT NULL
    , salary          NUMBER(8,2)
        CONSTRAINT      emp_salary_ck     CHECK (salary>0)
    , commission_pct NUMBER(2,2)
    , manager_id      NUMBER(6)
        CONSTRAINT emp_manager_fk REFERENCES
          employees (employee_id)
    , department_id  NUMBER(4)
        CONSTRAINT      emp_dept_fk       REFERENCES
          departments (department_id));
```

ORACLE

CREATE TABLE: Example

The example in the slide shows the statement that is used to create the EMPLOYEES table in the HR schema.

Violating Constraints

```
UPDATE  employees
SET     department_id = 55
WHERE   department_id = 110;
```

```
Error starting at line 1 in command:
UPDATE employees
SET    department_id = 55
WHERE  department_id = 110
Error report:
SQL Error: ORA-02291: integrity constraint (ORA1.EMP_DEPT_FK) violated - parent key not found
02291. 00000 - "integrity constraint (%s.%s) violated - parent key not found"
*Cause:    A foreign key value has no matching primary key value.
```

Department 55 does not exist.

Violating Constraints

When you have constraints in place on columns, an error is returned if you try to violate the constraint rule. For example, if you try to update a record with a value that is tied to an integrity constraint, an error is returned.

In the example in the slide, department 55 does not exist in the parent table, DEPARTMENTS, and so you receive the "parent key not found" violation ORA-02291.

Violating Constraints

You cannot delete a row that contains a primary key that is used as a foreign key in another table.

```
DELETE FROM departments
WHERE department_id = 60;
```

```
Error starting at line 1 in command:
DELETE FROM departments
WHERE department_id = 60
Error report:
SQL Error: ORA-02292: integrity constraint (ORA1.JHIST_DEPT_FK) violated - child record found
02292. 00000 -  "integrity constraint (%s.%s) violated - child record found"
*Cause:    attempted to delete a parent key value that had a foreign
           dependency.
*Action:   delete dependencies first then parent or disable constraint.
```

ORACLE

Violating Constraints (continued)

If you attempt to delete a record with a value that is tied to an integrity constraint, an error is returned.

The example in the slide tries to delete department 60 from the DEPARTMENTS table, but it results in an error because that department number is used as a foreign key in the EMPLOYEES table. If the parent record that you attempt to delete has child records, you receive the "child record found" violation ORA-02292.

The following statement works because there are no employees in department 70:

```
DELETE FROM  departments
WHERE department_id = 70;
```

```
1 rows deleted
```

Lesson Agenda

- Database objects
  - Naming rules
- CREATE TABLE statement:
  - Access another user's tables
  - DEFAULT option
- Data types
- Overview of constraints: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK constraints
- **Creating a table using a subquery**
- ALTER TABLE
  - Read-only tables
- DROP TABLE statement

Oracle University and NETEC, S.A. de C.V. use only.

Creating a Table Using a Subquery

- Create a table and insert rows by combining the `CREATE TABLE` statement and the `AS subquery` option.

```
CREATE TABLE table
            [(column, column...)]
AS subquery;
```

- Match the number of specified columns to the number of subquery columns.
- Define columns with column names and default values.

Creating a Table Using a Subquery

A second method for creating a table is to apply the `AS subquery` clause, which both creates the table and inserts rows returned from the subquery.

In the syntax:

table	Is the name of the table
column	Is the name of the column, default value, and integrity constraint
subquery	Is the `SELECT` statement that defines the set of rows to be inserted into the new table

Guidelines

- The table is created with the specified column names, and the rows retrieved by the `SELECT` statement are inserted into the table.
- The column definition can contain only the column name and default value.
- If column specifications are given, the number of columns must equal the number of columns in the subquery `SELECT` list.
- If no column specifications are given, the column names of the table are the same as the column names in the subquery.
- The column data type definitions and the `NOT NULL` constraint are passed to the new table. Note that only the explicit `NOT NULL` constraint will be inherited. The `PRIMARY KEY` column will not pass the `NOT NULL` feature to the new column. Any other constraint rules are not passed to the new table. However, you can add constraints in the column definition.

Creating a Table Using a Subquery

```
CREATE TABLE  dept80
   AS
      SELECT   employee_id, last_name,
               salary*12 ANNSAL,
               hire_date
      FROM     employees
      WHERE    department id = 80;
```
CREATE TABLE succeeded.

```
DESCRIBE dept80
```

```
Name                            Null     Type
------------------------------- -------- -------------
EMPLOYEE_ID                              NUMBER(6)
LAST_NAME                       NOT NULL VARCHAR2(25)
ANNSAL                                   NUMBER
HIRE_DATE                       NOT NULL DATE
```

Creating a Table Using a Subquery (continued)

The example in the slide creates a table named DEPT80, which contains details of all the employees working in department 80. Notice that the data for the DEPT80 table comes from the EMPLOYEES table.

You can verify the existence of a database table and check the column definitions by using the DESCRIBE command.

However, be sure to provide a column alias when selecting an expression. The expression SALARY*12 is given the alias ANNSAL. Without the alias, the following error is generated:

```
Error starting at line 1 in command:
CREATE TABLE  dept80
  AS       SELECT  employee_id, last_name,
           salary*12 ,
           hire_date□    FROM     employees    WHERE    department_id = 80
Error at Command Line:3 Column:18
Error report:
SQL Error: ORA-00998: must name this expression with a column alias
00998. 00000 -   "must name this expression with a column alias"
*Cause:
*Action:
```

Lesson Agenda

- Database objects
    - Naming rules
- `CREATE TABLE` statement:
    - Access another user's tables
    - `DEFAULT` option
- Data types
- Overview of constraints: `NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`, `CHECK` constraints
- Creating a table using a subquery
- `ALTER TABLE`
    - Read-only tables
- `DROP TABLE` statement

ALTER TABLE Statement

Use the `ALTER TABLE` statement to:

- Add a new column
- Modify an existing column definition
- Define a default value for the new column
- Drop a column
- Rename a column
- Change table to read-only status

ALTER TABLE Statement

After you create a table, you may need to change the table structure for any of the following reasons:
- You omitted a column.
- Your column definition or its name needs to be changed.
- You need to remove columns.
- You want to put the table into the read-only mode

You can do this by using the `ALTER TABLE` statement.

Read-Only Tables

You can use the `ALTER TABLE` syntax to:

- Put a table into read-only mode, which prevents DDL or DML changes during table maintenance
- Put the table back into read/write mode

```
ALTER TABLE employees READ ONLY;

-- perform table maintenance and then
-- return table back to read/write mode

ALTER TABLE employees READ WRITE;
```

Read-Only Tables

With Oracle Database 11*g,* you can specify `READ ONLY` to place a table in the read-only mode. When the table is in the `READ-ONLY` mode, you cannot issue any DML statements that affect the table or any `SELECT ... FOR UPDATE` statements. You can issue DDL statements as long as they do not modify any data in the table. Operations on indexes associated with the table are allowed when the table is in the `READ ONLY` mode.

Specify `READ/WRITE` to return a read-only table to the read/write mode.

Note: You can drop a table that is in the `READ ONLY` mode. The `DROP` command is executed only in the data dictionary, so access to the table contents is not required. The space used by the table will not be reclaimed until the tablespace is made read/write again, and then the required changes can be made to the block segment headers, and so on.

For information about the `ALTER TABLE` statement, see the course titled *Oracle Database 10g SQL Fundamentals II*.

Lesson Agenda

- Database objects
  - Naming rules
- `CREATE TABLE` statement:
  - Access another user's tables
  - `DEFAULT` option
- Data types
- Overview of constraints: `NOT NULL`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`, `CHECK` constraints
- Creating a table using a subquery
- `ALTER TABLE`
  - Read-only tables
- `DROP TABLE` statement

ORACLE

Dropping a Table

- Moves a table to the recycle bin
- Removes the table and all its data entirely if the `PURGE` clause is specified
- Invalidates dependent objects and removes object privileges on the table

```
DROP TABLE dept80;
```
```
DROP TABLE dept80 succeeded.
```

Dropping a Table

The `DROP TABLE` statement moves a table to the recycle bin or removes the table and all its data from the database entirely. Unless you specify the `PURGE` clause, the `DROP TABLE` statement does not result in space being released back to the tablespace for use by other objects, and the space continues to count towards the user's space quota. Dropping a table invalidates the dependent objects and removes object privileges on the table.

When you drop a table, the database loses all the data in the table and all the indexes associated with it.

Syntax

```
DROP TABLE table [PURGE]
```

In the syntax, *table* is the name of the table.

Guidelines
- All the data is deleted from the table.
- Any views and synonyms remain, but are invalid.
- Any pending transactions are committed.
- Only the creator of the table or a user with the `DROP ANY TABLE` privilege can remove a table.

Note: Use the `FLASHBACK TABLE` statement to restore a dropped table from the recycle bin. This is discussed in detail in the course titled *Oracle Database 11g: SQL Fundamentals II*.

Quiz

You can use constraints to do the following:

1. Enforce rules on the data in a table whenever a row is inserted, updated, or deleted.
2. Prevent the deletion of a table.
3. Prevent the creation of a table.
4. Prevent the creation of data in a table.

Answers: 1, 2, 4

Summary

In this lesson, you should have learned how to use the CREATE TABLE statement to create a table and include constraints:

- Categorize the main database objects
- Review the table structure
- List the data types that are available for columns
- Create a simple table
- Explain how constraints are created at the time of table creation
- Describe how schema objects work

Summary

In this lesson, you should have learned how to do the following:

CREATE TABLE
- Use the CREATE TABLE statement to create a table and include constraints.
- Create a table based on another table by using a subquery.

DROP TABLE
- Remove rows and a table structure.
- When executed, this statement cannot be rolled back.

Practice 10: Overview

This practice covers the following topics:

- Creating new tables
- Creating a new table by using the `CREATE TABLE AS` syntax
- Verifying that tables exist
- Setting a table to read-only status
- Dropping tables

ORACLE

Practice 10: Overview

Create new tables by using the `CREATE TABLE` statement. Confirm that the new table was added to the database. You also learn to set the status of a table as `READ ONLY` and then revert to `READ/WRITE`.

Note: For all the `DDL` and `DML` statements, click the Run Script icon (or press [F5]) to execute the query in SQL Developer. This way you get to see the feedback messages on the Script Output tabbed page. For SELECT queries, continue to click the Execute Statement icon or press [F9] to get the formatted output on the Results tabbed page.

11

Creating Other Schema Objects

Objectives

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

- Create simple and complex views
- Retrieve data from views
- Create, maintain, and use sequences
- Create and maintain indexes
- Create private and public synonyms

Objectives

In this lesson, you are introduced to the view, sequence, synonym, and index objects. You learn the basics of creating and using views, sequences, and indexes.

Lesson Agenda

- Overview of views:
  - Creating, modifying, and retrieving data from a view
  - Data manipulation language (DML) operations on a view
  - Dropping a view
- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
  - Creating, dropping indexes
- Overview of synonyms
  - Creating, dropping synonyms

Database Objects

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of data retrieval queries
Synonym	Gives alternative names to objects

ORACLE

Database Objects

There are several other objects in a database in addition to tables.

With views, you can present and hide data from the tables.

Many applications require the use of unique numbers as primary key values. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.

If you want to improve the performance of data retrieval queries, you should consider creating an index. You can also use indexes to enforce uniqueness on a column or a collection of columns.

You can provide alternative names for objects by using synonyms.

What Is a View?

EMPLOYEES table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100 Steven		King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000
101 Neena		Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000
102 Lex		De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000
103 Alexander		Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000
104 Br						ROG	6000
10						ROG	4200
						ACCOUNT	6900
						MAN	5800
						CLERK	3500
						CLERK	3100
						CLERK	2600
						CLERK	2500
						SA_MAN	10500
						SA_REP	11000
					R-98	SA_REP	8600
					4-MAY-99	SA_REP	7000
					17-SEP-87	AD_ASST	4400
					17-FEB-96	MK_MAN	13000
				6666	17-AUG-97	MK_REP	6000
205 Shelley		Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000
206 William		Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY
100 Steven		King	24000
101 Neena		Kochhar	17000
102 Lex		De Haan	17000
103 Alexander		Hunold	9000
104 Bruce		Ernst	6000

What Is a View?

You can present logical subsets or combinations of data by creating views of tables. A view is a logical table based on a table or another view. A view contains no data of its own, but is like a window through which data from tables can be viewed or changed. The tables on which a view is based are called *base tables*. The view is stored as a SELECT statement in the data dictionary.

Advantages of Views

To restrict data access		To make complex queries easy

To provide data independence		To present different views of the same data
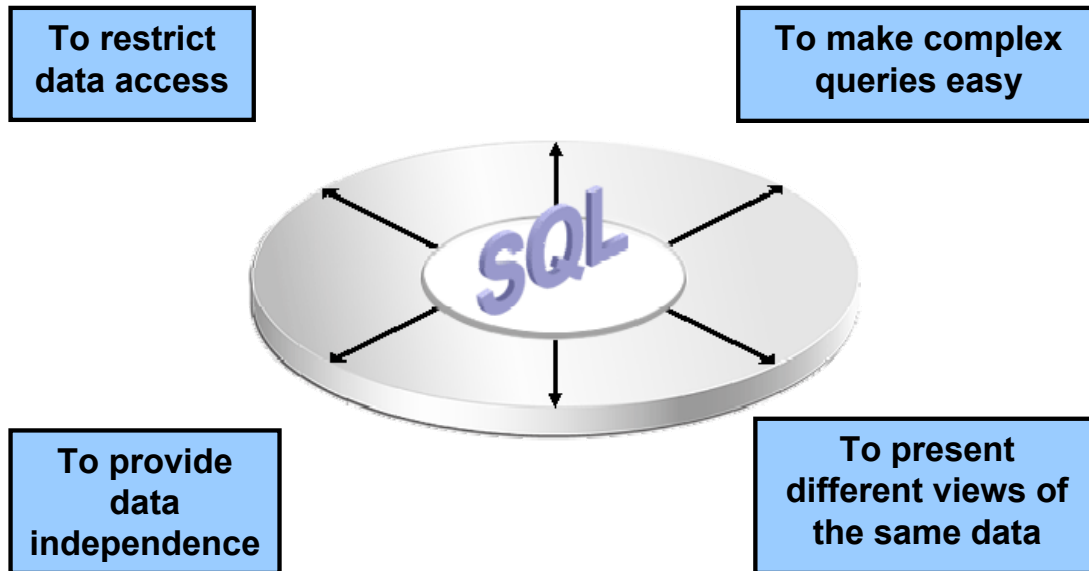
Advantages of Views

- Views restrict access to the data because it displays selected columns from the table.
- Views can be used to make simple queries to retrieve the results of complicated queries. For example, views can be used to query information from multiple tables without the user knowing how to write a join statement.
- Views provide data independence for ad hoc users and application programs. One view can be used to retrieve data from several tables.
- Views provide groups of users access to data according to their particular criteria.

For more information, see the section on "CREATE VIEW" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

Simple Views and Complex Views

There are two classifications for views: simple and complex. The basic difference is related to the DML (INSERT, UPDATE, and DELETE) operations.

- A simple view is one that:
  - Derives data from only one table
  - Contains no functions or groups of data
  - Can perform DML operations through the view
- A complex view is one that:
  - Derives data from many tables
  - Contains functions or groups of data
  - Does not always allow DML operations through the view

Creating a View

- You embed a subquery in the `CREATE VIEW` statement:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
 AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

- The subquery can contain complex `SELECT` syntax.

Creating a View

You can create a view by embedding a subquery in the `CREATE VIEW` statement.

In the syntax:

`OR REPLACE`	Re-creates the view if it already exists
`FORCE`	Creates the view regardless of whether or not the base tables exist
`NOFORCE`	Creates the view only if the base tables exist (This is the default.)
`view`	Is the name of the view
`alias`	Specifies names for the expressions selected by the view's query (The number of aliases must match the number of expressions selected by the view.)
`subquery`	Is a complete `SELECT` statement (You can use aliases for the columns in the `SELECT` list.)
`WITH CHECK OPTION`	Specifies that only those rows that are accessible to the view can be inserted or updated
`constraint`	Is the name assigned to the `CHECK OPTION` constraint
`WITH READ ONLY`	Ensures that no DML operations can be performed on this view

Note: In SQL Developer, click the Run Script icon or press [F5] to run the data definition language (DDL) statements. The feedback messages will be shown on the Script Output tabbed page.

Creating a View

- Create the `EMPVU80` view, which contains details of the employees in department 80:

```
CREATE VIEW   empvu80
 AS SELECT   employee_id, last_name, salary
    FROM     employees
    WHERE    department_id = 80;
```
```
CREATE VIEW succeeded.
```

- Describe the structure of the view by using the SQL*Plus `DESCRIBE` command:

```
DESCRIBE empvu80
```

ORACLE

Creating a View (continued)

The example in the slide creates a view that contains the employee number, last name, and salary for each employee in department 80.

You can display the structure of the view by using the `DESCRIBE` command.

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

Guidelines

- The subquery that defines a view can contain complex `SELECT` syntax, including joins, groups, and subqueries.
- If you do not specify a constraint name for the view created with the `WITH CHECK OPTION`, the system assigns a default name in the `SYS_Cn` format.
- You can use the `OR REPLACE` option to change the definition of the view without dropping and re-creating it, or regranting the object privileges previously granted on it.

Creating a View

- Create a view by using column aliases in the subquery:

```
CREATE VIEW   salvu50
 AS SELECT   employee_id ID_NUMBER, last_name NAME,
             salary*12 ANN_SALARY
    FROM     employees
    WHERE    department_id = 50;
CREATE VIEW succeeded.
```

- Select the columns from this view by the given alias names.

Oracle University and NETEC, S.A. de C.V. use only.

Creating a View (continued)

You can control the column names by including column aliases in the subquery.

The example in the slide creates a view containing the employee number (EMPLOYEE_ID) with the alias ID_NUMBER, name (LAST_NAME) with the alias NAME, and annual salary (SALARY) with the alias ANN_SALARY for every employee in department 50.

Alternatively, you can use an alias after the CREATE statement and before the SELECT subquery. The number of aliases listed must match the number of expressions selected in the subquery.

```
CREATE OR REPLACE VIEW salvu50 (ID_NUMBER, NAME, ANN_SALARY)
  AS SELECT   employee_id, last_name, salary*12
     FROM     employees
     WHERE    department_id = 50;

CREATE VIEW succeeded.
```

Retrieving Data from a View

```
SELECT *
FROM    salvu50;
```

	ID_NUMBER	NAME	ANN_SALARY
1	124	Mourgos	69600
2	141	Rajs	42000
3	142	Davies	37200
4	143	Matos	31200
5	144	Vargas	30000

Retrieving Data from a View

You can retrieve data from a view as you would from any table. You can display either the contents of the entire view or just specific rows and columns.

Modifying a View

- Modify the `EMPVU80` view by using a `CREATE OR REPLACE VIEW` clause. Add an alias for each column name:

```
CREATE OR REPLACE VIEW empvu80
  (id_number, name, sal, department_id)
AS SELECT   employee_id, first_name || ' '
            || last_name, salary, department_id
   FROM     employees
   WHERE    department_id = 80;
CREATE OR REPLACE VIEW succeeded.
```

- Column aliases in the `CREATE OR REPLACE VIEW` clause are listed in the same order as the columns in the subquery.

Oracle University and NETEC, S.A. de C.V. use only.

Modifying a View

With the `OR REPLACE` option, a view can be created even if one exists with this name already, thus replacing the old version of the view for its owner. This means that the view can be altered without dropping, re-creating, and regranting object privileges.

Note: When assigning column aliases in the `CREATE OR REPLACE VIEW` clause, remember that the aliases are listed in the same order as the columns in the subquery.

Creating a Complex View

Create a complex view that contains group functions to display values from two tables:

```
CREATE OR REPLACE VIEW dept_sum_vu
   (name, minsal, maxsal, avgsal)
AS SELECT    d.department_name, MIN(e.salary),
             MAX(e.salary),AVG(e.salary)
   FROM      employees e JOIN departments d
   ON        (e.department_id = d.department_id)
   GROUP BY d.department_name;
CREATE OR REPLACE VIEW succeeded.
```

Oracle University and NETEC, S.A. de C.V. use only.

Creating a Complex View

The example in the slide creates a complex view of department names, minimum salaries, maximum salaries, and the average salaries by department. Note that alternative names have been specified for the view. This is a requirement if any column of the view is derived from a function or an expression.

You can view the structure of the view by using the DESCRIBE command. Display the contents of the view by issuing a SELECT statement.

```
SELECT  *
FROM    dept_sum_vu;
```

	NAME	MINSAL	MAXSAL	AVGSAL
1	Administration	4400	4400	4400
2	Accounting	8300	12000	10150
3	IT	4200	9000	6400
4	Executive	17000	24000	19333.333333333...
5	Shipping	2500	5800	3500
6	Sales	8600	11000	10033.333333333...
7	Marketing	6000	13000	9500

Rules for Performing DML Operations on a View

- You can usually perform DML operations on simple views.
- You cannot remove a row if the view contains the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword

Rules for Performing DML Operations on a View

- You can perform DML operations on data through a view if those operations follow certain rules.
- You can remove a row from a view unless it contains any of the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword

Rules for Performing DML Operations on a View

You cannot modify data in a view if it contains:

- Group functions
- A GROUP BY clause
- The DISTINCT keyword
- The pseudocolumn ROWNUM keyword
- Columns defined by expressions

Rules for Performing DML Operations on a View (continued)

You can modify data through a view unless it contains any of the conditions mentioned in the previous slide or columns defined by expressions (for example, SALARY * 12).

Rules for Performing DML Operations on a View

You cannot add data through a view if the view includes:

- Group functions
- A `GROUP BY` clause
- The `DISTINCT` keyword
- The pseudocolumn `ROWNUM` keyword
- Columns defined by expressions
- `NOT NULL` columns in the base tables that are not selected by the view

Rules for Performing DML Operations on a View (continued)

You can add data through a view unless it contains any of the items listed in the slide. You cannot add data to a view if the view contains `NOT NULL` columns without default values in the base table. All the required values must be present in the view. Remember that you are adding values directly to the underlying table *through* the view.

For more information, see the section on "CREATE VIEW" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Using the WITH CHECK OPTION Clause

- You can ensure that DML operations performed on the view stay in the domain of the view by using the WITH CHECK OPTION clause:

```
CREATE OR REPLACE VIEW empvu20
AS SELECT      *
   FROM       employees
   WHERE      department_id = 20
   WITH CHECK OPTION CONSTRAINT empvu20_ck ;
```
```
CREATE OR REPLACE VIEW succeeded.
```

- Any attempt to INSERT a row with a department_id other than 20, or to UPDATE the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint.

Using the WITH CHECK OPTION Clause

It is possible to perform referential integrity checks through views. You can also enforce constraints at the database level. The view can be used to protect data integrity, but the use is very limited.

The WITH CHECK OPTION clause specifies that INSERTs and UPDATEs performed through the view cannot create rows that the view cannot select. Therefore, it enables integrity constraints and data validation checks to be enforced on data being inserted or updated. If there is an attempt to perform DML operations on rows that the view has not selected, an error is displayed, along with the constraint name if that has been specified.

```
UPDATE empvu20
SET    department_id = 10
WHERE  employee_id = 201;
```

causes:

```
Error report:
SQL Error: ORA-01402: view WITH CHECK OPTION where-clause violation
01402. 00000 -  "view WITH CHECK OPTION where-clause violation"
```

Note: No rows are updated because, if the department number were to change to 10, the view would no longer be able to see that employee. With the WITH CHECK OPTION clause, therefore, the view can see only the employees in department 20 and does not allow the department number for those employees to be changed through the view.

Denying DML Operations

- You can ensure that no DML operations occur by adding the `WITH READ ONLY` option to your view definition.
- Any attempt to perform a DML operation on any row in the view results in an Oracle server error.

Denying DML Operations

You can ensure that no DML operations occur on your view by creating it with the `WITH READ ONLY` option. The example in the next slide modifies the `EMPVU10` view to prevent any DML operations on the view.

Denying DML Operations

```
CREATE OR REPLACE VIEW empvu10
     (employee_number, employee_name, job_title)
AS SELECT    employee_id, last_name, job_id
   FROM      employees
   WHERE     department_id = 10
   WITH READ ONLY ;
CREATE OR REPLACE VIEW succeeded.
```

Denying DML Operations (continued)

Any attempt to remove a row from a view with a read-only constraint results in an error:

```
DELETE FROM empvu10
WHERE  employee_number = 200;
```

Similarly, any attempt to insert a row or modify a row using the view with a read-only constraint results in the same error.

```
Error report:
SQL Error: ORA-42399: cannot perform a DML operation on a read-only view
```

Removing a View

You can remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;
```
DROP VIEW empvu80 succeeded.

Removing a View

You use the DROP VIEW statement to remove a view. The statement removes the view definition from the database. However, dropping views has no effect on the tables on which the view was based. Alternatively, views or other applications based on the deleted views become invalid. Only the creator or a user with the DROP ANY VIEW privilege can remove a view.

In the syntax, *view* is the name of the view.

Practice 11: Overview of Part 1

This practice covers the following topics:
- Creating a simple view
- Creating a complex view
- Creating a view with a check constraint
- Attempting to modify data in the view
- Removing views

Practice 11: Overview of Part 1

Part 1 of this lesson's practice provides you with a variety of exercises in creating, using, and removing views. Complete questions 1–6 at the end of this lesson.

Lesson Agenda

- Overview of views:
  - Creating, modifying, and retrieving data from a view
  - DML operations on a view
  - Dropping a view
- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - `NEXTVAL` and `CURRVAL` pseudocolumns
- Overview of indexes
  - Creating, dropping indexes
- Overview of synonyms
  - Creating, dropping synonyms

Sequences

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Sequences

A sequence is a database object that creates integer values. You can create sequences and then use them to generate numbers.

Sequences

A sequence:

- Can automatically generate unique numbers
- Is a shareable object
- Can be used to create a primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cached in memory
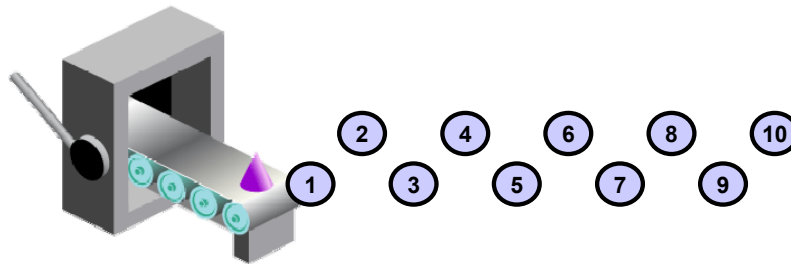
ORACLE

Sequences (continued)

A sequence is a user-created database object that can be shared by multiple users to generate integers.

You can define a sequence to generate unique values or to recycle and use the same numbers again.

A typical usage for sequences is to create a primary key value, which must be unique for each row. A sequence is generated and incremented (or decremented) by an internal Oracle routine. This can be a time-saving object because it can reduce the amount of application code needed to write a sequence-generating routine.

Sequence numbers are stored and generated independent of tables. Therefore, the same sequence can be used for multiple tables.

CREATE SEQUENCE Statement: Syntax

Define a sequence to generate sequential numbers automatically:

```
CREATE SEQUENCE sequence
       [INCREMENT BY n]
       [START WITH n]
       [{MAXVALUE n | NOMAXVALUE}]
       [{MINVALUE n | NOMINVALUE}]
       [{CYCLE | NOCYCLE}]
       [{CACHE n | NOCACHE}];
```

CREATE SEQUENCE Statement: Syntax

Automatically generate sequential numbers by using the CREATE SEQUENCE statement.

In the syntax:

sequence	Is the name of the sequence generator
INCREMENT BY n	Specifies the interval between sequence numbers, where n is an integer (If this clause is omitted, the sequence increments by 1.)
START WITH n	Specifies the first sequence number to be generated (If this clause is omitted, the sequence starts with 1.)
MAXVALUE n	Specifies the maximum value the sequence can generate
NOMAXVALUE	Specifies a maximum value of 10^{27} for an ascending sequence and –1 for a descending sequence (This is the default option.)
MINVALUE n	Specifies the minimum sequence value
NOMINVALUE	Specifies a minimum value of 1 for an ascending sequence and $-(10^{26})$ for a descending sequence (This is the default option.)

Creating a Sequence

- Create a sequence named `DEPT_DEPTID_SEQ` to be used for the primary key of the `DEPARTMENTS` table.
- Do not use the `CYCLE` option.

```
CREATE SEQUENCE dept_deptid_seq
                INCREMENT BY 10
                START WITH 120
                MAXVALUE 9999
                NOCACHE
                NOCYCLE;
```
```
CREATE SEQUENCE succeeded.
```

Creating a Sequence (continued)

`CYCLE  │  NOCYCLE`	Specifies whether the sequence continues to generate values after reaching its maximum or minimum value (`NOCYCLE` is the default option.)
`CACHE n  │  NOCACHE`	Specifies how many values the Oracle server preallocates and keeps in memory (By default, the Oracle server caches 20 values.)

The example in the slide creates a sequence named `DEPT_DEPTID_SEQ` to be used for the `DEPARTMENT_ID` column of the `DEPARTMENTS` table. The sequence starts at 120, does not allow caching, and does not cycle.

Do not use the `CYCLE` option if the sequence is used to generate primary key values, unless you have a reliable mechanism that purges old rows faster than the sequence cycles.

For more information, see the section on "`CREATE SEQUENCE`" in the *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Note: The sequence is not tied to a table. Generally, you should name the sequence after its intended use. However, the sequence can be used anywhere, regardless of its name.

NEXTVAL and CURRVAL Pseudocolumns

- NEXTVAL returns the next available sequence value. It returns a unique value every time it is referenced, even for different users.
- CURRVAL obtains the current sequence value.
- NEXTVAL must be issued for that sequence before CURRVAL contains a value.

NEXTVAL and CURRVAL Pseudocolumns

After you create your sequence, it generates sequential numbers for use in your tables. Reference the sequence values by using the NEXTVAL and CURRVAL pseudocolumns.

The NEXTVAL pseudocolumn is used to extract successive sequence numbers from a specified sequence. You must qualify NEXTVAL with the sequence name. When you reference *sequence*.NEXTVAL, a new sequence number is generated and the current sequence number is placed in CURRVAL.

The CURRVAL pseudocolumn is used to refer to a sequence number that the current user has just generated. However, NEXTVAL must be used to generate a sequence number in the current user's session before CURRVAL can be referenced. You must qualify CURRVAL with the sequence name. When you reference *sequence*.CURRVAL, the last value returned to that user's process is displayed.

NEXTVAL and CURRVAL Pseudocolumns (continued)

Rules for Using NEXTVAL and CURRVAL

You can use NEXTVAL and CURRVAL in the following contexts:
- The SELECT list of a SELECT statement that is not part of a subquery
- The SELECT list of a subquery in an INSERT statement
- The VALUES clause of an INSERT statement
- The SET clause of an UPDATE statement

You cannot use NEXTVAL and CURRVAL in the following contexts:
- The SELECT list of a view
- A SELECT statement with the DISTINCT keyword
- A SELECT statement with GROUP BY, HAVING, or ORDER BY clauses
- A subquery in a SELECT, DELETE, or UPDATE statement
- The DEFAULT expression in a CREATE TABLE or ALTER TABLE statement

For more information, see the sections on "Pseudocolumns" and "CREATE SEQUENCE" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1).*

Using a Sequence

- Insert a new department named "Support" in location ID 2500:

```
INSERT INTO departments(department_id,
            department_name, location_id)
VALUES      (dept_deptid_seq.NEXTVAL,
            'Support', 2500);
```
```
1 rows inserted
```

- View the current value for the DEPT_DEPTID_SEQ sequence:

```
SELECT    dept_deptid_seq.CURRVAL
FROM      dual;
```

Using a Sequence

The example in the slide inserts a new department in the DEPARTMENTS table. It uses the DEPT_DEPTID_SEQ sequence to generate a new department number as follows.

You can view the current value of the sequence using the *sequence_name*.CURRVAL, as shown in the second example in the slide. The output of the query is shown below:

	CURRVAL
1	120

Suppose that you now want to hire employees to staff the new department. The INSERT statement to be executed for all new employees can include the following code:

```
INSERT INTO employees (employee_id, department_id, ...)
VALUES (employees_seq.NEXTVAL, dept_deptid_seq .CURRVAL, ...);
```

Note: The preceding example assumes that a sequence called EMPLOYEE_SEQ has already been created to generate new employee numbers.

Caching Sequence Values

- Caching sequence values in memory gives faster access to those values.
- Gaps in sequence values can occur when:
  - A rollback occurs
  - The system crashes
  - A sequence is used in another table

Caching Sequence Values

You can cache sequences in memory to provide faster access to those sequence values. The cache is populated the first time you refer to the sequence. Each request for the next sequence value is retrieved from the cached sequence. After the last sequence value is used, the next request for the sequence pulls another cache of sequences into memory.

Gaps in the Sequence

Although sequence generators issue sequential numbers without gaps, this action occurs independent of a commit or rollback. Therefore, if you roll back a statement containing a sequence, the number is lost.

Another event that can cause gaps in the sequence is a system crash. If the sequence caches values in memory, those values are lost if the system crashes.

Because sequences are not tied directly to tables, the same sequence can be used for multiple tables. However, if you do so, each table can contain gaps in the sequential numbers.

Modifying a Sequence

Change the increment value, maximum value, minimum value, cycle option, or cache option:

```
ALTER SEQUENCE dept_deptid_seq
                INCREMENT BY 20
                MAXVALUE 999999
                NOCACHE
                NOCYCLE;
```
ALTER SEQUENCE dept_deptid_seq succeeded.

Modifying a Sequence

If you reach the MAXVALUE limit for your sequence, no additional values from the sequence are allocated and you will receive an error indicating that the sequence exceeds the MAXVALUE. To continue to use the sequence, you can modify it by using the ALTER SEQUENCE statement.

Syntax
```
ALTER  SEQUENCE    sequence
      [INCREMENT BY n]
      [{MAXVALUE n | NOMAXVALUE}]
      [{MINVALUE n | NOMINVALUE}]
      [{CYCLE | NOCYCLE}]
      [{CACHE n | NOCACHE}];
```
In the syntax, *sequence* is the name of the sequence generator.

For more information, see the section on "ALTER SEQUENCE" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Guidelines for Modifying a Sequence

- You must be the owner or have the `ALTER` privilege for the sequence.
- Only future sequence numbers are affected.
- The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed.
- To remove a sequence, use the `DROP` statement:

```
DROP SEQUENCE dept_deptid_seq;
```
```
DROP SEQUENCE dept_deptid_seq succeeded.
```

Guidelines for Modifying a Sequence

- You must be the owner or have the `ALTER` privilege for the sequence to modify it. You must be the owner or have the `DROP ANY SEQUENCE` privilege to remove it.
- Only future sequence numbers are affected by the `ALTER SEQUENCE` statement.
- The `START WITH` option cannot be changed using `ALTER SEQUENCE`. The sequence must be dropped and re-created to restart the sequence at a different number.
- Some validation is performed. For example, a new `MAXVALUE` that is less than the current sequence number cannot be imposed.

```
ALTER SEQUENCE dept_deptid_seq
      INCREMENT BY 20
      MAXVALUE 90
      NOCACHE
      NOCYCLE;
```

- The error:

```
Error report:
SQL Error: ORA-04009: MAXVALUE cannot be made to be less than the current value
04009. 00000 -  "MAXVALUE cannot be made to be less than the current value"
*Cause:    the current value exceeds the given MAXVALUE
*Action:   make sure that the new MAXVALUE is larger than the current value
```

Lesson Agenda

- Overview of views:
  - Creating, modifying, and retrieving data from a view
  - DML operations on a view
  - Dropping a view
- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
- **Overview of indexes**
  - Creating, dropping indexes
- Overview of synonyms
  - Creating, dropping synonyms

ORACLE

Indexes

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Indexes

Indexes are database objects that you can create to improve the performance of some queries. Indexes can also be created automatically by the server when you create a primary key or a unique constraint.

Indexes

An index:

- Is a schema object
- Can be used by the Oracle server to speed up the retrieval of rows by using a pointer
- Can reduce disk input/output (I/O) by using a rapid path access method to locate data quickly
- Is independent of the table that it indexes
- Is used and maintained automatically by the Oracle server
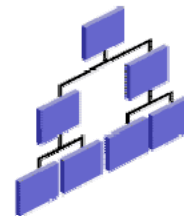
ORACLE

Indexes (continued)

An Oracle server index is a schema object that can speed up the retrieval of rows by using a pointer. Indexes can be created explicitly or automatically. If you do not have an index on the column, a full table scan occurs.

An index provides direct and fast access to rows in a table. Its purpose is to reduce the disk I/O by using an indexed path to locate data quickly. An index is used and maintained automatically by the Oracle server. After an index is created, no direct activity is required by the user.

Indexes are logically and physically independent of the table that they index. This means that they can be created or dropped at any time, and have no effect on the base tables or other indexes.

Note: When you drop a table, the corresponding indexes are also dropped.

For more information, see the section on "Schema Objects: Indexes" in *Oracle Database Concepts 11g, Release 1 (11.1)*.

How Are Indexes Created?

- Automatically: A unique index is created automatically when you define a `PRIMARY KEY` or `UNIQUE` constraint in a table definition.

- Manually: Users can create nonunique indexes on columns to speed up access to the rows.

How Are Indexes Created?

You can create two types of indexes.
- **Unique index:** The Oracle server automatically creates this index when you define a column in a table to have a `PRIMARY KEY` or a `UNIQUE` constraint. The name of the index is the name that is given to the constraint.
- **Nonunique index:** This is an index that a user can create. For example, you can create the `FOREIGN KEY` column index for a join in a query to improve the speed of retrieval.

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

Creating an Index

- Create an index on one or more columns:

```
CREATE [UNIQUE][BITMAP]INDEX index
ON table (column[, column]...);
```

- Improve the speed of query access to the LAST_NAME column in the EMPLOYEES table:

```
CREATE INDEX  emp_last_name_idx
ON            employees(last_name);
```
```
CREATE INDEX succeeded.
```

ORACLE

Creating an Index

Create an index on one or more columns by issuing the CREATE INDEX statement.

In the syntax:
- index   Is the name of the index
- table   Is the name of the table
- column   Is the name of the column in the table to be indexed

Specify UNIQUE to indicate that the value of the column (or columns) upon which the index is based must be unique. Specify BITMAP to indicate that the index is to be created with a bitmap for each distinct key, rather than indexing each row separately. Bitmap indexes store the rowids associated with a key value as a bitmap.

For more information, see the section on "CREATE INDEX" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Index Creation Guidelines

Create an index when:	
✓	A column contains a wide range of values
✓	A column contains a large number of null values
✓	One or more columns are frequently used together in a `WHERE` clause or a join condition
✓	The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table

Do not create an index when:	
✗	The columns are not often used as a condition in the query
✗	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
✗	The table is updated frequently
✗	The indexed columns are referenced as part of an expression

ORACLE

Index Creation Guidelines

More Is Not Always Better

Having more indexes on a table does not produce faster queries. Each DML operation that is committed on a table with indexes means that the indexes must be updated. The more indexes that you have associated with a table, the more effort the Oracle server must make to update all the indexes after a DML operation.

When to Create an Index

Therefore, you should create indexes only if:
- The column contains a wide range of values
- The column contains a large number of null values
- One or more columns are frequently used together in a `WHERE` clause or join condition
- The table is large and most queries are expected to retrieve less than 2% to 4% of the rows

Remember that if you want to enforce uniqueness, you should define a unique constraint in the table definition. A unique index is then created automatically.

Removing an Index

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

```
DROP INDEX index;
```

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

```
DROP INDEX emp_last_name_idx;
```
```
DROP INDEX emp_last_name_idx succeeded.
```

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

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.

Lesson Agenda

- Overview of views:
  - Creating, modifying, and retrieving data from a view
  - DML operations on a view
  - Dropping a view
- Overview of sequences:
  - Creating, using, and modifying a sequence
  - Cache sequence values
  - NEXTVAL and CURRVAL pseudocolumns
- Overview of indexes
  - Creating, dropping indexes
- **Overview of synonyms**
  - Creating, dropping synonyms

Synonyms

Object	Description
Table	Basic unit of storage; composed of rows
View	Logically represents subsets of data from one or more tables
Sequence	Generates numeric values
Index	Improves the performance of some queries
Synonym	Gives alternative names to objects

ORACLE

Synonyms

Synonyms are database objects that enable you to call a table by another name. You can create synonyms to give an alternative name to a table.

Creating a Synonym for an Object

Simplify access to objects by creating a synonym (another name for an object). With synonyms, you can:

- Create an easier reference to a table that is owned by another user
- Shorten lengthy object names

```
CREATE [PUBLIC] SYNONYM synonym
FOR    object;
```

Creating a Synonym for an Object

To refer to a table that is owned by another user, you need to prefix the table name with the name of the user who created it, followed by a period. Creating a synonym eliminates the need to qualify the object name with the schema and provides you with an alternative name for a table, view, sequence, procedure, or other objects. This method can be especially useful with lengthy object names, such as views.

In the syntax:

PUBLIC	Creates a synonym that is accessible to all users
synonym	Is the name of the synonym to be created
object	Identifies the object for which the synonym is created

Guidelines
- The object cannot be contained in a package.
- A private synonym name must be distinct from all other objects that are owned by the same user.

For more information, see the section on "CREATE SYNONYM" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Creating and Removing Synonyms

- Create a shortened name for the `DEPT_SUM_VU` view:

```
CREATE SYNONYM  d_sum
FOR  dept_sum_vu;
```
```
CREATE SYNONYM succeeded.
```

- Drop a synonym:

```
DROP SYNONYM d_sum;
```
```
DROP SYNONYM d_sum succeeded.
```

Creating and Removing Synonyms

Creating a Synonym

The slide example creates a synonym for the `DEPT_SUM_VU` view for quicker reference.

The database administrator can create a public synonym that is accessible to all users. The following example creates a public synonym named `DEPT` for Alice's `DEPARTMENTS` table:

```
CREATE PUBLIC SYNONYM  dept
```
```
CREATE SYNONYM succeeded.
```

Removing a Synonym

To remove a synonym, use the `DROP SYNONYM` statement. Only the database administrator can drop a public synonym.

```
DROP PUBLIC SYNONYM  dept;
```

For more information, see the section on "`DROP SYNONYM`" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Quiz

Indexes must be created manually and serve to speed up access to rows in a table.

1. True
2. False

Oracle University and NETEC, S.A. de C.V. use only.

Answer: 2

Note: Indexes are designed to speed up query performance. However, not all indexes are created manually. The Oracle server automatically creates an index when you define a column in a table to have a PRIMARY KEY or a UNIQUE constraint.

Summary

In this lesson, you should have learned how to:

- Create, use, and remove views
- Automatically generate sequence numbers by using a sequence generator
- Create indexes to improve speed of query retrieval
- Use synonyms to provide alternative names for objects

Summary

In this lesson, you should have learned about database objects such as views, sequences, indexes, and synonyms.

Practice 11: Overview of Part 2

This practice covers the following topics:
- Creating sequences
- Using sequences
- Creating nonunique indexes
- Creating synonyms

Practice 11: Overview of Part 2

Part 2 of this lesson's practice provides you with a variety of exercises in creating and using a sequence, an index, and a synonym.

Complete questions 7–10 at the end of this lesson.

Appendix A
Practices and Solutions

Table of Contents

In this practice, you perform the following:

- Start Oracle SQL Developer and create a new connection to the `ora1` account.
- Use Oracle SQL Developer to examine data objects in the `ora1` account. The `ora1` account contains the `HR` schema tables.

Note the following location for the lab files:
```
\home\oracle\labs\sql1\labs
```
If you are asked to save any lab files, save them in this location.

In any practice, there may be exercises that are prefaced with the phrases "If you have time" or "If you want an extra challenge." Work on these exercises only if you have completed all other exercises within the allocated time and would like a further challenge to your skills.

Perform the practices slowly and precisely. You can experiment with saving and running command files. If you have any questions at any time, ask your instructor.

Note

1) All written practices use Oracle SQL Developer as the development environment. Although it is recommended that you use Oracle SQL Developer, you can also use SQL*Plus that is available in this course.

2) For any query, the sequence of rows retrieved from the database may differ from the screenshots shown.

Practice I-1: Introduction

This is the first of many practices in this course. The solutions (if you require them) can be found at the end of this practice. Practices are intended to cover most of the topics that are presented in the corresponding lesson.

Starting Oracle SQL Developer

1) Start Oracle SQL Developer using the SQL Developer desktop icon.

Creating a New Oracle SQL Developer Database Connection

2) To create a new database connection, in the Connections Navigator, right-click Connections. Select New Connection from the menu. The New/Select Database Connection dialog box appears.

3) Create a database connection using the following information:

   a) Connection Name: `myconnection`

   b) Username: `ora1`

   c) Password: `ora1`

   d) Hostname: `localhost`

   e) Port: `1521`

   f) SID: `ORCL`

Ensure that you select the Save Password check box.

Testing and Connecting Using the Oracle SQL Developer Database Connection

4) Test the new connection.

5) If the status is Success, connect to the database using this new connection.

Browsing the Tables in the Connections Navigator

6) In the Connections Navigator, view the objects available to you in the Tables node. Verify that the following tables are present:

```
COUNTRIES
DEPARTMENTS
EMPLOYEES
JOB_GRADES
JOB_HISTORY
JOBS
LOCATIONS
REGIONS
```

7) Browse the structure of the `EMPLOYEES` table.

8) View the data of the `DEPARTMENTS` table.

Practice Solutions I-1: Introduction

Starting Oracle SQL Developer

1) Start Oracle SQL Developer using the SQL Developer desktop icon.

   a) Double-click the SQL Developer desktop icon.

The SQL Developer Interface appears.
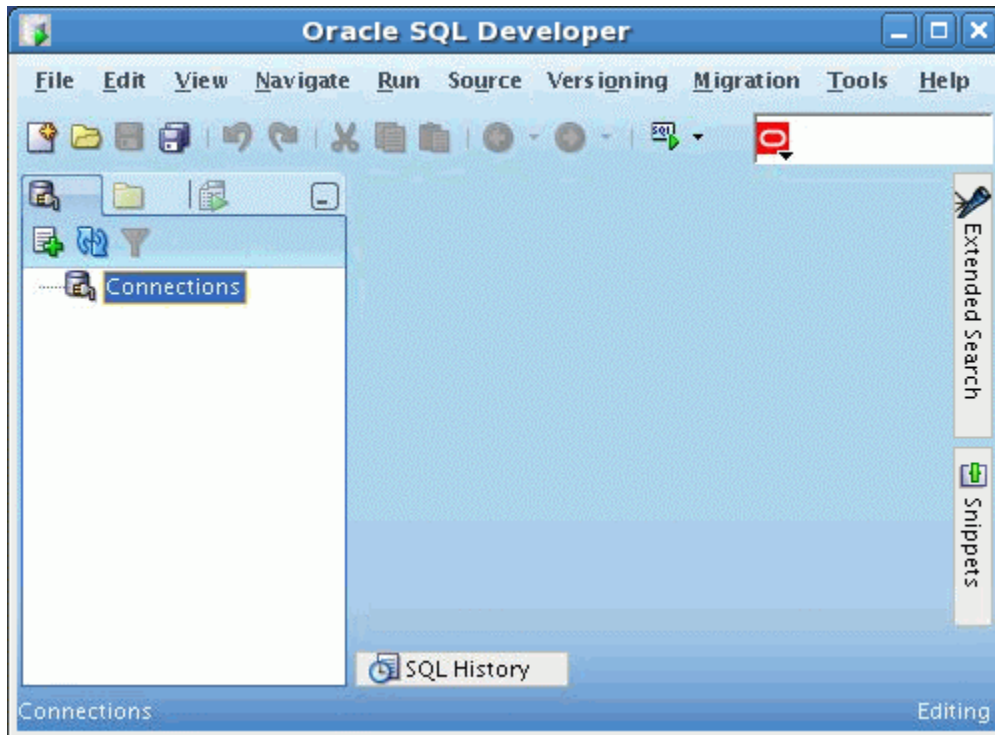
Creating a New Oracle SQL Developer Database Connection

2) To create a new database connection, in the Connections Navigator, right-click Connections and select New Connection from the menu.
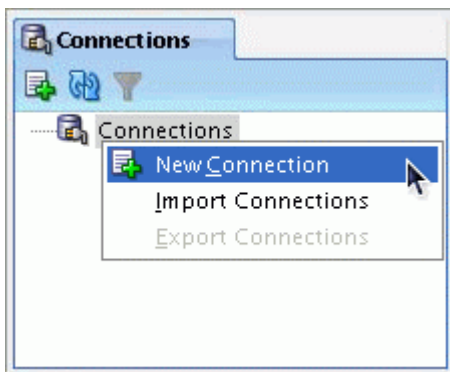
Practice Solutions I-1: Introduction (continued)

The New / Select Database Connection dialog box appears.

3) Create a database connection using the following information:

   a) Connection Name: `myconnection`

   b) Username: `ora1`

   c) Password: `ora1`

   d) Hostname: `localhost`

   e) Port: `1521`

   f) SID: `ORCL`

Ensure that you select the Save Password check box.

Practice Solutions I-1: Introduction (continued)

Testing and Connecting Using the Oracle SQL Developer Database Connection

4) Test the new connection.

5) If the status is Success, connect to the database using this new connection.

Practice Solutions I-1: Introduction (continued)
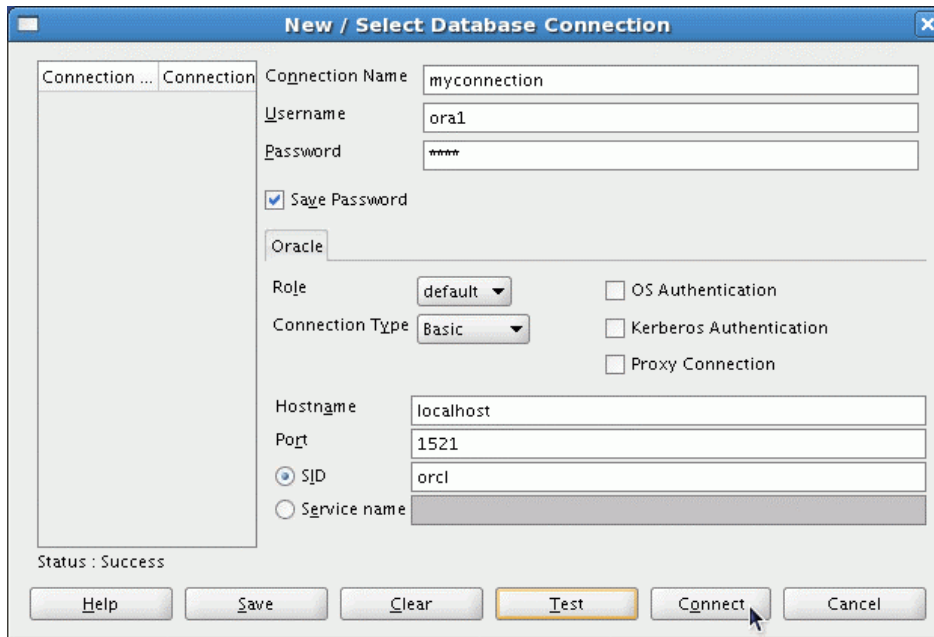
When you create a connection, a SQL Worksheet for that connection opens automatically.

Browsing the Tables in the Connections Navigator

6) In the Connections Navigator, view the objects available to you in the Tables node. Verify that the following tables are present:

```
COUNTRIES
DEPARTMENTS
EMPLOYEES
JOB_GRADES
JOB_HISTORY
JOBS
LOCATIONS
REGIONS
```

7) Browse the structure of the EMPLOYEES table.

8) View the data of the DEPARTMENTS table.

Practice Solutions I-1: Introduction (continued)

In this practice, you write simple SELECT queries. The queries cover most of the SELECT clauses and operations that you learned in this lesson.

Practice 1-1: Retrieving Data Using the SQL SELECT Statement
Part 1

Test your knowledge:

1) The following SELECT statement executes successfully:

```
SELECT last_name, job_id, salary AS Sal
FROM   employees;
```

   True/False

2) The following SELECT statement executes successfully:

```
SELECT *
FROM   job_grades;
```

   True/False

3) There are four coding errors in the following statement. Can you identify them?

```
SELECT     employee_id, last_name
sal x 12   ANNUAL SALARY
FROM       employees;
```

Part 2

Note the following points before you begin with the practices:

- Save all your lab files at the following location:
  /home/oracle/labs/sql1/labs

- Enter your SQL statements in a SQL Worksheet. To save a script in SQL
  Developer, make sure that the required SQL worksheet is active and then from the
  File menu, select Save As to save your SQL statement as a
  lab_<lessonno>_<stepno>.sql script. When you are modifying an existing
  script, make sure that you use Save As to save it with a different file name.

- To run the query, click the Execute Statement icon in the SQL Worksheet.
  Alternatively, you can press [F9]. For DML and DDL statements, use the Run
  Script icon or press [F5].

- After you have executed the query, make sure that you do not enter your next
  query in the same worksheet. Open a new worksheet.

You have been hired as a SQL programmer for Acme Corporation. Your first task is to
create some reports based on data from the Human Resources tables.

Practice 1-1: Retrieving Data Using the SQL SELECT Statement (continued)

4) Your first task is to determine the structure of the DEPARTMENTS table and its contents.

```
Name                          Null     Type
----------------------------- -------- --------------
DEPARTMENT_ID                 NOT NULL NUMBER(4)
DEPARTMENT_NAME               NOT NULL VARCHAR2(30)
MANAGER_ID                             NUMBER(6)
LOCATION_ID                            NUMBER(4)

4 rows selected
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

5) Determine the structure of the EMPLOYEES table.

```
Name                          Null     Type
----------------------------- -------- --------------
EMPLOYEE_ID                   NOT NULL NUMBER(6)
FIRST_NAME                             VARCHAR2(20)
LAST_NAME                     NOT NULL VARCHAR2(25)
EMAIL                         NOT NULL VARCHAR2(25)
PHONE_NUMBER                           VARCHAR2(20)
HIRE_DATE                     NOT NULL DATE
JOB_ID                        NOT NULL VARCHAR2(10)
SALARY                                 NUMBER(8,2)
COMMISSION_PCT                         NUMBER(2,2)
MANAGER_ID                             NUMBER(6)
DEPARTMENT_ID                          NUMBER(4)

11 rows selected
```

The HR department wants a query to display the last name, job ID, hire date, and employee ID for each employee, with the employee ID appearing first. Provide an alias STARTDATE for the HIRE_DATE column. Save your SQL statement to a file named lab_01_05.sql so that you can dispatch this file to the HR department.

6) Test your query in the lab_01_05.sql file to ensure that it runs correctly.

Note: After you have executed the query, make sure that you do not enter your next query in the same worksheet. Open a new worksheet.

Practice 1-1: Retrieving Data Using the SQL SELECT Statement (continued)

	EMPLOYEE_ID	LAST_NAME	JOB_ID	STARTDATE
1	200 Whalen		AD_ASST	17-SEP-87
2	201 Hartstein		MK_MAN	17-FEB-96
3	202 Fay		MK_REP	17-AUG-97
4	205 Higgins		AC_MGR	07-JUN-94
5	206 Gietz		AC_ACCOUNT	07-JUN-94

...

19	176 Taylor		SA_REP	24-MAR-98
20	178 Grant		SA_REP	24-MAY-99

7) The HR department wants a query to display all unique job IDs from the EMPLOYEES table.

	JOB_ID
1	AC_ACCOUNT
2	AC_MGR
3	AD_ASST
4	AD_PRES
5	AD_VP
6	IT_PROG
7	MK_MAN
8	MK_REP
9	SA_MAN
10	SA_REP
11	ST_CLERK
12	ST_MAN

Part 3

If you have time, complete the following exercises:

8) The HR department wants more descriptive column headings for its report on employees. Copy the statement from lab_01_05.sql to a new SQL Worksheet. Name the column headings Emp #, Employee, Job, and Hire Date, respectively. Then run the query again.

	Emp #	Employee	Job	Hire Date
1	200	Whalen	AD_ASST	17-SEP-87
2	201	Hartstein	MK_MAN	17-FEB-96
3	202	Fay	MK_REP	17-AUG-97
4	205	Higgins	AC_MGR	07-JUN-94
5	206	Gietz	AC_ACCOUNT	07-JUN-94

...

Practice 1-1: Retrieving Data Using the SQL SELECT Statement (continued)

19	176	Taylor	SA_REP	24-MAR-98
20	178	Grant	SA_REP	24-MAY-99

9) The HR department has requested a report of all employees and their job IDs. Display the last name concatenated with the job ID (separated by a comma and space) and name the column `Employee and Title`.

	Employee and Title
1	Abel, SA_REP
2	Davies, ST_CLERK
3	De Haan, AD_VP
4	Ernst, IT_PROG
5	Fay, MK_REP

...

19	Whalen, AD_ASST
20	Zlotkey, SA_MAN

If you want an extra challenge, complete the following exercise:

10) To familiarize yourself with the data in the `EMPLOYEES` table, create a query to display all the data from that table. Separate each column output by a comma. Name the column title `THE_OUTPUT`.

	THE_OUTPUT
1	200,Jennifer,Whalen,JWHALEN,515.123.4444,AD_ASST,101,17-SEP-87,4400,,10
2	201,Michael,Hartstein,MHARTSTE,515.123.5555,MK_MAN,100,17-FEB-96,13000,,20
3	202,Pat,Fay,PFAY,603.123.6666,MK_REP,201,17-AUG-97,6000,,20
4	205,Shelley,Higgins,SHIGGINS,515.123.8080,AC_MGR,101,07-JUN-94,12000,,110
5	206,William,Gietz,WGIETZ,515.123.8181,AC_ACCOUNT,205,07-JUN-94,8300,,110

...

19	176,Jonathon,Taylor,JTAYLOR,011.44.1644.429265,SA_REP,149,24-MAR-98,8600,.2,80
20	178,Kimberely,Grant,KGRANT,011.44.1644.429263,SA_REP,149,24-MAY-99,7000,.15,

Oracle University and NETEC, S.A. de C.V. use only.

Practice Solutions 1-1: Retrieving Data Using the SQL `SELECT` Statement

Part 1

Test your knowledge:

1) The following `SELECT` statement executes successfully:

```
SELECT last_name, job_id, salary AS Sal
FROM   employees;
```

   True/False

2) The following `SELECT` statement executes successfully:

```
SELECT *
FROM   job_grades;
```

   True/False

3) There are four coding errors in the following statement. Can you identify them?

```
SELECT    employee_id, last_name
sal x 12  ANNUAL SALARY
FROM      employees;
```

- **The `EMPLOYEES` table does not contain a column called `sal`. The column is called `SALARY`.**

- **The multiplication operator is *, not x, as shown in line 2.**

- **The `ANNUAL  SALARY` alias cannot include spaces. The alias should read `ANNUAL_SALARY` or should be enclosed within double quotation marks.**

- **A comma is missing after the `LAST_NAME` column.**

Part 2

You have been hired as a SQL programmer for Acme Corporation. Your first task is to create some reports based on data from the Human Resources tables.

4) Your first task is to determine the structure of the `DEPARTMENTS` table and its contents.

   a. To determine the `DEPARTMENTS` table structure:

```
DESCRIBE departments
```

Practice Solutions 1-1: Retrieving Data Using the SQL SELECT Statement (continued)

    b. To view the data contained in the DEPARTMENTS table:

```
SELECT *
FROM   departments;
```

5) Determine the structure of the EMPLOYEES table.

```
DESCRIBE employees
```

The HR department wants a query to display the last name, job ID, hire date, and employee ID for each employee, with the employee ID appearing first. Provide an alias STARTDATE for the HIRE_DATE column. Save your SQL statement to a file named lab_01_05.sql so that you can dispatch this file to the HR department.

```
SELECT employee_id, last_name, job_id, hire_date StartDate
FROM   employees;
```

6) Test your query in the lab_01_05.sql file to ensure that it runs correctly.

```
SELECT employee_id, last_name, job_id, hire_date StartDate
FROM   employees;
```

7) The HR department wants a query to display all unique job IDs from the EMPLOYEES table.

```
SELECT DISTINCT job_id
FROM   employees;
```

Part 3

If you have time, complete the following exercises:

8) The HR department wants more descriptive column headings for its report on employees. Copy the statement from lab_01_05.sql to a new SQL Worksheet. Name the column headings Emp #, Employee, Job, and Hire Date, respectively. Then run the query again.

```
SELECT employee_id "Emp #", last_name "Employee",
       job_id "Job", hire_date "Hire Date"
FROM   employees;
```

9) The HR department has requested a report of all employees and their job IDs. Display the last name concatenated with the job ID (separated by a comma and space) and name the column Employee and Title.

```
SELECT last_name||', '||job_id "Employee and Title"
FROM   employees;
```

Practice Solutions 1-1: Retrieving Data Using the SQL SELECT Statement (continued)

If you want an extra challenge, complete the following exercise:

10) To familiarize yourself with the data in the EMPLOYEES table, create a query to display all the data from that table. Separate each column output by a comma. Name the column title THE_OUTPUT.

```
SELECT employee_id || ',' || first_name || ',' || last_name
       || ',' || email || ',' || phone_number || ','|| job_id
       || ',' || manager_id || ',' || hire_date || ','
       || salary || ',' || commission_pct || ',' ||
department_id
       THE_OUTPUT
FROM   employees;
```

In this practice, you build more reports, including statements that use the WHERE clause and the ORDER BY clause. You make the SQL statements more reusable and generic by including the ampersand substitution.

Practice 2-1: Restricting and Sorting Data

The HR department needs your assistance in creating some queries.

1) Because of budget issues, the HR department needs a report that displays the last name and salary of employees who earn more than $12,000. Save your SQL statement as a file named `lab_02_01.sql`. Run your query.

	LAST_NAME	SALARY
1	Hartstein	13000
2	King	24000
3	Kochhar	17000
4	De Haan	17000

2) Open a new SQL Worksheet. Create a report that displays the last name and department number for employee number 176. Run the query.

	LAST_NAME	DEPARTMENT_ID
1	Taylor	80

3) The HR department needs to find high-salary and low-salary employees. Modify `lab_02_01.sql` to display the last name and salary for any employee whose salary is not in the range of $5,000 to $12,000. Save your SQL statement as `lab_02_03.sql`.

	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	King	24000
4	Kochhar	17000
5	De Haan	17000
6	Lorentz	4200
7	Rajs	3500
8	Davies	3100
9	Matos	2600
10	Vargas	2500

4) Create a report to display the last name, job ID, and hire date for employees with the last names of Matos and Taylor. Order the query in ascending order by the hire date.

	LAST_NAME	JOB_ID	HIRE_DATE
1	Matos	ST_CLERK	15-MAR-98
2	Taylor	SA_REP	24-MAR-98

Practice 2-1: Restricting and Sorting Data (continued)

5) Display the last name and department ID of all employees in departments 20 or 50 in ascending alphabetical order by name.

	LAST_NAME	DEPARTMENT_ID
1	Davies	50
2	Fay	20
3	Hartstein	20
4	Matos	50
5	Mourgos	50
6	Rajs	50
7	Vargas	50

6) Modify `lab_02_03.sql` to display the last name and salary of employees who earn between $5,000 and $12,000, and are in department 20 or 50. Label the columns `Employee` and `Monthly Salary`, respectively. Save `lab_02_03.sql` as `lab_02_06.sql` again. Run the statement in `lab_02_06.sql`.

	Employee	Monthly Salary
1	Fay	6000
2	Mourgos	5800

7) The HR department needs a report that displays the last name and hire date for all employees who were hired in 1994.

	LAST_NAME	HIRE_DATE
1	Higgins	07-JUN-94
2	Gietz	07-JUN-94

8) Create a report to display the last name and job title of all employees who do not have a manager.

	LAST_NAME	JOB_ID
1	King	AD_PRES

9) Create a report to display the last name, salary, and commission of all employees who earn commissions. Sort data in descending order of salary and commissions. Use the column's numeric position in the `ORDER BY` clause.

	LAST_NAME	SALARY	COMMISSION_PCT
1	Abel	11000	0.3
2	Zlotkey	10500	0.2
3	Taylor	8600	0.2
4	Grant	7000	0.15

Practice 2-1: Restricting and Sorting Data (continued)

10) Members of the HR department want to have more flexibility with the queries that you are writing. They would like a report that displays the last name and salary of employees who earn more than an amount that the user specifies after a prompt. Save this query to a file named `lab_02_10.sql`. If you enter 12000 when prompted, the report displays the following results:

	LAST_NAME	SALARY
1	Hartstein	13000
2	King	24000
3	Kochhar	17000
4	De Haan	17000

11) The HR department wants to run reports based on a manager. Create a query that prompts the user for a manager ID and generates the employee ID, last name, salary, and department for that manager's employees. The HR department wants the ability to sort the report on a selected column. You can test the data with the following values:

manager_id = 103, sorted by last_name:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	104 Ernst		6000	60
2	107 Lorentz		4200	60

manager_id = 201, sorted by salary:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	202 Fay		6000	20

manager_id = 124, sorted by employee_id:

	EMPLOYEE_ID	LAST_NAME	SALARY	DEPARTMENT_ID
1	141 Rajs		3500	50
2	142 Davies		3100	50
3	143 Matos		2600	50
4	144 Vargas		2500	50

If you have time, complete the following exercises:

12) Display all employee last names in which the third letter of the name is "a."

	LAST_NAME
1	Grant
2	Whalen

Oracle Database 11g: SQL Fundamentals I   A - 22

Practice 2-1: Restricting and Sorting Data (continued)

13) Display the last names of all employees who have both an "a" and an "e" in their last name.

	LAST_NAME
1	Davies
2	De Haan
3	Hartstein
4	Whalen

If you want an extra challenge, complete the following exercises:

14) Display the last name, job, and salary for all employees whose jobs are either those of a sales representative or of a stock clerk, and whose salaries are not equal to $2,500, $3,500, or $7,000.

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000
2	Taylor	SA_REP	8600
3	Davies	ST_CLERK	3100
4	Matos	ST_CLERK	2600

15) Modify `lab_02_06.sql` to display the last name, salary, and commission for all employees whose commission is 20%. Save `lab_02_06.sql` as `lab_02_15.sql` again. Rerun the statement in `lab_02_15.sql`.

	Employee	Monthly Salary	COMMISSION_PCT
1	Zlotkey	10500	0.2
2	Taylor	8600	0.2

Practice Solutions 2-1: Restricting and Sorting Data

The HR department needs your assistance in creating some queries.

1) Because of budget issues, the HR department needs a report that displays the last name and salary of employees earning more than $12,000. Save your SQL statement as a file named `lab_02_01.sql`. Run your query.

```
SELECT   last_name, salary
FROM     employees
WHERE    salary > 12000;
```

2) Open a new SQL Worksheet. Create a report that displays the last name and department number for employee number 176.

```
SELECT   last_name, department_id
FROM     employees
WHERE    employee_id = 176;
```

3) The HR department needs to find high-salary and low-salary employees. Modify `lab_02_01.sql` to display the last name and salary for all employees whose salary is not in the range $5,000 through $12,000. Save your SQL statement as `lab_02_03.sql`.

```
SELECT   last_name, salary
FROM     employees
WHERE    salary NOT BETWEEN 5000 AND 12000;
```

4) Create a report to display the last name, job ID, and hire date for employees with the last names of Matos and Taylor. Order the query in ascending order by hire date.

```
SELECT    last_name, job_id, hire_date
FROM      employees
WHERE     last_name IN ('Matos', 'Taylor')
ORDER BY hire_date;
```

5) Display the last name and department ID of all employees in departments 20 or 50 in ascending alphabetical order by name.

```
SELECT    last_name, department_id
FROM      employees
WHERE     department_id IN (20, 50)
ORDER BY last_name ASC;
```

6) Modify `lab_02_03.sql` to list the last name and salary of employees who earn between $5,000 and $12,000, and are in department 20 or 50. Label the columns Employee and Monthly Salary, respectively. Save `lab_02_03.sql` as `lab_02_06.sql` again. Run the statement in `lab_02_06.sql`.

```
SELECT    last_name "Employee", salary "Monthly Salary"
FROM      employees
WHERE     salary  BETWEEN 5000 AND 12000
AND       department_id IN (20, 50);
```

Practice Solutions 2-1: Restricting and Sorting Data (continued)

7) The HR department needs a report that displays the last name and hire date for all employees who were hired in 1994.

```
SELECT    last_name, hire_date
FROM      employees
WHERE     hire_date LIKE '%94';
```

8) Create a report to display the last name and job title of all employees who do not have a manager.

```
SELECT    last_name, job_id
FROM      employees
WHERE     manager_id IS NULL;
```

9) Create a report to display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions. Use the column's numeric position in the ORDER BY clause.

```
SELECT    last_name, salary, commission_pct
FROM      employees
WHERE     commission_pct IS NOT NULL
ORDER BY 2 DESC, 3 DESC;
```

10) Members of the HR department want to have more flexibility with the queries that you are writing. They would like a report that displays the last name and salary of employees who earn more than an amount that the user specifies after a prompt. (You can use the query created in practice exercise 1 and modify it.) Save this query to a file named lab_02_10.sql.

```
SELECT  last_name, salary
FROM    employees
WHERE   salary > &sal_amt;
```

Enter 12000 when prompted for a value in a dialog box. Click OK.
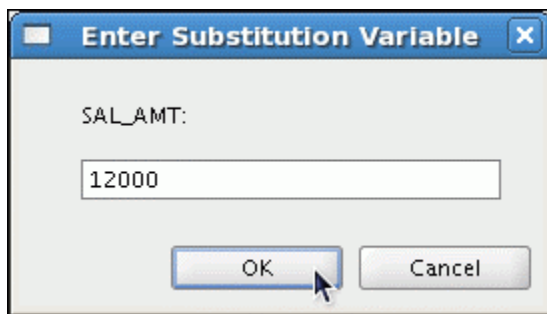
11) The HR department wants to run reports based on a manager. Create a query that prompts the user for a manager ID and generates the employee ID, last name, salary, and department for that manager's employees. The HR department wants the ability to sort the report on a selected column. You can test the data with the following values:

manager _id = 103, sorted by last_name
manager_id = 201, sorted by salary
manager_id = 124, sorted by employee_id

```
SELECT employee_id, last_name, salary, department_id
FROM employees
WHERE manager_id = &mgr_num
ORDER BY &order_col;
```

If you have the time, complete the following exercises:

12) Display all employee last names in which the third letter of the name is "a."

```
SELECT     last_name
FROM       employees
WHERE      last_name LIKE '__a%';
```

13) Display the last names of all employees who have both an "a" and an "e" in their last name.

```
SELECT     last_name
FROM       employees
WHERE      last_name LIKE '%a%'
AND        last_name LIKE '%e%';
```

If you want an extra challenge, complete the following exercises:

14) Display the last name, job, and salary for all employees whose job is that of a sales representative or a stock clerk, and whose salary is not equal to $2,500, $3,500, or $7,000.

```
SELECT     last_name, job_id, salary
FROM       employees
WHERE      job_id IN ('SA_REP', 'ST_CLERK')
AND        salary NOT IN (2500, 3500, 7000);
```

15) Modify `lab_02_06.sql` to display the last name, salary, and commission for all employees whose commission amount is 20%. Save `lab_02_06.sql` as `lab_02_15.sql` again. Rerun the statement in `lab_02_15.sql`.

```
SELECT     last_name "Employee", salary "Monthly Salary",
           commission_pct
FROM       employees
WHERE      commission_pct = .20;
```

This practice provides a variety of exercises using different functions that are available for character, number, and date data types.

Practice 3-1: Using Single-Row Functions to Customize Output

1) Write a query to display the system date. Label the column `Date`.

   Note: If your database is remotely located in a different time zone, the output will be the date for the operating system on which the database resides.

	Date
1	10-JUN-09

2) The HR department needs a report to display the employee number, last name, salary, and salary increased by 15.5% (expressed as a whole number) for each employee. Label the column `New Salary`. Save your SQL statement in a file named `lab_03_02.sql`.

3) Run your query in the `lab_03_02.sql` file.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary
1	200	Whalen	4400	5082
2	201	Hartstein	13000	15015
3	202	Fay	6000	6930
4	205	Higgins	12000	13860
5	206	Gietz	8300	9587

   ...

19	176	Taylor	8600	9933
20	178	Grant	7000	8085

4) Modify your query `lab_03_02.sql` to add a column that subtracts the old salary from the new salary. Label the column `Increase`. Save the contents of the file as `lab_03_04.sql`. Run the revised query.

	EMPLOYEE_ID	LAST_NAME	SALARY	New Salary	Increase
1	200	Whalen	4400	5082	682
2	201	Hartstein	13000	15015	2015
3	202	Fay	6000	6930	930
4	205	Higgins	12000	13860	1860
5	206	Gietz	8300	9587	1287

   ...

19	176	Taylor	8600	9933	1333
20	178	Grant	7000	8085	1085

Practice 3-1: Using Single-Row Functions to Customize Output (continued)
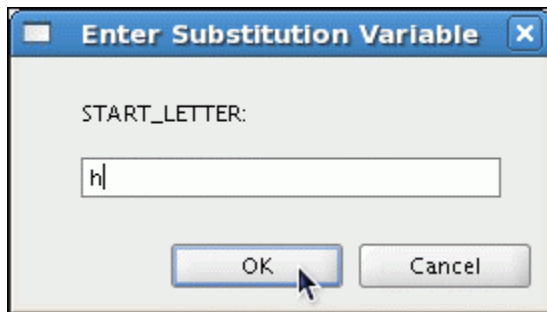
5) Write a query that displays the last name (with the first letter in uppercase and all the other letters in lowercase) and the length of the last name for all employees whose name starts with the letters "J," "A," or "M." Give each column an appropriate label. Sort the results by the employees' last names.

	Name	Length
1	Abel	4
2	Matos	5
3	Mourgos	7

Rewrite the query so that the user is prompted to enter a letter that the last name starts with. For example, if the user enters "H" (capitalized) when prompted for a letter, then the output should show all employees whose last name starts with the letter "H."

	Name	Length
1	Hartstein	9
2	Higgins	7
3	Hunold	6

Modify the query such that the case of the entered letter does not affect the output. The entered letter must be capitalized before being processed by the SELECT query.

Enter Substitution Variable

START_LETTER:

h

OK   Cancel

	Name	Length
1	Hartstein	9
2	Higgins	7
3	Hunold	6

6) The HR department wants to find the duration of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column as MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

Note: Because this query depends on the date when it was executed, the values in the MONTHS_WORKED column will differ for you.

Practice 3-1: Using Single-Row Functions to Customize Output (continued)

	LAST_NAME	MONTHS_WORKED
1	Zlotkey	112
2	Mourgos	115
3	Grant	121
4	Lorentz	124
5	Vargas	131

...

	LAST_NAME	MONTHS_WORKED
19	Whalen	261
20	King	264

If you have time, complete the following exercises:

7) Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the $ symbol. Label the column SALARY.

	LAST_NAME	SALARY
1	Whalen	$$$$$$$$$$$4400
2	Hartstein	$$$$$$$$$$13000
3	Fay	$$$$$$$$$$$6000
4	Higgins	$$$$$$$$$$12000
5	Gietz	$$$$$$$$$$$8300

...

	LAST_NAME	SALARY
19	Taylor	$$$$$$$$$$$8600
20	Grant	$$$$$$$$$$$7000

8) Create a query that displays the first eight characters of the employees' last names and indicates the amounts of their salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column EMPLOYEES_AND_THEIR_SALARIES.

	EMPLOYEES_AND_THEIR_SALARIES
1	King **************************
2	Kochhar *****************
3	De Haan *****************
4	Hartstei *************
5	Higgins ************

...

	EMPLOYEES_AND_THEIR_SALARIES
19	Matos ***
20	Vargas ***

Practice 3-1: Using Single-Row Functions to Customize Output (continued)

9) Create a query to display the last name and the number of weeks employed for all employees in department 90. Label the number of weeks column TENURE. Truncate the number of weeks value to 0 decimal places. Show the records in descending order of the employee's tenure.

Note: The TENURE value will differ as it depends on the date on which you run the query.

	LAST_NAME	TENURE
1	King	1147
2	Kochhar	1028
3	De Haan	856

Practice Solutions 3-1: Using Single-Row Functions to Customize Output

1) Write a query to display the system date. Label the column `Date`.

   Note: If your database is remotely located in a different time zone, the output will be the date for the operating system on which the database resides.

```
SELECT   sysdate "Date"
FROM     dual;
```

2) The HR department needs a report to display the employee number, last name, salary, and salary increased by 15.5% (expressed as a whole number) for each employee. Label the column `New Salary`. Save your SQL statement in a file named `lab_03_02.sql`.

```
SELECT   employee_id, last_name, salary,
         ROUND(salary * 1.155, 0) "New Salary"
FROM     employees;
```

3) Run your query in the file `lab_03_02.sql`.

```
SELECT   employee_id, last_name, salary,
         ROUND(salary * 1.155, 0) "New Salary"
FROM     employees;
```

4) Modify your query `lab_03_02.sql` to add a column that subtracts the old salary from the new salary. Label the column `Increase`. Save the contents of the file as `lab_03_04.sql`. Run the revised query.

```
SELECT   employee_id, last_name, salary,
         ROUND(salary * 1.155, 0) "New Salary",
         ROUND(salary * 1.155, 0) - salary "Increase"
FROM     employees;
```

5) Write a query that displays the last name (with the first letter in uppercase and all the other letters in lowercase) and the length of the last name for all employees whose name starts with the letters "J," "A," or "M." Give each column an appropriate label. Sort the results by the employees' last names.

```
SELECT   INITCAP(last_name) "Name",
         LENGTH(last_name) "Length"
FROM     employees
WHERE    last_name LIKE 'J%'
OR       last_name LIKE 'M%'
OR       last_name LIKE 'A%'
ORDER BY last_name ;
```

Rewrite the query so that the user is prompted to enter a letter that starts the last name. For example, if the user enters H (capitalized) when prompted for a letter, then the output should show all employees whose last name starts with the letter "H."

Practice Solutions 3-1: Using Single-Row Functions to Customize Output (continued)

```
SELECT  INITCAP(last_name) "Name",
        LENGTH(last_name) "Length"
FROM    employees
WHERE   last_name LIKE '&start_letter%'
ORDER BY last_name;
```

Modify the query such that the case of the entered letter does not affect the output. The entered letter must be capitalized before being processed by the SELECT query.

```
SELECT  INITCAP(last_name) "Name",
LENGTH(last_name) "Length"
FROM    employees
WHERE   last_name LIKE UPPER('&start_letter%' )
ORDER BY last_name;
```

6) The HR department wants to find the duration of employment for each employee. For each employee, display the last name and calculate the number of months between today and the date on which the employee was hired. Label the column MONTHS_WORKED. Order your results by the number of months employed. Round the number of months up to the closest whole number.

Note: Because this query depends on the date when it was executed, the values in the MONTHS_WORKED column will differ for you.

```
SELECT last_name, ROUND(MONTHS_BETWEEN(
        SYSDATE, hire_date)) MONTHS_WORKED
FROM    employees
ORDER BY months_worked;
```

If you have the time, complete the following exercises:

7) Create a query to display the last name and salary for all employees. Format the salary to be 15 characters long, left-padded with the $ symbol. Label the column SALARY.

```
SELECT last_name,
       LPAD(salary, 15, '$') SALARY
FROM   employees;
```

8) Create a query that displays the first eight characters of the employees' last names and indicates the amounts of their salaries with asterisks. Each asterisk signifies a thousand dollars. Sort the data in descending order of salary. Label the column EMPLOYEES_AND_THEIR_SALARIES.

```
SELECT rpad(last_name, 8)||' '||
       rpad(' ', salary/1000+1, '*')
               EMPLOYEES_AND_THEIR_SALARIES
FROM   employees
ORDER BY salary DESC;
```

Practice Solutions 3-1: Using Single-Row Functions to Customize Output (continued)

9) Create a query to display the last name and the number of weeks employed for all employees in department 90. Label the number of weeks column `TENURE`. Truncate the number of weeks value to 0 decimal places. Show the records in descending order of the employee's tenure.

Note: The `TENURE` value will differ as it depends on the date when you run the query.

```
SELECT last_name, trunc((SYSDATE-hire_date)/7) AS TENURE
FROM    employees
WHERE   department_id = 90
ORDER BY TENURE DESC
```

This practice provides a variety of exercises using TO_CHAR and TO_DATE functions, and conditional expressions such as DECODE and CASE. Remember that for nested functions, the results are evaluated from the innermost function to the outermost function.

Practice 4-1: Using Conversion Functions and Conditional Expressions

1) Create a report that produces the following for each employee:

   `<employee last name>` earns `<salary>` monthly but wants `<3 times salary.>`. Label the column `Dream Salaries`.

	Dream Salaries
1	Whalen earns $4,400.00 monthly but wants $13,200.00.
2	Hartstein earns $13,000.00 monthly but wants $39,000.00.
3	Fay earns $6,000.00 monthly but wants $18,000.00.
4	Higgins earns $12,000.00 monthly but wants $36,000.00.
5	Gietz earns $8,300.00 monthly but wants $24,900.00.

   ...

19	Taylor earns $8,600.00 monthly but wants $25,800.00.
20	Grant earns $7,000.00 monthly but wants $21,000.00.

2) Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column `REVIEW`. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."

	LAST_NAME	HIRE_DATE	REVIEW
1	Whalen	17-SEP-87	Monday, the Twenty-First of March, 1988
2	Hartstein	17-FEB-96	Monday, the Nineteenth of August, 1996
3	Fay	17-AUG-97	Monday, the Twenty-Third of February, 1998
4	Higgins	07-JUN-94	Monday, the Twelfth of December, 1994
5	Gietz	07-JUN-94	Monday, the Twelfth of December, 1994

   ...

19	Taylor	24-MAR-98	Monday, the Twenty-Eighth of September, 1998
20	Grant	24-MAY-99	Monday, the Twenty-Ninth of November, 1999

3) Display the last name, hire date, and day of the week on which the employee started. Label the column `DAY`. Order the results by the day of the week, starting with Monday.

	LAST_NAME	HIRE_DATE	DAY
1	Grant	24-MAY-99	MONDAY
2	Ernst	21-MAY-91	TUESDAY
3	Taylor	24-MAR-98	TUESDAY
4	Rajs	17-OCT-95	TUESDAY
5	Mourgos	16-NOV-99	TUESDAY

   ...

19	Matos	15-MAR-98	SUNDAY
20	Fay	17-AUG-97	SUNDAY

Practice 4-1: Using Conversion Functions and Conditional Expressions (continued)

4) Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, show "No Commission." Label the column COMM.

	LAST_NAME	COMM
1	Whalen	No Commission
2	Hartstein	No Commission
3	Fay	No Commission
4	Higgins	No Commission
5	Gietz	No Commission

...

16	Vargas	No Commission
17	Zlotkey	.2
18	Abel	.3
19	Taylor	.2
20	Grant	.15

If you have time, complete the following exercises:

5) Using the DECODE function, write a query that displays the grade of all employees based on the value of the column JOB_ID, using the following data:

Job	Grade
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
None of the above	0

	JOB_ID	GRADE
1	AC_ACCOUNT	0
2	AC_MGR	0
3	AD_ASST	0
4	AD_PRES	A
5	AD_VP	0
6	AD_VP	0
7	IT_PROG	C

...

14	SA_REP	D
15	SA_REP	D

...

19	ST_CLERK	E
20	ST_MAN	B

Practice 4-1: Using Conversion Functions and Conditional Expressions (continued)

6) Rewrite the statement in the preceding exercise by using the CASE syntax.

	JOB_ID	GRADE
1	AC_ACCOUNT	0
2	AC_MGR	0
3	AD_ASST	0
4	AD_PRES	A
5	AD_VP	0
6	AD_VP	0
7	IT_PROG	C

...

	JOB_ID	GRADE
14	SA_REP	D
15	SA_REP	D

...

	JOB_ID	GRADE
19	ST_CLERK	E
20	ST_MAN	B

Practice Solutions 4-1: Using Conversion Functions and Conditional Expressions

1) Create a report that produces the following for each employee:
   <employee last name> earns *<salary>* monthly but wants *<3 times salary.>*. Label the column `Dream Salaries`.

```
SELECT  last_name || ' earns '
        || TO_CHAR(salary, 'fm$99,999.00')
        || ' monthly but wants '
        || TO_CHAR(salary * 3, 'fm$99,999.00')
        || '.' "Dream Salaries"
FROM    employees;
```

2) Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column `REVIEW`. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."

```
SELECT last_name, hire_date,
       TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6),'MONDAY'),
       'fmDay, "the" Ddspth "of" Month, YYYY') REVIEW
FROM    employees;
```

3) Display the last name, hire date, and day of the week on which the employee started. Label the column `DAY`. Order the results by the day of the week, starting with Monday.

```
SELECT last_name, hire_date,
       TO_CHAR(hire_date, 'DAY') DAY
FROM    employees
ORDER BY TO_CHAR(hire_date - 1, 'd');
```

4) Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, show "No Commission." Label the column `COMM`.

```
SELECT last_name,
       NVL(TO_CHAR(commission_pct), 'No Commission') COMM
FROM   employees;
```

5) Using the `DECODE` function, write a query that displays the grade of all employees based on the value of the `JOB_ID` column, using the following data:

Job	Grade
AD_PRES	A
ST_MAN	B
IT_PROG	C
SA_REP	D
ST_CLERK	E
None of the above	0

Practice Solutions 4-1: Using Conversion Functions and Conditional Expressions (continued)

```
SELECT job_id, decode (job_id,
                        'ST_CLERK',  'E',
                        'SA_REP',    'D',
                        'IT_PROG',   'C',
                        'ST_MAN',    'B',
                        'AD_PRES',   'A',
                        '0')GRADE
FROM employees;
```

6) Rewrite the statement in the preceding exercise by using the CASE syntax.

```
SELECT job_id, CASE job_id
                WHEN 'ST_CLERK' THEN 'E'
                WHEN 'SA_REP'   THEN 'D'
                WHEN 'IT_PROG'  THEN 'C'
                WHEN 'ST_MAN'   THEN 'B'
                WHEN 'AD_PRES'  THEN 'A'
                ELSE '0'  END   GRADE
FROM employees;
```

At the end of this practice, you should be familiar with using group functions and selecting groups of data.

Practice 5-1: Reporting Aggregated Data Using the Group Functions

Determine the validity of the following three statements. Circle either True or False.

1) Group functions work across many rows to produce one result per group.
   True/False

2) Group functions include nulls in calculations.
   True/False

3) The WHERE clause restricts rows before inclusion in a group calculation.
   True/False

The HR department needs the following reports:

4) Find the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number. Save your SQL statement as lab_05_04.sql. Run the query.

	Maximum	Minimum	Sum	Average
1	24000	2500	175500	8775

5) Modify the query in lab_05_04.sql to display the minimum, maximum, sum, and average salary for each job type. Save lab_05_04.sql as lab_05_05.sql again. Run the statement in lab_05_05.sql.

	JOB_ID	Maximum	Minimum	Sum	Average
1	AC_MGR	12000	12000	12000	12000
2	AC_ACCOUNT	8300	8300	8300	8300
3	IT_PROG	9000	4200	19200	6400
4	ST_MAN	5800	5800	5800	5800
5	AD_ASST	4400	4400	4400	4400
6	AD_VP	17000	17000	34000	17000
7	MK_MAN	13000	13000	13000	13000
8	SA_MAN	10500	10500	10500	10500
9	MK_REP	6000	6000	6000	6000
10	AD_PRES	24000	24000	24000	24000
11	SA_REP	11000	7000	26600	8867
12	ST_CLERK	3500	2500	11700	2925

Practice 5-1: Reporting Aggregated Data Using the Group Functions (continued)

6) Write a query to display the number of people with the same job.

	JOB_ID	COUNT(*)
1	AC_ACCOUNT	1
2	AC_MGR	1
3	AD_ASST	1
4	AD_PRES	1
5	AD_VP	2
6	IT_PROG	3
7	MK_MAN	1
8	MK_REP	1
9	SA_MAN	1
10	SA_REP	3
11	ST_CLERK	4
12	ST_MAN	1

Generalize the query so that the user in the HR department is prompted for a job title. Save the script to a file named `lab_05_06.sql`. Run the query. Enter `IT_PROG` when prompted.

	JOB_ID	COUNT(*)
1	IT_PROG	3

7) Determine the number of managers without listing them. Label the column `Number of Managers`.
Hint: Use the `MANAGER_ID` column to determine the number of managers.

	Number of Managers
1	8

8) Find the difference between the highest and lowest salaries. Label the column `DIFFERENCE`.

	DIFFERENCE
1	21500

If you have time, complete the following exercises:

9) Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is $6,000 or less. Sort the output in descending order of salary.

	MANAGER_ID	MIN(SALARY)
1	102	9000
2	205	8300
3	149	7000

Practice 5-1: Reporting Aggregated Data Using the Group Functions (continued)

If you want an extra challenge, complete the following exercises:

10) Create a query to display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

	TOTAL	1995	1996	1997	1998
1	20	1	2	2	3

11) Create a matrix query to display the job, the salary for that job based on department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

	Job	Dept 20	Dept 50	Dept 80	Dept 90	Total
1	AC_MGR	(null)	(null)	(null)	(null)	12000
2	AC_ACCOUNT	(null)	(null)	(null)	(null)	8300
3	IT_PROG	(null)	(null)	(null)	(null)	19200
4	ST_MAN	(null)	5800	(null)	(null)	5800
5	AD_ASST	(null)	(null)	(null)	(null)	4400
6	AD_VP	(null)	(null)	(null)	34000	34000
7	MK_MAN	13000	(null)	(null)	(null)	13000
8	SA_MAN	(null)	(null)	10500	(null)	10500
9	MK_REP	6000	(null)	(null)	(null)	6000
10	AD_PRES	(null)	(null)	(null)	24000	24000
11	SA_REP	(null)	(null)	19600	(null)	26600
12	ST_CLERK	(null)	11700	(null)	(null)	11700

Practice Solutions 5-1: Reporting Aggregated Data Using the Group Functions

Determine the validity of the following three statements. Circle either True or False.

1) Group functions work across many rows to produce one result per group.
   True/False

2) Group functions include nulls in calculations.
   True/**False**

3) The WHERE clause restricts rows before inclusion in a group calculation.
   True/False

The HR department needs the following reports:

4) Find the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively. Round your results to the nearest whole number. Save your SQL statement as lab_05_04.sql. Run the query.

```
SELECT ROUND(MAX(salary),0)  "Maximum",
       ROUND(MIN(salary),0)  "Minimum",
       ROUND(SUM(salary),0)  "Sum",
       ROUND(AVG(salary),0)  "Average"
FROM   employees;
```

5) Modify the query in lab_05_04.sql to display the minimum, maximum, sum, and average salary for each job type. Save lab_05_04.sql as lab_05_05.sql again. Run the statement in lab_05_05.sql.

```
SELECT job_id, ROUND(MAX(salary),0)  "Maximum",
               ROUND(MIN(salary),0)  "Minimum",
               ROUND(SUM(salary),0)  "Sum",
               ROUND(AVG(salary),0)  "Average"
FROM   employees
GROUP BY job_id;
```

6) Write a query to display the number of people with the same job.

```
SELECT job_id, COUNT(*)
FROM   employees
GROUP BY job_id;
```

Generalize the query so that the user in the HR department is prompted for a job title. Save the script to a file named lab_05_06.sql. Run the query. Enter IT_PROG when prompted and click OK.

```
SELECT job_id, COUNT(*)
FROM   employees
WHERE  job_id = '&job_title'
GROUP BY job_id;
```

Practice Solutions 5-1: Reporting Aggregated Data Using the Group Functions (continued)

7) Determine the number of managers without listing them. Label the column `Number of Managers`.

Hint: Use the `MANAGER_ID` column to determine the number of managers.

```
SELECT COUNT(DISTINCT manager_id) "Number of Managers"
FROM   employees;
```

8) Find the difference between the highest and lowest salaries. Label the column `DIFFERENCE`.

```
SELECT   MAX(salary) - MIN(salary) DIFFERENCE
FROM     employees;
```

If you have the time, complete the following exercises:

9) Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is $6,000 or less. Sort the output in descending order of salary.

```
SELECT   manager_id, MIN(salary)
FROM     employees
WHERE    manager_id IS NOT NULL
GROUP BY manager_id
HAVING   MIN(salary) > 6000
ORDER BY MIN(salary) DESC;
```

If you want an extra challenge, complete the following exercises:

10) Create a query that will display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

```
SELECT  COUNT(*) total,
        SUM(DECODE(TO_CHAR(hire_date,
'YYYY'),1995,1,0))"1995",
        SUM(DECODE(TO_CHAR(hire_date,
'YYYY'),1996,1,0))"1996",
        SUM(DECODE(TO_CHAR(hire_date,
'YYYY'),1997,1,0))"1997",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1998,1,0))"1998"
FROM    employees;
```

Practice Solutions 5-1: Reporting Aggregated Data Using the Group Functions (continued)

11) Create a matrix query to display the job, the salary for that job based on the department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

```
SELECT    job_id "Job",
          SUM(DECODE(department_id , 20, salary)) "Dept 20",
          SUM(DECODE(department_id , 50, salary)) "Dept 50",
          SUM(DECODE(department_id , 80, salary)) "Dept 80",
          SUM(DECODE(department_id , 90, salary)) "Dept 90",
          SUM(salary) "Total"
FROM      employees
GROUP BY job_id;
```

This practice is intended to give you experience in extracting data from more than one table using the SQL:1999–compliant joins.

Practice 6-1: Displaying Data from Multiple Tables Using Joins

1) Write a query for the HR department to produce the addresses of all the departments.
   Use the LOCATIONS and COUNTRIES tables. Show the location ID, street address,
   city, state or province, and country in the output. Use a NATURAL JOIN to produce
   the results.

	LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	1400	2014 Jabberwocky Rd	Southlake	Texas	United States of America
2	1500	2011 Interiors Blvd	South San Francisco	California	United States of America
3	1700	2004 Charade Rd	Seattle	Washington	United States of America
4	1800	460 Bloor St. W.	Toronto	Ontario	Canada
5	2500	Magdalen Centre, The Oxford Science Park	Oxford	Oxford	United Kingdom

2) The HR department needs a report of only those employees with corresponding
   departments. Write a query to display the last name, department number, and
   department name for these employees.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting

3) The HR department needs a report of employees in Toronto. Display the last name,
   job, department number, and the department name for all employees who work in
   Toronto.

	LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	Hartstein	MK_MAN	20	Marketing
2	Fay	MK_REP	20	Marketing

4) Create a report to display employees' last name and employee number along with
   their manager's last name and manager number. Label the columns Employee, Emp#,
   Manager, and Mgr#, respectively. Save your SQL statement as lab_06_04.sql.
   Run the query.

	Employee	EMP#	Manager	Mgr#
1	Hunold	103	De Haan	102
2	Fay	202	Hartstein	201
3	Gietz	206	Higgins	205
4	Lorentz	107	Hunold	103
5	Ernst	104	Hunold	103

…

Practice 6-1: Displaying Data from Multiple Tables Using Joins (continued)

18 Taylor	176 Zlotkey	149
19 Abel	174 Zlotkey	149

5) Modify `lab_06_04.sql` to display all employees including King, who has no manager. Order the results by the employee number. Save your SQL statement as `lab_06_05.sql`. Run the query in `lab_06_05.sql`.

	Employee	EMP#	Manager	Mgr#
1	King	100	(null)	(null)
2	Kochhar	101	King	100
3	De Haan	102	King	100
4	Hunold	103	De Haan	102
5	Ernst	104	Hunold	103

...

19 Higgins	205 Kochhar	101
20 Gietz	206 Higgins	205

6) Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named `lab_06_06.sql`.

	DEPARTMENT	EMPLOYEE	COLLEAGUE
1	20	Fay	Hartstein
2	20	Hartstein	Fay
3	50	Davies	Matos
4	50	Davies	Mourgos
5	50	Davies	Rajs

...

41	110	Gietz	Higgins
42	110	Higgins	Gietz

7) The HR department needs a report on job grades and salaries. To familiarize yourself with the JOB_GRADES table, first show the structure of the JOB_GRADES table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES
Name                            Null     Type
------------------------------- -------- --------------
GRADE_LEVEL                              VARCHAR2(3)
LOWEST_SAL                               NUMBER
HIGHEST_SAL                              NUMBER

3 rows selected
```

Practice 6-1: Displaying Data from Multiple Tables Using Joins (continued)

	LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
1	King	AD_PRES	Executive	24000	E
2	Kochhar	AD_VP	Executive	17000	E
3	De Haan	AD_VP	Executive	17000	E
4	Hartstein	MK_MAN	Marketing	13000	D
5	Higgins	AC_MGR	Accounting	12000	D

...

	LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
18	Matos	ST_CLERK	Shipping	2600	A
19	Vargas	ST_CLERK	Shipping	2500	A

If you want an extra challenge, complete the following exercises:

8) The HR department wants to determine the names of all the employees who were hired after Davies. Create a query to display the name and hire date of any employee hired after employee Davies.

	LAST_NAME	HIRE_DATE
1	Fay	17-AUG-97
2	Lorentz	07-FEB-99
3	Mourgos	16-NOV-99
4	Matos	15-MAR-98
5	Vargas	09-JUL-98
6	Zlotkey	29-JAN-00
7	Taylor	24-MAR-98
8	Grant	24-MAY-99

9) The HR department needs to find the names and hire dates of all the employees who were hired before their managers, along with their managers' names and hire dates. Save the script to a file named `lab_06_09.sql`.

	LAST_NAME	HIRE_DATE	LAST_NAME_1	HIRE_DATE_1
1	Whalen	17-SEP-87	Kochhar	21-SEP-89
2	Hunold	03-JAN-90	De Haan	13-JAN-93
3	Vargas	09-JUL-98	Mourgos	16-NOV-99
4	Matos	15-MAR-98	Mourgos	16-NOV-99
5	Davies	29-JAN-97	Mourgos	16-NOV-99
6	Rajs	17-OCT-95	Mourgos	16-NOV-99
7	Grant	24-MAY-99	Zlotkey	29-JAN-00
8	Taylor	24-MAR-98	Zlotkey	29-JAN-00
9	Abel	11-MAY-96	Zlotkey	29-JAN-00

Practice Solutions 6-1: Displaying Data from Multiple Tables Using Joins

1) Write a query for the HR department to produce the addresses of all the departments. Use the LOCATIONS and COUNTRIES tables. Show the location ID, street address, city, state or province, and country in the output. Use a NATURAL JOIN to produce the results.

```
SELECT location_id, street_address, city, state_province,
country_name
FROM    locations
NATURAL JOIN  countries;
```

2) The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all the employees.

```
SELECT last_name, department_id, department_name
FROM    employees
JOIN    departments
USING (department_id);
```

3) The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.

```
SELECT e.last_name, e.job_id, e.department_id,
d.department_name
FROM    employees e JOIN departments d
ON      (e.department_id = d.department_id)
JOIN    locations l
ON      (d.location_id = l.location_id)
WHERE LOWER(l.city) = 'toronto';
```

4) Create a report to display employees' last names and employee number along with their managers' last names and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Save your SQL statement as lab_06_04.sql. Run the query.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM   employees w join employees m
ON     (w.manager_id = m.employee_id);
```

5) Modify lab_06_04.sql to display all employees including King, who has no manager. Order the results by the employee number. Save your SQL statement as lab_06_05.sql. Run the query in lab_06_05.sql.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM   employees w
LEFT   OUTER JOIN employees m
ON     (w.manager_id = m.employee_id)
ORDER BY 2;
```

Practice Solutions 6-1: Displaying Data from Multiple Tables Using Joins (continued)

6) Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named `lab_06_06.sql`. Run the query.

```
SELECT e.department_id department, e.last_name employee,
       c.last_name colleague
FROM   employees e JOIN employees c
ON     (e.department_id = c.department_id)
WHERE  e.employee_id <> c.employee_id
ORDER BY e.department_id, e.last_name, c.last_name;
```

7) The HR department needs a report on job grades and salaries. To familiarize yourself with the JOB_GRADES table, first show the structure of the JOB_GRADES table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES

SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   job_grades j
ON     (e.salary BETWEEN j.lowest_sal AND j.highest_sal);
```

If you want an extra challenge, complete the following exercises:

8) The HR department wants to determine the names of all employees who were hired after Davies. Create a query to display the name and hire date of any employee hired after employee Davies.

```
SELECT e.last_name, e.hire_date
FROM   employees e JOIN employees davies
ON     (davies.last_name = 'Davies')
WHERE  davies.hire_date < e.hire_date;
```

9) The HR department needs to find the names and hire dates for all employees who were hired before their managers, along with their managers' names and hire dates. Save the script to a file named `lab_06_09.sql`.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w JOIN employees m
ON     (w.manager_id = m.employee_id)
WHERE    w.hire_date <  m.hire_date;
```

In this practice, you write complex queries using nested SELECT statements.
For practice questions, you may want to create the inner query first. Make sure that it runs and produces the data that you anticipate before you code the outer query.

Practice 7-1: Using Subqueries to Solve Queries

1) The HR department needs a query that prompts the user for an employee last name. The query then displays the last name and hire date of any employee in the same department as the employee whose name they supply (excluding that employee). For example, if the user enters `Zlotkey`, find all employees who work with Zlotkey (excluding Zlotkey).

Enter Substitution Variable ☒

ENTER_NAME:

Zlotkey

OK   Cancel

	LAST_NAME	HIRE_DATE
1	Abel	11-MAY-96
2	Taylor	24-MAR-98

2) Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in order of ascending salary.

	EMPLOYEE_ID	LAST_NAME	SALARY
1	103	Hunold	9000
2	149	Zlotkey	10500
3	174	Abel	11000
4	205	Higgins	12000
5	201	Hartstein	13000
6	102	De Haan	17000
7	101	Kochhar	17000
8	100	King	24000

3) Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains the letter "u." Save your SQL statement as `lab_07_03.sql`. Run your query.

	EMPLOYEE_ID	LAST_NAME
1	124	Mourgos
2	141	Rajs
3	142	Davies
4	143	Matos
5	144	Vargas
6	103	Hunold
7	104	Ernst
8	107	Lorentz

Practice 7-1: Using Subqueries to Solve Queries (continued)

4) The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

	LAST_NAME	DEPARTMENT_ID	JOB_ID
1	Whalen	10	AD_ASST
2	King	90	AD_PRES
3	Kochhar	90	AD_VP
4	De Haan	90	AD_VP
5	Higgins	110	AC_MGR
6	Gietz	110	AC_ACCOUNT

Modify the query so that the user is prompted for a location ID. Save this to a file named `lab_07_04.sql`.

5) Create a report for HR that displays the last name and salary of every employee who reports to King.

	LAST_NAME	SALARY
1	Hartstein	13000
2	Kochhar	17000
3	De Haan	17000
4	Mourgos	5800
5	Zlotkey	10500

6) Create a report for HR that displays the department number, last name, and job ID for every employee in the Executive department.

	DEPARTMENT_ID	LAST_NAME	JOB_ID
1	90	King	AD_PRES
2	90	Kochhar	AD_VP
3	90	De Haan	AD_VP

7) Create a report that displays a list of all employees whose salary is more than the salary of any employee from department 60.

If you have the time, complete the following exercise:

8) Modify the query in `lab_07_03.sql` to display the employee number, last name, and salary of all employees who earn more than the average salary, and who work in a department with any employee whose last name contains a "u." Save `lab_07_03.sql` as `lab_07_08.sql` again. Run the statement in `lab_07_08.sql`.

	EMPLOYEE_ID	LAST_NAME	SALARY
1	103	Hunold	9000

Practice Solutions 7-1: Using Subqueries to Solve Queries

1) The HR department needs a query that prompts the user for an employee last name.
    The query then displays the last name and hire date of any employee in the same
    department as the employee whose name they supply (excluding that employee). For
    example, if the user enters Zlotkey, find all employees who work with Zlotkey
    (excluding Zlotkey).

```
UNDEFINE Enter_name

SELECT last_name, hire_date
FROM    employees
WHERE   department_id = (SELECT department_id
                        FROM    employees
                        WHERE   last_name = '&&Enter_name')
AND     last_name <> '&Enter_name';
```

2) Create a report that displays the employee number, last name, and salary of all
    employees who earn more than the average salary. Sort the results in order of
    ascending salary.

```
SELECT employee_id, last_name, salary
FROM    employees
WHERE   salary > (SELECT AVG(salary)
                  FROM    employees)
ORDER BY salary;
```

3) Write a query that displays the employee number and last name of all employees who
    work in a department with any employee whose last name contains a "u." Save your
    SQL statement as lab_07_03.sql. Run your query.

```
SELECT employee_id, last_name
FROM    employees
WHERE   department_id IN (SELECT department_id
                         FROM    employees
                         WHERE   last_name like '%u%');
```

4) The HR department needs a report that displays the last name, department number,
    and job ID of all employees whose department location ID is 1700.

```
SELECT last_name, department_id, job_id
FROM    employees
WHERE   department_id IN (SELECT department_id
                         FROM    departments
                         WHERE   location_id = 1700);
```

Practice Solutions 7-1: Using Subqueries to Solve Queries (continued)

Modify the query so that the user is prompted for a location ID. Save this to a file named `lab_07_04.sql`.

```
SELECT last_name, department_id, job_id
FROM    employees
WHERE   department_id IN (SELECT department_id
                          FROM   departments
                          WHERE  location_id =
&Enter_location);
```

5) Create a report for HR that displays the last name and salary of every employee who reports to King.

```
SELECT last_name, salary
FROM    employees
WHERE   manager_id = (SELECT employee_id
                      FROM   employees
                      WHERE  last_name = 'King');
```

6) Create a report for HR that displays the department number, last name, and job ID for every employee in the Executive department.

```
SELECT department_id, last_name, job_id
FROM    employees
WHERE   department_id IN (SELECT department_id
                          FROM   departments
                          WHERE  department_name =
'Executive');
```

7) Create a report that displays a list of all employees whose salary is more than the salary of any employee from department 60.

```
SELECT last_name FROM employees
WHERE salary > ANY (SELECT salary
                    FROM employees
                    WHERE department_id=60);
```

If you have the time, complete the following exercise:

8) Modify the query in `lab_07_03.sql` to display the employee number, last name, and salary of all employees who earn more than the average salary and who work in a department with any employee whose last name contains a "u." Save `lab_07_03.sql` to `lab_07_08.sql` again. Run the statement in `lab_07_08.sql`.

```
SELECT employee_id, last_name, salary
FROM    employees
WHERE   department_id IN (SELECT department_id
                          FROM   employees
                          WHERE  last_name like '%u%')
AND     salary > (SELECT AVG(salary)
                  FROM   employees);
```

In this practice, you write queries using the set operators.

Practice 8-1: Using the Set Operators

1) The HR department needs a list of department IDs for departments that do not contain the job ID ST_CLERK. Use the set operators to create this report.

	DEPARTMENT_ID
1	10
2	20
3	60
4	80
5	90
6	110
7	190

2) The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use the set operators to create this report.

	COUNTRY_ID	COUNTRY_NAME
1	DE	Germany

3) Produce a list of jobs for departments 10, 50, and 20, in that order. Display the job ID and department ID by using the set operators.

	JOB_ID	DEPARTMENT_ID
1	AD_ASST	10
2	ST_MAN	50
3	ST_CLERK	50
4	MK_MAN	20
5	MK_REP	20

4) Create a report that lists the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired by the company (that is, they changed jobs, but have now gone back to doing their original job).

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

5) The HR department needs a report with the following specifications:

- Last name and department ID of all employees from the EMPLOYEES table, regardless of whether or not they belong to a department

- Department ID and department name of all departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them

Write a compound query to accomplish this.

Practice 8-1: Using the Set Operators (continued)

	LAST_NAME	DEPARTMENT_ID	TO_CHAR(NULL)
1	Abel	80	(null)
2	Davies	50	(null)
3	De Haan	90	(null)
4	Ernst	60	(null)
5	Fay	20	(null)
6	Gietz	110	(null)
7	Grant	(null)	(null)
8	Hartstein	20	(null)
9	Higgins	110	(null)
10	Hunold	60	(null)
11	King	90	(null)
12	Kochhar	90	(null)
13	Lorentz	60	(null)
14	Matos	50	(null)
15	Mourgos	50	(null)
16	Rajs	50	(null)
17	Taylor	80	(null)
18	Vargas	50	(null)
19	Whalen	10	(null)
20	Zlotkey	80	(null)
21	(null)	10	Administration
22	(null)	20	Marketing
23	(null)	50	Shipping
24	(null)	60	IT
25	(null)	80	Sales
26	(null)	90	Executive
27	(null)	110	Accounting
28	(null)	190	Contracting

Oracle University and NETEC, S.A. de C.V. use only.

Practice Solutions 8-1: Using the Set Operators

1) The HR department needs a list of department IDs for departments that do not contain the job ID ST_CLERK. Use the set operators to create this report.

```
SELECT department_id
FROM   departments
MINUS
SELECT department_id
FROM   employees
WHERE  job_id = 'ST_CLERK';
```

2) The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use the set operators to create this report.

```
SELECT country_id,country_name
FROM countries
MINUS
SELECT l.country_id,c.country_name
FROM locations l JOIN countries c
ON (l.country_id = c.country_id)
JOIN departments d
ON d.location_id=l.location_id;
```

3) Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID using the set operators.

```
SELECT distinct job_id, department_id
FROM employees
WHERE department_id = 10
UNION ALL
SELECT DISTINCT job_id, department_id
FROM employees
WHERE department_id = 50
UNION ALL
SELECT DISTINCT job_id, department_id
FROM employees
WHERE department_id = 20
```

4) Create a report that lists the employee IDs and job IDs of those employees who currently have a job title that is the same as their job title when they were initially hired by the company (that is, they changed jobs, but have now gone back to doing their original job).

```
SELECT     employee_id,job_id
FROM       employees
INTERSECT
SELECT   employee_id,job_id
FROM     job_history;
```

Practice Solutions 8-1: Using the Set Operators (continued)

5) The HR department needs a report with the following specifications:

- Last name and department ID of all the employees from the EMPLOYEES table, regardless of whether or not they belong to a department

- Department ID and department name of all the departments from the DEPARTMENTS table, regardless of whether or not they have employees working in them

Write a compound query to accomplish this.

```
SELECT last_name,department_id,TO_CHAR(null)
FROM    employees
UNION
SELECT TO_CHAR(null),department_id,department_name
FROM  departments;
```

In this practice, you add rows to the MY_EMPLOYEE table, update and delete data from the table, and control your transactions. You run a script to create the MY_EMPLOYEE table.

Practice 9-1: Manipulating Data

The HR department wants you to create SQL statements to insert, update, and delete employee data. As a prototype, you use the MY_EMPLOYEE table before giving the statements to the HR department.

Note: For all the DML statements, use the Run Script icon (or press [F5]) to execute the query. This way you get to see the feedback messages on the Script Output tabbed page. For SELECT queries, continue to use the Execute Statement icon or press [F9] to get the formatted output on the Results tabbed page.

Insert data into the MY_EMPLOYEE table.

1) Run the statement in the lab_09_01.sql script to build the MY_EMPLOYEE table used in this practice.

2) Describe the structure of the MY_EMPLOYEE table to identify the column names.

```
DESCRIBE my_employee
Name                            Null     Type
------------------------------  -------- --------------
ID                              NOT NULL NUMBER(4)
LAST_NAME                                VARCHAR2(25)
FIRST_NAME                               VARCHAR2(25)
USERID                                   VARCHAR2(8)
SALARY                                   NUMBER(9,2)

5 rows selected
```

3) Create an INSERT statement to add the *first row* of data to the MY_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause. *Do not enter all rows yet.*

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

4) Populate the MY_EMPLOYEE table with the second row of the sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.

5) Confirm your addition to the table.

Practice 9-1: Manipulating Data (continued)

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	895
2	2	Dancs	Betty	bdancs	860

6) Write an INSERT statement in a dynamic reusable script file to load the remaining rows into the MY_EMPLOYEE table. The script should prompt for all the columns (ID, LAST_NAME, FIRST_NAME, USERID, and SALARY). Save this script to a lab_09_06.sql file.

7) Populate the table with the next two rows of the sample data listed in step 3 by running the INSERT statement in the script that you created.

8) Confirm your additions to the table.

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	895
2	2	Dancs	Betty	bdancs	860
3	3	Biri	Ben	bbiri	1100
4	4	Newman	Chad	cnewman	750

9) Make the data additions permanent.

Update and delete data in the MY_EMPLOYEE table.

10) Change the last name of employee 3 to Drexler.

11) Change the salary to $1,000 for all employees who have a salary less than $900.

12) Verify your changes to the table.

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	2	Dancs	Betty	bdancs	1000
3	3	Drexler	Ben	bbiri	1100
4	4	Newman	Chad	cnewman	1000

13) Delete Betty Dancs from the MY_EMPLOYEE table.

14) Confirm your changes to the table.

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	3	Drexler	Ben	bbiri	1100
3	4	Newman	Chad	cnewman	1000

Practice 9-1: Manipulating Data (continued)

15) Commit all pending changes.

Control data transaction to the MY_EMPLOYEE table.

16) Populate the table with the last row of the sample data listed in step 3 by using the statements in the script that you created in step 6. Run the statements in the script.

17) Confirm your addition to the table.

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	3	Drexler	Ben	bbiri	1100
3	4	Newman	Chad	cnewman	1000
4	5	Ropeburn	Audrey	aropebur	1550

18) Mark an intermediate point in the processing of the transaction.

19) Delete all the rows from the MY_EMPLOYEE table.

20) Confirm that the table is empty.

21) Discard the most recent DELETE operation without discarding the earlier INSERT operation.

22) Confirm that the new row is still intact.

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	1	Patel	Ralph	rpatel	1000
2	3	Drexler	Ben	bbiri	1100
3	4	Newman	Chad	cnewman	1000
4	5	Ropeburn	Audrey	aropebur	1550

23) Make the data addition permanent.

If you have the time, complete the following exercise:

24) Modify the lab_09_06.sql script such that the USERID is generated automatically by concatenating the first letter of the first name and the first seven characters of the last name. The generated USERID must be in lowercase. Therefore, the script should not prompt for the USERID. Save this script to a file named lab_09_24.sql.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230

25) Run the lab_09_24.sql script to insert the following record:

26) Confirm that the new row was added with correct USERID.

Practice 9-1: Manipulating Data (continued)

	ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	6	Anthony	Mark	manthony	1230

Practice Solutions 9-1: Manipulating Data

Insert data into the MY_EMPLOYEE table.

1) Run the statement in the lab_09_01.sql script to build the MY_EMPLOYEE table used in this practice.

   a) From File menu, select Open. In the Open dialog box, navigate to the /home/oracle/labs/sql1/labs folder, and then double-click lab_09_01.sql.

   b) After the statement is opened in a SQL Worksheet, click the Run Script icon to run the script. You get a Create Table succeeded message on the Script Output tabbed page.

2) Describe the structure of the MY_EMPLOYEE table to identify the column names.

```
DESCRIBE my_employee
```

3) Create an INSERT statement to add the first row of data to the MY_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause.

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

```
INSERT INTO my_employee
  VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);
```

4) Populate the MY_EMPLOYEE table with the second row of the sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.

```
INSERT INTO my_employee (id, last_name, first_name,
                         userid, salary)
  VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

5) Confirm your additions to the table.

```
SELECT    *
FROM      my_employee;
```

Practice Solutions 9-1: Manipulating Data (continued)

6) Write an INSERT statement in a dynamic reusable script file to load the remaining rows into the MY_EMPLOYEE table. The script should prompt for all the columns (ID, LAST_NAME, FIRST_NAME, USERID, and SALARY). Save this script to a file named lab_09_06.sql.

```
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
        '&p_userid', &p_salary);
```

7) Populate the table with the next two rows of sample data listed in step 3 by running the INSERT statement in the script that you created.

```
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
        '&p_userid', &p_salary);
```

8) Confirm your additions to the table.

```
SELECT   *
FROM my_employee;
```

9) Make the data additions permanent.

```
COMMIT;
```

Update and delete data in the MY_EMPLOYEE table.

10) Change the last name of employee 3 to Drexler.

```
UPDATE   my_employee
SET      last_name = 'Drexler'
WHERE    id = 3;
```

11) Change the salary to $1,000 for all employees with a salary less than $900.

```
UPDATE   my_employee
SET      salary = 1000
WHERE    salary < 900;
```

12) Verify your changes to the table.

```
SELECT   *
FROM     my_employee;
```

13) Delete Betty Dancs from the MY_EMPLOYEE table.

```
DELETE
FROM   my_employee
WHERE last_name = 'Dancs';
```

14) Confirm your changes to the table.

```
SELECT   *
FROM     my_employee;
```

Oracle University and NETEC, S.A. de C.V. use only.

Practice Solutions 9-1: Manipulating Data (continued)

15) Commit all pending changes.

```
COMMIT;
```

Control data transaction to the MY_EMPLOYEE table.

16) Populate the table with the last row of the sample data listed in step 3 by using the statements in the script that you created in step 6. Run the statements in the script.

```
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
    '&p_userid', &p_salary);
```

17) Confirm your addition to the table.

```
SELECT  *
FROM    my_employee;
```

18) Mark an intermediate point in the processing of the transaction.

```
SAVEPOINT step_17;
```

19) Delete all the rows from the MY_EMPLOYEE table.

```
DELETE
FROM  my_employee;
```

20) Confirm that the table is empty.

```
SELECT *
FROM    my_employee;
```

21) Discard the most recent DELETE operation without discarding the earlier INSERT operation.

```
ROLLBACK TO step_17;
```

22) Confirm that the new row is still intact.

```
SELECT *
FROM    my_employee;
```

23) Make the data addition permanent.

```
COMMIT;
```

Practice Solutions 9-1: Manipulating Data (continued)

If you have time, complete the following exercise:

24) Modify the `lab_09_06.sql` script such that the USERID is generated automatically by concatenating the first letter of the first name and the first seven characters of the last name. The generated USERID must be in lowercase. Therefore, the script should not prompt for the USERID. Save this script to a file named `lab_09_24.sql`.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&&p_last_name', '&&p_first_name',
    lower(substr('&p_first_name', 1, 1) ||
    substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
UNDEFINE p_first_name
UNDEFINE p_last_name
```

25) Run the `lab_09_24.sql` script to insert the following record:

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230

26) Confirm that the new row was added with the correct USERID.

```
SELECT *
FROM my_employee
WHERE ID='6';
```

Create new tables by using the `CREATE TABLE` statement. Confirm that the new table was added to the database. You also learn to set the status of a table as `READ ONLY` and then revert to `READ/WRITE`.

Note: For all the `DDL` and `DML` statements, click the Run Script icon (or press [F5]) to execute the query in SQL Developer. This way you get to see the feedback messages on the Script Output tabbed page. For `SELECT` queries, continue to click the Execute Statement icon or press [F9] to get the formatted output on the Results tabbed page.

Practice 10-1: Using DDL Statements to Create and Manage Tables

1) Create the DEPT table based on the following table instance chart. Save the statement in a script called `lab_10_01.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	NAME
Key Type	Primary key	
Nulls/Unique		
FK Table		
FK Column		
Data type	NUMBER	VARCHAR2
Length	7	25

```
Name                            Null     Type
------------------------------- -------- -------------
ID                              NOT NULL NUMBER(7)
NAME                                     VARCHAR2(25)
```

2) Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.

3) Create the EMP table based on the following table instance chart. Save the statement in a script called `lab_10_03.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

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

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

Practice 10-1: Using DDL Statements to Create and Manage Tables (continued)

4) 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.

5) Alter the `EMPLOYEES2` table status to read-only.

6) Try to insert the following row in the `EMPLOYEES2` table:

ID	FIRST_NAME	LAST_NAME	SALARY	DEPT_ID
34	Grant	Marcie	5678	10

You get the following error message:

```
Error starting at line 1 in command:
INSERT INTO employees2
VALUES (34, 'Grant','Marcie',5678,10)
Error at Command Line:1 Column:12
Error report:
SQL Error: ORA-12081: update operation not allowed on table "ORA1"."EMPLOYEES2"
12081. 00000 -  "update operation not allowed on table \"%s\".\"%s\""
*Cause:    An attempt was made to update a read-only materialized view.
*Action:   No action required. Only Oracle is allowed to update a
           read-only materialized view.
```

7) Revert the `EMPLOYEES2` table to the read/write status. Now, try to insert the same row again. You should get the following messages:

```
ALTER TABLE employees2 succeeded.
1 rows inserted
```

8) Drop the `EMPLOYEES2` table.

Practice Solutions 10-1: Using DDL Statements to Create and Manage Tables

1) Create the DEPT table based on the following table instance chart. Save the statement in a script called `lab_10_01.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	NAME
Key Type	Primary key	
Nulls/Unique		
FK Table		
FK Column		
Data type	NUMBER	VARCHAR2
Length	7	25

```
CREATE TABLE dept
 (id   NUMBER(7)CONSTRAINT department_id_pk PRIMARY KEY,
  name VARCHAR2(25));
```

To confirm that the table was created and to view its structure, issue the following command:

```
DESCRIBE dept
```

2) Populate the DEPT table with data from the DEPARTMENTS table. Include only those columns that you need.

```
INSERT INTO dept
  SELECT   department_id, department_name
  FROM     departments;
```

3) Create the EMP table based on the following table instance chart. Save the statement in a script called `lab_10_03.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

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

Practice Solutions 10-1: Using DDL Statements to Create and Manage Tables (continued)

```
CREATE TABLE  emp
 (id             NUMBER(7),
  last_name      VARCHAR2(25),
  first_name     VARCHAR2(25),
  dept_id        NUMBER(7)
    CONSTRAINT emp_dept_id_FK REFERENCES dept (id)
  );
```

To confirm that the table was created and to view its structure:

```
DESCRIBE emp
```

4) 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.

```
CREATE TABLE employees2 AS
  SELECT  employee_id id, first_name, last_name, salary,
          department_id dept_id
  FROM    employees;
```

5) Alter the EMPLOYEES2 table status to read-only.

```
ALTER TABLE employees2 READ ONLY
```

6) Try to insert the following row in the EMPLOYEES2 table.

ID	FIRST_NAME	LAST_NAME	SALARY	DEPT_ID
34	Grant	Marcie	5678	10

Note, you will get the "Update operation not allowed on table" error message. Therefore, you will not be allowed to insert any row into the table because it is assigned a read-only status.

```
INSERT INTO employees2
VALUES (34, 'Grant','Marcie',5678,10)
```

7) Revert the EMPLOYEES2 table to the read/write status. Now try to insert the same row again.

Now, because the table is assigned a READ WRITE status, you will be allowed to insert a row into the table.

```
ALTER TABLE employees2 READ WRITE

INSERT INTO employees2
VALUES (34, 'Grant','Marcie',5678,10)
```

Practice Solutions 10-1: Using DDL Statements to Create and Manage Tables *(continued)*

8) Drop the EMPLOYEES2 table.

> **Note:** You can even drop a table that is in the READ ONLY mode. To test this, alter the table again to READ ONLY status, and then issue the DROP TABLE command. The table EMPLOYEES2 will be dropped.

```
DROP TABLE employees2;
```

Part 1 of this lesson's practice provides you with a variety of exercises in creating, using, and removing views. Complete questions 1–6 of this lesson.

Part 2 of this lesson's practice provides you with a variety of exercises in creating and using a sequence, an index, and a synonym. Complete questions 7–10 of this lesson.

Practice 11-1: Creating Other Schema Objects

Part 1

1) The staff in the HR department wants to hide some of the data in the EMPLOYEES table. Create a view called EMPLOYEES_VU based on the employee numbers, employee last names, and department numbers from the EMPLOYEES table. The heading for the employee name should be EMPLOYEE.

2) Confirm that the view works. Display the contents of the EMPLOYEES_VU view.

	EMPLOYEE_ID	EMPLOYEE	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110
5	206	Gietz	110

...

19	205	Higgins	110
20	206	Gietz	110

3) Using your EMPLOYEES_VU view, write a query for the HR department to display all employee names and department numbers.

	EMPLOYEE	DEPARTMENT_ID
1	King	90
2	Kochhar	90
3	De Haan	90
4	Hunold	60
5	Ernst	60

...

19	Higgins	110
20	Gietz	110

4) Department 50 needs access to its employee data. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. You have been asked to label the view columns EMPNO, EMPLOYEE, and DEPTNO. For security purposes, do not allow an employee to be reassigned to another department through the view.

5) Display the structure and contents of the DEPT50 view.

```
DESCRIBE dept50
Name                            Null      Type
------------------------------- --------  -------------
EMPNO                           NOT NULL  NUMBER(6)
EMPLOYEE                        NOT NULL  VARCHAR2(25)
DEPTNO                                    NUMBER(4)
```

Practice 11-1: Creating Other Schema Objects (continued)

```
EMPNO                    EMPLOYEE                 DEPTNO
--------------------     --------------------     --------------
124                      Mourgos                  50
141                      Rajs                     50
142                      Davies                   50
143                      Matos                    50
144                      Vargas                   50
```

6) Test your view. Attempt to reassign Matos to department 80.

Part 2

7) You need a sequence that can be used with the PRIMARY KEY column of the DEPT table. The sequence should start at 200 and have a maximum value of 1,000. Have your sequence increment by 10. Name the sequence DEPT_ID_SEQ.

8) To test your sequence, write a script to insert two rows in the DEPT table. Name your script lab_11_08.sql. Be sure to use the sequence that you created for the ID column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.

9) Create a nonunique index on the NAME column in the DEPT table.

10) Create a synonym for your EMPLOYEES table. Call it EMP.

Practice Solutions 11-1: Creating Other Schema Objects

Part 1

1) The staff in the HR department wants to hide some of the data in the EMPLOYEES table. Create a view called EMPLOYEES_VU based on the employee numbers, employee last names, and department numbers from the EMPLOYEES table. The heading for the employee name should be EMPLOYEE.

```
CREATE OR REPLACE VIEW employees_vu AS
    SELECT employee_id, last_name employee, department_id
    FROM employees;
```

2) Confirm that the view works. Display the contents of the EMPLOYEES_VU view.

```
SELECT    *
FROM      employees_vu;
```

3) Using your EMPLOYEES_VU view, write a query for the HR department to display all employee names and department numbers.

```
SELECT    employee, department_id
FROM      employees_vu;
```

4) Department 50 needs access to its employee data. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. They have requested that you label the view columns EMPNO, EMPLOYEE, and DEPTNO. For security purposes, do not allow an employee to be reassigned to another department through the view.

```
CREATE VIEW dept50 AS
   SELECT    employee_id empno, last_name employee,
             department_id deptno
   FROM      employees
   WHERE     department_id = 50
   WITH CHECK OPTION CONSTRAINT emp_dept_50;
```

5) Display the structure and contents of the DEPT50 view.

```
DESCRIBE dept50

SELECT    *
FROM      dept50;
```

6) Test your view. Attempt to reassign Matos to department 80.

```
UPDATE    dept50
SET       deptno = 80
WHERE     employee = 'Matos';
```

The error is because the DEPT50 view has been created with the WITH CHECK OPTION constraint. This ensures that the DEPTNO column in the view is protected from being changed.

Practice Solutions 11-1: Creating Other Schema Objects (continued)

Part 2

7) You need a sequence that can be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1,000. Have your sequence increment by 10. Name the sequence DEPT_ID_SEQ.

```
CREATE SEQUENCE dept_id_seq
  START WITH 200
  INCREMENT BY 10
  MAXVALUE 1000;
```

8) To test your sequence, write a script to insert two rows in the DEPT table. Name your script lab_11_08.sql. Be sure to use the sequence that you created for the ID column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.

```
INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Education');

INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Administration');
```

9) Create a nonunique index on the NAME column in the DEPT table.

```
CREATE INDEX dept_name_idx ON dept (name);
```

10) Create a synonym for your EMPLOYEES table. Call it EMP.

```
CREATE SYNONYM emp FOR EMPLOYEES;
```

This practice is intended to give you practical experience in extracting data from more than one table using the Oracle join syntax.

Practice F-1: Oracle Join Syntax

1) Write a query for the HR department to produce the addresses of all the departments. Use the `LOCATIONS` and `COUNTRIES` tables. Show the location ID, street address, city, state or province, and country in the output. Run the query.

	LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1	1400	2014 Jabberwocky Rd	Southlake	Texas	United States of America
2	1500	2011 Interiors Blvd	South San Francisco	California	United States of America
3	1700	2004 Charade Rd	Seattle	Washington	United States of America
4	1800	460 Bloor St. W.	Toronto	Ontario	Canada
5	2500	Magdalen Centre, The Oxford Science Park	Oxford	Oxford	United Kingdom

2) The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all employees. Run the query.

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping

…

18	Higgins	110	Accounting
19	Gietz	110	Accounting

3) The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.

	LAST_NAME	JOB_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	Hartstein	MK_MAN	20	Marketing
2	Fay	MK_REP	20	Marketing

4) Create a report to display the employees' last names and employee number along with their managers' last names and manager number. Label the columns `Employee`, `Emp#`, `Manager`, and `Mgr#`, respectively. Save your SQL statement as `lab_f_04.sql`.

	Employee	EMP#	Manager	Mgr#
1	Hunold	103	De Haan	102
2	Fay	202	Hartstein	201
3	Gietz	206	Higgins	205
4	Lorentz	107	Hunold	103
5	Ernst	104	Hunold	103

…

18	Taylor	176	Zlotkey	149
19	Abel	174	Zlotkey	149

Practice F-1: Oracle Join Syntax (continued)

5) Modify `lab_f_04.sql` to display all employees including `King`, who has no manager. Order the results by the employee number. Save your SQL statement as `lab_f_05.sql`. Run the query in `lab_f_05.sql`.

	Employee	EMP#	Manager	Mgr#
1	Hunold	103	De Haan	102
2	Fay	202	Hartstein	201
3	Gietz	206	Higgins	205
4	Lorentz	107	Hunold	103
5	Ernst	104	Hunold	103

...

	Employee	EMP#	Manager	Mgr#
19	Abel	174	Zlotkey	149
20	King	100	(null)	(null)

6) Create a report for the HR department that displays employee last names, department numbers, and all employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named `lab_f_06.sql`.

	DEPARTMENT	EMPLOYEE	COLLEAGUE
1	20	Fay	Hartstein
2	20	Hartstein	Fay
3	50	Davies	Matos
4	50	Davies	Mourgos
5	50	Davies	Rajs

...

	DEPARTMENT	EMPLOYEE	COLLEAGUE
39	90	Kochhar	De Haan
40	90	Kochhar	King
41	110	Gietz	Higgins
42	110	Higgins	Gietz

7) The HR department needs a report on job grades and salaries. To familiarize yourself with the `JOB_GRADES` table, first show the structure of the `JOB_GRADES` table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
Name                              Null      Type
--------------------------------  --------  -----------
GRADE_LEVEL                                 VARCHAR2(3)
LOWEST_SAL                                  NUMBER
HIGHEST_SAL                                 NUMBER
```

Practice F-1: Oracle Join Syntax (continued)

	LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
1	King	AD_PRES	Executive	24000	E
2	De Haan	AD_VP	Executive	17000	E
3	Kochhar	AD_VP	Executive	17000	E
4	Hartstein	MK_MAN	Marketing	13000	D
5	Higgins	AC_MGR	Accounting	12000	D

...

	LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRADE_LEVEL
18	Matos	ST_CLERK	Shipping	2600	A
19	Vargas	ST_CLERK	Shipping	2500	A

If you want an extra challenge, complete the following exercises:

8) The HR department wants to determine the names of all employees who were hired after `Davies`. Create a query to display the name and hire date of any employee hired after employee `Davies`.

	LAST_NAME	HIRE_DATE
1	Lorentz	07-FEB-99
2	Mourgos	16-NOV-99
3	Matos	15-MAR-98
4	Vargas	09-JUL-98
5	Zlotkey	29-JAN-00
6	Taylor	24-MAR-98
7	Grant	24-MAY-99
8	Fay	17-AUG-97

9) The HR department needs to find the names and hire dates for all employees who were hired before their managers, along with their managers' names and hire dates. Save the script to a file named `lab_f_09.sql`.

	LAST_NAME	HIRE_DATE	LAST_NAME_1	HIRE_DATE_1
1	Whalen	17-SEP-87	Kochhar	21-SEP-89
2	Hunold	03-JAN-90	De Haan	13-JAN-93
3	Vargas	09-JUL-98	Mourgos	16-NOV-99
4	Matos	15-MAR-98	Mourgos	16-NOV-99
5	Davies	29-JAN-97	Mourgos	16-NOV-99
6	Rajs	17-OCT-95	Mourgos	16-NOV-99
7	Grant	24-MAY-99	Zlotkey	29-JAN-00
8	Taylor	24-MAR-98	Zlotkey	29-JAN-00
9	Abel	11-MAY-96	Zlotkey	29-JAN-00

Practice Solutions F-1: Oracle Join Syntax

1) Write a query for the HR department to produce the addresses of all the departments. Use the LOCATIONS and COUNTRIES tables. Show the location ID, street address, city, state or province, and country in the output. Run the query.

```
SELECT location_id, street_address, city, state_province,
country_name
FROM    locations, countries
WHERE   locations.country_id = countries.country_id;
```

2) The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all employees. Run the query.

```
SELECT e.last_name, e.department_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id = d.department_id;
```

3) The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.

```
SELECT e.last_name, e.job_id, e.department_id,
d.department_name
FROM    employees e, departments d , locations l
WHERE   e.department_id = d.department_id
AND     d.location_id = l.location_id
AND     LOWER(l.city) = 'toronto';
```

4) Create a report to display the employee last name and the employee number along with the last name of the employee's manager and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Save your SQL statement as lab_f_04.sql.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM    employees w, employees m
WHERE   w.manager_id = m.employee_id;
```

5) Modify lab_f_04.sql to display all employees including King, who has no manager. Order the results by the employee number. Save the SQL statement as lab_f_05.sql. Run the query in lab_f_05.sql.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM    employees w, employees m
WHERE   w.manager_id  = m.employee_id (+);
```

Practice Solutions F-1: Oracle Join Syntax (continued)

6) Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named `lab_f_06.sql`.

```
SELECT e1.department_id department, e1.last_name employee,
       e2.last_name colleague
FROM   employees e1, employees e2
WHERE  e1.department_id = e2.department_id
AND    e1.employee_id <> e2.employee_id
ORDER BY e1.department_id, e1.last_name, e2.last_name;
```

7) The HR department needs a report on job grades and salaries. To familiarize yourself with the JOB_GRADES table, first show the structure of the JOB_GRADES table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES

SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
FROM   employees e, departments d, job_grades j
WHERE  e.department_id = d.department_id
AND    e.salary BETWEEN j.lowest_sal AND j.highest_sal;
```

If you want an extra challenge, complete the following exercises:

8) The HR department wants to determine the names of all employees hired after `Davies`. Create a query to display the name and hire date of any employee hired after `Davies`.

```
SELECT e.last_name, e.hire_date
FROM   employees e , employees davies
WHERE  davies.last_name = 'Davies'
AND    davies.hire_date < e.hire_date;
```

9) The HR department needs to find the names and hire dates for all employees who were hired before their managers, along with their managers' names and hire dates. Label the columns `Employee`, `Emp Hired`, `Manager`, and `Mgr Hired`, respectively. Save the script to a file named `lab_f_09.sql`.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w , employees m
WHERE  w.manager_id = m.employee_id
AND    w.hire_date <  m.hire_date;
```

B

Table Descriptions

Oracle University and NETEC , S.A. de C.V. use only.

Schema Description

Overall Description

The Oracle Database sample schemas portray a sample company that operates worldwide to fill orders for several different products. The company has three divisions:

- **Human Resources:** Tracks information about the employees and facilities
- **Order Entry:** Tracks product inventories and sales through various channels
- **Sales History:** Tracks business statistics to facilitate business decisions

Each of these divisions is represented by a schema. In this course, you have access to the objects in all the schemas. However, the emphasis of the examples, demonstrations, and practices is on the `Human Resources (HR)` schema.

All scripts necessary to create the sample schemas reside in the `$ORACLE_HOME/demo/schema/` folder.

Human Resources (HR)

This is the schema that is used in this course. In the Human Resource (HR) records, each employee has an identification number, email address, job identification code, salary, and manager. Some employees earn commissions in addition to their salary.

The company also tracks information about jobs within the organization. Each job has an identification code, job title, and a minimum and maximum salary range for the job. Some employees have been with the company for a long time and have held different positions within the company. When an employee resigns, the duration the employee was working for, the job identification number, and the department are recorded.

The sample company is regionally diverse, so it tracks the locations of its warehouses and departments. Each employee is assigned to a department, and each department is identified either by a unique department number or a short name. Each department is associated with one location, and each location has a full address that includes the street name, postal code, city, state or province, and the country code.

In places where the departments and warehouses are located, the company records details such as the country name, currency symbol, currency name, and the region where the country is located geographically.

HR

DEPARTMENTS
department_id
department_name
manager_id
location_id

LOCATIONS
location_id
street_address
postal_code
city
state_province
country_id

JOB_HISTORY
employee_id
start_date
end_date
job_id
department_id

EMPLOYEES
employee_id
first_name
last_name
email
phone_number
hire_date
job_id
salary
commission_pct
manager_id
department_id

COUNTRIES
country_id
country_name
region_id

JOBS
job_id
job_title
min_salary
max_salary

REGIONS
region_id
region_name

Human Resources (HR) Table Descriptions

DESCRIBE countries

```
Name                            Null      Type
------------------------------- --------- ---------------
COUNTRY_ID                      NOT NULL  CHAR(2)
COUNTRY_NAME                              VARCHAR2(40)
REGION_ID                                NUMBER
```

SELECT * FROM countries;

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	CA	Canada	2
2	DE	Germany	1
3	UK	United Kingdom	1
4	US	United States of America	2

Human Resources (HR) Table Descriptions (continued)

DESCRIBE departments

```
Name                            Null      Type
------------------------------- --------  ---------------
DEPARTMENT_ID                   NOT NULL  NUMBER(4)
DEPARTMENT_NAME                 NOT NULL  VARCHAR2(30)
MANAGER_ID                                NUMBER(6)
LOCATION_ID                               NUMBER(4)
```

SELECT * FROM departments;

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

DESCRIBE employees

```
Name                             Null      Type
------------------------------   --------  ----------------
EMPLOYEE_ID                      NOT NULL  NUMBER(6)
FIRST_NAME                                 VARCHAR2(20)
LAST_NAME                        NOT NULL  VARCHAR2(25)
EMAIL                            NOT NULL  VARCHAR2(25)
PHONE_NUMBER                               VARCHAR2(20)
HIRE_DATE                        NOT NULL  DATE
JOB_ID                           NOT NULL  VARCHAR2(10)
SALARY                                     NUMBER(8,2)
COMMISSION_PCT                             NUMBER(2,2)
MANAGER_ID                                 NUMBER(6)
DEPARTMENT_ID                              NUMBER(4)
```

SELECT * FROM employees;

	EMPLOYEE_ID	FIRST_...	LAS...	EMAIL	PHONE_N...	HIRE_DATE	JOB_ID	SALARY	COMMISSIO...	MANAG...	DEPAR...
1	100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	(null)	(null)	90
2	101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	(null)	100	90
3	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	(null)	100	90
4	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	(null)	102	60
5	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	(null)	103	60
6	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	4200	(null)	103	60
7	124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	5800	(null)	100	50
8	141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	3500	(null)	124	50
9	142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	3100	(null)	124	50
10	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600	(null)	124	50
11	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500	(null)	124	50
12	149	Eleni	Zlotkey	EZLOTKEY	011.44.1344....	29-JAN-00	SA_MAN	10500	0.2	100	80
13	174	Ellen	Abel	EABEL	011.44.1644....	11-MAY-96	SA_REP	11000	0.3	149	80
14	176	Jonathon	Taylor	JTAYLOR	011.44.1644....	24-MAR-98	SA_REP	8600	0.2	149	80
15	178	Kimberely	Grant	KGRANT	011.44.1644....	24-MAY-99	SA_REP	7000	0.15	149	(null)
16	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	(null)	101	10
17	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	13000	(null)	100	20
18	202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	6000	(null)	201	20
19	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	12000	(null)	101	110
20	206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	8300	(null)	205	110

Human Resources (HR) Table Descriptions (continued)

DESCRIBE job_history

```
Name                             Null      Type
-------------------------------- --------  --------------
EMPLOYEE_ID                      NOT NULL  NUMBER(6)
START_DATE                       NOT NULL  DATE
END_DATE                         NOT NULL  DATE
JOB_ID                           NOT NULL  VARCHAR2(10)
DEPARTMENT_ID                              NUMBER(4)
```

SELECT * FROM job_history

	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
1	102	13-JAN-93	24-JUL-98	IT_PROG	60
2	101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
3	101	28-OCT-93	15-MAR-97	AC_MGR	110
4	201	17-FEB-96	19-DEC-99	MK_REP	20
5	114	24-MAR-98	31-DEC-99	ST_CLERK	50
6	122	01-JAN-99	31-DEC-99	ST_CLERK	50
7	200	17-SEP-87	17-JUN-93	AD_ASST	90
8	176	24-MAR-98	31-DEC-98	SA_REP	80
9	176	01-JAN-99	31-DEC-99	SA_MAN	80
10	200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

DESCRIBE jobs

```
Name                          Null      Type
----------------------------- --------  --------------
JOB_ID                        NOT NULL  VARCHAR2(10)
JOB_TITLE                     NOT NULL  VARCHAR2(35)
MIN_SALARY                              NUMBER(6)
MAX_SALARY                              NUMBER(6)
```

SELECT * FROM jobs

	JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
1	AD_PRES	President	20000	40000
2	AD_VP	Administration Vice President	15000	30000
3	AD_ASST	Administration Assistant	3000	6000
4	AC_MGR	Accounting Manager	8200	16000
5	AC_ACCOUNT	Public Accountant	4200	9000
6	SA_MAN	Sales Manager	10000	20000
7	SA_REP	Sales Representative	6000	12000
8	ST_MAN	Stock Manager	5500	8500
9	ST_CLERK	Stock Clerk	2000	5000
10	IT_PROG	Programmer	4000	10000
11	MK_MAN	Marketing Manager	9000	15000
12	MK_REP	Marketing Representative	4000	9000

Human Resources (HR) Table Descriptions (continued)

DESCRIBE locations

```
Name                            Null      Type
------------------------------- --------  ----------------
LOCATION_ID                     NOT NULL  NUMBER(4)
STREET_ADDRESS                            VARCHAR2(40)
POSTAL_CODE                               VARCHAR2(12)
CITY                            NOT NULL  VARCHAR2(30)
STATE_PROVINCE                            VARCHAR2(25)
COUNTRY_ID                                CHAR(2)
```

SELECT * FROM locations

	LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
1	1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
2	1500	2011 Interiors Blvd	99236	South San Francisco	California	US
3	1700	2004 Charade Rd	98199	Seattle	Washington	US
4	1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
5	2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

DESCRIBE regions

```
Name                              Null      Type
-----------------------------     --------  --------------
REGION_ID                         NOT NULL  NUMBER
REGION_NAME                                 VARCHAR2(25)
```

SELECT * FROM regions

	REGION_ID	REGION_NAME
1	1	Europe
2	2	Americas
3	3	Asia
4	4	Middle East and Africa

C

Using SQL Developer

Oracle University and NETEC, S.A. de C.V. use only.

Objectives

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

- List the key features of Oracle SQL Developer
- Identify the menu items of Oracle SQL Developer
- Create a database connection
- Manage database objects
- Use SQL Worksheet
- Save and run SQL scripts
- Create and save reports

Objectives

In this appendix, you are introduced to the graphical tool called SQL Developer. You learn how to use SQL Developer for your database development tasks. You learn how to use SQL Worksheet to execute SQL statements and SQL scripts.

What Is Oracle SQL Developer?

- Oracle SQL Developer is a graphical tool that enhances productivity and simplifies database development tasks.
- You can connect to any target Oracle database schema by using standard Oracle database authentication.
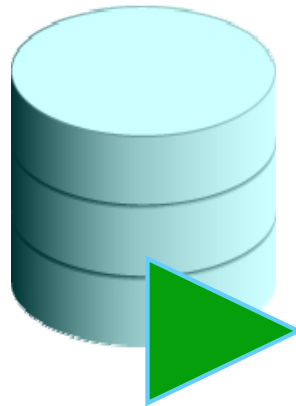
SQL Developer

What Is Oracle SQL Developer?

Oracle SQL Developer is a free graphical tool designed to improve your productivity and simplify the development of everyday database tasks. With just a few clicks, you can easily create and debug stored procedures, test SQL statements, and view optimizer plans.

SQL Developer, which is the visual tool for database development, simplifies the following tasks:

- Browsing and managing database objects
- Executing SQL statements and scripts
- Editing and debugging PL/SQL statements
- Creating reports

You can connect to any target Oracle database schema by using standard Oracle database authentication. When connected, you can perform operations on objects in the database.

The SQL Developer 1.2 release tightly integrates with *Developer Migration Workbench* that provides users with a single point to browse database objects and data in third-party databases, and to migrate from these databases to Oracle. You can also connect to schemas for selected third-party (non-Oracle) databases, such as MySQL, Microsoft SQL Server, and Microsoft Access, and view metadata and data in these databases.

Additionally, SQL Developer includes support for Oracle Application Express 3.0.1 (Oracle APEX).

Specifications of SQL Developer

- Shipped along with Oracle Database 11*g* Release 2
- Developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Enables default connectivity using the JDBC Thin driver
- Connects to Oracle Database version 9.2.0.1 and later
- Freely downloadable from the following link:
  - http://www.oracle.com/technology/products/database/ sql_developer/index.html

ORACLE

Specifications of SQL Developer

Oracle SQL Developer 1.5 is shipped along with Oracle Database 11*g* Release 2. SQL Developer is developed in Java leveraging the Oracle JDeveloper integrated development environment (IDE). Therefore, it is a cross-platform tool. The tool runs on Windows, Linux, and Mac operating system (OS) X platforms.

The default connectivity to the database is through the Java Database Connectivity (JDBC) Thin driver, and therefore, no Oracle Home is required. SQL Developer does not require an installer and you need to simply unzip the downloaded file. With SQL Developer, users can connect to Oracle Databases 9.2.0.1 and later, and all Oracle database editions including Express Edition.

Note

For Oracle Database versions before Oracle Database 11*g* Release 2, you will have to download and install SQL Developer. SQL Developer 1.5 is freely downloadable from the following link:
 http://www.oracle.com/technology/products/database/sql_developer/index.html.

For instructions on how to install SQL Developer, see the Web site at:
 http://download.oracle.com/docs/cd/E12151_01/index.htm

SQL Developer 1.5 Interface
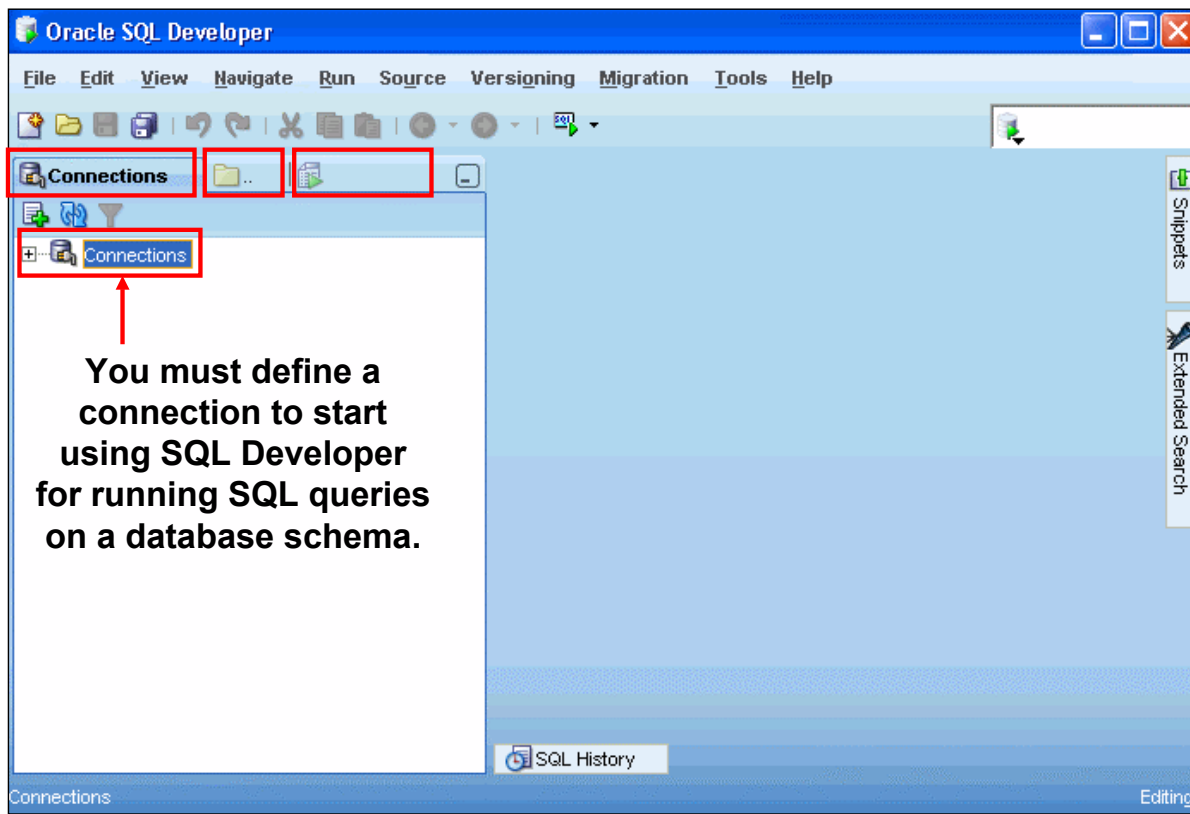
SQL Developer 1.5 Interface

The SQL Developer 1.5 interface contains three main navigation tabs, from left to right:
- **Connections tab:** By using this tab, you can browse database objects and users to which you have access.
- **Files tab:** Identified by the Files folder icon, this tab enables you to access files from your local machine without having to use the File > Open menu.
- **Reports tab:** Identified by the Reports icon, this tab enables you to run predefined reports or create and add your own reports.

General Navigation and Use

SQL Developer uses the left side for navigation to find and select objects, and the right side to display information about selected objects. You can customize many aspects of the appearance and behavior of SQL Developer by setting preferences.

Note: You need to define at least one connection to be able to connect to a database schema and issue SQL queries or run procedures/functions.

Menus

The following menus contain standard entries, plus entries for features specific to SQL Developer:

- **View:** Contains options that affect what is displayed in the SQL Developer interface
- **Navigate:** Contains options for navigating to panes and for executing subprograms
- **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected, and also debugging options.
- **Source:** Contains options for use when you edit functions and procedures
- **Versioning:** Provides integrated support for the following versioning and source control systems: CVS (Concurrent Versions System) and Subversion.
- **Migration:** Contains options related to migrating third-party databases to Oracle
- **Tools:** Invokes SQL Developer tools such as SQL*Plus, Preferences, and SQL Worksheet

Note: The Run menu also contains options that are relevant when a function or procedure is selected for debugging. These are the same options that are found in the Debug menu in version 1.2.

Creating a Database Connection

- You must have at least one database connection to use SQL Developer.
- You can create and test connections for:
  - Multiple databases
  - Multiple schemas
- SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.
- You can export connections to an Extensible Markup Language (XML) file.
- Each additional database connection created is listed in the Connections Navigator hierarchy.

Creating a Database Connection

A connection is a SQL Developer object that specifies the necessary information for connecting to a specific database as a specific user of that database. To use SQL Developer, you must have at least one database connection, which may be existing, created, or imported.

You can create and test connections for multiple databases and for multiple schemas.

By default, the `tnsnames.ora` file is located in the `$ORACLE_HOME/network/admin` directory, but it can also be in the directory specified by the `TNS_ADMIN` environment variable or registry value. When you start SQL Developer and display the Database Connections dialog box, SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.

Note: On Windows, if the `tnsnames.ora` file exists, but its connections are not being used by SQL Developer, define `TNS_ADMIN` as a system environment variable.

You can export connections to an XML file so that you can reuse it.

You can create additional connections as different users to the same database or to connect to the different databases.

Creating a Database Connection
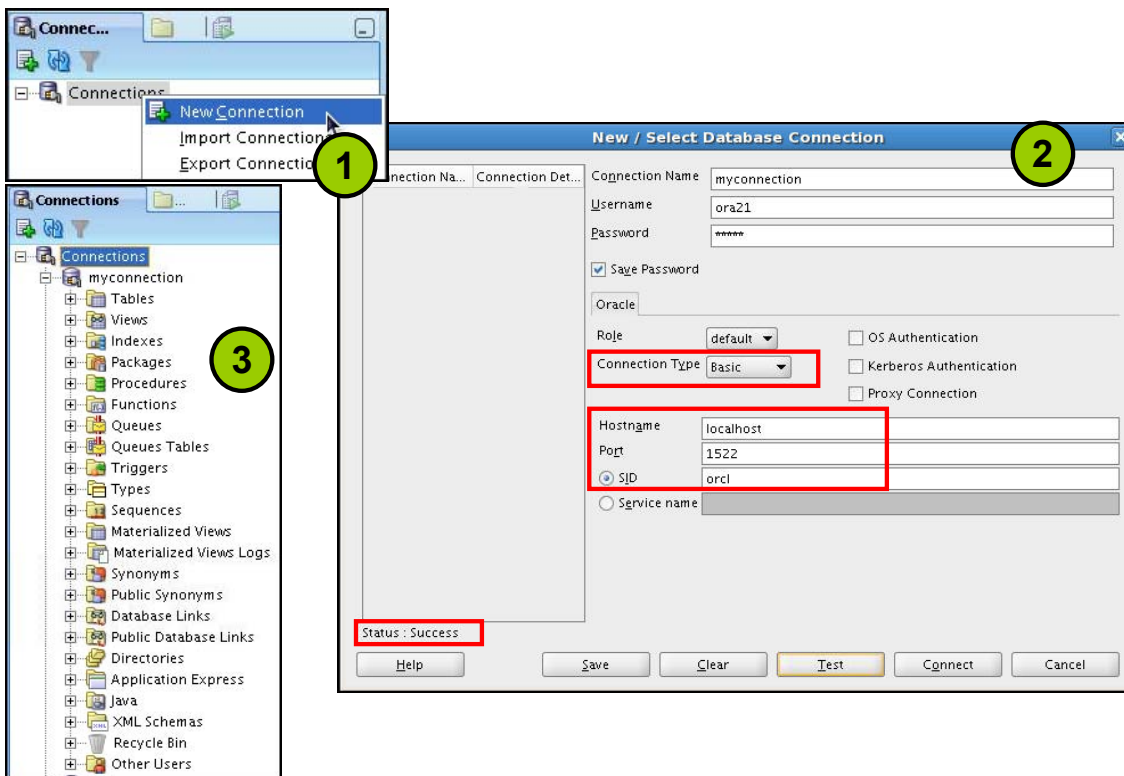
Creating a Database Connection (continued)

To create a database connection, perform the following steps:

1. On the Connections tabbed page, right-click **Connections** and select **New Connection**.
2. In the New/Select Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to.
   a) From the Role drop-down list, you can select either *default* or SYSDBA. (You choose SYSDBA for the sys user or any user with database administrator privileges.)
   b) You can select the connection type as:
      Basic: In this type, enter host name and SID for the database you want to connect to. Port is already set to 1521. You can also choose to enter the Service name directly if you use a remote database connection.
      TNS: You can select any one of the database aliases imported from the tnsnames.ora file.
      LDAP: You can look up database services in Oracle Internet Directory, which is a component of Oracle Identity Management.
      Advanced: You can define a custom Java Database Connectivity (JDBC) URL to connect to the database.
   c) Click Test to ensure that the connection has been set correctly.
   d) Click Connect.

Creating a Database Connection (continued)

> If you select the Save Password check box, the password is saved to an XML file. So, after you close the SQL Developer connection and open it again, you are not prompted for the password.

> 3. The connection gets added in the Connections Navigator. You can expand the connection to view the database objects and view object definitions—for example, dependencies, details, statistics, and so on.

Note: From the same New/Select Database Connection window, you can define connections to non-Oracle data sources using the Access, MySQL, and SQL Server tabs. However, these connections are read-only connections that enable you to browse objects and data in that data source.

Browsing Database Objects

Use the Connections Navigator to:
- Browse through many objects in a database schema
- Review the definitions of objects at a glance
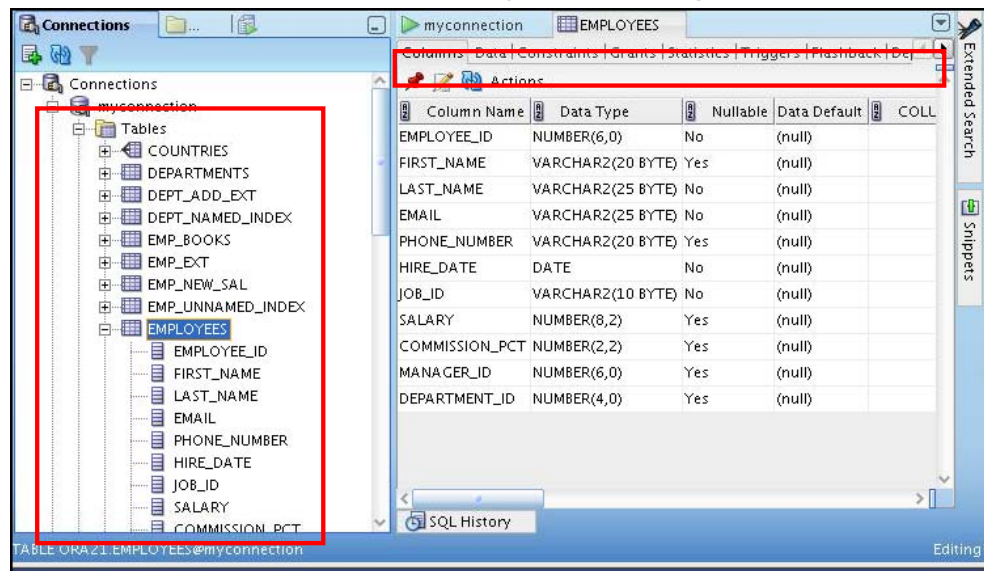
Browsing Database Objects

After you create a database connection, you can use the Connections Navigator to browse through many objects in a database schema including Tables, Views, Indexes, Packages, Procedures, Triggers, and Types.

SQL Developer uses the left side for navigation to find and select objects, and the right side to display information about the selected objects. You can customize many aspects of the appearance of SQL Developer by setting preferences.
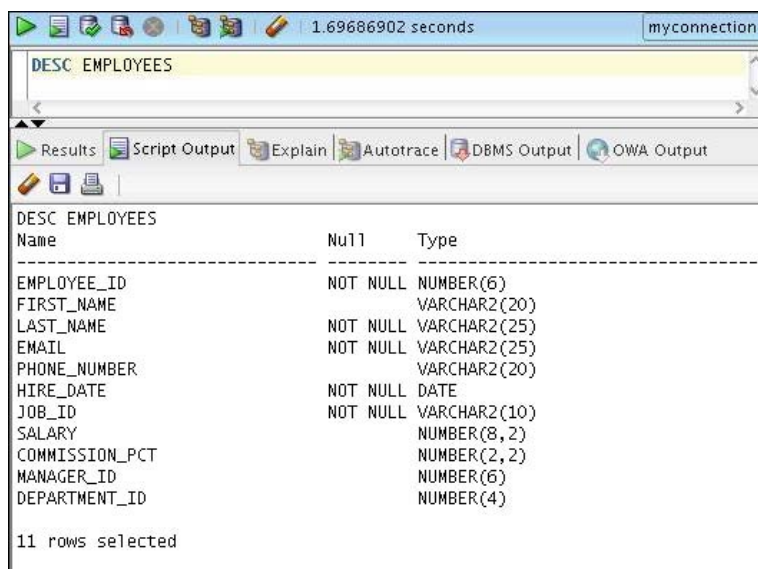
You can see the definition of the objects broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, the details about columns, constraints, grants, statistics, triggers, and so on are displayed on an easy-to-read tabbed page.

If you want to see the definition of the EMPLOYEES table as shown in the slide, perform the following steps:
1. Expand the Connections node in the Connections Navigator.
2. Expand Tables.
3. Click EMPLOYEES. By default, the Columns tab is selected. It shows the column description of the table. Using the Data tab, you can view the table data and also enter new rows, update data, and commit these changes to the database.

Displaying the Table Structure

Use the `DESCRIBE` command to display the structure of a table:

Displaying the Table Structure

In SQL Developer, you can also display the structure of a table using the `DESCRIBE` command. The result of the command is a display of column names and data types as well as an indication if a column must contain data.

Browsing Files

Use the File Navigator to explore the file system and open system files.
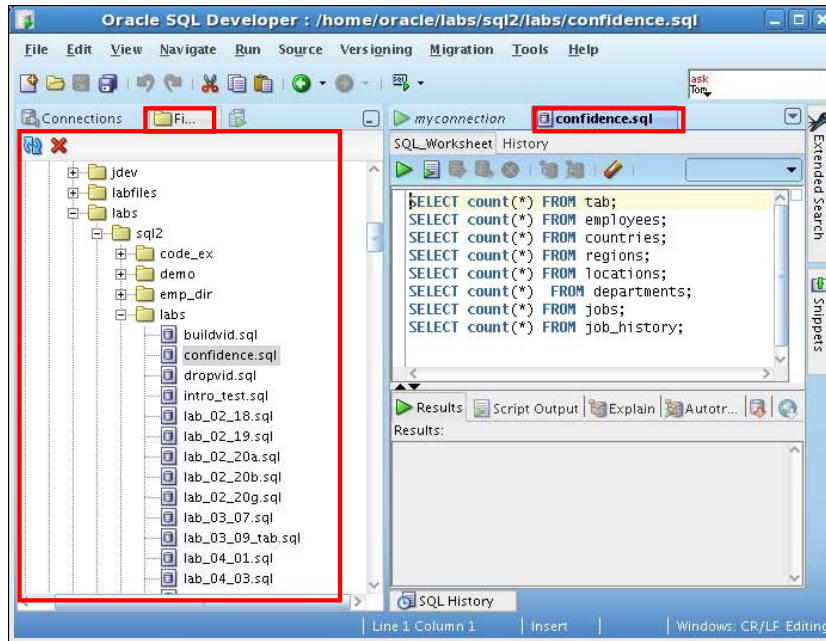
Browsing Database Objects

You can use the File Navigator to browse and open system files.

- To view the files navigator, click the Files tab, or select View > Files.
- To view the contents of a file, double-click a file name to display its contents in the SQL worksheet area.

Creating a Schema Object

- SQL Developer supports the creation of any schema object by:
  – Executing a SQL statement in SQL Worksheet
  – Using the context menu
- Edit the objects by using an edit dialog box or one of the many context-sensitive menus.
- View the data definition language (DDL) for adjustments such as creating a new object or editing an existing schema object.
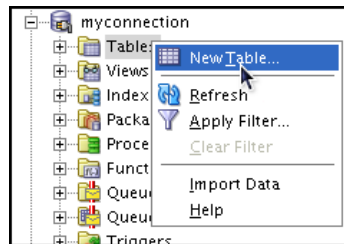
Creating a Schema Object

SQL Developer supports the creation of any schema object by executing a SQL statement in SQL Worksheet. Alternatively, you can create objects using the context menus. When created, you can edit the objects using an edit dialog box or one of the many context-sensitive menus.

As new objects are created or existing objects are edited, the DDL for those adjustments is available for review. An Export DDL option is available if you want to create the full DDL for one or more objects in the schema.

The slide shows how to create a table using the context menu. To open a dialog box for creating a new table, right-click Tables and select New Table. The dialog boxes to create and edit database objects have multiple tabs, each reflecting a logical grouping of properties for that type of object.

Creating a New Table: Example
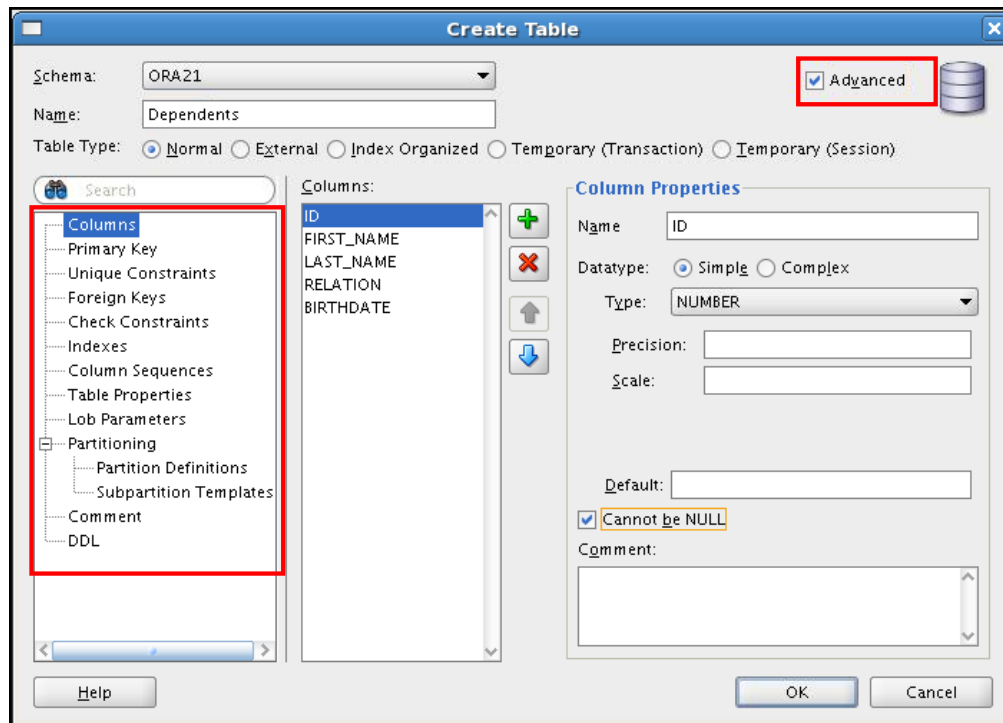
Creating a New Table: Example

In the Create Table dialog box, if you do not select the Advanced check box, you can create a table quickly by specifying columns and some frequently used features.

If you select the Advanced check box, the Create Table dialog box changes to one with multiple options, in which you can specify an extended set of features while you create the table.

The example in the slide shows how to create the DEPENDENTS table by selecting the Advanced check box.

To create a new table, perform the following steps:
1. In the Connections Navigator, right-click Tables.
2. Select Create TABLE.
3. In the Create Table dialog box, select Advanced.
4. Specify the column information.
5. Click OK.

Although it is not required, you should also specify a primary key by using the Primary Key tab in the dialog box. Sometimes, you may want to edit the table that you have created; to do so, right-click the table in the Connections Navigator and select Edit.

Using the SQL Worksheet

- Use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL *Plus statements.
- Specify any actions that can be processed by the database connection associated with the worksheet.
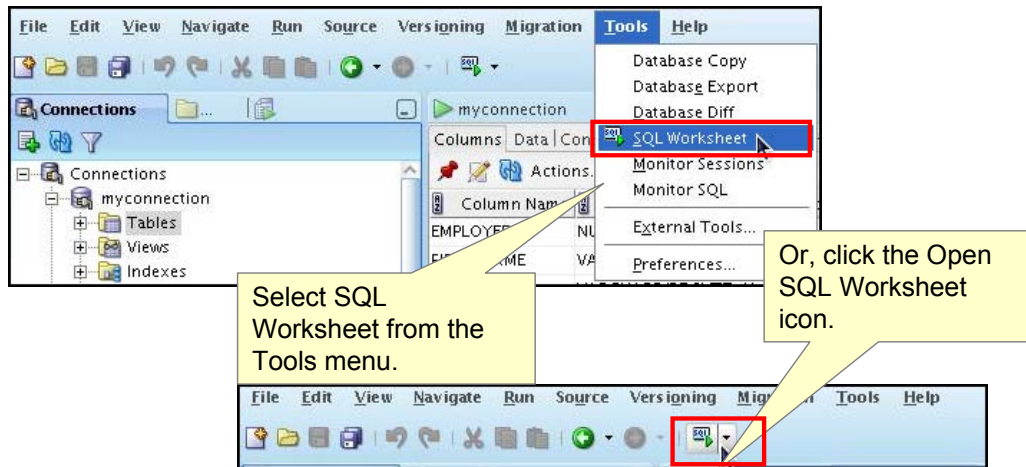
Using the SQL Worksheet

When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. The SQL Worksheet supports SQL*Plus statements to a certain extent. SQL*Plus statements that are not supported by the SQL Worksheet are ignored and not passed to the database.

You can specify actions that can be processed by the database connection associated with the worksheet, such as:
- Creating a table
- Inserting data
- Creating and editing a trigger
- Selecting data from a table
- Saving the selected data to a file

You can display a SQL Worksheet by using one of the following:
- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon.

Using the SQL Worksheet
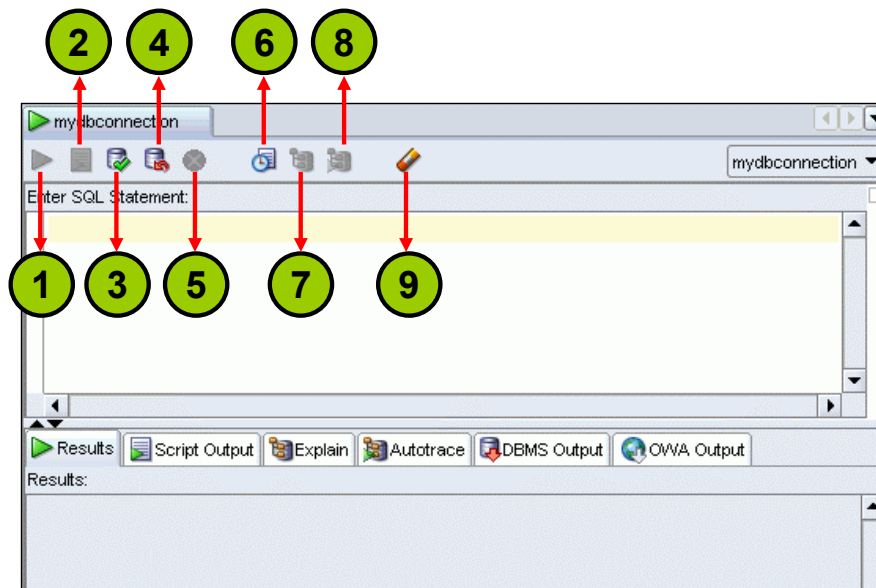
Using the SQL Worksheet (continued)

You may want to use the shortcut keys or icons to perform certain tasks such as executing a SQL statement, running a script, and viewing the history of SQL statements that you have executed. You can use the SQL Worksheet toolbar that contains icons to perform the following tasks:

1. **Execute Statement:** Executes the statement where the cursor is located in the Enter SQL Statement box. You can use bind variables in the SQL statements, but not substitution variables.
2. **Run Script:** Executes all statements in the Enter SQL Statement box by using the Script Runner. You can use substitution variables in the SQL statements, but not bind variables.
3. **Commit:** Writes any changes to the database and ends the transaction
4. **Rollback:** Discards any changes to the database, without writing them to the database, and ends the transaction
5. **Cancel:** Stops the execution of any statements currently being executed
6. **SQL History:** Displays a dialog box with information about SQL statements that you have executed
7. **Execute Explain Plan:** Generates the execution plan, which you can see by clicking the Explain tab
8. **Autotrace:** Generates trace information for the statement
9. **Clear:** Erases the statement or statements in the Enter SQL Statement box

Using the SQL Worksheet

Enter SQL statements.

Results are shown here.
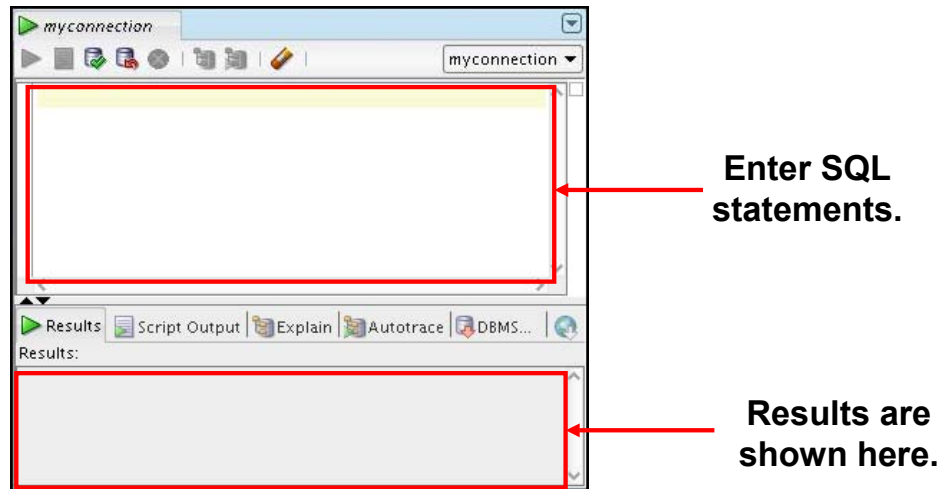
Using the SQL Worksheet (continued)

When you connect to a database, a SQL Worksheet window for that connection automatically opens. You can use the SQL Worksheet to enter and execute SQL, PL/SQL, and SQL*Plus statements. All SQL and PL/SQL commands are supported as they are passed directly from the SQL Worksheet to the Oracle database. SQL*Plus commands used in the SQL Developer have to be interpreted by the SQL Worksheet before being passed to the database.

The SQL Worksheet currently supports a number of SQL*Plus commands. Commands not supported by the SQL Worksheet are ignored and are not sent to the Oracle database. Through the SQL Worksheet, you can execute SQL statements and some of the SQL*Plus commands.

You can display a SQL Worksheet by using any of the following options:
- Select Tools > SQL Worksheet.
- Click the Open SQL Worksheet icon.

Executing SQL Statements

Use the Enter SQL Statement box to enter single or multiple SQL statements.

Executing SQL Statements

The example in the slide shows the difference in output for the same query when the [F9] key or Execute Statement is used versus the output when [F5] or Run Script is used.

Saving SQL Scripts

1 Click the Save icon to save your SQL statement to a file.

2 Identify a location, enter a file name, and click Save.

3 The contents of the saved file are visible and editable in your SQL Worksheet window.
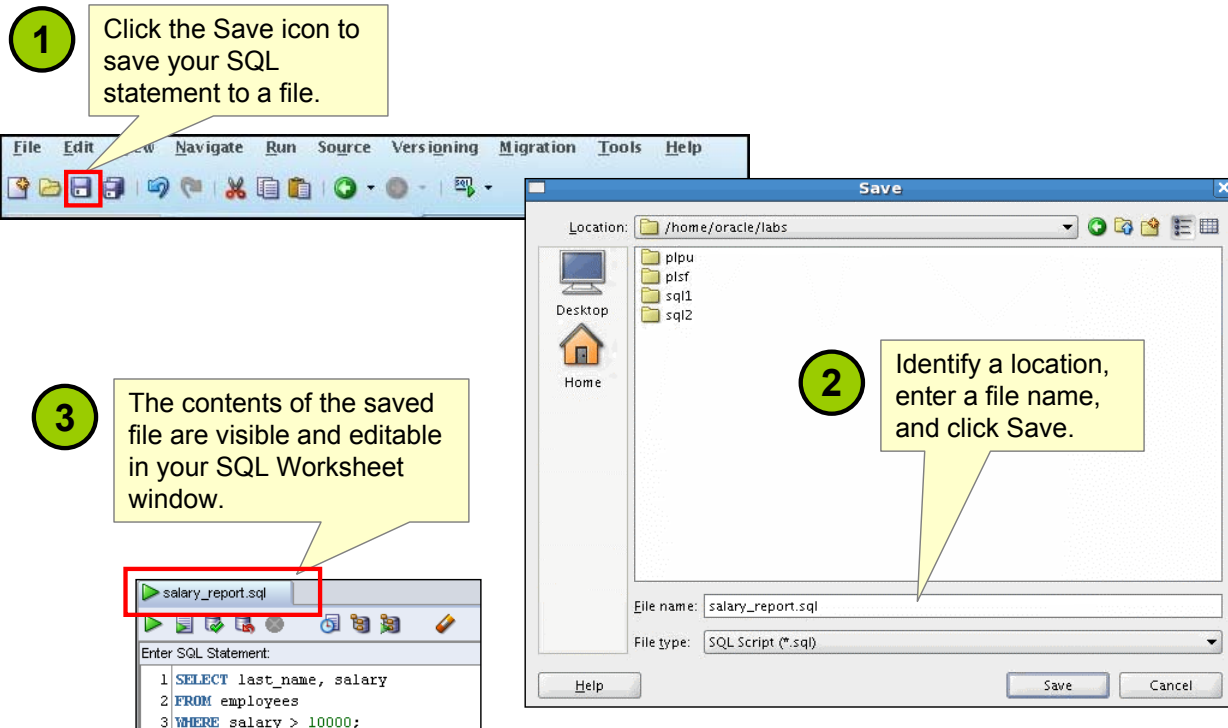
Saving SQL Scripts

You can save your SQL statements from the SQL Worksheet into a text file. To save the contents of the Enter SQL Statement box, perform the following steps:

1. Click the Save icon or use the File > Save menu item.
2. In the Save dialog box, enter a file name and the location where you want the file saved.
3. Click Save.

After you save the contents to a file, the Enter SQL Statement window displays a tabbed page of your file contents. You can have multiple files open at the same time. Each file displays as a tabbed page.

Script Pathing

You can select a default path to look for scripts and to save scripts. Under Tools > Preferences > Database > Worksheet Parameters, enter a value in the "Select default path to look for scripts" field.

Executing Saved Script Files: Method 1

1. Use the Files tab to locate the script file that you want to open.
2. Double-click the script to display the code in the SQL Worksheet.

To run the code, click either:
- Execute Script (F9), or
- Run Script (F5)

Select a connection from the drop-down list.

```
SELECT count(*) FROM tab;
SELECT count(*) FROM employees;
SELECT count(*) FROM countries;
SELECT count(*) FROM regions;
SELECT count(*) FROM locations;
SELECT count(*)  FROM departments;
SELECT count(*) FROM jobs;
SELECT count(*) FROM job_history;
```
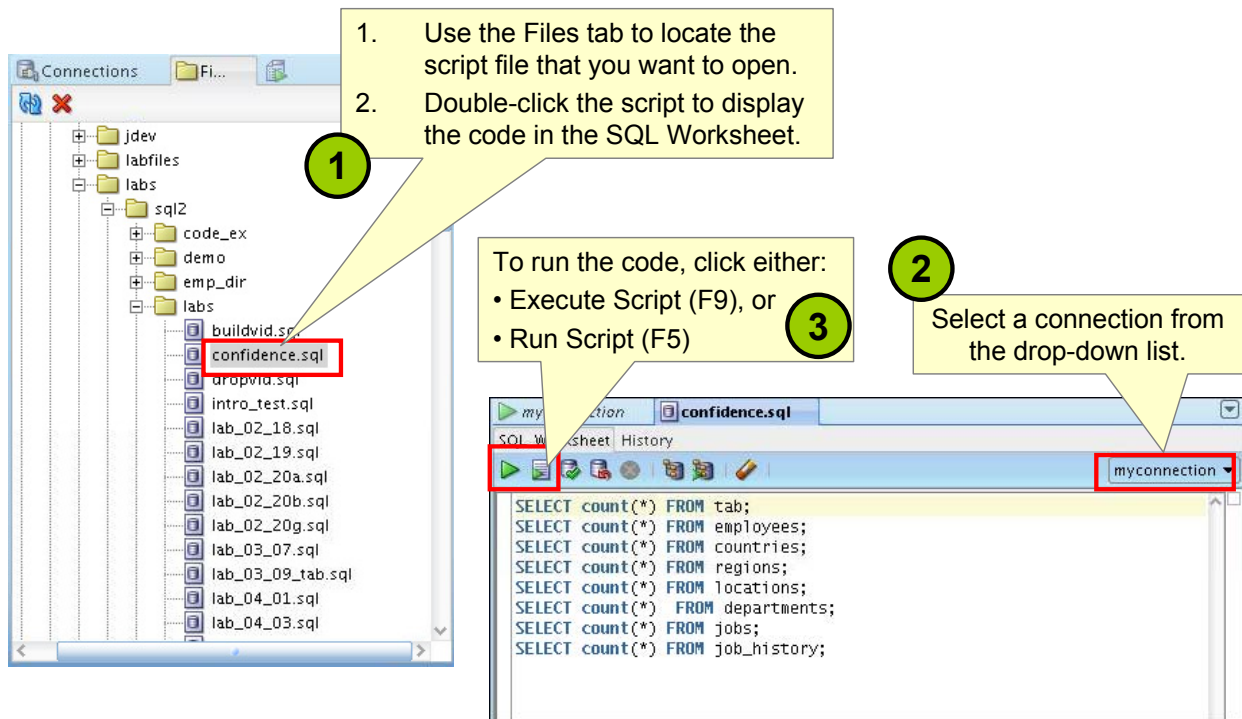
Executing Saved Script Files: Method 1

To open a script file and display the code in the SQL Worksheet area, perform the following steps:

1. In the files navigator, select (or navigate to) the script file that you want to open.
2. Double-click to open. The code of the script file is displayed in the SQL Worksheet area.
3. Select a connection from the connection drop-down list.
4. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection you want to use for the script execution.

Alternatively, you can also do the following:

1. Select File > Open. The Open dialog box is displayed.
2. In the Open dialog box, select (or navigate to) the script file that you want to open.
3. Click Open. The code of the script file is displayed in the SQL Worksheet area.
4. Select a connection from the connection drop-down list.
5. To run the code, click the Run Script (F5) icon on the SQL Worksheet toolbar. If you have not selected a connection from the connection drop-down list, a connection dialog box will appear. Select the connection you want to use for the script execution

Executing Saved Script Files: Method 2

Use the @ command followed by the location and name of the file that you want to execute, and click the Run Script icon.

myconnection

1.731089 seconds   myconnection ▼

@/home/oracle/labs/sql2/labs/confidence.sql

The output from the script is displayed on the Script Output tabbed page.

Results ▸ | 📄 Script Output | 📊 Explain | 🔍 Autotrace | 📋 DBMS Output | 🌐

1 rows selected

COUNT(*)

10
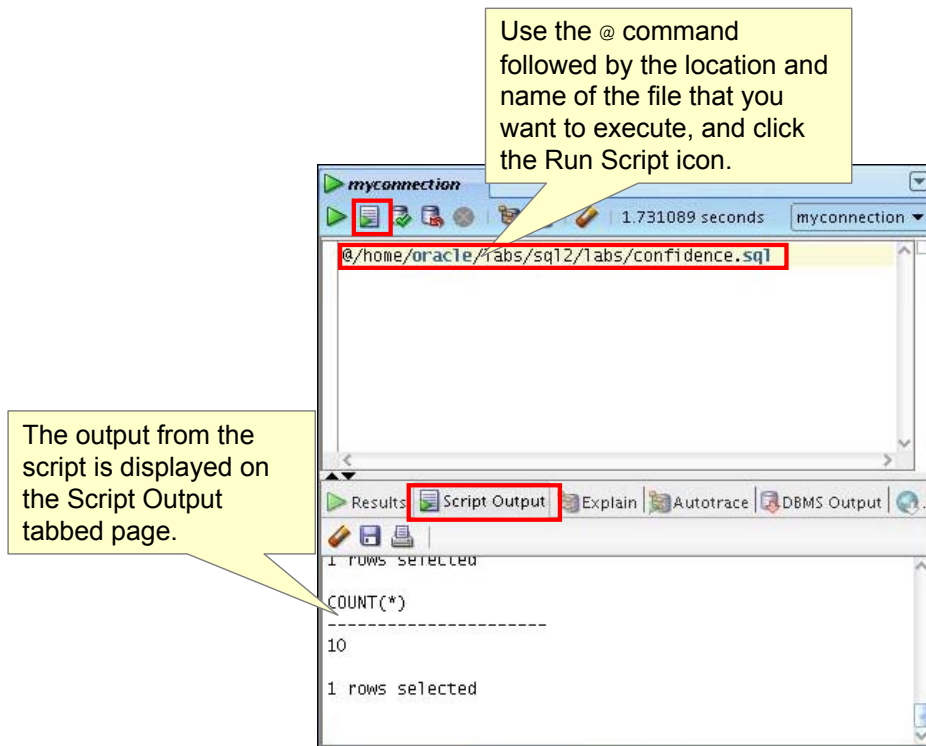
1 rows selected

Executing Saved Script Files: Method 2

To run a saved SQL script, perform the following steps:

1. Use the **@** command, followed by the location, and name of the file you want to run, in the Enter SQL Statement window.
2. Click the Run Script icon.

The results from running the file are displayed on the Script Output tabbed page. You can also save the script output by clicking the Save icon on the Script Output tabbed page. The File Save dialog box appears and you can identify a name and location for your file.

Formatting the SQL Code

Before formatting

```
select employee_id,  first_name,  salary from employees e,  departments d
where e.department_id= d.department_id and e.salary          >3000;
```

▶ Execute Statement	F9
📊 Execute Explain Plan	F6
📊 Autotrace	F10
📄 Run Script	F5
🖨 Print File	Ctrl-P
✏ Clear	Ctrl-D
⊘ Cancel	Ctrl-Q
📊 SQL History	F8
✂ Cut	Ctrl-X
📋 Copy	Ctrl-C
📋 Paste	Ctrl-V
Select All	Ctrl-A
Compile	Ctrl+Shift-F9
Query Builder	
Refactoring	▶
Format	Ctrl-F7
Code Template	Ctrl+Shift-T
Popup Describe	Shift-F4

▶ Results 📊 Script Output 📊 Expl   . Output

esults:

After formatting

```
SELECT employee_id,
   first_name,
   salary
FROM employees e,
   departments d
WHERE e.department_id= d.department_id
AND e.salary          > 3000;
```

Formatting the SQL Code

You may want to format the indentation, spacing, capitalization, and line separation of the SQL code. SQL Developer has a feature for formatting SQL code.

To format the SQL code, right-click in the statement area, and select Format SQL.

In the example in the slide, before formatting, the SQL code has the keywords not capitalized and the statement not properly indented. After formatting, the SQL code is beautified with the keywords capitalized and the statement properly indented.

Using Snippets

Snippets are code fragments that may be just syntax or examples.
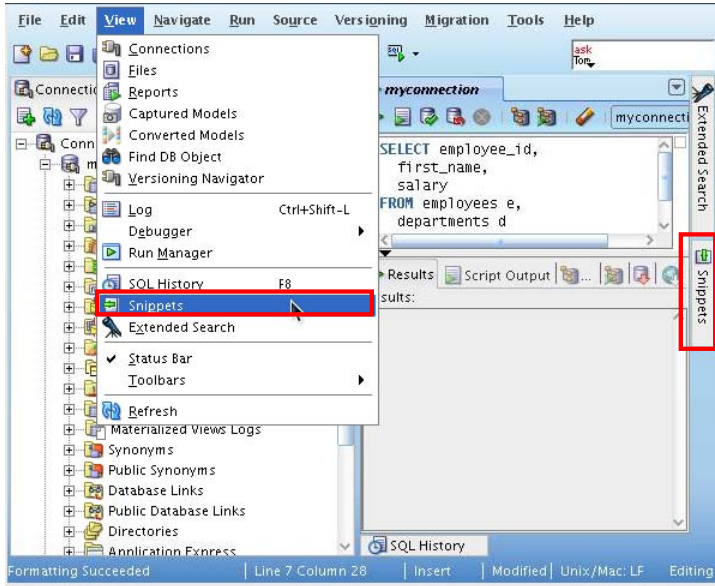
When you place your cursor here, it shows the Snippets window. From the drop-down list, you can select the functions category that you want.
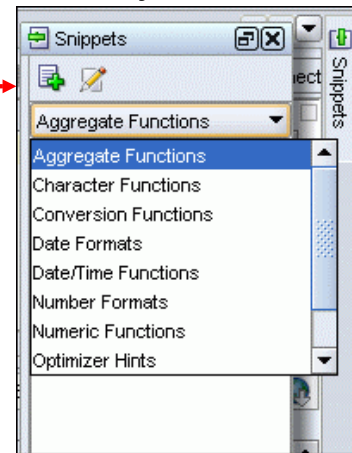
Using Snippets

You may want to use certain code fragments when you use the SQL Worksheet or create or edit a PL/SQL function or procedure. SQL Developer has the feature called Snippets. Snippets are code fragments such as SQL functions, Optimizer hints, and miscellaneous PL/SQL programming techniques. You can drag snippets into the Editor window.

To display Snippets, select View > Snippets.

The Snippets window is displayed at the right. You can use the drop-down list to select a group. A Snippets button is placed in the right window margin, so that you can display the Snippets window if it becomes hidden.

Using Snippets: Example
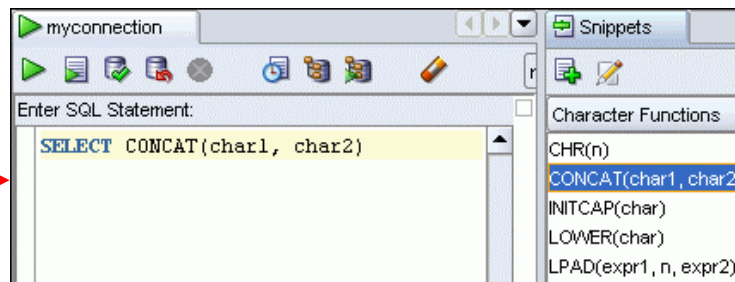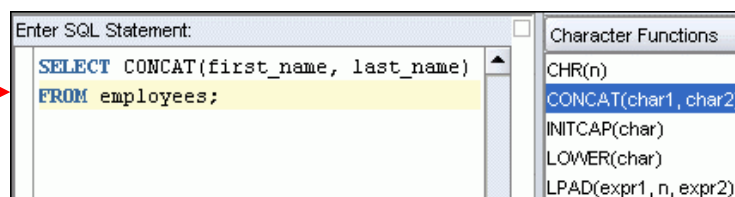
Using Snippets: Example

To insert a Snippet into your code in a SQL Worksheet or in a PL/SQL function or procedure, drag the snippet from the Snippets window to the desired place in your code. Then you can edit the syntax so that the SQL function is valid in the current context. To see a brief description of a SQL function in a tool tip, place the cursor over the function name.

The example in the slide shows that `CONCAT(char1, char2)` is dragged from the Character Functions group in the Snippets window. Then the `CONCAT` function syntax is edited and the rest of the statement is added as in the following:

```
SELECT CONCAT(first_name, last_name)
FROM employees;
```

Debugging Procedures and Functions

- Use SQL Developer to debug PL/SQL functions and procedures.
- Use the Compile for Debug option to perform a PL/SQL compilation so that the procedure can be debugged.
- Use the Debug menu options to set breakpoints, and to perform step into, step over tasks.
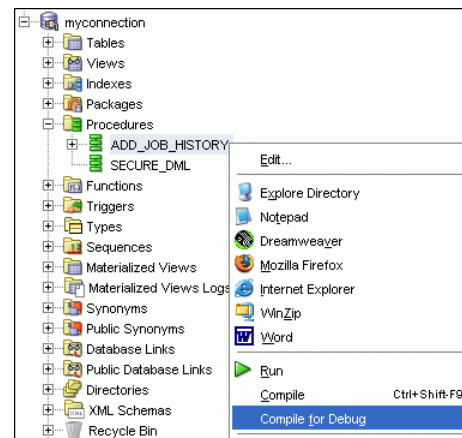
Debugging Procedures and Functions
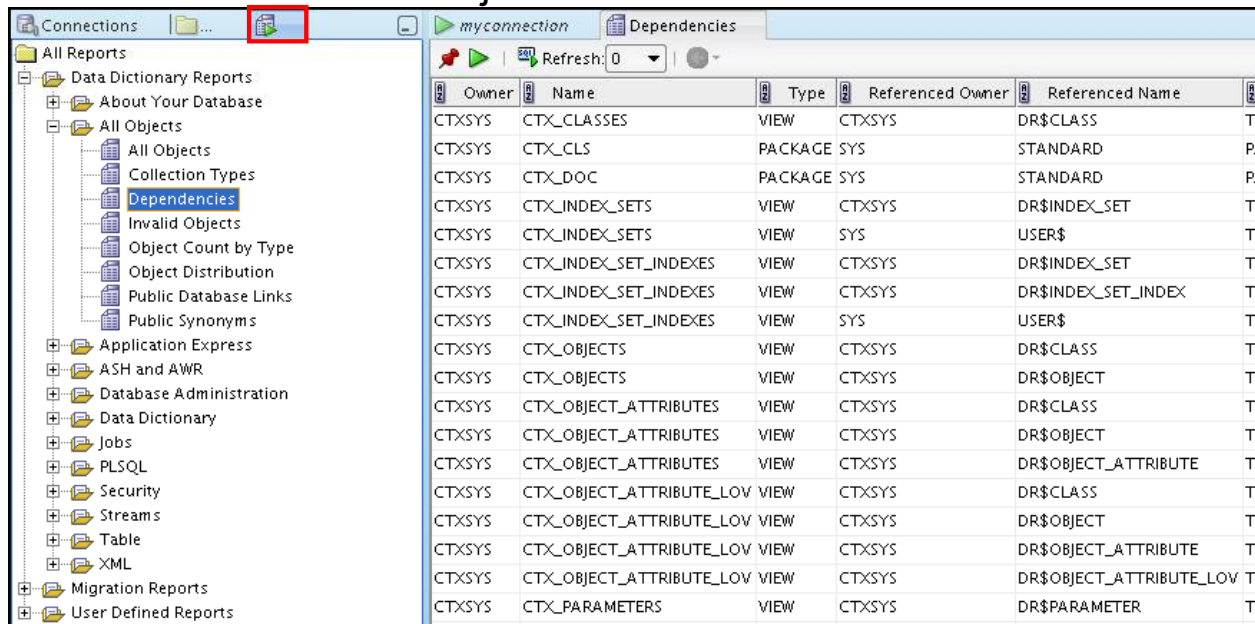
In SQL Developer, you can debug PL/SQL procedures and functions. Using the Debug menu options, you can perform the following debugging tasks:

- **Find Execution Point** goes to the next execution point.
- **Resume** continues execution.
- **Step Over** bypasses the next method and goes to the next statement after the method.
- **Step Into** goes to the first statement in the next method.
- **Step Out** leaves the current method and goes to the next statement.
- **Step to End of Method** goes to the last statement of the current method.
- **Pause** halts execution, but does not exit, thus allowing you to resume execution.
- **Terminate** halts and exits the execution. You cannot resume execution from this point; instead, to start running or debugging from the beginning of the function or procedure, click the Run or Debug icon on the Source tab toolbar.
- **Garbage Collection** removes invalid objects from the cache in favor of more frequently accessed and more valid objects.

These options are also available as icons on the debugging toolbar.

Database Reporting

SQL Developer provides a number of predefined reports about the database and its objects.

Database Reporting

SQL Developer provides many reports about the database and its objects. These reports can be grouped into the following categories:

- About Your Database reports
- Database Administration reports
- Table reports
- PL/SQL reports
- Security reports
- XML reports
- Jobs reports
- Streams reports
- All Objects reports
- Data Dictionary reports
- User-Defined reports

To display reports, click the Reports tab at the left of the window. Individual reports are displayed in tabbed panes at the right of the window; and for each report, you can select (using a drop-down list) the database connection for which to display the report. For reports about objects, the objects shown are only those visible to the database user associated with the selected database connection, and the rows are usually ordered by Owner. You can also create your own user-defined reports.

Creating a User-Defined Report

Create and save user-defined reports for repeated use.
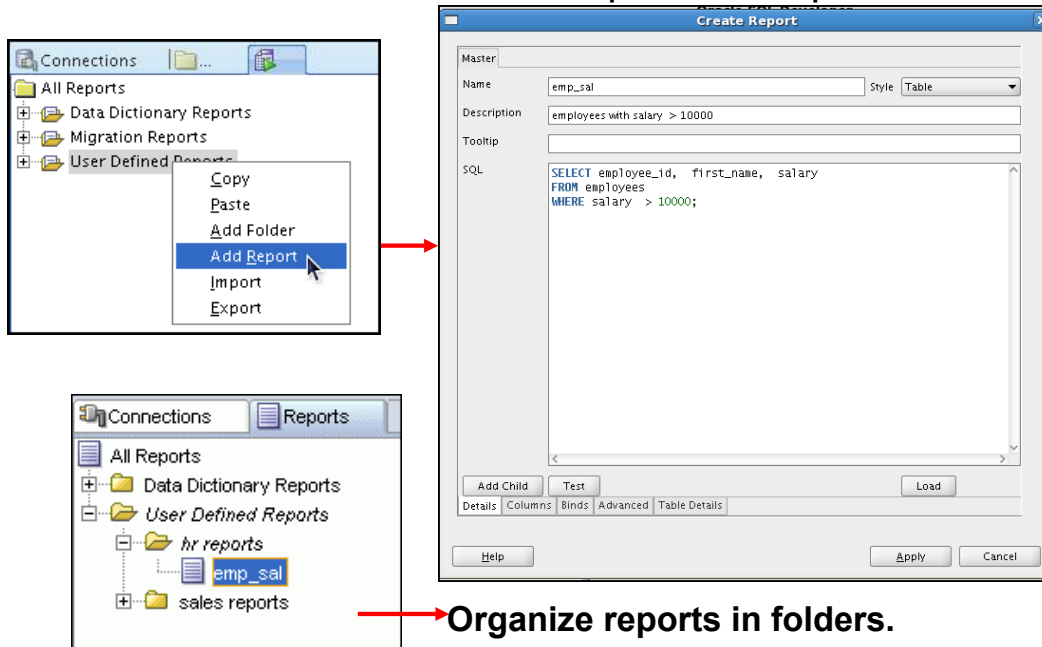
Organize reports in folders.

Creating a User-Defined Report

User-defined reports are reports created by SQL Developer users. To create a user-defined report, perform the following steps:
1. Right-click the User Defined Reports node under Reports, and select Add Report.
2. In the Create Report dialog box, specify the report name and the SQL query to retrieve information for the report. Then click Apply.

In the example in the slide, the report name is specified as emp_sal. An optional description is provided indicating that the report contains details of employees with `salary >= 10000`. The complete SQL statement for retrieving the information to be displayed in the user-defined report is specified in the SQL box. You can also include an optional tool tip to be displayed when the cursor stays briefly over the report name in the Reports navigator display.

You can organize user-defined reports in folders, and you can create a hierarchy of folders and subfolders. To create a folder for user-defined reports, right-click the User Defined Reports node or any folder name under that node and select Add Folder. Information about user-defined reports, including any folders for these reports, is stored in a file named `UserReports.xml` under the directory for user-specific information.

Search Engines and External Tools

Shortcuts to frequently used tools

1

2
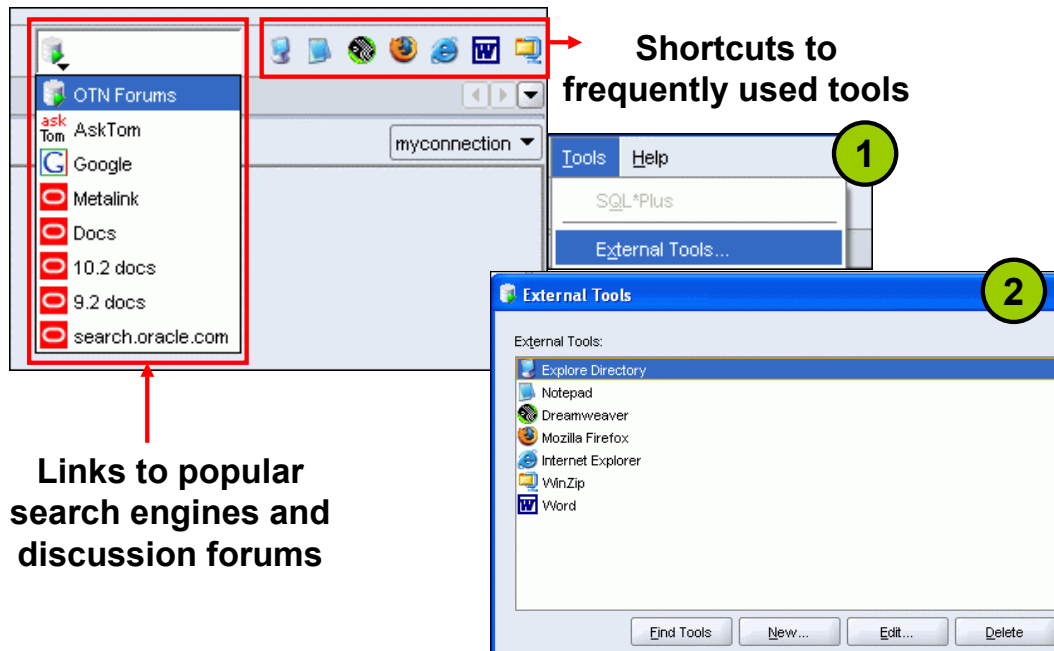
Links to popular search engines and discussion forums

Search Engines and External Tools

To enhance productivity of the SQL developers, SQL Developer has added quick links to popular search engines and discussion forums such as AskTom, Google, and so on. Also, you have shortcut icons to some of the frequently used tools such as Notepad, Microsoft Word, and Dreamweaver, available to you.

You can add external tools to the existing list or even delete shortcuts to tools that you do not use frequently. To do so, perform the following steps:

1. From the Tools menu, select External Tools.
2. In the External Tools dialog box, select New to add new tools. Select Delete to remove any tool from the list.

Setting Preferences

- Customize the SQL Developer interface and environment.
- In the Tools menu, select Preferences.
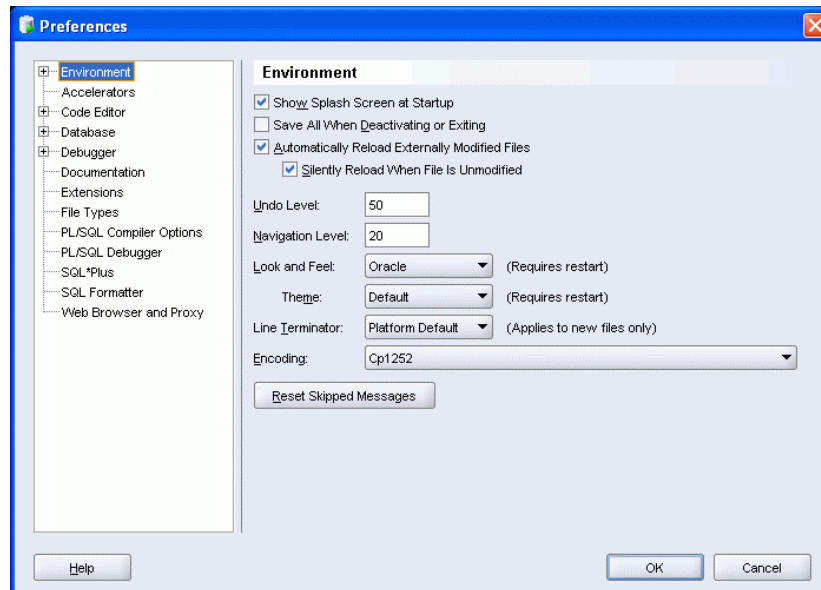
ORACLE
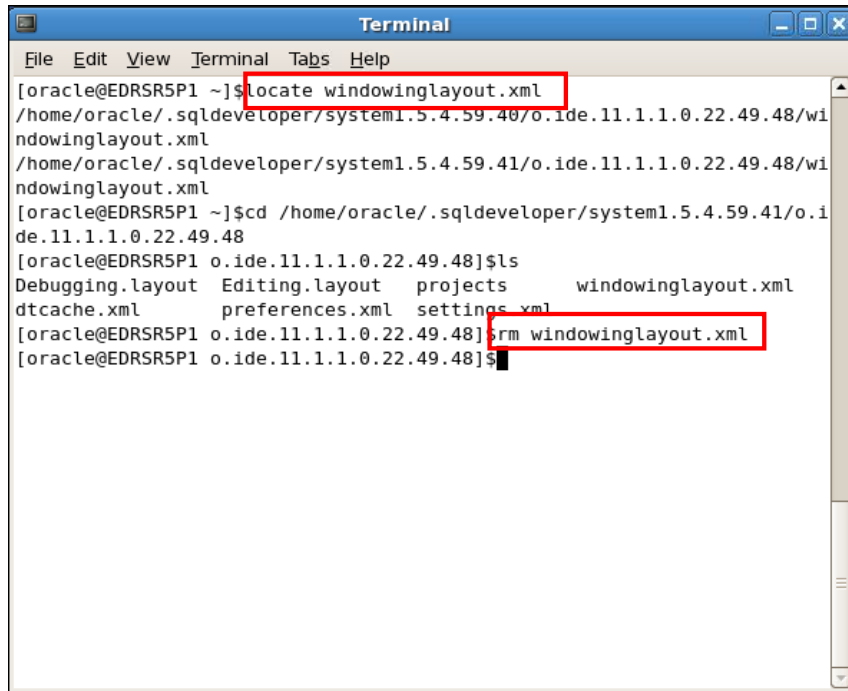
Setting Preferences

You can customize many aspects of the SQL Developer interface and environment by modifying SQL Developer preferences according to your preferences and needs. To modify SQL Developer preferences, select Tools, then Preferences.

The preferences are grouped into the following categories:
- Environment
- Accelerators (Keyboard shortcuts)
- Code Editors
- Database
- Debugger
- Documentation
- Extensions
- File Types
- Migration
- PL/SQL Compilers
- PL/SQL Debugger

Resetting the SQL Developer Layout

Resetting the SQL Developer Layout

While working with SQL Developer, if the Connections Navigator disappears or if you cannot dock the Log window in its original place, perform the following steps to fix the problem:

1. Exit from SQL Developer.
2. Open a terminal window and use the locate command to find the location of `windowinglayout.xml`.
3. Go to the directory that has `windowinglayout.xml` and delete it.
4. Restart SQL Developer.

Summary

In this appendix, you should have learned how to use SQL Developer to do the following:

- Browse, create, and edit database objects
- Execute SQL statements and scripts in SQL Worksheet
- Create and save custom reports

Summary

SQL Developer is a free graphical tool to simplify database development tasks. Using SQL Developer, you can browse, create, and edit database objects. You can use SQL Worksheet to run SQL statements and scripts. SQL Developer enables you to create and save your own special set of reports for repeated use.

D

Using SQL*Plus

Objectives

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

- Log in to SQL*Plus
- Edit SQL commands
- Format the output using SQL*Plus commands
- Interact with script files

Objectives

You might want to create SELECT statements that can be used again and again. This appendix also covers the use of SQL*Plus commands to execute SQL statements. You learn how to format output using SQL*Plus commands, edit SQL commands, and save scripts in SQL*Plus.

SQL and SQL*Plus Interaction

SQL statements

Server

SQL*Plus

Query results

Buffer

SQL scripts
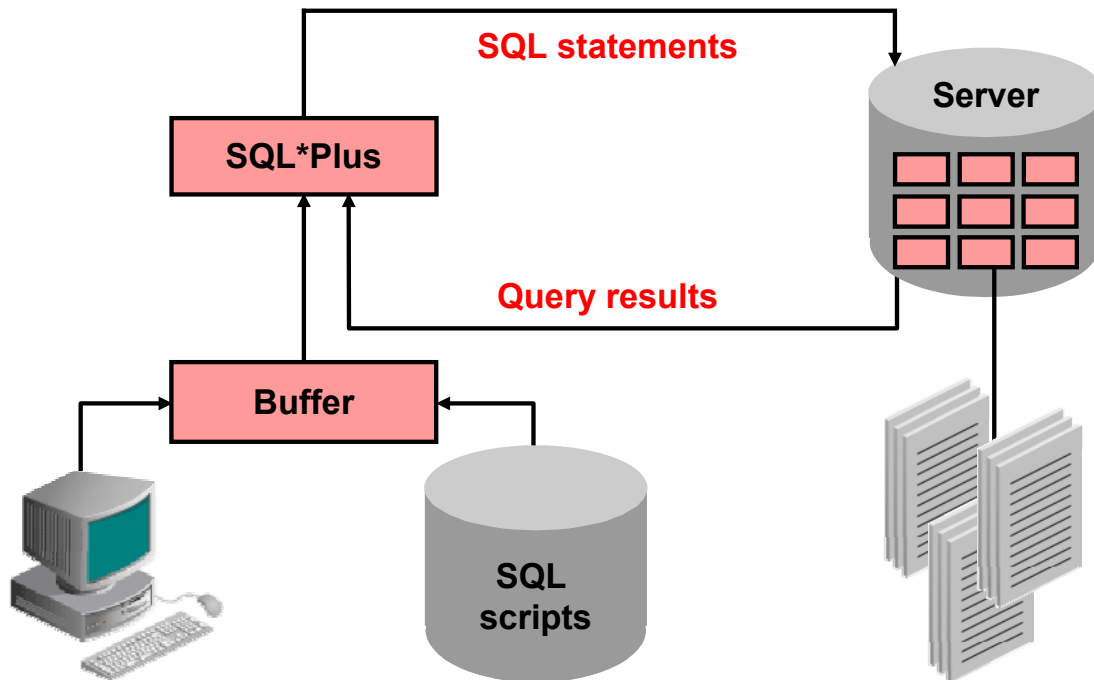
SQL and SQL*Plus

SQL is a command language used for communication with the Oracle server from any tool or application. Oracle SQL contains many extensions. When you enter a SQL statement, it is stored in a part of memory called the *SQL buffer* and remains there until you enter a new SQL statement. SQL*Plus is an Oracle tool that recognizes and submits SQL statements to the Oracle9*i* Server for execution. It contains its own command language.

Features of SQL

- Can be used by a range of users, including those with little or no programming experience
- Is a nonprocedural language
- Reduces the amount of time required for creating and maintaining systems
- Is an English-like language

Features of SQL*Plus

- Accepts ad hoc entry of statements
- Accepts SQL input from files
- Provides a line editor for modifying SQL statements
- Controls environmental settings
- Formats query results into basic reports
- Accesses local and remote databases

SQL Statements Versus SQL*Plus Commands

SQL

- A language
- ANSI-standard
- Keywords cannot be abbreviated.
- Statements manipulate data and table definitions in the database.

| SQL statements | → | SQL buffer |
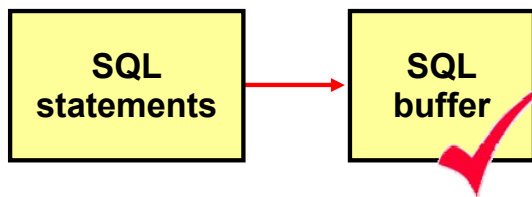
SQL*Plus

- An environment
- Oracle-proprietary
- Keywords can be abbreviated.
- Commands do not allow manipulation of values in the database.

| SQL*Plus commands | → | SQL*Plus buffer |
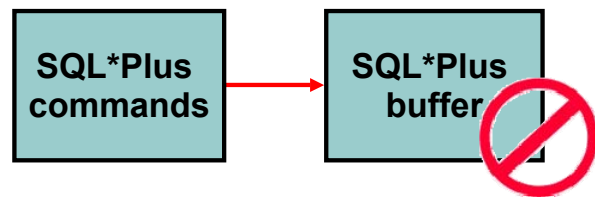
SQL and SQL*Plus (continued)

The following table compares SQL and SQL*Plus:

SQL	SQL*Plus
Is a language for communicating with the Oracle server to access data	Recognizes SQL statements and sends them to the server
Is based on American National Standards Institute (ANSI)–standard SQL	Is the Oracle-proprietary interface for executing SQL statements
Manipulates data and table definitions in the database	Does not allow manipulation of values in the database
Is entered into the SQL buffer on one or more lines	Is entered one line at a time, not stored in the SQL buffer
Does not have a continuation character	Uses a dash (–) as a continuation character if the command is longer than one line
Cannot be abbreviated	Can be abbreviated
Uses a termination character to execute commands immediately	Does not require termination characters; executes commands immediately
Uses functions to perform some formatting	Uses commands to format data

Overview of SQL*Plus

- Log in to SQL*Plus.
- Describe the table structure.
- Edit your SQL statement.
- Execute SQL from SQL*Plus.
- Save SQL statements to files and append SQL statements to files.
- Execute saved files.
- Load commands from the file to buffer to edit.
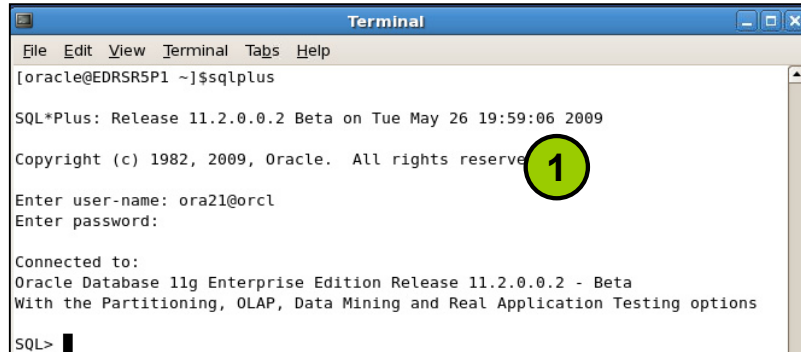
SQL*Plus

SQL*Plus is an environment in which you can:

- Execute SQL statements to retrieve, modify, add, and remove data from the database
- Format, perform calculations on, store, and print query results in the form of reports
- Create script files to store SQL statements for repeated use in the future

SQL*Plus commands can be divided into the following main categories:

Category	Purpose
Environment	Affect the general behavior of SQL statements for the session
Format	Format query results
File manipulation	Save, load, and run script files
Execution	Send SQL statements from the SQL buffer to the Oracle server
Edit	Modify SQL statements in the buffer
Interaction	Create and pass variables to SQL statements, print variable values, and print messages to the screen
Miscellaneous	Connect to the database, manipulate the SQL*Plus environment, and display column definitions

Logging In to SQL*Plus
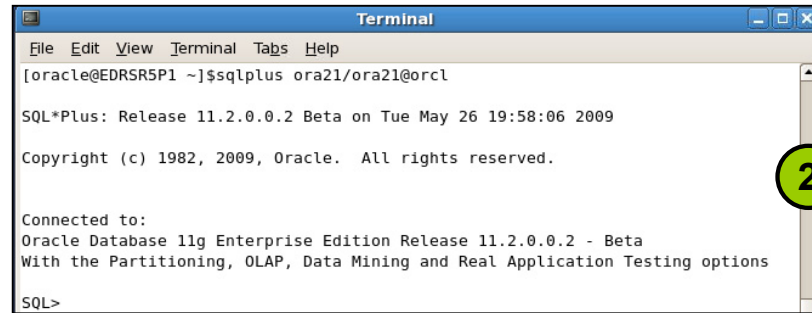

```
sqlplus [username[/password[@database]]]
```

Logging In to SQL*Plus

How you invoke SQL*Plus depends on which type of operating system you are running Oracle Database.

To log in from a Linux environment, perform the following steps:
1. Right-click your Linux desktop and select terminal.
2. Enter the `sqlplus` command shown in the slide.
3. Enter the username, password, and database name.

In the syntax:

username	Your database username
password	Your database password (Your password is visible if you enter it here.)
@database	The database connect string

Note: To ensure the integrity of your password, do not enter it at the operating system prompt. Instead, enter only your username. Enter your password at the password prompt.

Displaying the Table Structure

Use the SQL*Plus `DESCRIBE` command to display the structure of a table:

```
DESC[RIBE] tablename
```

Displaying the Table Structure

In SQL*Plus, you can display the structure of a table using the `DESCRIBE` command. The result of the command is a display of column names and data types as well as an indication if a column must contain data.

In the syntax:

tablename The name of any existing table, view, or synonym that is accessible to the user

To describe the `DEPARTMENTS` table, use this command:

```
SQL> DESCRIBE DEPARTMENTS
Name                     Null?    Type
----------------------- -------- ---------------

DEPARTMENT_ID            NOT NULL NUMBER(4)
DEPARTMENT_NAME          NOT NULL VARCHAR2(30)
MANAGER_ID               NUMBER(6)
LOCATION_ID              NUMBER(4)
```

Displaying the Table Structure

```
DESCRIBE departments
```

```
Name                          Null?    Type
----------------------- -------- ------------
DEPARTMENT_ID                 NOT NULL NUMBER(4)
DEPARTMENT_NAME               NOT NULL VARCHAR2(30)
MANAGER_ID                    NUMBER(6)
LOCATION_ID                   NUMBER(4)
```

Displaying the Table Structure (continued)

The example in the slide displays the information about the structure of the DEPARTMENTS table. In the result:

Null?: Specifies whether a column must contain data (NOT NULL indicates that a column must contain data.)

Type: Displays the data type for a column

SQL*Plus Editing Commands

- A[PPEND] *text*
- C[HANGE] / *old* / *new*
- C[HANGE] / *text* /
- CL[EAR] BUFF[ER]
- DEL
- DEL *n*
- DEL *m n*

ORACLE

SQL*Plus Editing Commands

SQL*Plus commands are entered one line at a time and are not stored in the SQL buffer.

Command	Description
A[PPEND] *text*	Adds text to the end of the current line
C[HANGE] / *old* / *new*	Changes *old* text to *new* in the current line
C[HANGE] / *text* /	Deletes *text* from the current line
CL[EAR] BUFF[ER]	Deletes all lines from the SQL buffer
DEL	Deletes current line
DEL *n*	Deletes line *n*
DEL *m n*	Deletes lines *m* to *n* inclusive

Guidelines

- If you press Enter before completing a command, SQL*Plus prompts you with a line number.
- You terminate the SQL buffer either by entering one of the terminator characters (semicolon or slash) or by pressing [Enter] twice. The SQL prompt appears.

SQL*Plus Editing Commands

- I[NPUT]
- I[NPUT] *text*
- L[IST]
- L[IST] *n*
- L[IST] *m n*
- R[UN]
- *n*
- *n text*
- 0 *text*

ORACLE

SQL*Plus Editing Commands (continued)

Command	Description
I[NPUT]	Inserts an indefinite number of lines
I[NPUT] *text*	Inserts a line consisting of *text*
L[IST]	Lists all lines in the SQL buffer
L[IST] *n*	Lists one line (specified by *n*)
L[IST] *m n*	Lists a range of lines (*m* to *n*) inclusive
R[UN]	Displays and runs the current SQL statement in the buffer
n	Specifies the line to make the current line
n text	Replaces line *n* with *text*
0 *text*	Inserts a line before line 1

Note: You can enter only one SQL*Plus command for each SQL prompt. SQL*Plus commands are not stored in the buffer. To continue a SQL*Plus command on the next line, end the first line with a hyphen (-).

Using `LIST`, n, and `APPEND`

```
LIST
  1  SELECT last_name
  2* FROM   employees
```

```
1
  1* SELECT last_name
```

```
A , job_id
  1* SELECT last_name, job_id
```

```
LIST
  1  SELECT last_name, job_id
  2* FROM   employees
```

Using `LIST`, n, and `APPEND`

- Use the `L[IST]` command to display the contents of the SQL buffer. The asterisk (`*`) beside line 2 in the buffer indicates that line 2 is the current line. Any edits that you made apply to the current line.
- Change the number of the current line by entering the number (n) of the line that you want to edit. The new current line is displayed.
- Use the `A[PPEND]` command to add text to the current line. The newly edited line is displayed. Verify the new contents of the buffer by using the `LIST` command.

Note: Many SQL*Plus commands, including `LIST` and `APPEND`, can be abbreviated to just their first letter. `LIST` can be abbreviated to `L`; `APPEND` can be abbreviated to `A`.

Using the CHANGE Command

```
LIST
  1* SELECT * from employees
```

```
c/employees/departments
  1* SELECT * from departments
```

```
LIST
  1* SELECT * from departments
```

Using the CHANGE Command

- Use L[IST] to display the contents of the buffer.
- Use the C[HANGE] command to alter the contents of the current line in the SQL buffer. In this case, replace the employees table with the departments table. The new current line is displayed.
- Use the L[IST] command to verify the new contents of the buffer.

SQL*Plus File Commands

- SAVE *filename*
- GET *filename*
- START *filename*
- @ *filename*
- EDIT *filename*
- SPOOL *filename*
- EXIT

SQL*Plus File Commands

SQL statements communicate with the Oracle server. SQL*Plus commands control the environment, format query results, and manage files. You can use the commands described in the following table:

Command	Description
SAV[E] *filename* [.ext] [REP[LACE] APP[END]]	Saves current contents of SQL buffer to a file. Use APPEND to add to an existing file; use REPLACE to overwrite an existing file. The default extension is .sql.
GET *filename* [.ext]	Writes the contents of a previously saved file to the SQL buffer. The default extension for the file name is .sql.
STA[RT] *filename* [.ext]	Runs a previously saved command file
@ *filename*	Runs a previously saved command file (same as START)
ED[IT]	Invokes the editor and saves the buffer contents to a file named afiedt.buf
ED[IT] [*filename*[.ext]]	Invokes the editor to edit the contents of a saved file
SPO[OL] [*filename*[.ext]| OFF|OUT]	Stores query results in a file. OFF closes the spool file. OUT closes the spool file and sends the file results to the printer.
EXIT	Quits SQL*Plus

Using the SAVE, START Commands

```
LIST
  1  SELECT last_name, manager_id, department_id
  2* FROM employees
```

```
SAVE my_query
  Created file my_query
```

```
START my_query

LAST_NAME                       MANAGER_ID DEPARTMENT_ID
------------------------        ---------- -------------
King                                                  90
Kochhar                                100            90
...
107 rows selected.
```

Oracle University and NETEC, S.A. de C.V. use only.

Using the SAVE, START, and EDIT Commands

SAVE

Use the SAVE command to store the current contents of the buffer in a file. In this way, you can store frequently used scripts for use in the future.

START

Use the START command to run a script in SQL*Plus. You can also, alternatively, use the symbol @ to run a script.

```
@my_query
```

SERVEROUTPUT Command

- Use the `SET SERVEROUT[PUT]` command to control whether to display the output of stored procedures or PL/SQL blocks in SQL*Plus.
- The `DBMS_OUTPUT` line length limit is increased from 255 bytes to 32767 bytes.
- The default size is now unlimited.
- Resources are not preallocated when `SERVEROUTPUT` is set.
- Because there is no performance penalty, use `UNLIMITED` unless you want to conserve physical memory.

```
SET SERVEROUT[PUT] {ON | OFF} [SIZE {n | UNL[IMITED]}]
   [FOR[MAT] {WRA[PPED] | WOR[D_WRAPPED] | TRU[NCATED]}]
```

ORACLE

SERVEROUTPUT Command

Most of the PL/SQL programs perform input and output through SQL statements, to store data in database tables or query those tables. All other PL/SQL input/output is done through APIs that interact with other programs. For example, the DBMS_OUTPUT package has procedures, such as PUT_LINE. To see the result outside of PL/SQL requires another program, such as SQL*Plus, to read and display the data passed to DBMS_OUTPUT.

SQL*Plus does not display DBMS_OUTPUT data unless you first issue the SQL*Plus command SET SERVEROUTPUT ON as follows:

```
SET SERVEROUTPUT ON
```

Note

- SIZE sets the number of bytes of the output that can be buffered within the Oracle Database server. The default is UNLIMITED. n cannot be less than 2000 or greater than 1,000,000.
- For additional information about SERVEROUTPUT, see *Oracle Database PL/SQL User's Guide and Reference 11g*.

Using the SQL*Plus `SPOOL` Command

```
SPO[OL] [file_name[.ext] [CRE[ATE] | REP[LACE] |
    APP[END]] | OFF | OUT]
```

Option	Description
`file_name[.ext]`	Spools output to the specified file name
`CRE[ATE]`	Creates a new file with the name specified
`REP[LACE]`	Replaces the contents of an existing file. If the file does not exist, `REPLACE` creates the file.
`APP[END]`	Adds the contents of the buffer to the end of the file you specify
`OFF`	Stops spooling
`OUT`	Stops spooling and sends the file to your computer's standard (default) printer

Using the SQL*Plus `SPOOL` Command

The `SPOOL` command stores query results in a file or optionally sends the file to a printer. The `SPOOL` command has been enhanced. You can now append to, or replace an existing file, where previously you could only use `SPOOL` to create (and replace) a file. `REPLACE` is the default.

To spool output generated by commands in a script without displaying the output on the screen, use `SET TERMOUT OFF`. `SET TERMOUT OFF` does not affect output from commands that run interactively.

You must use quotes around file names containing white space. To create a valid HTML file using `SPOOL APPEND` commands, you must use `PROMPT` or a similar command to create the HTML page header and footer. The `SPOOL APPEND` command does not parse HTML tags. `SET SQLPLUSCOMPAT[IBILITY]` to 9.2 or earlier to disable the `CREATE`, `APPEND` and `SAVE` parameters.

Using the `AUTOTRACE` Command

- It displays a report after the successful execution of SQL DML statements such as `SELECT`, `INSERT`, `UPDATE`, or `DELETE`.

- The report can now include execution statistics and the query execution path.

```
SET AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]]
    [STAT[ISTICS]]
```

```
SET AUTOTRACE ON
-- The AUTOTRACE report includes both the optimizer
-- execution path and the SQL statement execution
-- statistics
```

Using the `AUTOTRACE` Command

`EXPLAIN` shows the query execution path by performing an `EXPLAIN PLAN`. `STATISTICS` displays SQL statement statistics. The formatting of your `AUTOTRACE` report may vary depending on the version of the server to which you are connected and the configuration of the server. The `DBMS_XPLAN` package provides an easy way to display the output of the `EXPLAIN PLAN` command in several predefined formats.

Note

- For additional information about the package and subprograms, refer to *Oracle Database PL/SQL Packages and Types Reference 11g*.
- For additional information about the `EXPLAIN PLAN`, refer to *Oracle Database SQL Reference 11g*.
- For additional information about Execution Plans and the statistics, refer to *Oracle Database Performance Tuning Guide 11g*.

Summary

In this appendix, you should have learned how to use SQL*Plus as an environment to do the following:

- Execute SQL statements
- Edit SQL statements
- Format the output
- Interact with script files

Summary

SQL*Plus is an execution environment that you can use to send SQL commands to the database server and to edit and save SQL commands. You can execute commands from the SQL prompt or from a script file.

Using JDeveloper

Objectives

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

- List the key features of Oracle JDeveloper
- Create a database connection in JDeveloper
- Manage database objects in JDeveloper
- Use JDeveloper to execute SQL Commands
- Create and run PL/SQL Program Units

Objectives

In this appendix, you are introduced to JDeveloper. You learn how to use JDeveloper for your database development tasks.

Oracle JDeveloper
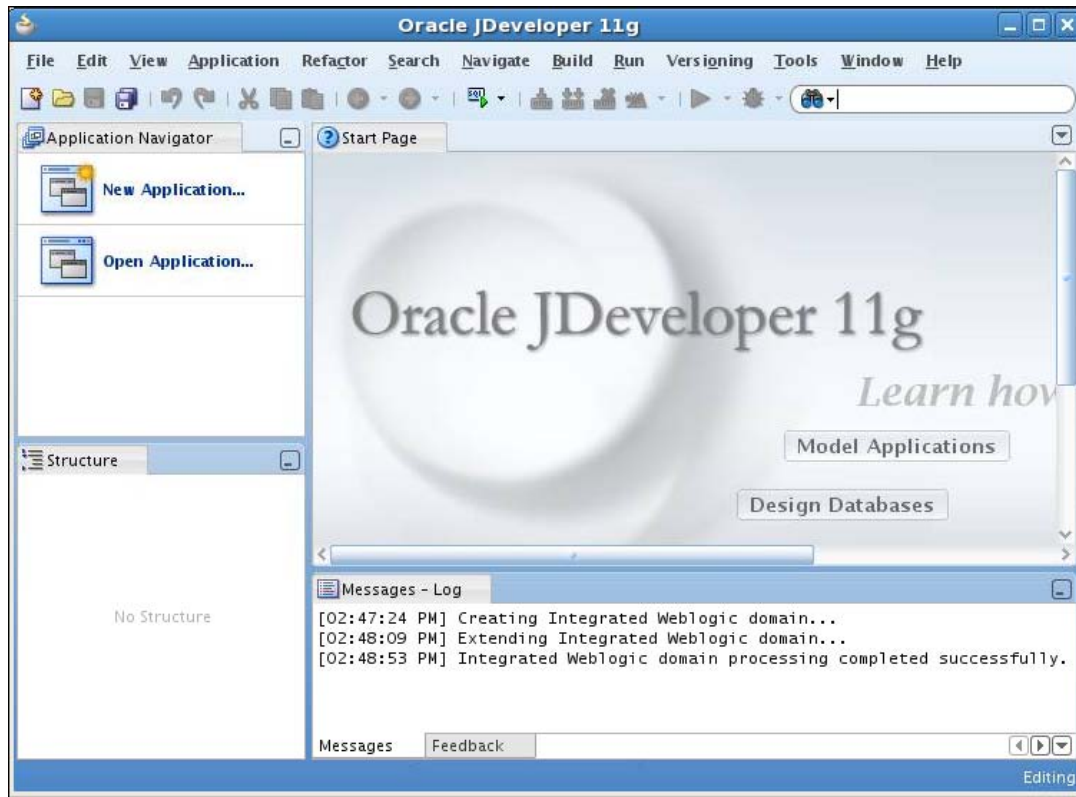
Oracle JDeveloper

Oracle JDeveloper is an integrated development environment (IDE) for developing and deploying Java applications and Web services. It supports every stage of the software development life cycle (SDLC) from modeling to deploying. It has the features to use the latest industry standards for Java, XML, and SQL while developing an application.

Oracle JDeveloper 11*g* initiates a new approach to J2EE development with features that enable visual and declarative development. This innovative approach makes J2EE development simple and efficient.

Database Navigator
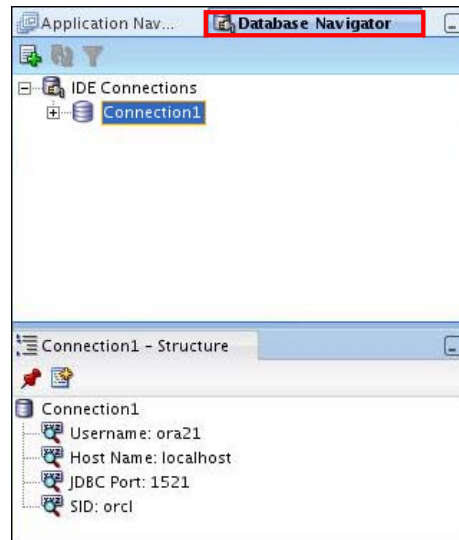
Database Navigator

Using Oracle JDeveloper, you can store the information necessary to connect to a database in an object called "connection." A connection is stored as part of the IDE settings, and can be exported and imported for easy sharing among groups of users. A connection serves several purposes from browsing the database and building applications, all the way through to deployment.

Creating Connection

1 Click the **New Connection** icon in the **Database Navigator**.

2 In the Create Database Connection window, enter the **Username**, **Password**, and the **SID**.

3 Click **Test Connection**.

4 Click **OK**

Creating Connection

A connection is an object that specifies the necessary information for connecting to a specific database as a specific user of that database. You can create and test connections for multiple databases and for multiple schemas.

To create a database connection, perform the following steps:
1. Click the New Connection icon in the Database Navigator.
2. In the Create Database Connection window, enter the connection name. Enter the username and password of the schema that you want to connect to. Enter the SID of the Database you want to connect.
3. Click Test to ensure that the connection has been set correctly.
4. Click OK.

Browsing Database Objects

Use the Database Navigator to:

- Browse through many objects in a database schema
- Review the definitions of objects at a glance
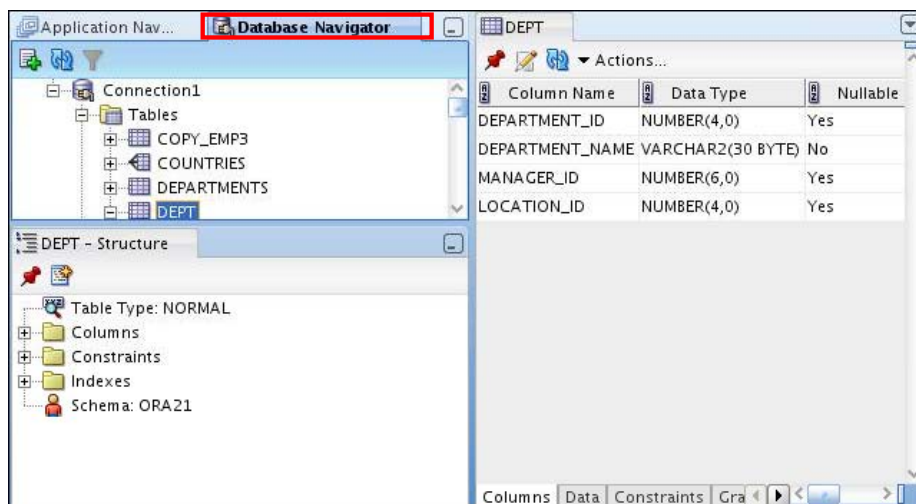
Browsing Database Objects

After you create a database connection, you can use the Database Navigator to browse through many objects in a database schema including Tables, Views, Indexes, Packages, Procedures, Triggers, and Types.

You can see the definition of the objects broken into tabs of information that is pulled out of the data dictionary. For example, if you select a table in the Navigator, the details about columns, constraints, grants, statistics, triggers, and so on are displayed on an easy-to-read Database Navigator.

Executing SQL Statements
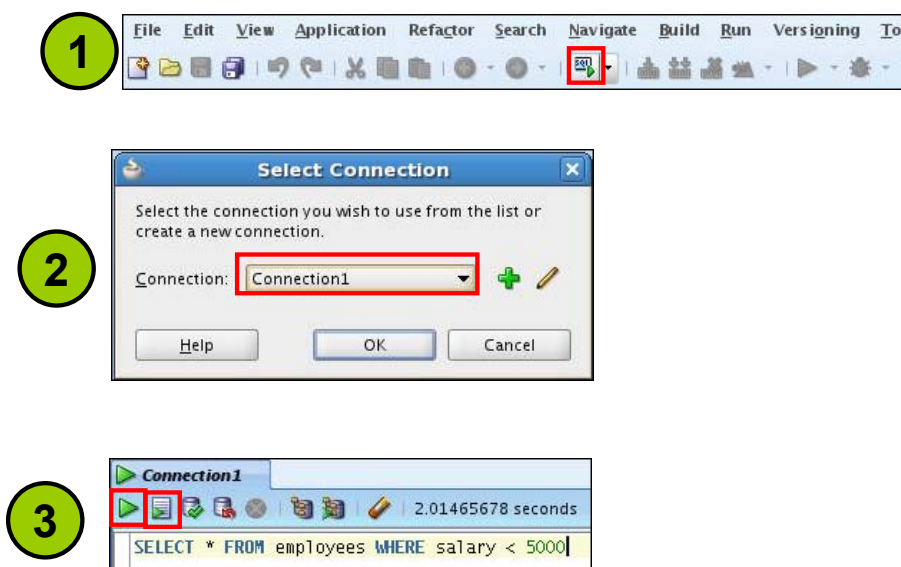
Executing SQL Statements

To execute a SQL statement, perform the following steps:

1. Click the Open SQL Worksheet icon.

2. Select the connection.

3. Execute the SQL command by clicking:
   1. The **Execute statement** button or by pressing F9. The output is as follows:

   2. The **Run Script** button or by pressing F5. The output is as follows:

Creating Program Units
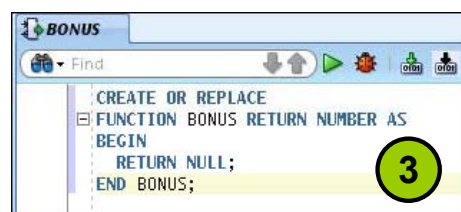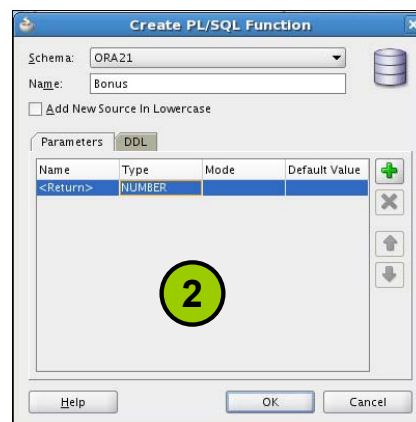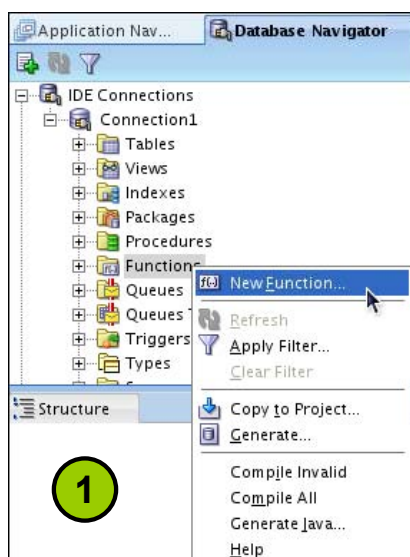
Skeleton of the function

Creating Program Units

To create a PL/SQL program unit, perform the following steps:

1. Select View > Database Navigator. Select and expand a database connection. Right-click a folder corresponding to the object type (Procedures, Packages, Functions). Choose "New [Procedures|Packages|Functions]."
2. Enter a valid name for the function, package, or procedure and click OK.
3. A skeleton definition is created and opened in the Code Editor. You can then edit the subprogram to suit your need.

Compiling
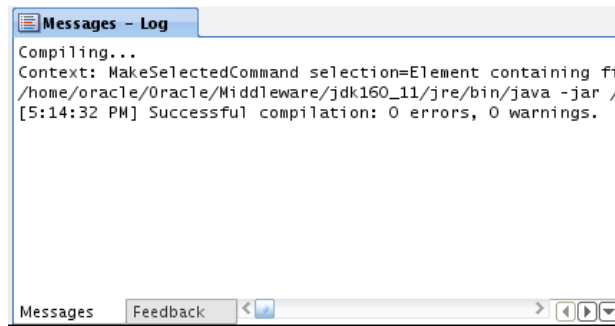
Compilation with errors
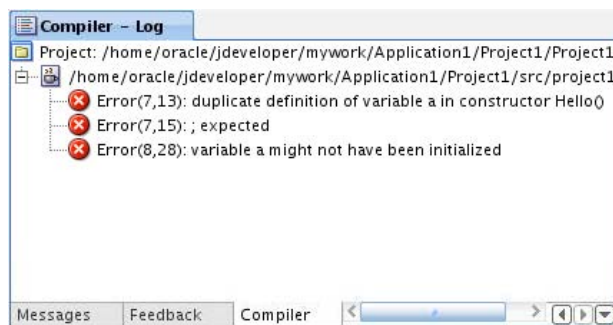
Compilation without errors

Compiling

After editing the skeleton definition, you need to compile the program unit. Right-click the PL/SQL object that you need to compile in the Connection Navigator, and then select Compile. Alternatively, you can also press [Ctrl] + [Shift] + [F9] to compile.

Running a Program Unit
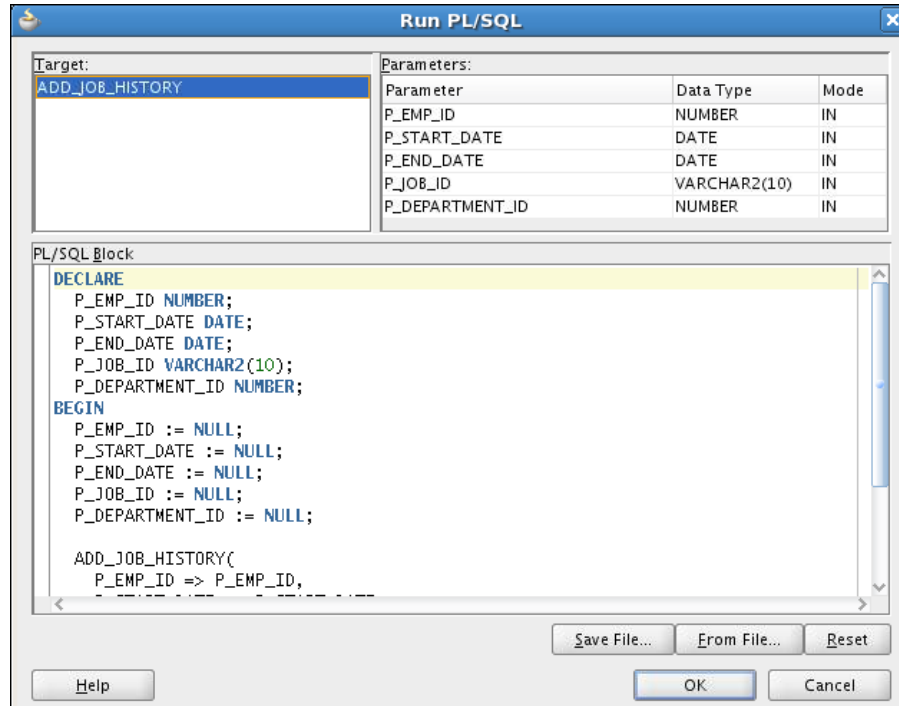
Running a Program Unit

To execute the program unit, right-click the object and click Run. The Run PL/SQL dialog box appears. You may need to change the NULL values with reasonable values that are passed into the program unit. After you change the values, click OK. The output is displayed in the Message-Log window.

Dropping a Program Unit
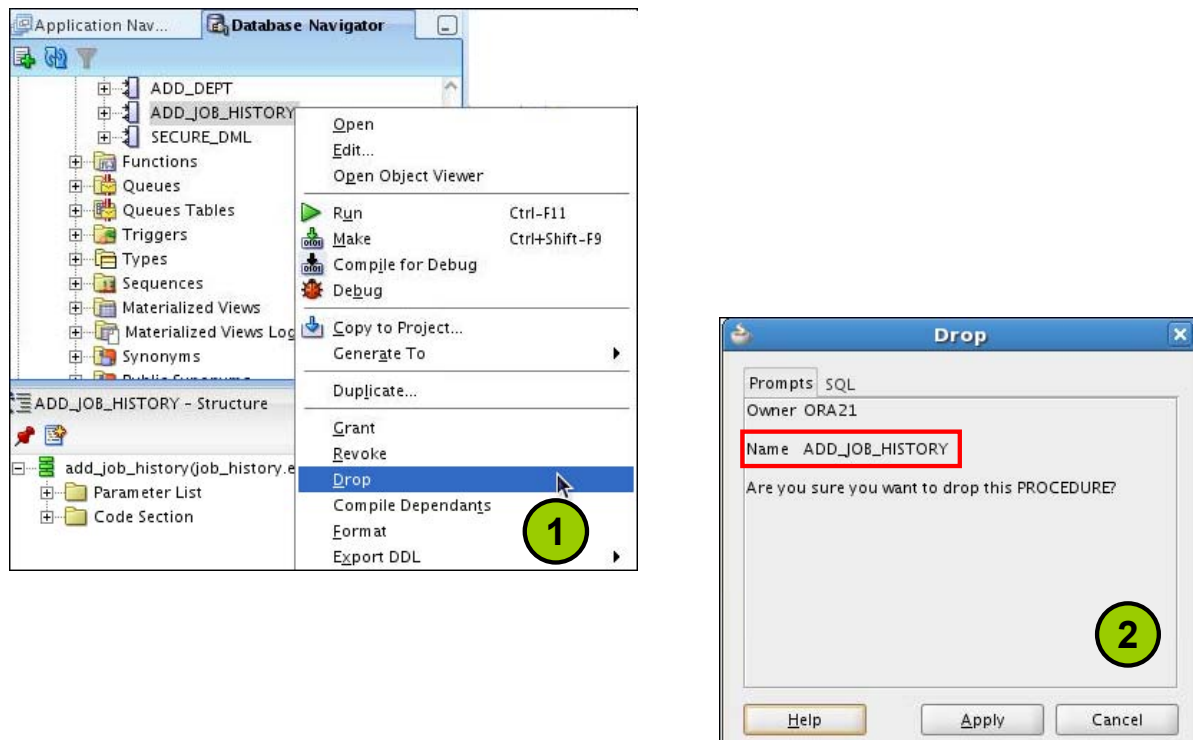
Dropping a Program Unit

To drop a program unit, right-click the object and select Drop. The Drop Confirmation dialog box appears; click **Apply**. The object is dropped from the database.

Structure Window
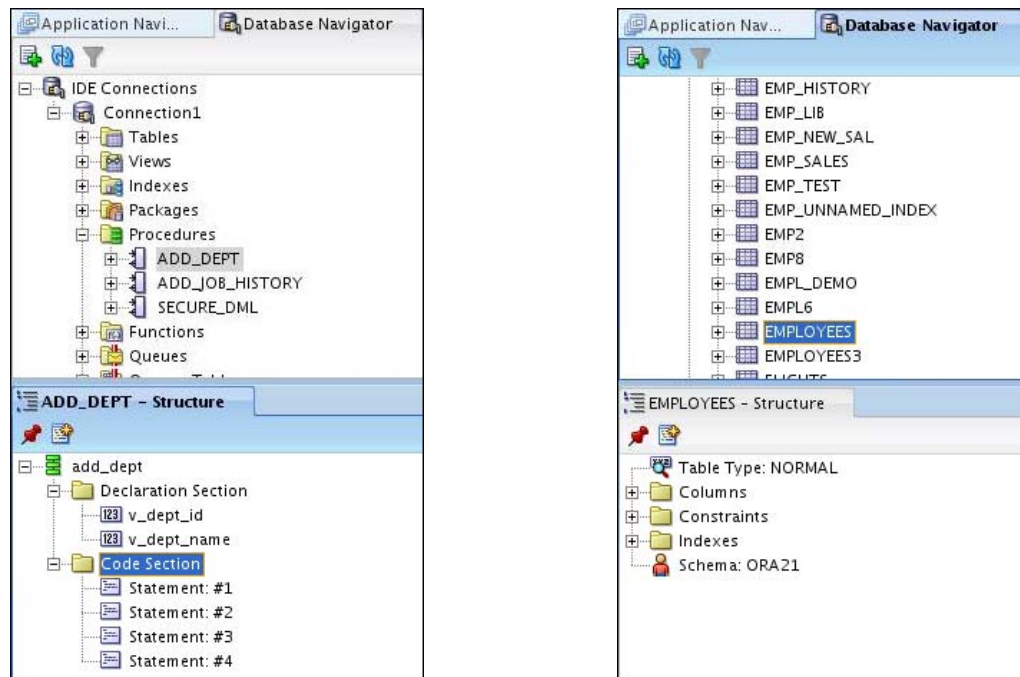
Structure Window

The Structure window offers a structural view of the data in the document currently selected in the active window of those windows that participate in providing structure: the navigators, the editors and viewers, and the Property Inspector.

Click View > Structure window to view the Structure window.

In the Structure window, you can view the document data in a variety of ways. The structures available for display are based upon document type. For a Java file, you can view code structure, UI structure, or UI model data. For an XML file, you can view XML structure, design structure, or UI model data.

The Structure window is dynamic, tracking always the current selection of the active window (unless you freeze the window's contents on a particular view), as is pertinent to the currently active editor. When the current selection is a node in the navigator, the default editor is assumed. To change the view on the structure for the current selection, click a different structure tab.

Editor Window
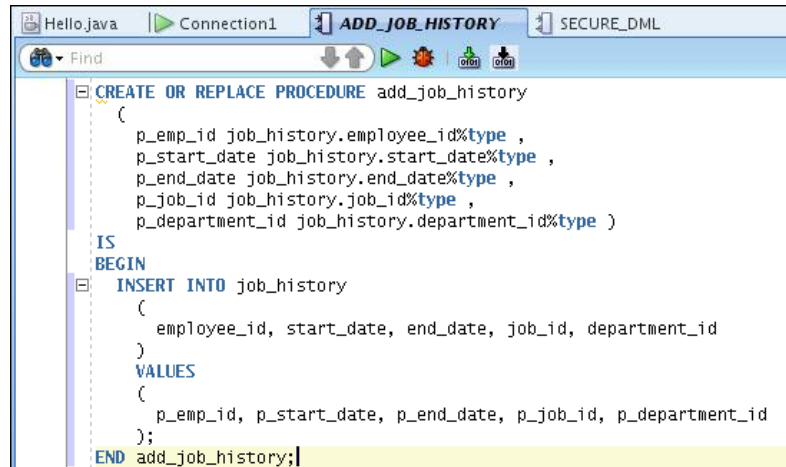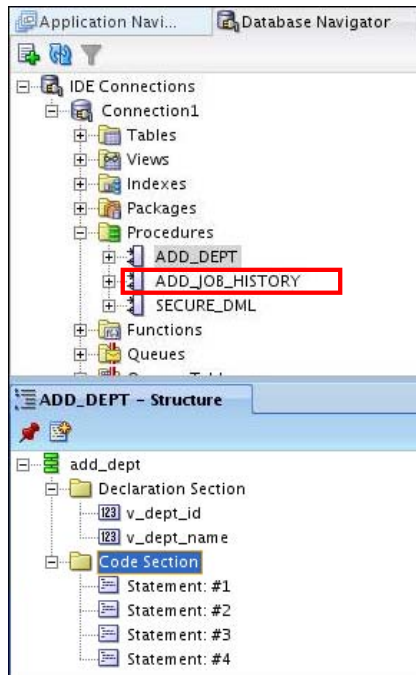
Editor Window

Double-clicking the name of a program unit opens it in the Editor window. You can view your project files all in one single editor window, you can open multiple views of the same file, or you can open multiple views of different files.

The tabs at the top of the editor window are the document tabs. Clicking a document tab gives that file focus, bringing it to the foreground of the window in the current editor.

The tabs at the bottom of the editor window for a given file are the editor tabs. Selecting an editor tab opens the file in that editor.

Application Navigator
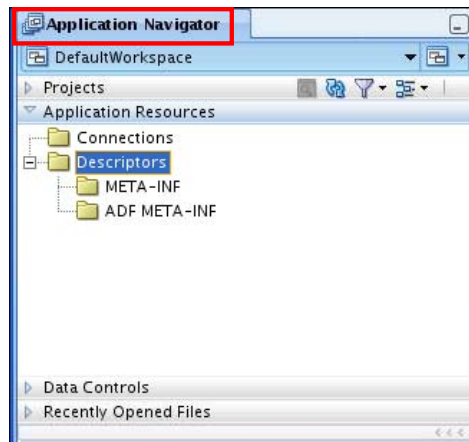
Application Navigator

The Applications - Navigator gives you a logical view of your application and the data it contains. The Applications - Navigator provides an infrastructure that the different extensions can plug in to and use to organize their data and menus in a consistent, abstract manner. While the Applications - Navigator can contain individual files (such as Java source files), it is designed to consolidate complex data. Complex data types such as entity objects, UML diagrams, EJB, or Web services appear in this navigator as single nodes. The raw files that make up these abstract nodes appear in the Structure window.

Deploying Java Stored Procedures

Before deploying Java stored procedures, perform the following steps:

1. Create a database connection.
2. Create a deployment profile.
3. Deploy the objects.
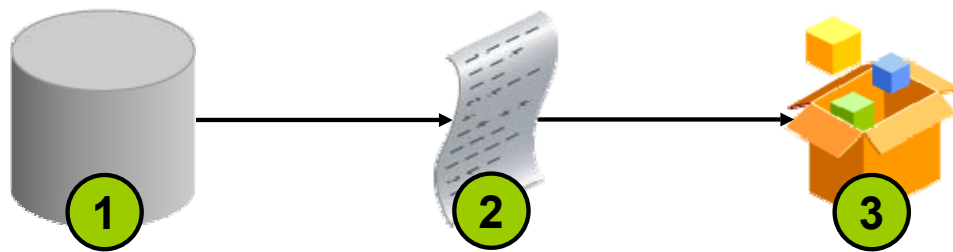
Deploying Java Stored Procedures

Create a deployment profile for Java stored procedures, and then deploy the classes and, optionally, any public static methods in JDeveloper using the settings in the profile.

Deploying to the database uses the information provided in the Deployment Profile Wizard and two Oracle Database utilities:

- `loadjava` loads the Java class containing the stored procedures to an Oracle database.
- `publish` generates the PL/SQL call–specific wrappers for the loaded public static methods. Publishing enables the Java methods to be called as PL/SQL functions or procedures.

Publishing Java to PL/SQL

Publishing Java to PL/SQL

The slide shows the Java code and illustrates how to publish the Java code in a PL/SQL procedure.

Oracle Database 11g: SQL Fundamentals I   E - 16

How Can I Learn More About JDeveloper 11*g*?

Topic	Website
Oracle JDeveloper Product Page	http://www.oracle.com/technology/products/jdev/index.html
Oracle JDeveloper 11g Tutorials	http://www.oracle.com/technology/obe/obe11jdev/11/index.html
Oracle JDeveloper 11g Product Documentation	http://www.oracle.com/technology/documentation/jdev.html
Oracle JDeveloper 11g Discussion Forum	http://forums.oracle.com/forums/forum.jspa?forumID=83

ORACLE

Summary

In this appendix, you should have learned to do the following:

- List the key features of Oracle JDeveloper
- Create a database connection in JDeveloper
- Manage database objects in JDeveloper
- Use JDeveloper to execute SQL Commands
- Create and run PL/SQL Program Units

ORACLE

Oracle Join Syntax

F

Objectives

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

- Write `SELECT` statements to access data from more than one table using equijoins and nonequijoins
- Join a table to itself by using a self-join
- View data that generally does not meet a join condition by using outer joins
- Generate a Cartesian product of all rows from two or more tables

Oracle University and NETEC, S.A. de C.V. use only.

Objectives

This lesson explains how to obtain data from more than one table. A *join* is used to view information from multiple tables. Therefore, you can *join* tables together to view information from more than one table.

Note: Information about joins is found in the section on "SQL Queries and Subqueries: Joins" in *Oracle Database SQL Language Reference 11g, Release 1 (11.1)*.

Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
...			
18	174	Abel	80
19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

Obtaining Data from Multiple Tables

Sometimes you need to use data from more than one table. In the example in the slide, the report displays data from two separate tables:

- Employee IDs exist in the EMPLOYEES table.
- Department IDs exist in both the EMPLOYEES and DEPARTMENTS tables.
- Department names exist in the DEPARTMENTS table.

To produce the report, you need to link the EMPLOYEES and DEPARTMENTS tables, and access data from both of them.

Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition in a `WHERE` clause.

Cartesian Products

When a join condition is invalid or omitted completely, the result is a *Cartesian product,* in which all combinations of rows are displayed. In other words, all rows in the first table are joined to all rows in the second table.

A Cartesian product tends to generate a large number of rows and the result is rarely useful. Therefore, you should always include a valid join condition unless you have a specific need to combine all rows from all tables.

However, Cartesian products are useful for some tests when you need to generate a large number of rows to simulate a reasonable amount of data.

Generating a Cartesian Product

EMPLOYEES (20 rows)

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200 Whalen		10
2	201 Hartstein		20
3	202 Fay		20
4	205 Higgins		110

...

| 19 | 176 Taylor | | 80 |
| 20 | 178 Grant | | (null) |

DEPARTMENTS (8 rows)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700

Cartesian product:
20 x 8 = 160 rows

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10	1700
2	201	20	1700

...

| 21 | 200 | 10 | 1800 |
| 22 | 201 | 20 | 1800 |

...

| 159 | 176 | 80 | 1700 |
| 160 | 178 | (null) | 1700 |

Generating a Cartesian Product

A Cartesian product is generated when a join condition is omitted. The example in the slide displays the last name of the employee and the department name from the EMPLOYEES and DEPARTMENTS tables, respectively. Because no join condition has been specified, all rows (20 rows) from the EMPLOYEES table are joined with all rows (8 rows) in the DEPARTMENTS table, thereby generating 160 rows in the output.

```
SELECT last_name, department_name dept_name
FROM employees, departments;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration

...

158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting

Types of Oracle-Proprietary Joins

- Equijoin
- Nonequijoin
- Outer join
- Self-join

Types of Joins

To join tables, you can use Oracle's join syntax.

Note: Before the Oracle9*i* release, the join syntax was proprietary. The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax.

Oracle does not have an equivalent syntax to support the FULL OUTER JOIN of the SQL:1999–compliant join syntax.

Joining Tables Using Oracle Syntax

Use a join to query data from more than one table:

```
SELECT    table1.column, table2.column
FROM      table1, table2
WHERE     table1.column1 = table2.column2;
```

- Write the join condition in the `WHERE` clause.
- Prefix the column name with the table name when the same column name appears in more than one table.

Joining Tables Using Oracle Syntax

When data from more than one table in the database is required, a *join* condition is used. Rows in one table can be joined to rows in another table according to common values that exist in the corresponding columns (that is, usually primary and foreign key columns).

To display data from two or more related tables, write a simple join condition in the `WHERE` clause.

In the syntax:

table1.column	Denotes the table and column from which data is retrieved
table1.column1 = table2.column2	Is the condition that joins (or relates) the tables together

Guidelines

- When writing a `SELECT` statement that joins tables, precede the column name with the table name for clarity and to enhance database access.
- If the same column name appears in more than one table, the column name must be prefixed with the table name.
- To join *n* tables together, you need a minimum of n-1 join conditions. For example, to join four tables, a minimum of three joins is required. This rule may not apply if your table has a concatenated primary key, in which case more than one column is required to uniquely identify each row.

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use table aliases, instead of full table name prefixes.
- Table aliases give a table a shorter name.
  - Keeps SQL code smaller, uses less memory
- Use column aliases to distinguish columns that have identical names, but reside in different tables.

Qualifying Ambiguous Column Names

When joining two or more tables, you need to qualify the names of the columns with the table name to avoid ambiguity. Without the table prefixes, the DEPARTMENT_ID column in the SELECT list could be from either the DEPARTMENTS table or the EMPLOYEES table. Therefore, it is necessary to add the table prefix to execute your query. If there are no common column names between the two tables, there is no need to qualify the columns. However, using a table prefix improves performance, because you tell the Oracle server exactly where to find the columns.

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. Therefore, you can use *table aliases*, instead of table names. Just as a column alias gives a column another name, a table alias gives a table another name. Table aliases help to keep SQL code smaller, thereby using less memory.

The table name is specified in full, followed by a space and then the table alias. For example, the EMPLOYEES table can be given an alias of e, and the DEPARTMENTS table an alias of d.

Guidelines

- Table aliases can be up to 30 characters in length, but shorter aliases are better than longer ones.
- If a table alias is used for a particular table name in the FROM clause, that table alias must be substituted for the table name throughout the SELECT statement.
- Table aliases should be meaningful.
- A table alias is valid only for the current SELECT statement.

Equijoins

EMPLOYEES   DEPARTMENTS

Foreign key

Primary key

Equijoins

To determine an employee's department name, you compare the value in the DEPARTMENT_ID column in the EMPLOYEES table with the DEPARTMENT_ID values in the DEPARTMENTS table. The relationship between the EMPLOYEES and DEPARTMENTS tables is an *equijoin*; that is, values in the DEPARTMENT_ID column in both tables must be equal. Often, this type of join involves primary and foreign key complements.

Note: Equijoins are also called *simple joins* or *inner joins*.

Retrieving Records with Equijoins

```
SELECT  e.employee_id, e.last_name, e.department_id,
        d.department_id, d.location_id
FROM    employees e, departments d
WHERE   e.department id = d.department id;
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	144	Vargas	50	50	1500
5	143	Matos	50	50	1500
6	142	Davies	50	50	1500
7	141	Rajs	50	50	1500
8	124	Mourgos	50	50	1500
9	103	Hunold	60	60	1400
10	104	Ernst	60	60	1400
11	107	Lorentz	60	60	1400

...

Retrieving Records with Equijoins

In the example in the slide:

- **The SELECT clause specifies the column names to retrieve:**
    - Employee last name, employee number, and department number, which are columns in the EMPLOYEES table
    - Department number, department name, and location ID, which are columns in the DEPARTMENTS table
- **The FROM clause specifies the two tables that the database must access:**
    - EMPLOYEES table
    - DEPARTMENTS table
- **The WHERE clause specifies how the tables are to be joined:**
    - e.department_id = d.department_id

Because the DEPARTMENT_ID column is common to both tables, it must be prefixed with the table alias to avoid ambiguity. Other columns that are not present in both the tables need not be qualified by a table alias, but it is recommended for better performance.

Note: When you use the Execute Statement icon to run the query, SQL Developer suffixes a "_1" to differentiate between the two DEPARTMENT_IDs.

Retrieving Records with Equijoins: Example

```
SELECT d.department_id, d.department_name,
       d.location_id, l.city
FROM   departments d, locations l
WHERE  d.location_id = l.location_id;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

ORACLE

Oracle University and NETEC, S.A. de C.V. use only.

Retrieving Records with Equijoins: Example

In the example in the slide, the LOCATIONS table is joined to the DEPARTMENTS table by the
LOCATION_ID column, which is the only column of the same name in both the tables. Table aliases
are used to qualify the columns and avoid ambiguity.

Additional Search Conditions
Using the AND Operator

```
SELECT   d.department_id, d.department_name, l.city
FROM     departments d, locations l
WHERE    d.location_id = l.location_id
AND d.department id IN (20, 50);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	CITY
1	20	Marketing	Toronto
2	50	Shipping	South San Francisco

Additional Search Conditions Using the AND Operator

In addition to the join, you may have criteria for your WHERE clause to restrict the rows under consideration for one or more tables in the join. The example in the slide limits the rows of output to those with a department ID equal to 20 or 50:

For example, to display employee Matos' department number and department name, you need an additional condition in the WHERE clause.

```
SELECT  e.last_name, e.department_id,
        d.department_name
FROM    employees e, departments d
WHERE   e.department_id = d.department_id
AND     last_name = 'Matos';
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Matos	50	Shipping

Joining More than Two Tables

EMPLOYEES

	LAST_NAME	DEPARTMENT_ID
1	King	90
2	Kochhar	90
3	De Haan	90
4	Hunold	60
5	Ernst	60
6	Lorentz	60
7	Mourgos	50
8	Rajs	50
9	Davies	50
10	Matos	50

...

DEPARTMENTS

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	50	1500
4	60	1400
5	80	2500
6	90	1700
7	110	1700
8	190	1700

LOCATIONS

	LOCATION_ID	CITY
1	1400	Southlake
2	1500	South San Francisco
3	1700	Seattle
4	1800	Toronto
5	2500	Oxford

To join *n* tables together, you need a minimum of n–1 join conditions. For example, to join three tables, a minimum of two joins is required.

Joining More than Two Tables

Sometimes you may need to join more than two tables. For example, to display the last name, the department name, and the city for each employee, you have to join the EMPLOYEES, DEPARTMENTS, and LOCATIONS tables.

```
SELECT e.last_name, d.department_name, l.city
FROM   employees e, departments d, locations l
WHERE  e.department_id = d.department_id
AND    d.location_id = l.location_id;
```

	LAST_NAME	DEPARTMENT_NAME	CITY
1	Abel	Sales	Oxford
2	Davies	Shipping	South San Francisco
3	De Haan	Executive	Seattle
4	Ernst	IT	Southlake
5	Fay	Marketing	Toronto

...

Nonequijoins

EMPLOYEES

	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000
...		
19	Taylor	8600
20	Grant	7000

JOB_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

`JOB_GRADES` **table defines** `LOWEST_SAL`
and `HIGHEST_SAL` **range of values for**
each `GRADE_LEVEL`**. Therefore, the**
`GRADE_LEVEL` **column can be used to**
assign grades to each employee.

ORACLE

Nonequijoins

A nonequijoin is a join condition containing something other than an equality operator.

The relationship between the EMPLOYEES table and the JOB_GRADES table is an example of a nonequijoin. The SALARY column in the EMPLOYEES table ranges between the values in the LOWEST_SAL and HIGHEST_SAL columns of the JOB_GRADES table. Therefore, each employee can be graded based on the salary. The relationship is obtained using an operator other than the equality operator (=).

Retrieving Records with Nonequijoins

```
SELECT e.last_name, e.salary, j.grade_level
FROM    employees e, job_grades j
WHERE   e.salary
        BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

Retrieving Records with Nonequijoins

The example in the slide creates a nonequijoin to evaluate an employee's salary grade. The salary must be *between* any pair of the low and high salary ranges.

It is important to note that all employees appear exactly once when this query is executed. No employee is repeated in the list. There are two reasons for this:

- None of the rows in the job grade table contain grades that overlap. That is, the salary value for an employee can lie only between the low salary and high salary values of one of the rows in the salary grade table.
- All of the employees' salaries lie within the limits that are provided by the job grade table. That is, no employee earns less than the lowest value contained in the LOWEST_SAL column or more than the highest value contained in the HIGHEST_SAL column.

Note: Other conditions (such as <= and >=) can be used, but BETWEEN is the simplest. Remember to specify the low value first and the high value last when using the BETWEEN condition. The Oracle server translates the BETWEEN condition to a pair of AND conditions. Therefore, using BETWEEN has no performance benefits, but should be used only for logical simplicity.

Table aliases have been specified in the example in the slide for performance reasons, not because of possible ambiguity.

Returning Records with No Direct Match with Outer Joins

DEPARTMENTS

	DEPARTMENT_NAME	DEPARTMENT_ID
1	Administration	10
2	Marketing	20
3	Shipping	50
4	IT	60
5	Sales	80
6	Executive	90
7	Accounting	110
8	Contracting	190

EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1	10	Whalen
2	20	Hartstein
3	20	Fay
4	110	Higgins
5	110	Gietz
6	90	King
7	90	Kochhar
8	90	De Haan
9	60	Hunold
10	60	Ernst

...

18	80	Abel
19	80	Taylor

There are no employees in department 190.

Returning Records with No Direct Match with Outer Joins

If a row does not satisfy a join condition, the row does not appear in the query result. For example, in the equijoin condition of the EMPLOYEES and DEPARTMENTS tables, department ID 190 does not appear because there are no employees with that department ID recorded in the EMPLOYEES table. Similarly, there is an employee whose DEPARTMENT_ID is set to NULL, so this row will also not appear in the query result of an equijoin. To return the department record that does not have any employees, or to return the employee record that does not belong to any department, you can use the outer join.

Outer Joins: Syntax

- You use an outer join to see rows that do not meet the join condition.
- The outer join operator is the plus sign (+).

```
SELECT  table1.column, table2.column
FROM    table1, table2
WHERE   table1.column(+) = table2.column;
```

```
SELECT  table1.column, table2.column
FROM    table1, table2
WHERE   table1.column = table2.column(+);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Outer Joins: Syntax

Missing rows can be returned if an *outer join* operator is used in the join condition. The operator is a plus sign enclosed with parentheses (+), and is placed on the "side" of the join that is deficient in the information. This operator has the effect of creating one or more null rows, to which one or more rows from the nondeficient table can be joined.

In the syntax:

table1.column =	Is the condition that joins (or relates) the tables together
table2.column (+)	Is the outer join symbol, which can be placed on either side of the WHERE clause condition, but not on both sides (Place the outer join symbol following the name of the column in the table without the matching rows.)

Oracle University and NETEC, S.A. de C.V. use only.

Development Program (WDP) eKit materials are provided for WDP in-class use only. Copying eKit materials is strictly prohibited and is in violation of Oracle copyright. All WDP students must receive an eKit watermarked with their name and email. Contact OracleWDP_ww@oracle.com if you have not received your personalized eKit.

Oracle Database 11g: SQL Fundamentals I   F - 17

Using Outer Joins

```
SELECT e.last_name, e.department_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id(+) = d.department_id ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping
9	Hunold	60	IT
10	Ernst	60	IT

...

19	Gietz	110	Accounting
20	(null)	(null)	Contracting

ORACLE

Using Outer Joins

The example in the slide displays employee last names, department IDs, and department names. The Contracting department does not have any employees. The empty value is shown in the output.

Outer Join Restrictions

- The outer join operator can appear only on *one* side of the expression—the side in which the information is missing. It returns those rows, from one table, that have no direct match in the other table.
- A condition involving an outer join cannot use the IN operator or be linked to another condition by the OR operator.

Note: Oracle's join syntax does not have an equivalent for the FULL OUTER JOIN of the SQL:1999– compliant join syntax.

Outer Join: Another Example

```
SELECT e.last_name, e.department_id, d.department_name
FROM    employees e, departments d
WHERE   e.department_id = d.department_id(+) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

...

16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

Outer Join: Another Example

The query in the example in the slide retrieves all rows in the EMPLOYEES table, even if there is no match in the DEPARTMENTS table.

Joining a Table to Itself

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst

...

`MANAGER_ID` in the `WORKER` table is equal to
`EMPLOYEE_ID` in the `MANAGER` table.

Joining a Table to Itself

Sometimes you need to join a table to itself. To find the name of each employee's manager, you need to join the EMPLOYEES table to itself, or perform a self-join. For example, to find the name of Lorentz's manager, you need to:

- Find Lorentz in the EMPLOYEES table by looking at the LAST_NAME column
- Find the manager number for Lorentz by looking at the MANAGER_ID column. Lorentz's manager number is 103.
- Find the name of the manager with EMPLOYEE_ID 103 by looking at the LAST_NAME column. Hunold's employee number is 103, so Hunold is Lorentz's manager.

In this process, you look in the table twice. The first time you look in the table to find Lorentz in the LAST_NAME column and the MANAGER_ID value of 103. The second time you look in the EMPLOYEE_ID column to find 103 and the LAST_NAME column to find Hunold.

Self-Join: Example

```
SELECT worker.last_name || ' works for '
       || manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id ;
```

	WORKER.LAST_NAME||'WORKSFOR'||MANAGER.LAST_NAME
1	Hunold works for De Haan
2	Fay works for Hartstein
3	Gietz works for Higgins
4	Lorentz works for Hunold
5	Ernst works for Hunold
6	Zlotkey works for King
7	Mourgos works for King
8	Kochhar works for King
9	Hartstein works for King
10	De Haan works for King

...

Self-Join: Example

The example in the slide joins the EMPLOYEES table to itself. To simulate two tables in the FROM clause, there are two aliases, namely worker and manager, for the same table, EMPLOYEES.

In this example, the WHERE clause contains the join that means "where a worker's manager number matches the employee number for the manager."

Summary

In this appendix, you should have learned how to use joins to display data from multiple tables by using Oracle-proprietary syntax.

Summary

There are multiple ways to join tables.

Types of Joins

- Equijoins
- Nonequijoins
- Outer joins
- Self-joins

Cartesian Products

A Cartesian product results in a display of all combinations of rows. This is done by omitting the WHERE clause.

Table Aliases

- Table aliases speed up database access.
- Table aliases can help to keep SQL code smaller by conserving memory.

Practice F: Overview

This practice covers the following topics:
- Joining tables by using an equijoin
- Performing outer and self-joins
- Adding conditions

Practice F: Overview

This practice is intended to give you practical experience in extracting data from more than one table using the Oracle join syntax.

Appendix AP
Additional Practices and Solutions

Table of Contents

Practice 1-1

These exercises can be used for extra practice after you have discussed the following topics: basic `SQL SELECT` statement, basic SQL Developer commands, and SQL functions.

1) The HR department needs to find data for all the clerks who were hired after the year 1997.

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
1	143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98	ST_CLERK	2600
2	144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98	ST_CLERK	2500

2) The HR department needs a report of employees who earn commission. Show the last name, job, salary, and commission of those employees. Sort the data by salary in descending order.

	LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
1	Abel	SA_REP	11000	0.3
2	Zlotkey	SA_MAN	10500	0.2
3	Taylor	SA_REP	8600	0.2
4	Grant	SA_REP	7000	0.15

3) For budgeting purposes, the HR department needs a report on projected raises. The report should display those employees who have no commission, but who have a 10% raise in salary (round off the salaries).

	New salary
1	The salary of King after a 10% raise is 26400
2	The salary of Kochhar after a 10% raise is 18700
3	The salary of De Haan after a 10% raise is 18700
4	The salary of Hunold after a 10% raise is 9900
5	The salary of Ernst after a 10% raise is 6600
6	The salary of Lorentz after a 10% raise is 4620
7	The salary of Mourgos after a 10% raise is 6380
8	The salary of Rajs after a 10% raise is 3850
9	The salary of Davies after a 10% raise is 3410
10	The salary of Matos after a 10% raise is 2860
11	The salary of Vargas after a 10% raise is 2750
12	The salary of Whalen after a 10% raise is 4840
13	The salary of Hartstein after a 10% raise is 14300
14	The salary of Fay after a 10% raise is 6600
15	The salary of Higgins after a 10% raise is 13200
16	The salary of Gietz after a 10% raise is 9130

Practice 1-1 (continued)

4) Create a report of employees and their length of employment. Show the last names of all the employees together with the number of years and the number of completed months that they have been employed. Order the report by the length of their employment. The employee who has been employed the longest should appear at the top of the list.

	LAST_NAME	YEARS	MONTHS
1	King	22	0
2	Whalen	21	9
3	Kochhar	19	9
4	Hunold	19	6
5	Ernst	18	1
6	De Haan	16	6
7	Higgins	15	1
8	Gietz	15	1
9	Rajs	13	8
10	Hartstein	13	4
11	Abel	13	2
12	Davies	12	5
13	Fay	11	10
14	Matos	11	4
15	Taylor	11	3
16	Vargas	11	0
17	Lorentz	10	5
18	Grant	10	1
19	Mourgos	9	7
20	Zlotkey	9	5

5) Show those employees who have a last name starting with the letters "J," "K," "L," or "M."

	LAST_NAME
1	King
2	Kochhar
3	Lorentz
4	Matos
5	Mourgos

Oracle Database 11g: SQL Fundamentals I   AP - 5

Practice 1-1 (continued)

6) Create a report that displays all employees, and indicate with the words *Yes* or *No* whether they receive a commission. Use the DECODE expression in your query.

	LAST_NAME	SALARY	COMMISSION
1	King	24000	No
2	Kochhar	17000	No
3	De Haan	17000	No
4	Hunold	9000	No
5	Ernst	6000	No
6	Lorentz	4200	No
7	Mourgos	5800	No
8	Rajs	3500	No
9	Davies	3100	No
10	Matos	2600	No
11	Vargas	2500	No
12	Zlotkey	10500	Yes
13	Abel	11000	Yes
14	Taylor	8600	Yes
15	Grant	7000	Yes
16	Whalen	4400	No
17	Hartstein	13000	No
18	Fay	6000	No
19	Higgins	12000	No
20	Gietz	8300	No

These exercises can be used for extra practice after you have discussed the following topics: basic SQL SELECT statement, basic SQL Developer commands, SQL functions, joins, and group functions.

7) Create a report that displays the department name, location ID, last name, job title, and salary of those employees who work in a specific location. Prompt the user for the location. For example, if the user enters 1800, these are the results:

	DEPARTMENT_NAME	LOCATION_ID	LAST_NAME	JOB_ID	SALARY
1	Marketing	1800	Hartstein	MK_MAN	13000
2	Marketing	1800	Fay	MK_REP	6000

8) Find the number of employees who have a last name that ends with the letter "n." Create two possible solutions.

	COUNT(*)
1	3

Practice 1-1 (continued)

9) Create a report that shows the name, location, and number of employees for each department. Make sure that the report also includes departments without employees.

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	COUNT(E.EMPLOYEE_ID)
1	80	Sales	2500	3
2	110	Accounting	1700	2
3	10	Administration	1700	1
4	60	IT	1400	3
5	20	Marketing	1800	2
6	90	Executive	1700	3
7	50	Shipping	1500	5
8	190	Contracting	1700	0

10) The HR department needs to find the job titles in departments 10 and 20. Create a report to display the job IDs for those departments.

	JOB_ID
1	AD_ASST
2	MK_MAN
3	MK_REP

11) Create a report that displays the jobs that are found in the Administration and Executive departments. Also display the number of employees for these jobs. Show the job with the highest number of employees first.

	JOB_ID	FREQUENCY
1	AD_VP	2
2	AD_PRES	1
3	AD_ASST	1

These exercises can be used for extra practice after you have discussed the following topics: basic SQL SELECT statements, basic SQL Developer commands, SQL functions, joins, group functions, and subqueries.

12) Show all the employees who were hired in the first half of the month (before the 16th of the month).

	LAST_NAME	HIRE_DATE
1	De Haan	13-JAN-93
2	Hunold	03-JAN-90
3	Lorentz	07-FEB-99
4	Matos	15-MAR-98
5	Vargas	09-JUL-98
6	Abel	11-MAY-96
7	Higgins	07-JUN-94
8	Gietz	07-JUN-94

Practice 1-1 (continued)

13) Create a report that displays the following for all employees: last name, salary, and salary expressed in terms of thousands of dollars.

	LAST_NAME	SALARY	THOUSANDS
1	King	24000	24
2	Kochhar	17000	17
3	De Haan	17000	17
4	Hunold	9000	9
5	Ernst	6000	6
6	Lorentz	4200	4
7	Mourgos	5800	5
8	Rajs	3500	3
9	Davies	3100	3
10	Matos	2600	2
11	Vargas	2500	2
12	Zlotkey	10500	10
13	Abel	11000	11
14	Taylor	8600	8
15	Grant	7000	7
16	Whalen	4400	4
17	Hartstein	13000	13
18	Fay	6000	6
19	Higgins	12000	12
20	Gietz	8300	8

14) Show all the employees who have managers with a salary higher than $15,000. Show the following data: employee name, manager name, manager salary, and salary grade of the manager.

	LAST_NAME	MANAGER	SALARY	GRADE_LEVEL
1	De Haan	King	24000	E
2	Hartstein	King	24000	E
3	Higgins	Kochhar	17000	E
4	Hunold	De Haan	17000	E
5	Kochhar	King	24000	E
6	Mourgos	King	24000	E
7	Whalen	Kochhar	17000	E
8	Zlotkey	King	24000	E

Practice 1-1 (continued)

15) Show the department number, name, number of employees, and average salary of all the departments, together with the names, salaries, and jobs of the employees working in each department.

	DEPARTMENT_ID	DEPARTMENT_NAME	EMPLOYEES	AVG_SAL	LAST_NAME	SALARY	JOB_ID
1	10	Administration	1	4400.00	Whalen	4400	AD_ASST
2	20	Marketing	2	9500.00	Hartstein	13000	MK_MAN
3	20	Marketing	2	9500.00	Fay	6000	MK_REP
4	50	Shipping	5	3500.00	Davies	3100	ST_CLERK
5	50	Shipping	5	3500.00	Matos	2600	ST_CLERK
6	50	Shipping	5	3500.00	Rajs	3500	ST_CLERK
7	50	Shipping	5	3500.00	Mourgos	5800	ST_MAN
8	50	Shipping	5	3500.00	Vargas	2500	ST_CLERK
9	60	IT	3	6400.00	Hunold	9000	IT_PROG
10	60	IT	3	6400.00	Lorentz	4200	IT_PROG
11	60	IT	3	6400.00	Ernst	6000	IT_PROG
12	80	Sales	3	10033.33	Zlotkey	10500	SA_MAN
13	80	Sales	3	10033.33	Taylor	8600	SA_REP
14	80	Sales	3	10033.33	Abel	11000	SA_REP
15	90	Executive	3	19333.33	Kochhar	17000	AD_VP
16	90	Executive	3	19333.33	De Haan	17000	AD_VP
17	90	Executive	3	19333.33	King	24000	AD_PRES
18	110	Accounting	2	10150.00	Gietz	8300	AC_ACCOUNT
19	110	Accounting	2	10150.00	Higgins	12000	AC_MGR
20	(null)	(null)	0	No average	Grant	7000	SA_REP

16) Create a report to display the department number and lowest salary of the department with the highest average salary.

	DEPARTMENT_ID	MIN(SALARY)
1	90	17000

17) Create a report that displays departments where no sales representatives work. Include the department number, department name, manager ID, and the location in the output.

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	90	Executive	100	1700
6	110	Accounting	205	1700
7	190	Contracting	(null)	1700

Practice 1-1 (continued)

18) Create the following statistical reports for the HR department: Include the department number, department name, and the number of employees working in each department that:

   a) Employs fewer than three employees:

	DEPARTMENT_ID	DEPARTMENT_NAME	COUNT(*)
1	10	Administration	1
2	110	Accounting	2
3	20	Marketing	2

   b) Has the highest number of employees:

	DEPARTMENT_ID	DEPARTMENT_NAME	COUNT(*)
1	50	Shipping	5

   c) Has the lowest number of employees:

	DEPARTMENT_ID	DEPARTMENT_NAME	COUNT(*)
1	10	Administration	1

19) Create a report that displays the employee number, last name, salary, department number, and the average salary in their department for all employees.

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	SALARY	AVG(S.SALARY)
1	149	Zlotkey	80	10500	10033.3333333333...
2	174	Abel	80	11000	10033.3333333333...
3	144	Vargas	50	2500	3500
4	101	Kochhar	90	17000	19333.3333333333...
5	100	King	90	24000	19333.3333333333...
6	103	Hunold	60	9000	6400
7	142	Davies	50	3100	3500
8	205	Higgins	110	12000	10150
9	104	Ernst	60	6000	6400
10	143	Matos	50	2600	3500
11	102	De Haan	90	17000	19333.3333333333...
12	107	Lorentz	60	4200	6400
13	141	Rajs	50	3500	3500
14	200	Whalen	10	4400	4400
15	202	Fay	20	6000	9500
16	176	Taylor	80	8600	10033.3333333333...
17	201	Hartstein	20	13000	9500
18	206	Gietz	110	8300	10150
19	124	Mourgos	50	5800	3500

Practice 1-1 (continued)

20) Show all the employees who were hired on the day of the week on which the highest number of employees were hired.

	LAST_NAME	DAY
1	Ernst	TUESDAY
2	Mourgos	TUESDAY
3	Rajs	TUESDAY
4	Taylor	TUESDAY
5	Higgins	TUESDAY
6	Gietz	TUESDAY

21) Create an anniversary overview based on the hire date of the employees. Sort the anniversaries in ascending order.

	LAST_NAME	BIRTHDAY
1	Hunold	January 03
2	De Haan	January 13
3	Davies	January 29
4	Zlotkey	January 29
5	Lorentz	February 07
6	Hartstein	February 17
7	Matos	March 15
8	Taylor	March 24
9	Abel	May 11
10	Ernst	May 21
11	Grant	May 24
12	Higgins	June 07
13	Gietz	June 07
14	King	June 17
15	Vargas	July 09
16	Fay	August 17
17	Whalen	September 17
18	Kochhar	September 21
19	Rajs	October 17
20	Mourgos	November 16

Oracle University and NETEC, S.A. de C.V. use only.

Practice Solutions 1-1

These exercises can be used for extra practice after you have discussed the following topics: basic SQL SELECT statement, basic SQL Developer commands, and SQL functions.

1) The HR department needs to find data for all of the clerks who were hired after the year 1997.

```
SELECT *
FROM    employees
WHERE   job_id = 'ST_CLERK'
AND hire_date > '31-DEC-1997';
```

2) The HR department needs a report of employees who earn commission. Show the last name, job, salary, and commission of those employees. Sort the data by salary in descending order.

```
SELECT last_name, job_id, salary, commission_pct
FROM    employees
WHERE   commission_pct IS NOT NULL
ORDER BY salary DESC;
```

3) For budgeting purposes, the HR department needs a report on projected raises. The report should display those employees who do not get a commission but who have a 10% raise in salary (round off the salaries).

```
SELECT 'The salary of '||last_name||' after a 10% raise is '
        || ROUND(salary*1.10) "New salary"
FROM    employees
WHERE   commission_pct IS NULL;
```

4) Create a report of employees and their duration of employment. Show the last names of all employees together with the number of years and the number of completed months that they have been employed. Order the report by the duration of their employment. The employee who has been employed the longest should appear at the top of the list.

```
SELECT last_name,
       TRUNC(MONTHS_BETWEEN(SYSDATE, hire_date) / 12) YEARS,
       TRUNC(MOD(MONTHS_BETWEEN(SYSDATE, hire_date), 12))
    MONTHS
FROM employees
ORDER BY years DESC, MONTHS desc;
```

5) Show those employees who have a last name starting with the letters "J," "K," "L," or "M."

```
SELECT  last_name
FROM    employees
WHERE   SUBSTR(last_name, 1,1) IN ('J', 'K', 'L', 'M');
```

Practice Solutions 1-1 (continued)

6) Create a report that displays all employees, and indicate with the words *Yes* or *No* whether they receive a commission. Use the DECODE expression in your query.

```
SELECT last_name, salary,
       decode(commission_pct, NULL, 'No', 'Yes') commission
FROM   employees;
```

These exercises can be used for extra practice after you have discussed the following topics: basic SQL SELECT statement, basic SQL Developer commands, SQL functions, joins, and group functions.

7) Create a report that displays the department name, location ID, name, job title, and salary of those employees who work in a specific location. Prompt the user for the location.

   a) Enter 1800 for location_id when prompted.

```
SELECT d.department_name, d.location_id, e.last_name,
e.job_id, e.salary
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND    d.location_id = &location_id;
```

8) Find the number of employees who have a last name that ends with the letter "n." Create two possible solutions.

```
SELECT COUNT(*)
FROM   employees
WHERE  last_name LIKE '%n';
--or
SELECT COUNT(*)
FROM   employees
WHERE  SUBSTR(last_name, -1) = 'n';
```

9) Create a report that shows the name, location, and number of employees for each department. Make sure that the report also includes departments without employees.

```
SELECT d.department_id, d.department_name,
       d.location_id,   COUNT(e.employee_id)
FROM   employees e RIGHT OUTER JOIN  departments d
ON     e.department_id = d.department_id
GROUP BY d.department_id, d.department_name, d.location_id;
```

10) The HR department needs to find the job titles in departments 10 and 20. Create a report to display the job IDs for those departments.

```
SELECT DISTINCT job_id
FROM   employees
WHERE  department_id IN (10, 20);
```

Practice Solutions 1-1 (continued)

11) Create a report that displays the jobs that are found in the Administration and
Executive departments. Also display the number of employees for these jobs. Show
the job with the highest number of employees first.

```
SELECT e.job_id, count(e.job_id) FREQUENCY
FROM       employees e JOIN departments d
ON  e.department_id = d.department_id
WHERE    d.department_name IN ('Administration', 'Executive')
GROUP BY e.job_id
ORDER BY FREQUENCY DESC;
```

These exercises can be used for extra practice after you have discussed the following
topics: basic SQL SELECT statements, basic SQL Developer commands, SQL functions,
joins, group functions, and subqueries.

12) Show all employees who were hired in the first half of the month (before the 16th of
the month).

```
SELECT last_name, hire_date
FROM   employees
WHERE  TO_CHAR(hire_date, 'DD') < 16;
```

13) Create a report that displays the following for all employees: last name, salary, and
salary expressed in terms of thousands of dollars.

```
SELECT last_name, salary, TRUNC(salary, -3)/1000  Thousands
FROM   employees;
```

14) Show all employees who have managers with a salary higher than $15,000. Show the
following data: employee name, manager name, manager salary, and salary grade of
the manager.

```
SELECT e.last_name, m.last_name manager, m.salary,
j.grade_level
FROM   employees e JOIN employees m
ON     e.manager_id = m.employee_id
JOIN   job_grades j
ON     m.salary BETWEEN j.lowest_sal AND j.highest_sal
AND    m.salary > 15000;
```

Practice Solutions 1-1 (continued)

15) Show the department number, name, number of employees, and average salary of all departments, together with the names, salaries, and jobs of the employees working in each department.

```
SELECT  d.department_id, d.department_name,
        count(e1.employee_id) employees,
        NVL(TO_CHAR(AVG(e1.salary), '99999.99'), 'No average'
) avg_sal,
        e2.last_name, e2.salary, e2.job_id
FROM    departments d RIGHT OUTER JOIN employees e1
ON      d.department_id = e1.department_id
RIGHT OUTER JOIN  employees e2
ON    d.department_id = e2.department_id
GROUP BY d.department_id, d.department_name, e2.last_name,
e2.salary,
        e2.job_id
ORDER BY d.department_id, employees;
```

16) Create a report to display the department number and lowest salary of the department with the highest average salary.

```
SELECT department_id, MIN(salary)
FROM    employees
GROUP BY department_id
HAVING AVG(salary) = (SELECT MAX(AVG(salary))
                      FROM    employees
                      GROUP BY department_id);
```

17) Create a report that displays the departments where no sales representatives work. Include the department number, department name, and location in the output.

```
SELECT *
FROM    departments
WHERE   department_id NOT IN(SELECT department_id
                            FROM employees
                            WHERE job_id = 'SA_REP'
                            AND department_id IS NOT NULL);
```

18) Create the following statistical reports for the HR department: Include the department number, department name, and the number of employees working in each department that:

a) Employs fewer than three employees:

```
SELECT d.department_id, d.department_name, COUNT(*)
FROM    departments d JOIN employees e
ON      d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
HAVING COUNT(*) < 3;
```

Practice Solutions 1-1 (continued)

b) Has the highest number of employees:

```
SELECT d.department_id, d.department_name, COUNT(*)
FROM   departments d JOIN employees e
ON     d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                   FROM   employees
                   GROUP BY department_id);
```

c) Has the lowest number of employees:

```
SELECT d.department_id, d.department_name, COUNT(*)
FROM   departments d JOIN employees e
ON     d.department_id = e.department_id
GROUP BY d.department_id, d.department_name
HAVING COUNT(*) = (SELECT MIN(COUNT(*))
                   FROM   employees
                   GROUP BY department_id);
```

19) Create a report that displays the employee number, last name, salary, department number, and the average salary in their department for all employees.

```
SELECT e.employee_id, e.last_name, e.department_id, e.salary,
AVG(s.salary)
FROM   employees e JOIN employees s
ON     e.department_id = s.department_id
GROUP BY e.employee_id, e.last_name, e.department_id,
e.salary;
```

20) Show all employees who were hired on the day of the week on which the highest number of employees were hired.

```
SELECT last_name, TO_CHAR(hire_date, 'DAY') day
FROM   employees
WHERE  TO_CHAR(hire_date, 'Day') =
       (SELECT TO_CHAR(hire_date, 'Day')
        FROM   employees
        GROUP BY TO_CHAR(hire_date, 'Day')
        HAVING COUNT(*) = (SELECT MAX(COUNT(*))
                           FROM   employees
                           GROUP BY TO_CHAR(hire_date,
'Day')));
```

21) Create an anniversary overview based on the hire date of the employees. Sort the anniversaries in ascending order.

```
SELECT last_name, TO_CHAR(hire_date, 'Month DD') BIRTHDAY
FROM   employees
ORDER BY TO_CHAR(hire_date, 'DDD');
```

In this case study, you build a set of database tables for a video application. After you create the tables, you insert, update, and delete records in a video store database and generate a report. The database contains only the essential tables.

The following is a diagram of the entities and attributes for the video application:
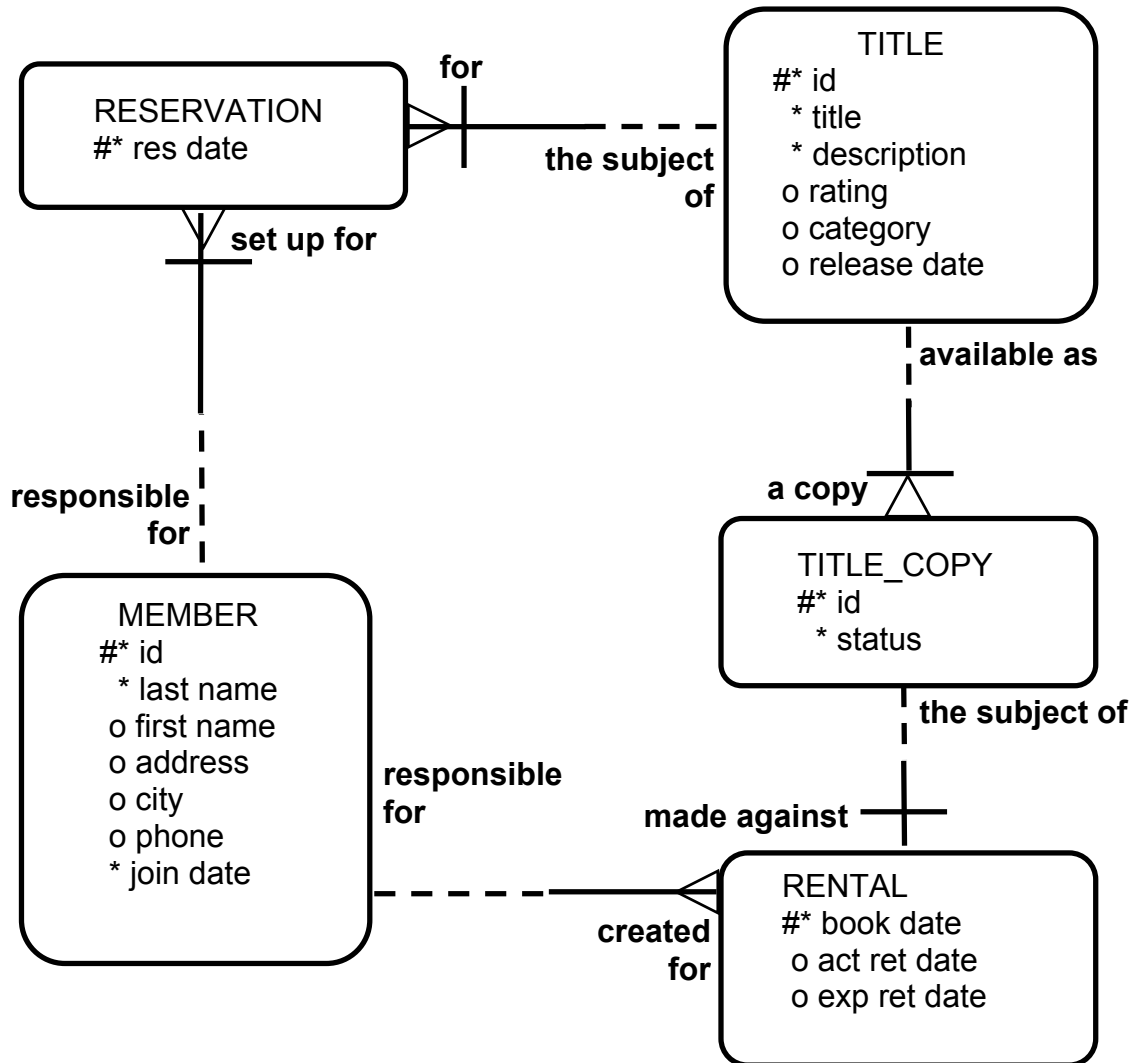
Note: If you want to build the tables, you can execute the commands in the `buildtab.sql` script in SQL Developer. If you want to drop the tables, you can execute the commands in the `dropvid.sql` script in SQL Developer. Then you can execute the commands in the `buildvid.sql` script in SQL Developer to create and populate the tables.

All the three SQL scripts are present in the `/home/oracle/labs/sql1/labs` folder.

- If you use the `buildtab.sql` script to build the tables, start with step 4.

Practice Solutions 1-1 (continued)

- If you use the `dropvid.sql` script to remove the video tables, start with step 1.
- If you use the `buildvid.sql` script to build and populate the tables, start with step 6(b).

Practice 2-1

1) Create the tables based on the following table instance charts. Choose the appropriate data types and be sure to add integrity constraints.

   a) Table name: MEMBER

Column_ Name	MEMBER_ ID	LAST_ NAME	FIRST_NAME	ADDRESS	CITY	PHONE	JOIN _DATE
Key Type	PK						
Null/ Unique	NN,U	NN					NN
Default Value							System Date
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	25	25	100	30	15	

   b) Table name: TITLE

Column_ Name	TITLE_ID	TITLE	DESCRIPTION	RATING	CATEGORY	RELEASE_ DATE
Key Type	PK					
Null/ Unique	NN,U	NN	NN			
Check				G, PG, R, NC17, NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMEN TARY	
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	60	400	4	20	

c) Table name: TITLE_COPY

Column Name	COPY_ID	TITLE_ID	STATUS
Key Type	PK	PK,FK	
Null/ Unique	NN,U	NN,U	NN
Check			AVAILABLE, DESTROYED, RENTED, RESERVED
FK Ref Table		TITLE	
FK Ref Col		TITLE_ID	
Data Type	NUMBER	NUMBER	VARCHAR2
Length	10	10	15

d) Table name: RENTAL

Column Name	BOOK_ DATE	MEMBER_ ID	COPY_ ID	ACT_RET_ DATE	EXP_RET_ DATE	TITLE_ ID
Key Type	PK	PK,FK1	PK,FK2			PK,FK2
Default Value	System Date				System Date + 2 days	
FK Ref Table		MEMBER	TITLE_ COPY			TITLE_ COPY
FK Ref Col		MEMBER_I D	COPY_ ID			TITLE_ID
Data Type	DATE	NUMBER	NUMBER	DATE	DATE	NUMBER
Length		10	10			10

Practice 2-1 (continued)

e) Table name: RESERVATION

Column Name	RES_ DATE	MEMBER_ ID	TITLE_ ID
Key Type	PK	PK,FK1	PK,FK2
Null/ Unique	NN,U	NN,U	NN
FK Ref Table		MEMBER	TITLE
FK Ref Column		MEMBER_ID	TITLE_ID
Data Type	DATE	NUMBER	NUMBER
Length		10	10

2) Verify that the tables were created properly by checking in the Connections Navigator in SQL Developer.
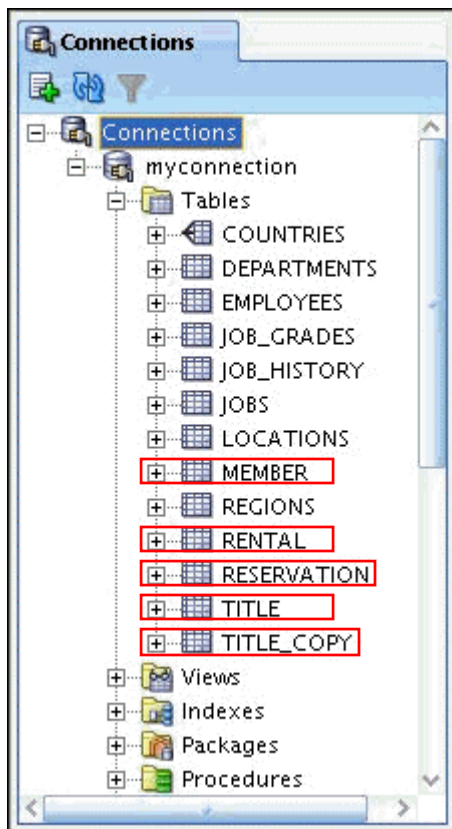
Practice 2-1 (continued)

3) Create sequences to uniquely identify each row in the MEMBER table and the TITLE table.

   a) Member number for the MEMBER table: Start with 101; do not allow caching of the values. Name the sequence MEMBER_ID_SEQ.

   b) Title number for the TITLE table: Start with 92; do not allow caching of the values. Name the sequence TITLE_ID_SEQ.

   c) Verify the existence of the sequences in the Connections Navigator in SQL Developer.
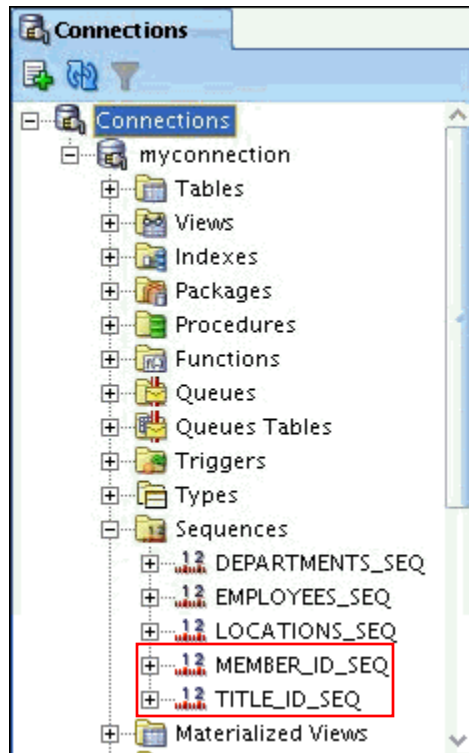
4) Add data to the tables. Create a script for each set of data to be added.

   a) Add movie titles to the TITLE table. Write a script to enter the movie information. Save the statements in a script named lab_apcs_4a.sql. Use the sequences to uniquely identify each title. Enter the release dates in the DD-MON-YYYY format. Remember that single quotation marks in a character field must be specially handled. Verify your additions.

	TITLE
1	Willie and Christmas Too
2	Alien Again
3	The Glob
4	My Day Off
5	Miracles on Ice
6	Soda Gang

Practice 2-1 (continued)

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends make a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-1995
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-1995
The Glob	A meteor crashes near a small American town and unleashes carnivorous goo in this classic.	NR	SCIFI	12-AUG-1995
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York.	PG	COMEDY	12-JUL-1995
Miracles on Ice	A six-year-old has doubts about Santa Claus, but she discovers that miracles really do exist.	PG	DRAMA	12-SEP-1995
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-1995

b) Add data to the MEMBER table. Save the insert statements in a script named
    `lab_apcs_4b.sql`. Execute commands in the script. Be sure to use the
    sequence to add the member numbers.

First_ Name	Last_Name	Address	City	Phone	Join_Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-1990
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-1990
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-1991
Mark	Quick-to-See	6921 King Way	Lagos	63-559-7777	07-APR-1990
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-1991
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-1991

Oracle University and NETEC, S.A. de C.V. use only.

Practice 2-1 (continued)

c) Add the following movie copies in the `TITLE_COPY` table:

Note: Have the `TITLE_ID` numbers available for this exercise.

Title	Copy_Id	Status	Title	Copy_Id
Willie and Christmas Too	1	AVAILABLE	Willie and Christmas Too	1
Alien Again	1	AVAILABLE	Alien Again	1
	2	RENTED		2
The Glob	1	AVAILABLE	The Glob	1
My Day Off	1	AVAILABLE	My Day Off	1
	2	AVAILABLE		2
	3	RENTED		3
Miracles on Ice	1	AVAILABLE	Miracles on Ice	1
Soda Gang	1	AVAILABLE	Soda Gang	1

d) Add the following rentals to the `RENTAL` table:

Note: The title number may be different depending on the sequence number.

Title_ Id	Copy_ Id	Member_Id	Book_date	Exp_Ret_Date
92	1	101	3 days ago	1 day ago
93	2	101	1 day ago	1 day from now
95	3	102	2 days ago	Today
97	1	106	4 days ago	2 days ago

Practice 2-1 (continued)

5) Create a view named `TITLE_AVAIL` to show the movie titles, the availability of each copy, and its expected return date if rented. Query all rows from the view. Order the results by title.

Note: Your results may be different.

	TITLE	COPY_ID	STATUS	EXP_RET_DATE
1	Alien Again	1	AVAILABLE	(null)
2	Alien Again	2	RENTED	15-JUL-09
3	Miracles on Ice	1	AVAILABLE	(null)
4	My Day Off	1	AVAILABLE	(null)
5	My Day Off	2	AVAILABLE	(null)
6	My Day Off	3	RENTED	16-JUL-09
7	Soda Gang	1	AVAILABLE	14-JUL-09
8	The Glob	1	AVAILABLE	(null)
9	Willie and Christmas Too	1	AVAILABLE	15-JUL-09

6) Make changes to the data in the tables.

a) Add a new title. The movie is "Interstellar Wars," which is rated PG and classified as a science fiction movie. The release date is 07-JUL-77. The description is "Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?" Be sure to add a title copy record for two copies.

b) Enter two reservations. One reservation is for Carmen Velasquez, who wants to rent "Interstellar Wars." The other is for Mark Quick-to-See, who wants to rent "Soda Gang."

Practice 2-1 (continued)

7) Make a modification to one of the tables.

a) Run the `lab_apcs_7a.sql` script located in the
`/home/oracle/labs/sql1/labs` folder, to add a `PRICE` column to the `TITLE`
table to record the purchase price of the video. Verify your modifications.

```
DESCRIBE title
Name                                    Null      Type
--------------------------------  --------  --------------
TITLE_ID                          NOT NULL  NUMBER(10)
TITLE                             NOT NULL  VARCHAR2(60)
DESCRIPTION                       NOT NULL  VARCHAR2(400)
RATING                                      VARCHAR2(4)
CATEGORY                                    VARCHAR2(20)
RELEASE_DATE                                DATE
PRICE                                       NUMBER(8,2)
```

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	30
Soda Gang	35
Interstellar Wars	29

b) Create a script named `lab_apcs_7b.sql` that contains update statements that
update each video with a price according to the preceding list. Run the commands
in the script.

Note: Have the `TITLE_ID` numbers available for this exercise.

8) Create a report that contains each customer's history of renting videos. Be sure to
include the customer name, movie rented, dates of the rental, and duration of rentals.
Total the number of rentals for all customers for the reporting period. Save the
commands that generate the report in a script file named `lab_apcs_8.sql`.

Note: Your results may be different.

	MEMBER	TITLE	BOOK_DATE	DURATION
1	Carmen Velasquez	Willie and Christmas Too	13-JUL-09	1
2	Carmen Velasquez	Alien Again	15-JUL-09	(null)
3	LaDoris Ngao	My Day Off	14-JUL-09	(null)
4	Molly Urguhart	Soda Gang	12-JUL-09	2

Practice Solutions 2-1

1) Create the tables based on the following table instance charts. Choose the appropriate data types and be sure to add integrity constraints.

a) Table name: MEMBER

```
CREATE TABLE member
      (member_id          NUMBER(10)
         CONSTRAINT member_member_id_pk PRIMARY KEY,
       last_name       VARCHAR2(25)
         CONSTRAINT member_last_name_nn NOT NULL,
       first_name      VARCHAR2(25),
       address         VARCHAR2(100),
       city            VARCHAR2(30),
       phone           VARCHAR2(15),
       join_date       DATE DEFAULT SYSDATE
         CONSTRAINT member_join_date_nn NOT NULL);
```

b) Table name: TITLE

```
CREATE TABLE title
      (title_id          NUMBER(10)
         CONSTRAINT title_title_id_pk PRIMARY KEY,
       title           VARCHAR2(60)
         CONSTRAINT title_title_nn NOT NULL,
       description     VARCHAR2(400)
         CONSTRAINT title_description_nn NOT NULL,
       rating          VARCHAR2(4)
         CONSTRAINT title_rating_ck CHECK
         (rating IN ('G', 'PG', 'R', 'NC17', 'NR')),
       category        VARCHAR2(20)
         CONSTRAINT title_category_ck CHECK
         (category IN ('DRAMA', 'COMEDY', 'ACTION',
         'CHILD', 'SCIFI', 'DOCUMENTARY')),
       release_date    DATE);
```

c) Table name: TITLE_COPY

```
CREATE TABLE title_copy
      (copy_id         NUMBER(10),
       title_id        NUMBER(10)
         CONSTRAINT title_copy_title_if_fk REFERENCES
title(title_id),
       status          VARCHAR2(15)
         CONSTRAINT title_copy_status_nn NOT NULL
         CONSTRAINT title_copy_status_ck CHECK (status IN
         ('AVAILABLE', 'DESTROYED','RENTED', 'RESERVED')),
       CONSTRAINT title_copy_copy_id_title_id_pk
         PRIMARY KEY (copy_id, title_id));
```

Practice Solutions 2-1 (continued)

d) Table name: RENTAL

```
CREATE TABLE rental
      (book_date    DATE DEFAULT SYSDATE,
       member_id       NUMBER(10)
         CONSTRAINT rental_member_id_fk REFERENCES
member(member_id),
       copy_id      NUMBER(10),
       act_ret_date DATE,
       exp_ret_date DATE DEFAULT SYSDATE + 2,
       title_id     NUMBER(10),
       CONSTRAINT rental_book_date_copy_title_pk
         PRIMARY KEY (book_date, member_id, copy_id,title_id),
       CONSTRAINT rental_copy_id_title_id_fk
         FOREIGN KEY (copy_id, title_id)
         REFERENCES title_copy(copy_id, title_id));
```

e) Table name: RESERVATION

```
CREATE TABLE reservation
      (res_date       DATE,
       member_id        NUMBER(10)
         CONSTRAINT reservation_member_id REFERENCES
member(member_id),
       title_id         NUMBER(10)
         CONSTRAINT reservation_title_id REFERENCES
title(title_id),
       CONSTRAINT reservation_resdate_mem_tit_pk PRIMARY KEY
         (res_date, member_id, title_id));
```

2) Verify that the tables were created properly by checking in the Connections Navigator in SQL Developer.

a) In the Connections Navigator, expand Connections > myconnection > Tables.

3) Create sequences to uniquely identify each row in the MEMBER table and the TITLE table.

a) Member number for the MEMBER table: Start with 101; do not allow caching of the values. Name the sequence MEMBER_ID_SEQ.

```
CREATE SEQUENCE member_id_seq
START WITH 101
NOCACHE;
```

b) Title number for the TITLE table: Start with 92; do not allow caching of the values. Name the sequence TITLE_ID_SEQ.

```
CREATE SEQUENCE title_id_seq
START WITH 92
NOCACHE;
```

Practice Solutions 2-1 (continued)

    c) Verify the existence of the sequences in the Connections Navigator in SQL Developer.

        i) In the Connections Navigator, assuming that the myconnection node is expanded, expand Sequences.

4) Add data to the tables. Create a script for each set of data to be added.

    a) Add movie titles to the TITLE table. Write a script to enter the movie information. Save the statements in a script named `lab_apcs_4a.sql`. Use the sequences to uniquely identify each title. Enter the release dates in the DD-MON-YYYY format. Remember that single quotation marks in a character field must be specially handled. Verify your additions.

```
INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES   (title_id_seq.NEXTVAL, 'Willie and Christmas Too',
          'All of Willie''s friends make a Christmas list for
          Santa, but Willie has yet to add his own wish list.',
          'G', 'CHILD', TO_DATE('05-OCT-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id , title, description, rating,
                  category, release_date)
VALUES   (title_id_seq.NEXTVAL, 'Alien Again', 'Yet another
           installment of science fiction history.  Can the
           heroine save the planet from the alien life form?',
           'R', 'SCIFI', TO_DATE( '19-MAY-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES   (title_id_seq.NEXTVAL, 'The Glob', 'A meteor crashes
          near a small American town and unleashes carnivorous
          goo in this classic.', 'NR', 'SCIFI',
          TO_DATE( '12-AUG-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES    (title_id_seq.NEXTVAL, 'My Day Off', 'With a little
           luck and a lot ingenuity, a teenager skips school
for
           a day in New York.', 'PG', 'COMEDY',
           TO_DATE( '12-JUL-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES    (title_id_seq.NEXTVAL, 'Miracles on Ice', 'A six-
year-old has      doubts about Santa Claus, but she discovers
that miracles really do exist.', 'PG', 'DRAMA',
           TO_DATE('12-SEP-1995','DD-MON-YYYY'))
/
```

Practice Solutions 2-1 (continued)

```
INSERT INTO title(title_id, title, description, rating,
                  category, release_date)
VALUES    (title_id_seq.NEXTVAL, 'Soda Gang', 'After
discovering a cache of drugs, a young couple find themselves
pitted against a vicious gang.', 'NR', 'ACTION', TO_DATE('01-
JUN-1995','DD-MON-YYYY'))
/
COMMIT
/
SELECT  title
FROM    title;
```

b) Add data to the MEMBER table. Place the insert statements in a script named
   `lab_apcs_4b.sql`. Execute the commands in the script. Be sure to use the
   sequence to add the member numbers.

```
SET VERIFY OFF
INSERT INTO member(member_id, first_name, last_name,
            address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Carmen', 'Velasquez',
        '283 King Street', 'Seattle', '206-899-6666',
TO_DATE('08-MAR-1990',
        'DD-MM-YYYY'))
/

INSERT INTO member(member_id, first_name, last_name,
            address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'LaDoris', 'Ngao',
        '5 Modrany', 'Bratislava', '586-355-8882',
TO_DATE('08-MAR-1990',
        'DD-MM-YYYY'))
/

INSERT INTO member(member_id, first_name, last_name,
            address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Midori', 'Nagayama',
        '68 Via Centrale', 'Sao Paolo', '254-852-5764',
TO_DATE('17-JUN-1991',
        'DD-MM-YYYY'))
/

INSERT INTO member(member_id, first_name, last_name,
            address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Mark', 'Quick-to-See',
        '6921 King Way', 'Lagos', '63-559-7777', TO_DATE('07-
APR-1990',
        'DD-MM-YYYY'))
/


INSERT INTO member(member_id, first_name, last_name,
```

Practice Solutions 2-1 (continued)

```
                address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Audry', 'Ropeburn',
        '86 Chu Street', 'Hong Kong', '41-559-87',
TO_DATE('18-JAN-1991',
        'DD-MM-YYYY'))
/

INSERT INTO member(member_id, first_name, last_name,
            address, city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, 'Molly', 'Urguhart',
        '3035 Laurier', 'Quebec', '418-542-9988', TO_DATE('18-
JAN-1991',
        'DD-MM-YYYY'));
/

COMMIT
SET VERIFY ON
```

c) Add the following movie copies in the TITLE_COPY table:

   Note: Have the TITLE_ID numbers available for this exercise.

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 92, 'AVAILABLE')
/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 93, 'AVAILABLE')
/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (2, 93, 'RENTED')
/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 94, 'AVAILABLE')
/
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 95, 'AVAILABLE')
/
INSERT INTO title_copy(copy_id, title_id,status)
VALUES (2, 95, 'AVAILABLE')
/
INSERT INTO title_copy(copy_id, title_id,status)
VALUES (3, 95, 'RENTED')
/
INSERT INTO title_copy(copy_id, title_id,status)
VALUES (1, 96, 'AVAILABLE')
/
INSERT INTO title_copy(copy_id, title_id,status)
VALUES (1, 97, 'AVAILABLE')
/
```

Practice Solutions 2-1 (continued)

d) Add the following rentals to the RENTAL table:

Note: The title number may be different depending on the sequence number.

```
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (92, 1, 101, sysdate-3, sysdate-1, sysdate-2)
/
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (93, 2, 101, sysdate-1, sysdate-1, NULL)
/
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date, act_ret_date)
VALUES (95, 3, 102, sysdate-2, sysdate, NULL)
/
INSERT INTO rental(title_id, copy_id, member_id,
                   book_date, exp_ret_date,act_ret_date)
VALUES (97, 1, 106, sysdate-4, sysdate-2, sysdate-2)
/
COMMIT
/
```

5) Create a view named TITLE_AVAIL to show the movie titles, the availability of each copy, and its expected return date if rented. Query all rows from the view. Order the results by title.

Note: Your results may be different.

```
CREATE VIEW title_avail AS
  SELECT   t.title, c.copy_id, c.status, r.exp_ret_date
  FROM     title t JOIN title_copy c
  ON       t.title_id = c.title_id
  FULL OUTER JOIN rental r
  ON       c.copy_id = r.copy_id
  AND      c.title_id = r.title_id;

SELECT   *
FROM     title_avail
ORDER BY title, copy_id;
```

Practice Solutions 2-1 (continued)

6) Make changes to data in the tables.

a) Add a new title. The movie is "Interstellar Wars," which is rated PG and classified as a science fiction movie. The release date is 07-JUL-77. The description is "Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?" Be sure to add a title copy record for two copies.

```
INSERT INTO title(title_id, title, description, rating,
            category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Interstellar Wars',
      'Futuristic interstellar action movie.  Can the
       rebels save the humans from the evil empire?',
      'PG', 'SCIFI', '07-JUL-77')
/
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (1, 98, 'AVAILABLE')
/
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (2, 98, 'AVAILABLE')
/
```

b) Enter two reservations. One reservation is for Carmen Velasquez, who wants to rent "Interstellar Wars." The other is for Mark Quick-to-See, who wants to rent "Soda Gang."

```
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 101, 98)
/
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 104, 97)
/
```

7) Make a modification to one of the tables.

a) Run the `lab_apcs_7a.sql` script located in the `/home/oracle/labs/sql1/labs` folder, to add a PRICE column to the TITLE table to record the purchase price of the video. Verify your modifications.

```
ALTER TABLE title
ADD  (price NUMBER(8,2));

DESCRIBE title
```

Practice Solutions 2-1 (continued)

b) Create a script named `lab_apcs_7b.sql` that contains update statements that update each video with a price according to the list provided. Run the commands in the script.

Note: Have the `TITLE_ID` numbers available for this exercise.

```
SET ECHO OFF
SET VERIFY OFF
UPDATE title
SET    price = &price
WHERE  title_id = &title_id;
SET VERIFY OFF
SET ECHO OFF
```

8) Create a report that contains each customer's history of renting videos. Be sure to include the customer name, movie rented, dates of the rental, and duration of rentals. Total the number of rentals for all customers for the reporting period. Save the commands that generate the report in a script file named `lab_apcs_8.sql`.

Note: Your results may be different.

```
SELECT  m.first_name||' '||m.last_name MEMBER, t.title,
        r.book_date, r.act_ret_date - r.book_date DURATION
FROM  member m
JOIN rental r
ON   r.member_id = m.member_id
JOIN title t
ON   r.title_id = t.title_id
ORDER BY member;
```

Index

Oracle University and NETEC, S.A. de C.V. use only.

F

Format Model 3-28, 3-30, 4-11, 4-12, 4-14, 4-15, 4-16, 4-19, 4-20, 4-43

Functions 2-5, 2-7, 3-1, 3-2, 3-3, 3-4, 3-5, 3-6, 3-7, 3-8, 3-9, 3-10, 3-11, 3-12, 3-13, 3-14, 3-15, 3-16, 3-17, 3-18, 3-20, 3-24, 3-27, 3-28, 3-29, 3-30, 3-31, 3-32, 3-33, 4-1, 4-2, 4-3, 4-4, 4-7, 4-9, 4-10, 4-20, 4-21, 4-23, 4-24, 4-25, 4-26, 4-27, 4-28, 4-36, 4-44, 4-45, 5-1, 5-2, 5-3, 5-4, 5-5, 5-6, 5-7, 5-8, 5-11, 5-12, 5-13, 5-14, 5-15, 5-19, 5-20, 5-22, 5-25, 5-26, 5-27, 5-28, 5-29, 7-3, 7-9, 7-12, 7-16, 7-21, 7-25, 8-25, 9-9, 10-9, 10-27, C-5, C-6, C-23, C-24, C-25, E-8, E-15, I-4, I-5

G

GROUP BY Clause 5-2, 5-3, 5-12, 5-13, 5-14, 5-15, 5-16, 5-18, 5-19, 5-22, 5-23, 5-25, 5-26, 5-27, 5-28, 7-14

Group Functions 3-5, 5-1, 5-2, 5-3, 5-4, 5-5, 5-6, 5-11, 5-12, 5-13, 5-14, 5-15, 5-19, 5-20, 5-22, 5-25, 5-26, 5-27, 5-28, 5-29, 7-3, 7-9, 7-12, 7-16, 7-21, 7-25, I-5

Group Functions in a Subquery 7-3, 7-9, 7-12, 7-16, 7-21

H

HAVING Clause 5-2, 5-3, 5-12, 5-20, 5-21, 5-22, 5-23, 5-24, 5-25, 5-28, 5-29, 7-3, 7-5, 7-9, 7-13, 7-16, 7-21, 7-25

I

Implicit Data Type Conversion 4-4, 4-5, 4-6

IN Operator 2-11, 7-22, F-17, F-18

Index 10-4, 10-8, 10-10, 10-22, 10-23, 10-36, 10-38, C-4, C-10, E-6, I-38

INSERT Statement 9-3, 9-6, 9-8, 9-12, 9-13, 9-19, 9-25, 9-39, 9-42, 10-10, 10-18

International Standards Organization I-31

INTERSECT Operator 8-3, 8-4, 8-5, 8-7, 8-12, 8-18, 8-19, 8-20, 8-21, 8-24, 8-27, 8-30, 8-31

INTERVAL YEAR TO MONTH 10-14

J

Java C-4, C-8, E-3, E-12, E-14, E-15, E-16, I-9, I-12, I-39

<div style="writing-mode: vertical">Oracle University and NETEC , S.A. de C.V. use only.</div>

R

Relational Database 9-43, I-2, I-3, I-4, I-8, I-15, I-16,
  I-18, I-19, I-27, I-28, I-29, I-30, I-31, I-33, I-36, I-39

REPLACE 2-36, 3-9, 3-13, D-12, D-16

ROUND and TRUNC Functions 3-30, 3-32

ROUND Function 3-17, 3-18, 4-26

RPAD 3-9, 3-13

RR Date Format 3-22, 3-23, 4-22

Rules of Precedence 1-12, 1-13, 2-3, 2-19, 2-20, 2-21, 2-22,
  2-26, 2-34

S

Schema 10-2, 10-5, 10-7, 10-8, 10-18, 10-28, 10-40, B-2, C-3, C-5,
  C-7, C-8, C-10, C-13, E-5, E-6, I-2, I-3, I-4, I-6, I-8,
  I-15, I-29, I-33, I-34, I-36, I-40

SELECT Statement 1-1, 1-2, 1-3, 1-4, 1-5, 1-7, 1-10, 1-16,
  1-19, 1-21, 1-22, 1-25, 1-28, 1-29, 2-6, 2-9, 2-10, 2-12, 2-21,
  2-23, 2-28, 2-32, 2-35, 2-38, 2-39, 3-2, 3-12, 4-2, 4-44, 5-9,
  5-15, 5-16, 5-17, 5-18, 5-19, 5-20, 6-2, 6-7, 7-2, 7-5, 7-8,
  7-10, 7-11, 7-25, 7-26, 8-3, 8-6, 8-7, 8-12, 8-18, 8-19, 8-20,
  8-21, 8-22, 8-24, 8-25, 8-26, 8-27, 9-3, 9-13, 9-19, 9-22, 9-25,
  9-33, 9-39, 9-40, 9-41, 9-42, 9-43, 9-44, 9-47, 10-32, D-2, F-2,
  F-7, F-8, I-4, I-5

Selection 1-4, 2-4, E-12

Self join 6-35

Sequences 2-24, 10-8

Set operators 8-1, 8-2, 8-3, 8-4, 8-5, 8-6, 8-7, 8-12,
  8-18, 8-21, 8-24, 8-27, 8-31, I-5

SET VERIFY ON 2-36

Sorting 2-1, 2-3, 2-19, 2-22, 2-24, 2-25, 2-26, 2-34, 2-38,
  2-39, I-5