

Oracle Database: Introduction to SQL Accelerated - IBM Graduate Program

**Student Guide – Vol 2 (Oracle Database 10g: SQL
Fundamentals I)**

D59008GC10

Edition 1.0

June 2010

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the U.S. Government or anyone using the documentation on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

I Introduction

Lesson Objectives	I-2
Goals of the Course	I-3
Oracle10g	I-4
Oracle Database 10g	I-6
Oracle Application Server 10g	I-7
Oracle Enterprise Manager 10g Grid Control	I-8
Relational and Object Relational Database Management Systems	I-9
Oracle Internet Platform	I-10
System Development Life Cycle	I-11
Data Storage on Different Media	I-13
Relational Database Concept	I-14
Definition of a Relational Database	I-15
Data Models	I-16
Entity Relationship Model	I-17
Entity Relationship Modeling Conventions	I-19
Relating Multiple Tables	I-21
Relational Database Terminology	I-23
Relational Database Properties	I-25
Communicating with an RDBMS Using SQL	I-26
Oracle's Relational Database Management System	I-27
SQL Statements	I-28
Tables Used in the Course	I-29
Summary	I-30

1 Retrieving Data Using the SQL `SELECT` Statement

Objectives	1-2
Capabilities of SQL <code>SELECT</code> Statements	1-3
Basic <code>SELECT</code> Statement	1-4
Selecting All Columns	1-5
Selecting Specific Columns	1-6
Writing SQL Statements	1-7
Column Heading Defaults	1-8

Arithmetic Expressions	1-9
Using Arithmetic Operators	1-10
Operator Precedence	1-11
Defining a Null Value	1-12
Null Values in Arithmetic Expressions	1-13
Defining a Column Alias	1-14
Using Column Aliases	1-15
Concatenation Operator	1-16
Literal Character Strings	1-17
Using Literal Character Strings	1-18
Alternative Quote (q) Operator	1-19
Duplicate Rows	1-20
SQL and iSQL*Plus Interaction	1-21
SQL Statements Versus iSQL*Plus Commands	1-22
Overview of iSQL*Plus	1-23
Logging In to iSQL*Plus	1-24
iSQL*Plus Environment	1-25
Displaying Table Structure	1-26
Interacting with Script Files	1-28
iSQL*Plus History Page	1-32
Setting iSQL*Plus Preferences	1-34
Setting the Output Location Preference	1-35
Summary	1-36
Practice 1: Overview	1-37

2 Restricting and Sorting Data

Objectives	2-2
Limiting Rows Using a Selection	2-3
Limiting the Rows That Are Selected	2-4
Using the WHERE Clause	2-5
Character Strings and Dates	2-6
Comparison Conditions	2-7
Using Comparison Conditions	2-8
Using the BETWEEN Condition	2-9
Using the IN Condition	2-10
Using the LIKE Condition	2-11
Using the NULL Conditions	2-13
Logical Conditions	2-14
Using the AND Operator	2-15

Using the <code>OR</code> Operator	2-16
Using the <code>NOT</code> Operator	2-17
Rules of Precedence	2-18
Using the <code>ORDER BY</code> Clause	2-20
Sorting	2-21
Substitution Variables	2-22
Using the <code>&</code> Substitution Variable	2-24
Character and Date Values with Substitution Variables	2-26
Specifying Column Names, Expressions, and Text	2-27
Using the <code>&&</code> Substitution Variable	2-28
Using the <code>iSQL*Plus DEFINE</code> Command	2-29
Using the <code>VERIFY</code> Command	2-30
Summary	2-31
Practice 2: Overview	2-32

3 Using Single-Row Functions to Customize Output

Objectives	3-2
SQL Functions	3-3
Two Types of SQL Functions	3-4
Single-Row Functions	3-5
Character Functions	3-7
Case-Manipulation Functions	3-9
Using Case-Manipulation Functions	3-10
Character-Manipulation Functions	3-11
Using the Character-Manipulation Functions	3-12
Number Functions	3-13
Using the <code>ROUND</code> Function	3-14
Using the <code>TRUNC</code> Function	3-15
Using the <code>MOD</code> Function	3-16
Working with Dates	3-17
Arithmetic with Dates	3-20
Using Arithmetic Operators with Dates	3-21
Date Functions	3-22
Using Date Functions	3-23
Practice 3: Overview of Part 1	3-25
Conversion Functions	3-26
Implicit Data Type Conversion	3-27
Explicit Data Type Conversion	3-29
Using the <code>TO_CHAR</code> Function with Dates	3-32
Elements of the Date Format Model	3-33

Using the TO_CHAR Function with Dates	3-37
Using the TO_CHAR Function with Numbers	3-38
Using the TO_NUMBER and TO_DATE Functions	3-41
RR Date Format	3-43
Example of RR Date Format	3-44
Nesting Functions	3-45
General Functions	3-47
NVL Function	3-48
Using the NVL Function	3-49
Using the NVL2 Function	3-50
Using the NULLIF Function	3-51
Using the COALESCE Function	3-52
Conditional Expressions	3-54
CASE Expression	3-55
Using the CASE Expression	3-56
DECODE Function	3-57
Using the DECODE Function	3-58
Summary	3-60
Practice 3: Overview of Part 2	3-61

4 Reporting Aggregated Data Using the Group Functions

Objectives	4-2
What Are Group Functions?	4-3
Types of Group Functions	4-4
Group Functions: Syntax	4-5
Using the AVG and SUM Functions	4-6
Using the MIN and MAX Functions	4-7
Using the COUNT Function	4-8
Using the DISTINCT Keyword	4-9
Group Functions and Null Values	4-10
Creating Groups of Data	4-11
Creating Groups of Data: GROUP BY Clause Syntax	4-12
Using the GROUP BY Clause	4-13
Grouping by More Than One Column	4-15
Using the GROUP BY Clause on Multiple Columns	4-16
Illegal Queries Using Group Functions	4-17
Restricting Group Results	4-19
Restricting Group Results with the HAVING Clause	4-20
Using the HAVING Clause	4-21

Nesting Group Functions 4-23
Summary 4-24
Practice 4: Overview 4-25

5 Displaying Data from Multiple Tables

Objectives 5-2
Obtaining Data from Multiple Tables 5-3
Types of Joins 5-4
Joining Tables Using SQL:1999 Syntax 5-5
Creating Natural Joins 5-6
Retrieving Records with Natural Joins 5-7
Creating Joins with the `USING` Clause 5-8
Joining Column Names 5-9
Retrieving Records with the `USING` Clause 5-10
Qualifying Ambiguous Column Names 5-11
Using Table Aliases 5-12
Creating Joins with the `ON` Clause 5-13
Retrieving Records with the `ON` Clause 5-14
Self-Joins Using the `ON` Clause 5-15
Applying Additional Conditions to a Join 5-17
Creating Three-Way Joins with the `ON` Clause 5-18
Nonequijoins 5-19
Retrieving Records with Nonequijoins 5-20
Outer Joins 5-21
INNER Versus OUTER Joins 5-22
LEFT OUTER JOIN 5-23
RIGHT OUTER JOIN 5-24
FULL OUTER JOIN 5-25
Cartesian Products 5-26
Generating a Cartesian Product 5-27
Creating Cross Joins 5-28
Summary 5-29
Practice 5: Overview 5-30

6 Using Subqueries to Solve Queries

Objectives 6-2
Using a Subquery to Solve a Problem 6-3
Subquery Syntax 6-4
Using a Subquery 6-5
Guidelines for Using Subqueries 6-6

Types of Subqueries	6-7
Single-Row Subqueries	6-8
Executing Single-Row Subqueries	6-9
Using Group Functions in a Subquery	6-10
The <code>HAVING</code> Clause with Subqueries	6-11
What Is Wrong with This Statement?	6-12
Will This Statement Return Rows?	6-13
Multiple-Row Subqueries	6-14
Using the <code>ANY</code> Operator in Multiple-Row Subqueries	6-15
Using the <code>ALL</code> Operator in Multiple-Row Subqueries	6-16
Null Values in a Subquery	6-17
Summary	6-19
Practice 6: Overview	6-20

7 Using the Set Operators

Objectives	7-2
Set Operators	7-3
Tables Used in This Lesson	7-4
<code>UNION</code> Operator	7-8
Using the <code>UNION</code> Operator	7-9
<code>UNION ALL</code> Operator	7-11
Using the <code>UNION ALL</code> Operator	7-12
<code>INTERSECT</code> Operator	7-13
Using the <code>INTERSECT</code> Operator	7-14
<code>MINUS</code> Operator	7-15
Set Operator Guidelines	7-17
The Oracle Server and Set Operators	7-18
Matching the <code>SELECT</code> Statements	7-19
Matching the <code>SELECT</code> Statement: Example	7-20
Controlling the Order of Rows	7-21
Summary	7-23
Practice 7: Overview	7-24

8 Manipulating Data

Objectives	8-2
Data Manipulation Language	8-3
Adding a New Row to a Table	8-4
<code>INSERT</code> Statement Syntax	8-5
Inserting New Rows	8-6
Inserting Rows with Null Values	8-7

Inserting Special Values	8-8
Inserting Specific Date Values	8-9
Creating a Script	8-10
Copying Rows from Another Table	8-11
Using a Subquery in an <code>INSERT</code> Statement	8-12
Changing Data in a Table	8-14
<code>UPDATE</code> Statement Syntax	8-15
Updating Rows in a Table	8-16
Updating Two Columns with a Subquery	8-17
Updating Rows Based on Another Table	8-18
Removing a Row from a Table	8-19
<code>DELETE</code> Statement	8-20
Deleting Rows from a Table	8-21
Deleting Rows Based on Another Table	8-22
<code>TRUNCATE</code> Statement	8-23
Database Transactions	8-24
Advantages of <code>COMMIT</code> and <code>ROLLBACK</code> Statements	8-26
Controlling Transactions	8-27
Rolling Back Changes to a Marker	8-28
Implicit Transaction Processing	8-29
State of the Data Before <code>COMMIT</code> or <code>ROLLBACK</code>	8-31
State of the Data After <code>COMMIT</code>	8-32
Committing Data	8-33
State of the Data After <code>ROLLBACK</code>	8-34
Statement-Level Rollback	8-36
Read Consistency	8-37
Implementation of Read Consistency	8-38
Summary	8-39
Practice 8: Overview	8-40

9 Using DDL Statements to Create and Manage Tables

Objectives	9-2
Database Objects	9-3
Naming Rules	9-4
<code>CREATE TABLE</code> Statement	9-5
Referencing Another User's Tables	9-6
<code>DEFAULT</code> Option	9-7
Creating Tables	9-8
Data Types	9-9
Datetime Data Types	9-11

Including Constraints 9-17
Constraint Guidelines 9-18
Defining Constraints 9-19
NOT NULL Constraint 9-21
UNIQUE Constraint 9-22
PRIMARY KEY Constraint 9-24
FOREIGN KEY Constraint 9-25
FOREIGN KEY Constraint: Keywords 9-27
CHECK Constraint 9-28
CREATE TABLE: Example 9-29
Violating Constraints 9-30
Creating a Table by Using a Subquery 9-32
ALTER TABLE Statement 9-34
Dropping a Table 9-35
Summary 9-36
Practice 9: Overview 9-37

10 Creating Other Schema Objects

Objectives 10-2
Database Objects 10-3
What Is a View? 10-4
Advantages of Views 10-5
Simple Views and Complex Views 10-6
Creating a View 10-7
Retrieving Data from a View 10-10
Modifying a View 10-11
Creating a Complex View 10-12
Rules for Performing DML Operations on a View 10-13
Using the WITH CHECK OPTION Clause 10-16
Denying DML Operations 10-17
Removing a View 10-19
Practice 10: Overview of Part 1 10-20
Sequences 10-21
CREATE SEQUENCE Statement: Syntax 10-23
Creating a Sequence 10-24
NEXTVAL and CURRVAL Pseudocolumns 10-25
Using a Sequence 10-27
Caching Sequence Values 10-28
Modifying a Sequence 10-29
Guidelines for Modifying a Sequence 10-30

Indexes	10-31
How Are Indexes Created?	10-33
Creating an Index	10-34
Index Creation Guidelines	10-35
Removing an Index	10-36
Synonyms	10-37
Creating and Removing Synonyms	10-39
Summary	10-40
Practice 10: Overview of Part 2	10-41

11 Managing Objects with Data Dictionary Views

Objectives	11-2
The Data Dictionary	11-3
Data Dictionary Structure	11-4
How to Use the Dictionary Views	11-6
USER_OBJECTS and ALL_OBJECTS Views	11-7
USER_OBJECTS View	11-8
Table Information	11-9
Column Information	11-10
Constraint Information	11-12
View Information	11-15
Sequence Information	11-16
Synonym Information	11-18
Adding Comments to a Table	11-19
Summary	11-20
Practice 11: Overview	11-21

A Practice Solutions

B Table Descriptions and Data

C Oracle Join Syntax

D Using SQL*Plus

E Using SQL Developer

Index

chetan ballur (chetanballur567@gmail.com) has a non-transferable
license to use this Student Guide.

A

Practice Solutions

chetan ballur (chetanballur567@gmail.com) has a non-transferable
license to use this Student Guide.

Practice 1: Solutions

Part 1

Test your knowledge:

1. Initiate an *iSQL*Plus* session using the user ID and password that are provided by the instructor.
2. *iSQL*Plus* commands access the database.
True/False

3. The following **SELECT** statement executes successfully:

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

True/False

4. The following **SELECT** statement executes successfully:

```
SELECT *
FROM job_grades;
```

True/False

5. There are four coding errors in this 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 in 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.

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

```
DESCRIBE departments

SELECT *
FROM departments;
```

Practice 1: Solutions (continued)

7. You need to determine the structure of the EMPLOYEES table.

```
DESCRIBE employees
```

The HR department wants a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first. Provide an alias STARTDATE for the HIRE_DATE column. Save your SQL statement to a file named lab_01_07.sql so that you can dispatch this file to the HR department.

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

8. Test your query in the file lab_01_07.sql to ensure that it runs correctly.

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

9. The HR department needs a query to display all unique job codes from the EMPLOYEES table.

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

Part 3

If you have time, complete the following exercises:

10. The HR department wants more descriptive column headings for its report on employees. Copy the statement from lab_01_07.sql to the iSQL*Plus Edit window. Name the column headings Emp #, Employee, Job, and Hire Date, respectively. Then run your query again.

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

11. 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 1: Solutions (continued)

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

12. To familiarize yourself with the data in the EMPLOYEES table, create a query to display all the data from the EMPLOYEES 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;
```


Practice 2: Solutions

The HR department needs your assistance with 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. Place your SQL statement in a text file named `lab_02_01.sql`. Run your query.

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

2. 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 departments 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 of \$5,000 to \$12,000. Place your SQL statement in a text file named `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 start date for the employees with the last names of Matos and Taylor. Order the query in ascending order by start 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 number 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;
```

Practice 2: Solutions (continued)

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. Resave lab_02_03.sql as lab_02_06.sql. 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);
```

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. Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

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

Practice 2: Solutions (continued)

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 employee 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 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 name 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 sales representative or 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%. Resave lab_02_06.sql as lab_02_15.sql. Rerun the statement in lab_02_15.sql.

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

Practice 3: Solutions

1. Write a query to display the current date. Label the column Date.

```
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. Place your SQL statement in a text 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 uppercase and all other letters 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 ;
```

Practice 3: Solutions (continued)

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 when prompted for a letter, then the output should show all employees whose last name starts with the letter *H*.

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

6. The HR department wants to find the length 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: Your results will differ.

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

7. 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;
```

If you have time, complete the following exercises:

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

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

Practice 3: Solutions (continued)

9. 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;
```

10. 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');
```

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

11. 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;
```

12. 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 3: Solutions (continued)

13. 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:

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

```
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;
```

14. Rewrite the statement in the preceding exercise 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;
```

Practice 4: Solutions

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. Place your SQL statement in a text file named lab_04_04.sql.

```
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_04_04.sql to display the minimum, maximum, sum, and average salary for each job type. Resave lab_04_04.sql as lab_04_05.sql. Run the statement in lab_04_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_04_06.sql.

```
SELECT job_id, COUNT(*)
FROM   employees
WHERE  job_id = '&job_title'
GROUP BY job_id;
```


Practice 4: Solutions (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 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 4: Solutions (continued)

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.

```
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;
```

Practice 5: Solutions

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 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 name and employee number along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Place your SQL statement in a text file named lab_05_04.sql.

```
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_05_04.sql to display all employees including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named lab_05_05.sql. Run the query in lab_05_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);
```

Practice 5: Solutions (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_05_06.sql`.

```
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_05_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;
```

Practice 6: Solutions

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*. Place your SQL statement in a text file named lab_06_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);
```

Modify the query so that the user is prompted for a location ID. Save this to a file named lab_06_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);
```

Practice 6: Solutions (continued)

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');
```

If you have time, complete the following exercise:

7. Modify the query in lab_06_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*. Resave lab_06_03.sql to lab_06_07.sql. Run the statement in lab_06_07.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);
```

Practice 7: Solutions

1. The HR department needs a list of department IDs for departments that do not contain the job ID ST_CLERK. Use 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 set operators to create this report.

```
SELECT country_id, country_name
FROM countries
MINUS
SELECT country_id, country_name
FROM countries
NATURAL JOIN locations
NATURAL JOIN departments;
```

3. Produce a list of jobs for departments 10, 50, and 20, in that order. Display job ID and department ID using set operators.

```
COLUMN dummy NOPRINT
SELECT job_id, department_id, 'x' dummy
FROM employees
WHERE department_id = 10
UNION
SELECT job_id, department_id, 'y' dummy
FROM employees
WHERE department_id = 50
UNION
SELECT job_id, department_id, 'z' dummy
FROM employees
WHERE department_id = 20
ORDER BY dummy;
COLUMN dummy PRINT
```

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 7: Solutions (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;
```


Practice 8: Solutions

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.

Insert data into the MY_EMPLOYEE table.

1. Run the statement in the lab_08_01.sql script to build the MY_EMPLOYEE table to be used for the lab.

```
CREATE TABLE my_employee
(id NUMBER(4) CONSTRAINT my_employee_id_nn NOT NULL,
last_name VARCHAR2(25),
first_name VARCHAR2(25),
userid VARCHAR2(8),
salary NUMBER(9,2));
```

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);
```

Practice 8: Solutions (continued)

4. Populate the MY_EMPLOYEE table with the second row of 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 addition to the table.

```
SELECT *
FROM my_employee;
```

6. Write an insert statement in a dynamic reusable script file named loademp.sql to load rows into the MY_EMPLOYEE table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the user ID. Save this script to a file named lab_08_06.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
```

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.

```
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
```

8. Confirm your additions to the table.

```
SELECT *
FROM my_employee;
```

9. Make the data additions permanent.

```
COMMIT;
```

Practice 8: Solutions (continued)

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 last_name, salary
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;
```

15. Commit all pending changes.

```
COMMIT;
```

Control data transaction to the MY_EMPLOYEE table.

16. Populate the table with the last row of 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.

```
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
```

Practice 8: Solutions (continued)

17. Confirm your addition to the table.

```
SELECT  *  
FROM    my_employee;
```

18. Mark an intermediate point in the processing of the transaction.

```
SAVEPOINT step_18;
```

19. Empty the entire 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_18;
```

22. Confirm that the new row is still intact.

```
SELECT  *  
FROM    my_employee;
```

23. Make the data addition permanent.

```
COMMIT;
```

Practice 9: Solutions

1. Create the DEPT table based on the following table instance chart. Place the syntax in a script called lab_09_01.sql, then execute the statement in the script to create the table. Confirm that the table is created.

```
CREATE TABLE dept
(id    NUMBER(7) CONSTRAINT department_id_pk PRIMARY KEY,
name  VARCHAR2(25));

DESCRIBE dept
```

2. Populate the DEPT table with data from the DEPARTMENTS table. Include only 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. Place the syntax in a script called lab_09_03.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

```
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)
);

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. Drop the EMP table.

```
DROP TABLE emp;
```

Practice 10: Solutions

Part 1

1. The staff in the HR department wants to hide some of the data in the EMPLOYEES table. They want a view called EMPLOYEES_VU based on the employee numbers, employee last names, and department numbers from the EMPLOYEES table. They want the heading for the employee name to 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 due to that fact that the view 'DEPT50' has been created with CHECK OPTION CONSTRAINT. This ensures the deptno column in the view is protected from being changed.

Practice 10: Solutions (continued)

Part 2

You cannot make modifications to the deptno column that will result in the row being removed from the view.

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 1000. 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_10_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;
```

Practice 11: Solutions

1. For a specified table, create a script that reports the column names, data types, and data types' lengths, as well as whether nulls are allowed. Prompt the user to enter the table name. Give appropriate aliases to the columns DATA_PRECISION and DATA_SCALE. Save this script in a file named lab_11_01.sql.

```
SELECT column_name, data_type, data_length,  
       data_precision PRECISION, data_scale SCALE, nullable  
FROM   user_tab_columns  
WHERE  table_name = UPPER('&tab_name');
```

2. Create a script that reports the column name, constraint name, constraint type, search condition, and status for a specified table. You must join the USER_CONSTRAINTS and USER_CONS_COLUMNS tables to obtain all of this information. Prompt the user to enter the table name. Save the script in a file named lab_11_02.sql.

```
SELECT ucc.column_name, uc.constraint_name, uc.constraint_type,  
       uc.search_condition, uc.status  
FROM   user_constraints uc JOIN user_cons_columns ucc  
ON     uc.table_name = ucc.table_name  
AND    uc.constraint_name = ucc.constraint_name  
AND    uc.table_name = UPPER('&tab_name');
```

3. Add a comment to the DEPARTMENTS table. Then query the USER_TAB_COMMENTS view to verify that the comment is present.

```
COMMENT ON TABLE departments IS  
    'Company department information including name, code, and location.';  
  
SELECT COMMENTS  
FROM   user_tab_comments  
WHERE  table_name = 'DEPARTMENTS';
```

4. Find the names of all synonyms that are in your schema.

```
SELECT *  
FROM   user_synonyms;
```


Practice 11: Solutions (continued)

5. You need to determine the names and definitions of all of the views in your schema. Create a report that retrieves view information (the view name and text) from the USER_VIEWS data dictionary view.

Note: Another view already exists. The EMP_DETAILS_VIEW was created as part of your schema. Also, if you completed practice 10, you will see the DEPT50 view.

Note: To see more contents of a LONG column, use the *iSQL*Plus* command SET LONG *n*, where *n* is the value of the number of characters of the LONG column that you want to see.

```
SET LONG 600
```

```
SELECT    view_name, text
FROM      user_views;
```

6. Find the names of your sequences. Write a query in a script to display the following information about your sequences: sequence name, maximum value, increment size, and last number. Name the script lab_11_06.sql. Run the statement in your script.

```
SELECT    sequence_name, max_value, increment_by, last_number
FROM      user_sequences;
```

Practice C: Solutions

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.

```
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.

```
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 employee number along with the employee's manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Place your SQL statement in a text file named lab_c_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_c_04.sql to display all employees including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named lab_c_05.sql. Run the query in lab_c_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 C: Solutions (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_c_06.sql`.

```
SELECT e.department_id department, e.last_name employee,  
       c.last_name colleague  
FROM   employees e, employees c  
WHERE  e.department_id = c.department_id  
AND    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, 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 employee 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 manager's names and hire dates. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively. Save the script to a file named `lab_c_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;
```

chetan ballur (chetanballur567@gmail.com) has a non-transferable
license to use this Student Guide.

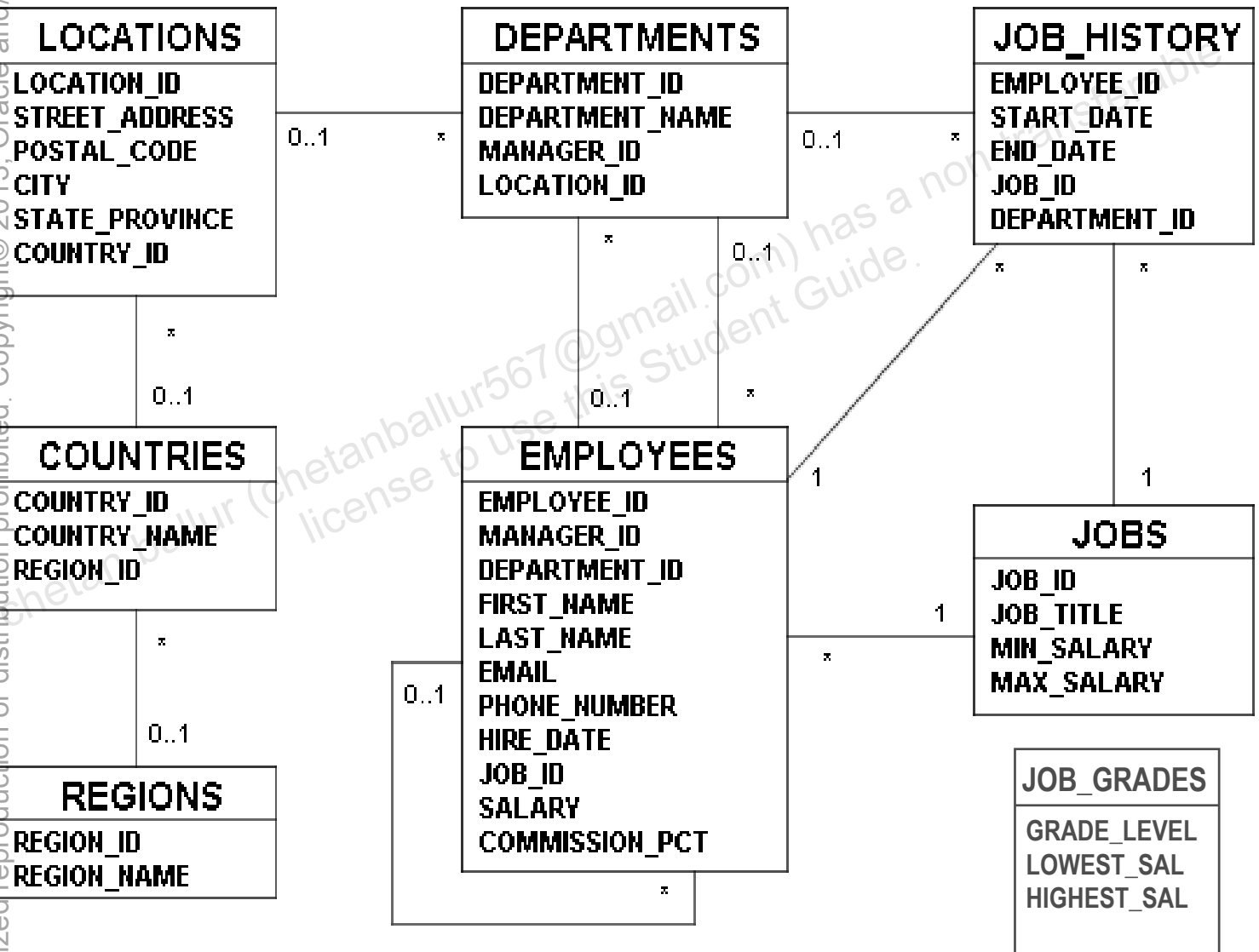
B

Table Descriptions and Data

chetan ballur (chetanballur567@gmail.com) has a non-transferable license to use this Student Guide.

Human Resources (HR) Data Set

Unauthorized reproduction or distribution prohibited. Copyright © 2013, Oracle and/or its affiliates. Chetan Ballur (chetanballur567@gmail.com) has a non-exclusive license to use this Student Guide.



Human Resources (HR) Data Set

The Human Resources (HR) schema is part of the Oracle Common Schema that can be installed in an Oracle database. The practices in this course use the data from the HR schema.

Table Descriptions

REGIONS contains rows representing a region (such as Americas, Asia, and so on).

COUNTRIES contains rows for countries, each of which are associated with a region.

LOCATIONS contains the addresses of specific offices, warehouses, and/or production sites of a company in a particular country.

DEPARTMENTS shows details of the departments in which employees work. Each department can have a relationship representing the department manager in the EMPLOYEES table.

EMPLOYEES contains details about each employee who works for a department. Some employees may not be assigned to any department.

JOBS contains the job types that can be held by each employee.

JOB_HISTORY contains the job history of the employees. If an employee changes departments within the job or changes jobs within the department, a new row is inserted in this table with the old job information of the employee.

JOB_GRADES identifies a salary range per job grade. The salary ranges do not overlap.

COUNTRIES Table

DESCRIBE countries

Name	Null?	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

SELECT * FROM countries;

CO	COUNTRY_NAME	REGION_ID
CA	Canada	2
DE	Germany	1
UK	United Kingdom	1
US	United States of America	2

DEPARTMENTS Table

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
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

8 rows selected.

EMPLOYEES Table

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_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE
100	Steven	King	SKING	515.123.4567	17-JUN-87
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97
143	Randall	Matos	RMATOS	650.121.2874	15-MAR-98
144	Peter	Vargas	PVARGAS	650.121.2004	09-JUL-98
149	Eleni	Zlotkey	EZLOTKEY	011.44.1344.429018	29-JAN-00
174	Ellen	Abel	EABEL	011.44.1644.429267	11-MAY-96
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	24-MAR-98
178	Kimberely	Grant	KGRANT	011.44.1644.429263	24-MAY-99
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94
206	William	Gietz	WGIEZT	515.123.8181	07-JUN-94

20 rows selected.

EMPLOYEES Table (continued)

JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
AD_PRES	24000			90
AD_VP	17000		100	90
AD_VP	17000		100	90
IT_PROG	9000		102	60
IT_PROG	6000		103	60
IT_PROG	4200		103	60
ST_MAN	5800		100	50
ST_CLERK	3500		124	50
ST_CLERK	3100		124	50
ST_CLERK	2600		124	50
ST_CLERK	2500		124	50
SA_MAN	10500	.2	100	80
SA_REP	11000	.3	149	80
SA_REP	8600	.2	149	80
SA_REP	7000	.15	149	
AD_ASST	4400		101	10
MK_MAN	13000		100	20
MK_REP	6000		201	20
AC_MGR	12000		101	110
AC_ACCOUNT	8300		205	110

20 rows selected.

JOBS Table

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
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000
AD_ASST	Administration Assistant	3000	6000
AC_MGR	Accounting Manager	8200	16000
AC_ACCOUNT	Public Accountant	4200	9000
SA_MAN	Sales Manager	10000	20000
SA_REP	Sales Representative	6000	12000
ST_MAN	Stock Manager	5500	8500
ST_CLERK	Stock Clerk	2000	5000
IT_PROG	Programmer	4000	10000
MK_MAN	Marketing Manager	9000	15000
MK_REP	Marketing Representative	4000	9000

12 rows selected.

JOB_GRADES Table

```
DESCRIBE job_grades
```

Name	Null?	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

```
SELECT * FROM job_grades;
```

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

6 rows selected.

JOB_HISTORY Table

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
102	13-JAN-93	24-JUL-98	IT_PROG	60
101	21-SEP-89	27-OCT-93	AC_ACCOUNT	110
101	28-OCT-93	15-MAR-97	AC_MGR	110
201	17-FEB-96	19-DEC-99	MK_REP	20
114	24-MAR-98	31-DEC-99	ST_CLERK	50
122	01-JAN-99	31-DEC-99	ST_CLERK	50
200	17-SEP-87	17-JUN-93	AD_ASST	90
176	24-MAR-98	31-DEC-98	SA_REP	80
176	01-JAN-99	31-DEC-99	SA_MAN	80
200	01-JUL-94	31-DEC-98	AC_ACCOUNT	90

10 rows selected.

LOCATIONS Table

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	CO
1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US
1500	2011 Interiors Blvd	99236	South San Francisco	California	US
1700	2004 Charade Rd	98199	Seattle	Washington	US
1800	460 Bloor St. W.	ON M5S 1X8	Toronto	Ontario	CA
2500	Magdalen Centre, The Oxford Science Park	OX9 9ZB	Oxford	Oxford	UK

REGIONS Table

DESCRIBE regions

Name	Null?	Type
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

SELECT * FROM regions;

REGION_ID	REGION_NAME
1	Europe
2	Americas
3	Asia
4	Middle East and Africa

Oracle Join Syntax

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

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

- Write `SELECT` statements to access data from more than one table using equijoins and nonequijoins
- Use outer joins to view data that generally does not meet a join condition
- Join a table to itself by using a self-join

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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 on joins is found in “SQL Queries and Subqueries: Joins” in *Oracle SQL Reference*.

Obtaining Data from Multiple Tables

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700



EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Data from Multiple Tables

Sometimes you need to use data from more than one table. In the slide example, 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.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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. 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. You should always include a valid join condition unless you have a specific need to combine all rows from all tables.

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
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

20 rows selected.

DEPARTMENTS (8 rows)

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

8 rows selected.

Cartesian product:
20 x 8 = 160 rows

EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
100	90	1700
101	90	1700
102	90	1700
103	60	1700
104	60	1700
107	60	1700
...		

160 rows selected.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Cartesian Products (continued)

A Cartesian product is generated if a join condition is omitted. The example in the slide displays employee last name and department name from the EMPLOYEES and DEPARTMENTS tables. 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	DEPT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
...	

160 rows selected.

Types of Joins

Oracle-proprietary joins (8i and earlier releases)

- Equijoin
- Nonequijoin
- Outer join
- Self-join

SQL:1999–compliant joins

- Cross join
- Natural join
- Using clause
- Full (or two-sided) outer join
- Arbitrary join condition for outer join

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Types of Joins

Before the release of Oracle9i Database, the join syntax was proprietary.

Note: The SQL:1999–compliant join syntax does not offer any performance benefits over the Oracle-proprietary join syntax that existed in prior releases. For detailed information about the SQL:1999–compliant join syntax, see Lesson 5.

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.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining Joins

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 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 = is the condition that joins (or relates) the tables together
table2.column2

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.

Equijoins

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
149	80
174	80
176	80

...

Foreign key

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT
60	IT
80	Sales
80	Sales
80	Sales

...

Primary key

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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. Frequently, 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 employees.employee_id, employees.last_name,  
       employees.department_id, departments.department_id,  
       departments.location_id  
FROM   employees, departments  
WHERE  employees.department_id = departments.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
144	Vargas	50	50	1500

...

19 rows selected.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Retrieving Records with Equijoins

In the slide example:

- 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:

EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID

Because the DEPARTMENT_ID column is common to both tables, it must be prefixed by the table name to avoid ambiguity.

Additional Search Conditions Using the AND Operator

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
Whalen	10
Hartstein	20
Fay	20
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Hunold	60
Ernst	60

...

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
50	Shipping
50	Shipping
50	Shipping
50	Shipping
60	IT
60	IT

...

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Additional Search Conditions

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. For example, to display employee Matos's department number and department name, you need an additional condition in the WHERE clause.

```
SELECT last_name, employees.department_id,
       department_name
FROM   employees, departments
WHERE  employees.department_id = departments.department_id
AND    last_name = 'Matos';
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Matos	50	Shipping

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Use column aliases to distinguish columns that have identical names but reside in different tables.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Qualifying Ambiguous Column Names

You need to qualify the names of the columns in the WHERE clause with the table name to avoid ambiguity. Without the table prefixes, the DEPARTMENT_ID column could be from either the DEPARTMENTS table or the EMPLOYEES table. 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 the table prefix improves performance, because you tell the Oracle server exactly where to find the columns.

The requirement to qualify ambiguous column names is also applicable to columns that may be ambiguous in other clauses, such as the SELECT clause or the ORDER BY clause.

Using Table Aliases

- Use table aliases to simplify queries.
- Use table prefixes to improve performance.

```
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;
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Table Aliases

Qualifying column names with table names can be very time consuming, particularly if table names are lengthy. 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, therefore using less memory.

Notice how table aliases are identified in the FROM clause in the example. The table name is specified in full, followed by a space and then the table alias. The EMPLOYEES table has been given an alias of e, and the DEPARTMENTS table has 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, then that table alias must be substituted for the table name throughout the SELECT statement.
- Table aliases should be meaningful.
- The table alias is valid for only the current SELECT statement.

Joining More Than Two Tables

EMPLOYEES

LAST_NAME	DEPARTMENT_ID
King	90
Kochhar	90
De Haan	90
Hunold	60
Ernst	60
Lorentz	60
Mourgos	50
Rajs	50
Davies	50
Matos	50
Vargas	50
Zlotkey	80
Abel	80
Taylor	80

20 rows selected.

DEPARTMENTS

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

LOCATIONS

LOCATION_ID	CITY
1400	Southlake
1500	South San Francisco
1700	Seattle
1800	Toronto
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.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Additional Search Conditions

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
Hunold	IT	Southlake
Ernst	IT	Southlake
Lorentz	IT	Southlake
Mourgos	Shipping	South San Francisco
Rajs	Shipping	South San Francisco
Davies	Shipping	South San Francisco

19 rows selected.

Nonequijoins

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600

...

20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

Salary in the **EMPLOYEES** table must be between lowest salary and highest salary in the **JOB_GRADES** table.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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. A relationship between the two tables is that the **SALARY** column in the **EMPLOYEES** table must be between the values in the **LOWEST_SALARY** and **HIGHEST_SALARY** columns of the **JOB_GRADES** table. The relationship is obtained using an operator other than equality (=).

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	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C

20 rows selected.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Nonequijoins (continued)

The slide example 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 \leq and \geq) can be used, but BETWEEN is the simplest. Remember to specify the low value first and the high value last when using BETWEEN.

Table aliases have been specified in the slide example for performance reasons, not because of possible ambiguity.

Outer Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

...
20 rows selected.

There are no employees
in department 190.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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, employee Grant does not appear because there is no department ID recorded for her in the EMPLOYEES table. Instead of seeing 20 employees in the result set, you see 19 records.

```
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
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping

...
19 rows selected.

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 © 2010, Oracle and/or its affiliates. All rights reserved.

Using Outer Joins to Return Records with No Direct Match

The missing rows can be returned if an *outer join* operator is used in the join condition. The operator is a plus sign enclosed in parentheses (+), and it is placed on the “side” of the join that is deficient in 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.)

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
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Matos	50	Shipping
...		
Gietz	110	Accounting
		Contracting

20 rows selected.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Outer Joins to Return Records with No Direct Match (continued)

The slide example 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 on only *one* side of the expression - the side that has information 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.

Self-Joins

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

...



**MANAGER_ID in the WORKER table is equal to
EMPLOYEE_ID in the MANAGER table.**

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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; this type of join is called a *self-join*.

For example, to find the name of Lorentz's manager, you need to do the following:

- 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 who has 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.

Joining a Table to Itself

```
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
Kochhar works for King
De Haan works for King
Mourgos works for King
Zlotkey works for King
Hartstein works for King
Whalen works for Kochhar
Higgins works for Kochhar
Hunold works for De Haan
Ernst works for Hunold
...
19 rows selected.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Joining a Table to Itself (continued)

The slide example joins the EMPLOYEES table to itself. To simulate two tables in the FROM clause, there are two aliases, namely w and m, 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 for versions 8i and earlier.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Summary

There are multiple ways to join tables.

Types of Joins

- Cartesian products
- 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 C: Overview

This practice covers writing queries to join tables using Oracle syntax.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice C: Overview

This practice is designed to give you a variety of exercises that join tables using the Oracle syntax that was covered in this appendix.

Practice C

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.

LOCATION_ID	STREET_ADDRESS	CITY	STATE_PROVINCE	COUNTRY_NAME
1400	2014 Jabberwocky Rd	Southlake	Texas	United States of America
1500	2011 Interiors Blvd	South San Francisco	California	United States of America
1700	2004 Charade Rd	Seattle	Washington	United States of America
1800	460 Bloor St. W.	Toronto	Ontario	Canada
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.

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
Fay	20	Marketing
Mourgos	50	Shipping
Rajs	50	Shipping
Davies	50	Shipping
Vargas	50	Shipping
■ ■ ■		
De Haan	90	Executive
Higgins	110	Accounting
Gietz	110	Accounting

19 rows selected.

Practice C (continued)

- 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
Hartstein	MK_MAN	20	Marketing
Fay	MK_REP	20	Marketing

- Create a report to display the employee last name and employee number along with the employee's manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Place your SQL statement in a text file named lab_c_04.sql.

Employee	EMP#	Manager	Mgr#
Kochhar	101	King	100
De Haan	102	King	100
Mourgos	124	King	100
Zlotkey	149	King	100
Hartstein	201	King	100
Whalen	200	Kochhar	101
Higgins	205	Kochhar	101
Hunold	103	De Haan	102
Ernst	104	Hunold	103
Lorentz	107	Hunold	103
Rajs	141	Mourgos	124
Davies	142	Mourgos	124
Matos	143	Mourgos	124
Vargas	144	Mourgos	124
Employee	EMP#	Manager	Mgr#
Abel	174	Zlotkey	149
Taylor	176	Zlotkey	149
Grant	178	Zlotkey	149
Fay	202	Hartstein	201
Gietz	206	Higgins	205

19 rows selected.

Practice C (continued)

- Modify `lab_c_04.sql` to display all employees including King, who has no manager. Order the results by the employee number. Place your SQL statement in a text file named `lab_c_05.sql`. Run the query in `lab_c_05.sql`.

Employee	EMP#	Manager	Mgr#
King	100		
Kochhar	101	King	100
De Haan	102	King	100
Hunold	103	De Haan	102
Ernst	104	Hunold	103
Lorentz	107	Hunold	103
Mourgos	124	King	100

■ ■ ■

20 rows selected.

- 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_c_06.sql`.

DEPARTMENT	EMPLOYEE	COLLEAGUE
20	Fay	Hartstein
20	Hartstein	Fay
50	Davies	Matos
50	Davies	Mourgos
50	Davies	Rajs
50	Davies	Vargas
50	Matos	Davies
50	Matos	Mourgos
50	Matos	Rajs
50	Matos	Vargas
50	Mourgos	Davies
50	Mourgos	Matos
50	Mourgos	Rajs
50	Mourgos	Vargas

■ ■ ■

42 rows selected.

Practice C (continued)

- 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. Second, create a query that displays the last name, job, department name, salary, and grade for all employees.

Name	Null?	Type
GRADE_LEVEL		VARCHAR2(3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

LAST_NAME	JOB_ID	DEPARTMENT_NAME	SALARY	GRA
Matos	ST_CLERK	Shipping	2600	A
Vargas	ST_CLERK	Shipping	2500	A
Lorentz	IT_PROG	IT	4200	B
Mourgos	ST_MAN	Shipping	5800	B
Rajs	ST_CLERK	Shipping	3500	B
Davies	ST_CLERK	Shipping	3100	B
Whalen	AD_ASST	Administration	4400	B

■ ■ ■

19 rows selected.

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

- 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 employee Davies.

LAST_NAME	HIRE_DATE
Lorentz	07-FEB-99
Mourgos	16-NOV-99
Matos	15-MAR-98
Vargas	09-JUL-98
Zlotkey	29-JAN-00
Taylor	24-MAR-98
Grant	24-MAY-99
Fay	17-AUG-97

8 rows selected.

Practice C (continued)

9. The HR department needs to find the name and hire date for all employees who were hired before their managers, along with their manager's name and hire date. Label the columns Employee, Emp Hired, Manager, and Mgr Hired, respectively. Save the script to a file named `lab_c_09.sql`.

LAST_NAME	HIRE_DATE	LAST_NAME	HIRE_DATE
Whalen	17-SEP-87	Kochhar	21-SEP-89
Hunold	03-JAN-90	De Haan	13-JAN-93
Rajs	17-OCT-95	Mourgos	16-NOV-99
Davies	29-JAN-97	Mourgos	16-NOV-99
Matos	15-MAR-98	Mourgos	16-NOV-99
Vargas	09-JUL-98	Mourgos	16-NOV-99
Abel	11-MAY-96	Zlotkey	29-JAN-00
Taylor	24-MAR-98	Zlotkey	29-JAN-00
Grant	24-MAY-99	Zlotkey	29-JAN-00

9 rows selected.

chetan ballur (chetanballur567@gmail.com) has a non-transferable
license to use this Student Guide.

D

Using SQL*Plus

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

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

- Log in to SQL*Plus
- Edit SQL commands
- Format output using SQL*Plus commands
- Interact with script files

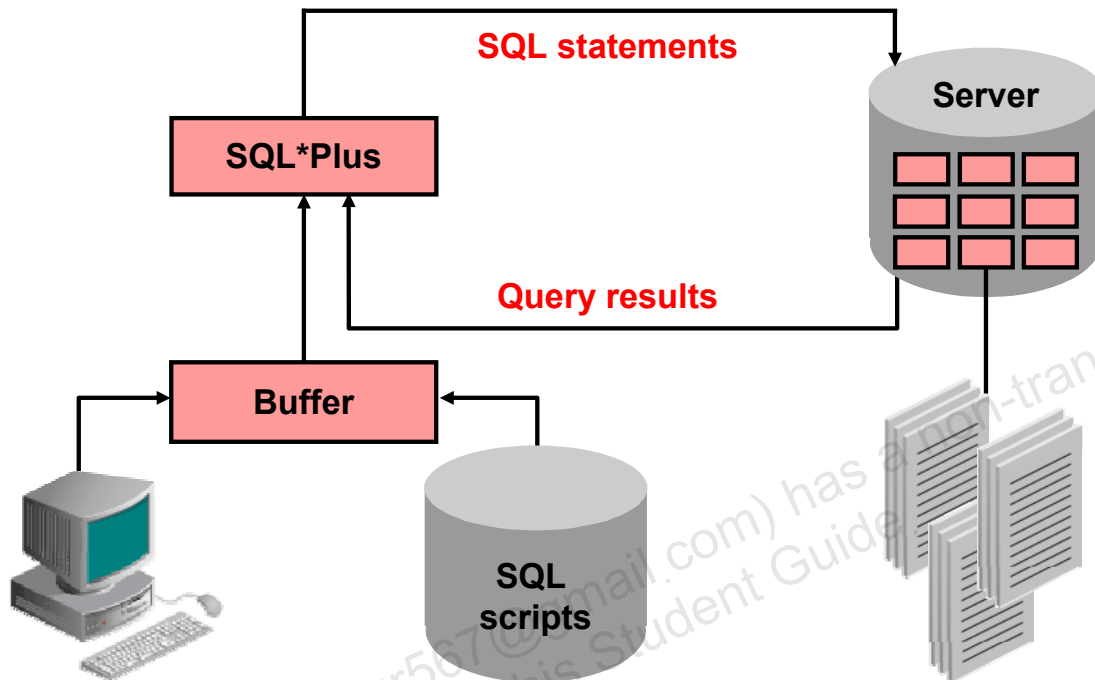
ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

SQL and SQL*Plus

SQL is a command language for communication with the Oracle9i 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 Oracle9i 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

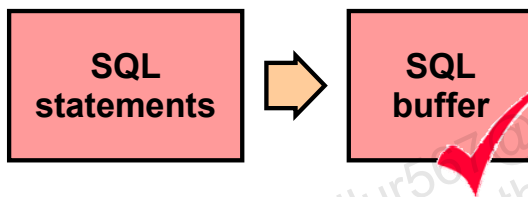
Features of SQL*Plus

- 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

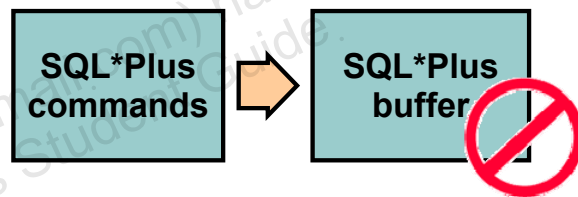
SQL

- A language
- ANSI-standard
- Keywords cannot be abbreviated
- Statements manipulate data and table definitions in the database



SQL*Plus

- An environment
- Oracle-proprietary
- Keywords can be abbreviated
- Commands do not allow manipulation of values in the database



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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 file to buffer to edit

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

SQL*Plus

SQL*Plus is an environment in which you can do the following:

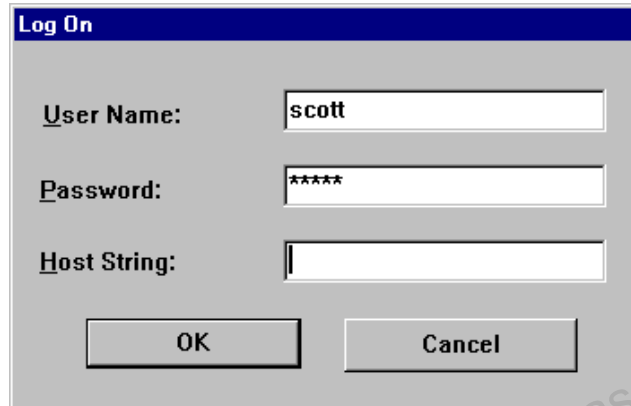
- Execute SQL statements to retrieve, modify, add, and remove data from the database
- Format, perform calculations on, store, and print query results in the form of reports
- Create script files to store SQL statements for repeated use in the future

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

- From a Windows environment:



- From a command line:

```
sqlplus [username[/password  
[@database]]]
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Logging In to SQL*Plus

How you invoke SQL*Plus depends on which type of operating system or Windows environment you are running.

To log in from a Windows environment:

1. Select Start > Programs > Oracle > Application Development > SQL*Plus.
2. Enter the username, password, and database name.

To log in from a command-line environment:

1. Log on to your machine.
2. Enter the SQL*Plus command shown in the slide.

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.

After you log in to SQL*Plus, you see the following message (if you are using SQL*Plus version 10.2.0.1.0):

```
SQL*Plus: Release 10.2.0.1.0 - Production on Fri May 19 12:12:26 2006  
Copyright (c) 1982, 2005, Oracle. All rights reserved.
```

Displaying Table Structure

Use the SQL*Plus DESCRIBE command to display the structure of a table:

```
DESC[RIBE] tablename
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Displaying 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 JOB_GRADES table, use this command:

```
SQL> DESCRIBE job_grades
```

Name	Null?	Type
-----	-----	-----
GRADE_LEVEL		VARCHAR2 (3)
LOWEST_SAL		NUMBER
HIGHEST_SAL		NUMBER

Displaying Table Structure

```
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)

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Displaying 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

The following table describes the data types:

Data Type	Description
NUMBER (<i>p</i> , <i>s</i>)	Number value that has a maximum number of digits <i>p</i> , which is the number of digits to the right of the decimal point <i>s</i>
VARCHAR2 (<i>s</i>)	Variable-length character value of maximum size <i>s</i>
DATE	Date and time value between January 1, 4712 B.C., and December 31, 9999 A.D.
CHAR (<i>s</i>)	Fixed-length character value of size <i>s</i>

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

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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] <i>text</i>	Adds <i>text</i> to the end of the current line
C[HANGE] / <i>old</i> / <i>new</i>	Changes <i>old</i> text to <i>new</i> in the current line
C[HANGE] / <i>text</i> /	Deletes <i>text</i> from the current line
CL[EAR] BUFF[ER]	Deletes all lines from the SQL buffer
DEL	Deletes current line
DEL <i>n</i>	Deletes line <i>n</i>
DEL <i>m n</i>	Deletes lines <i>m</i> to <i>n</i> 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 then 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

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

SQL*Plus Editing Commands (continued)

Command	Description
I [NPUT]	Inserts an indefinite number of lines
I [NPUT] <i>text</i>	Inserts a line consisting of <i>text</i>
L [IST]	Lists all lines in the SQL buffer
L [IST] <i>n</i>	Lists one line (specified by <i>n</i>)
L [IST] <i>m n</i>	Lists a range of lines (<i>m</i> to <i>n</i>) inclusive
R [UN]	Displays and runs the current SQL statement in the buffer
<i>n</i>	Specifies the line to make the current line
<i>n text</i>	Replaces line <i>n</i> with <i>text</i>
0 <i>text</i>	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

```
SQL> LIST
```

```
1  SELECT last_name  
2* FROM    employees
```

```
SQL> 1
```

```
1* SELECT last_name
```

```
SQL> A , job_id
```

```
1* SELECT last_name, job_id
```

```
SQL> L
```

```
1  SELECT last_name, job_id  
2* FROM    employees
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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

```
SQL> L
```

```
1* SELECT * from employees
```

```
SQL> c/employees/departments
```

```
1* SELECT * from departments
```

```
SQL> L
```

```
1* SELECT * from departments
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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`

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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
<code>SAV[E] filename [.ext]</code> <code>[REP[LACE] APP[END]]</code>	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.
<code>GET filename [.ext]</code>	Writes the contents of a previously saved file to the SQL buffer. The default extension for the file name is .sql.
<code>STA[RT] filename [.ext]</code>	Runs a previously saved command file
<code>@ filename</code>	Runs a previously saved command file (same as START)
<code>ED[IT]</code>	Invokes the editor and saves the buffer contents to a file named <code>afiedt.buf</code>
<code>ED[IT] [filename [.ext]]</code>	Invokes the editor to edit the contents of a saved file
<code>SPO[OL]</code> <code>[filename [.ext]] </code> <code>OFF OUT</code>	Stores query results in a file. OFF closes the spool file. OUT closes the spool file and sends the file results to the printer.
<code>EXIT</code>	Quits SQL*Plus

Using the SAVE and START Commands

```
SQL> L
  1  SELECT last_name, manager_id, department_id
  2*  FROM    employees
SQL> SAVE my_query
```

```
Created file my_query
```

```
SQL> START my_query
```

LAST_NAME	MANAGER_ID	DEPARTMENT_ID
King		90
Kochhar	100	90
...		
20 rows selected.		

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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.

EDIT

Use the EDIT command to edit an existing script. This opens an editor with the script file in it. When you have made the changes, quit the editor to return to the SQL*Plus command line.

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 output
- Interact with script files

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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.

chetan ballur (chetanballur567@gmail.com) has a non-transferable
license to use this Student Guide.

Using SQL Developer

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

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

- List the key features of Oracle SQL Developer
- Install Oracle SQL Developer
- Identify menu items of Oracle SQL Developer
- Create a database connection
- Manage database objects
- Use SQL Worksheet
- Execute SQL statements and SQL scripts
- Create and save reports

ORACLE

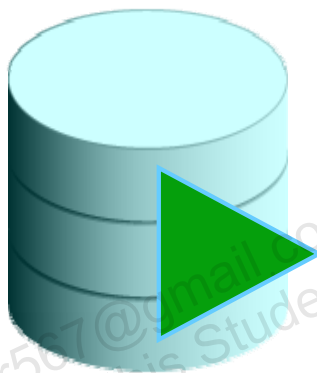
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

In this appendix, you are introduced to the graphical tool 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 using standard Oracle database authentication.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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, 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 using standard Oracle database authentication. When connected, you can perform operations on objects in the database.

Key Features

- Developed in Java
- Supports Windows, Linux, and Mac OS X platforms
- Default connectivity by using the JDBC Thin driver
- Does not require an installer
- Connects to any Oracle Database version 9.2.0.1 and later
- Bundled with JRE 1.5

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Key Features of SQL Developer

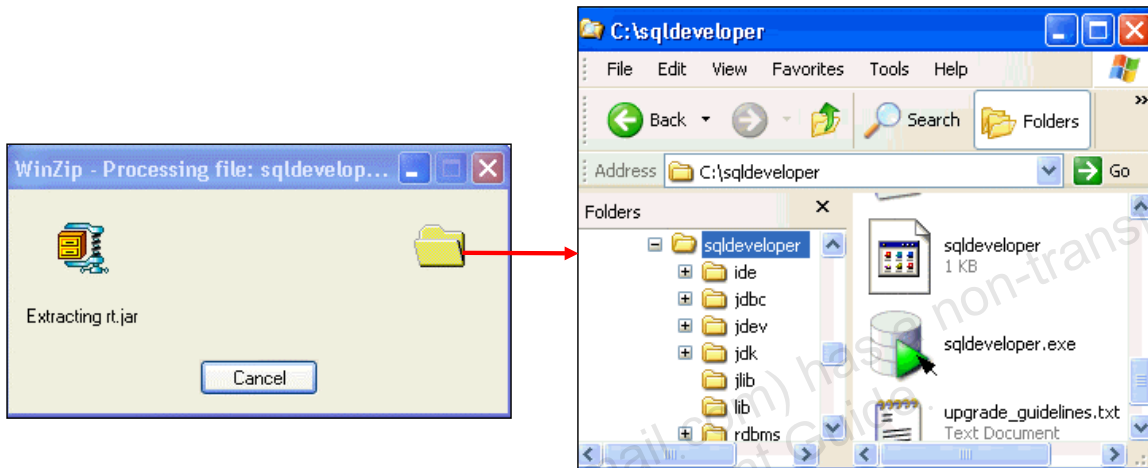
Oracle SQL Developer is developed in Java, leveraging the Oracle JDeveloper IDE. The tool runs on the Windows, Linux, and Mac OS X platforms. You can install SQL Developer on the Database Server and connect remotely from your desktop, thus avoiding client server network traffic.

Default connectivity to the database is through the JDBC Thin driver; so, no Oracle Home is required. SQL Developer does not require an installer and you need to just unzip the downloaded file.

With SQL Developer, users can connect to Oracle Database 9.2.0.1 and later versions, and all Oracle database editions including Express Edition. SQL Developer is bundled with JRE 1.5, with an additional `tools.jar` to support Windows clients. Non-Windows clients need only JDK 1.5.

Installing SQL Developer

Download the Oracle SQL Developer kit and unzip into any directory on your machine.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Installing SQL Developer

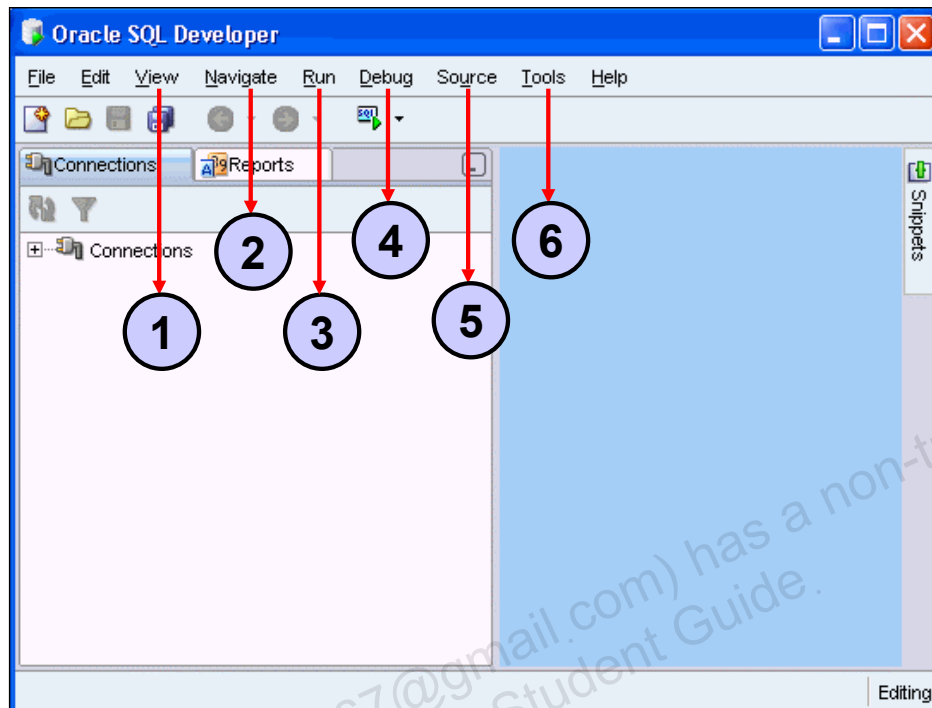
Oracle SQL Developer does not require an installer. To install SQL Developer, you need an unzip tool.

To install SQL Developer, perform the following steps:

1. Create a folder as <local drive>:\SQL Developer.
2. Download the SQL Developer kit from <http://www.oracle.com/technology/software/products/sql/index.html>.
3. Unzip the downloaded SQL Developer kit into the folder created in step 1.

To start SQL Developer, go to <local drive>:\SQL Developer, and double-click `sqldeveloper.exe`.

Menus for SQL Developer



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Menus for SQL Developer

SQL Developer has two main navigation tabs.

- **Connections Navigator:** By using this tab, you can browse database objects and users to which you have access.
- **Reporting Tab:** By using this tab, you can run predefined reports or create and add your own reports.

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.

The menus at the top contain standard entries, plus entries for features specific to SQL Developer.

1. **View:** Contains options that affect what is displayed in the SQL Developer interface
2. **Navigate:** Contains options for navigating to panes and for the execution of subprograms
3. **Run:** Contains the Run File and Execution Profile options that are relevant when a function or procedure is selected
4. **Debug:** Contains options that are relevant when a function or procedure is selected
5. **Source:** Contains options for use when editing functions and procedures
6. **Tools:** Invokes SQL Developer tools such as SQL*Plus, Preferences, and SQL Worksheet

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
 - For multiple schemas
- SQL Developer automatically imports any connections defined in the `tnsnames.ora` file on your system.
- You can export connections to an XML file.
- Each additional database connection created is listed in the connections navigator hierarchy.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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 systems, 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 later.

You can create additional connections to connect to the same database but as different users, or to connect to different databases. Each database connection is listed in the Connections navigator hierarchy.

Creating a Database Connection

The screenshot shows the 'New / Select Database Connection' dialog box. The 'Basic' tab is selected. The fields are filled as follows: Connection Name: DBConnection1, Username: hr, Password: **, Role: default, Host Name: localhost, Port: 1521, SID: orcl. The 'Save Password' checkbox is unchecked. The 'Test' button is highlighted with a mouse cursor.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Database Connection (continued)

To create a database connection, perform the following steps:

1. Double-click `<your_path>\sqldeveloper\sqldeveloper.exe`.
2. In the Connections tab, right-click **Connections** and select **New Database Connection**.
3. Enter the connection name, username, password, hostname, and SID for the database you want to connect to.
4. Click **Test** to make sure that the connection has been set correctly.
5. Click **Connect**.

In the Basic tabbed page, at the bottom, fill in the following options:

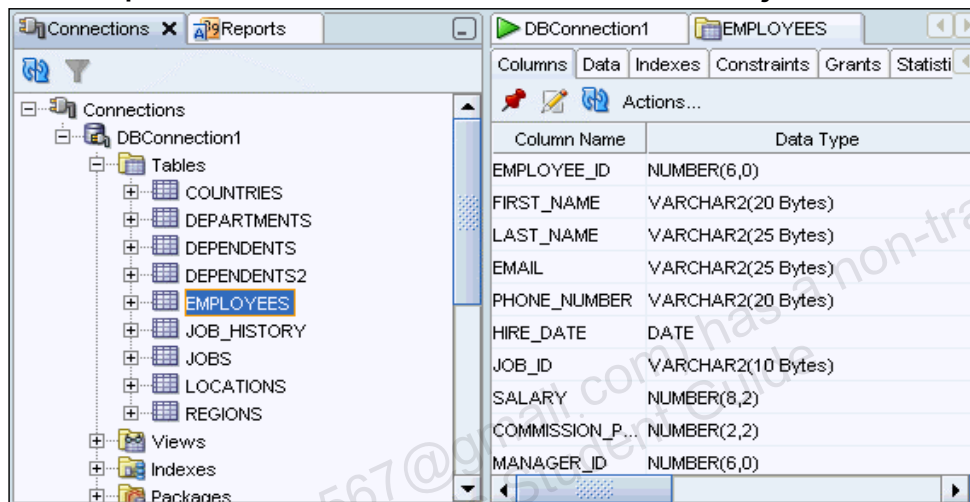
- **Hostname:** the Host system for the Oracle database
- **Port:** Listener port
- **SID:** Database name
- **Service Name:** Network service name for a remote database connection

If you select the Save Password check box, the password is saved to an XML file. So, when you close SQL Developer connection and open again, you will not be prompted for the password.

Browsing Database Objects

Use the Database Navigator to to:

- Browse through many objects in a database schema
- Do a quick review of the definitions of objects



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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, Types, and so on.

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 more are all displayed in an easy-to-read tabbed page.

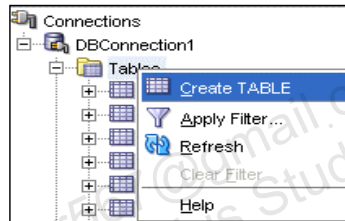
If you want to see the definition of the EMPLOYEES table as shown on the slide, perform the following steps:

1. Expand the connection node in the Connections Navigator.
2. Expand **Tables**.
3. Double-click **EMPLOYEES**.

Using the Data tab, you can enter new rows, update data, and commit these changes to the database.

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 or one of many context-sensitive menus.
- View the DDL for adjustments such as creating a new object or editing an existing schema object.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

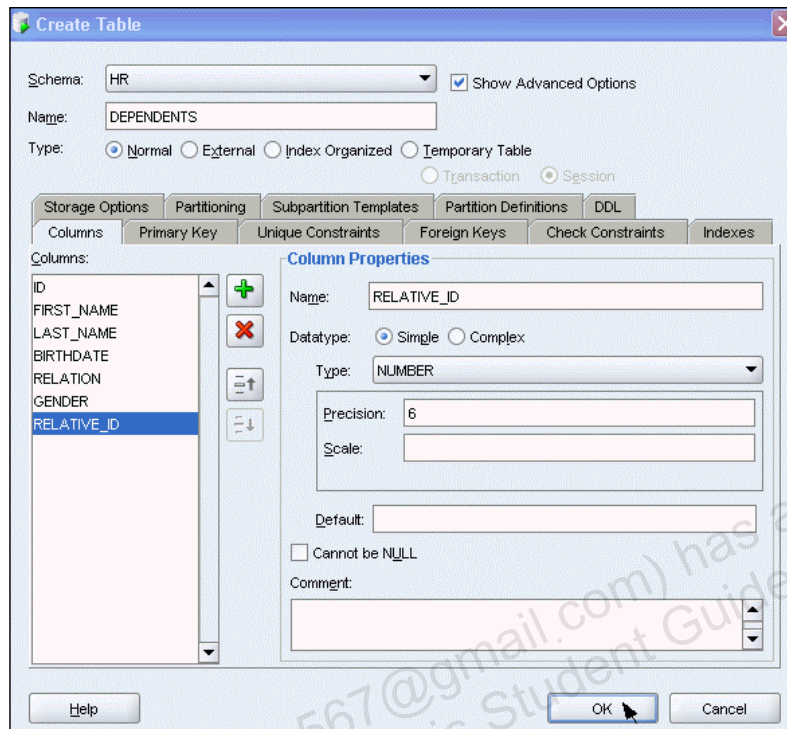
Creating a Schema Object

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. Once created, you can edit the objects using an edit dialog or one of 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 creating a table using the context menu. To open a dialog box for creating a new table, right-click **Tables** and select **Create TABLE**. The dialog boxes for creating and editing database objects have multiple tabs, each reflecting a logical grouping of properties for that type of object.

Creating a New Table: Example



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a New Table: Example

In the Create Table dialog box, if you do not select the **Show Advanced Options** check box, you can create a table quickly by specifying columns and some frequently used features.

If you select the **Show Advanced Options** check box, the Create Table dialog box changes to one with multiple tabs, in which you can specify an extended set of features while creating the table.

The example in the slide shows creating the DEPENDENTS table by selecting the **Show Advanced Options** check box.

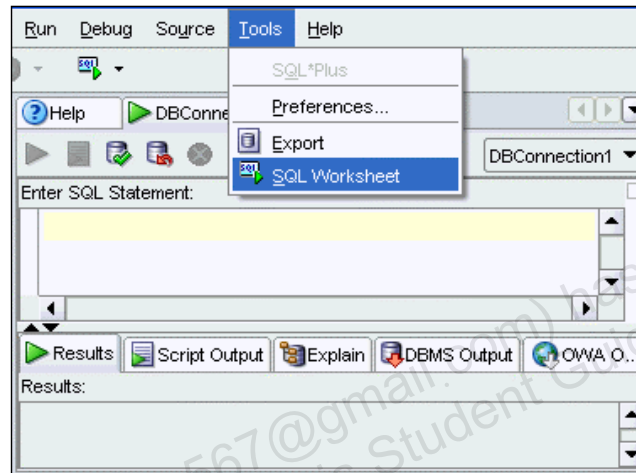
To create a new table, perform the following steps:

1. In the Connections Navigator, right-click **Tables**.
2. Select **Create TABLE**.
3. In the Create Table dialog box, select **Show Advanced Options**.
4. Specify column information.
5. Click **OK**.

Although it is not required, you should also specify a primary key using the Primary Key tab in the dialog box. Sometimes, you may want to edit the table that you have created. To edit a table, right-click the table in the Connections Navigator and select **Edit**.

Using SQL Worksheet

- Use 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.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using SQL Worksheet

When you connect to a database, a SQL Worksheet window for that connection is automatically opened. You can use 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.

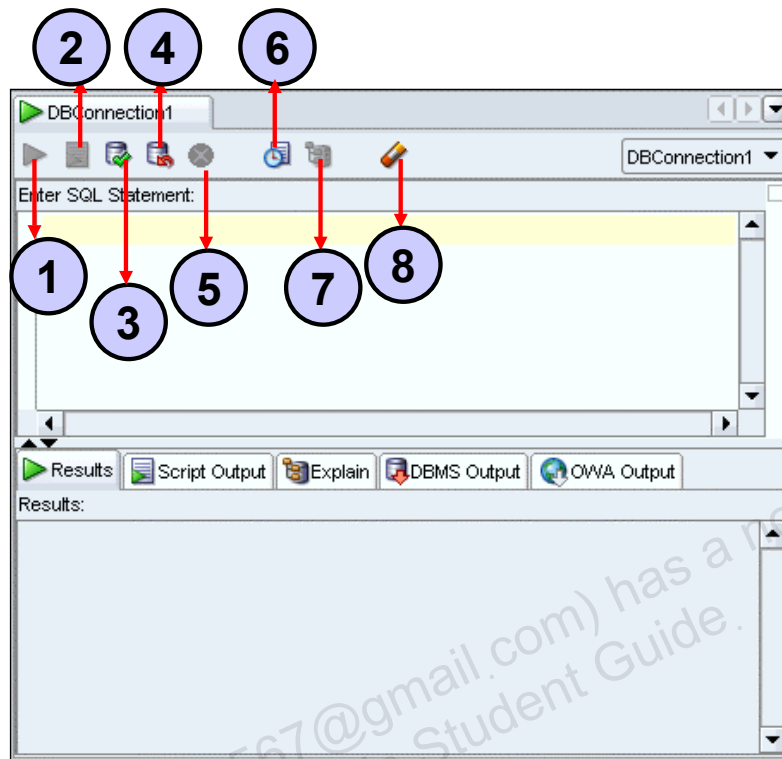
You can specify any actions that can be processed by the database connection associated with the worksheet, such as:

- 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 any of the following two options:

- Select **Tools > SQL Worksheet**.
- Click the **Open SQL Worksheet** icon.

Using SQL Worksheet



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

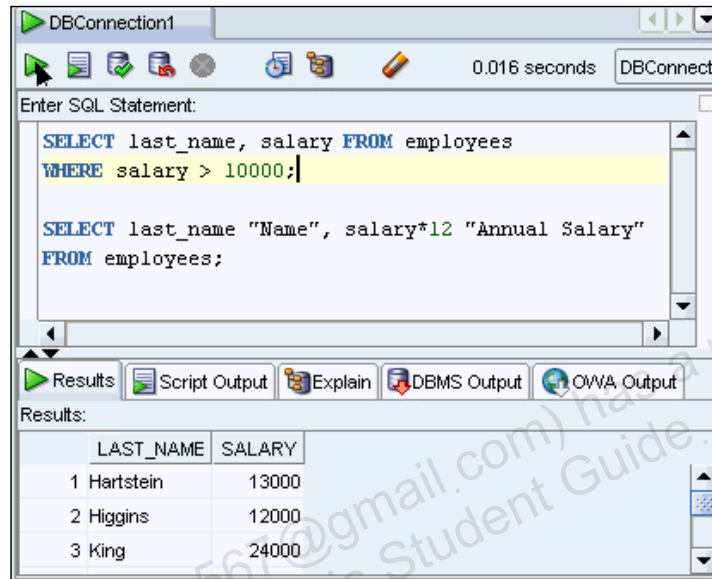
Using SQL Worksheet (continued)

You may want to use 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 at the cursor 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. **Clear:** Erases the statement or statements in the Enter SQL Statement box

Executing SQL Statements

Use the Enter SQL Statement box to enter single or multiple SQL statements.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Executing SQL Statements

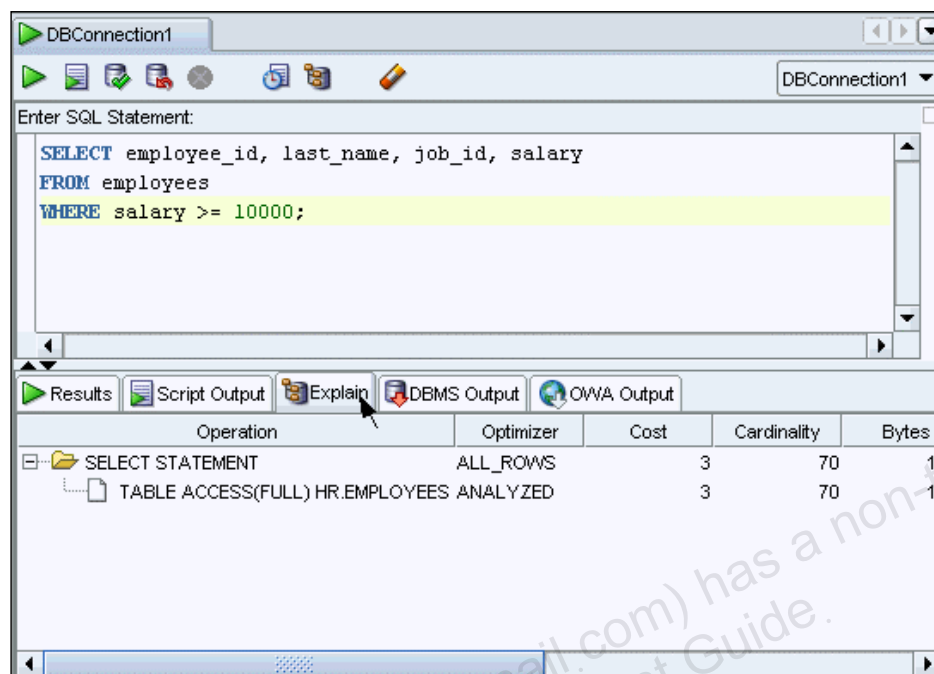
In SQL Worksheet, you can use the Enter SQL Statement box to enter a single or multiple SQL statements. For a single statement, the semicolon at the end is optional.

When you type in the statement, the SQL keywords are automatically highlighted. To execute a SQL statement, ensure that your cursor is within the statement and click the **Execute Statement** icon. Alternatively, you can press the **F9** key.

To execute multiple SQL statements and see the results, click the **Run Script** icon. Alternatively, you can press the **F5** key.

In the example in the slide, as there are multiple SQL statements, the first statement is terminated with a semicolon. The cursor is in the first statement and, therefore, when the statement is executed, results corresponding to the first statement are displayed in the Results box.

Viewing the Execution Plan



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Viewing the Execution Plan

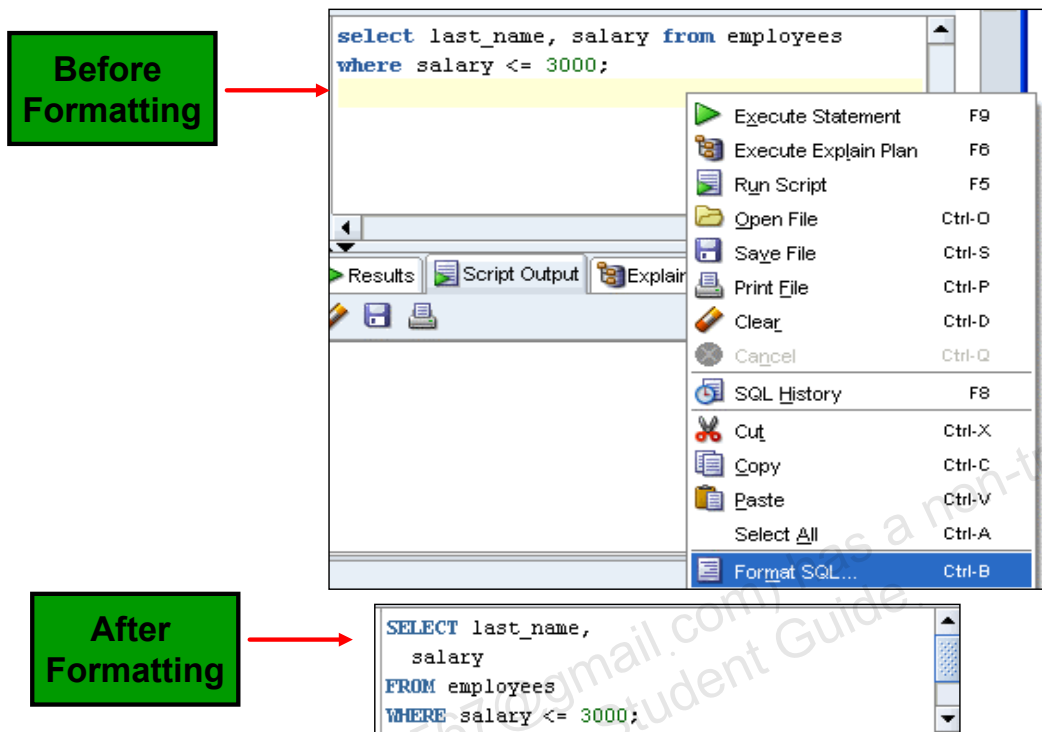
You can execute a SQL script and view the execution plan. To execute a SQL script file, perform the following steps:

1. Right-click in the Enter SQL Statement box, and select **Open File** from the drop-down menu.
2. In the Open dialog box, double-click the **.sql** file.
3. Click the **Run Script** icon.

When you double-click the **.sql** file, the SQL statements are loaded into the Enter SQL Statement box. You can execute the script or each line individually. The results are displayed in the Script Output area.

The example in the slide shows the execution plan. The Execute Explain Plan icon generates the execution plan. An execution plan is the sequence of operations that are performed to execute the statement. You can see the execution plan by clicking the **Explain** tab.

Formatting the SQL Code



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Formatting the SQL Code

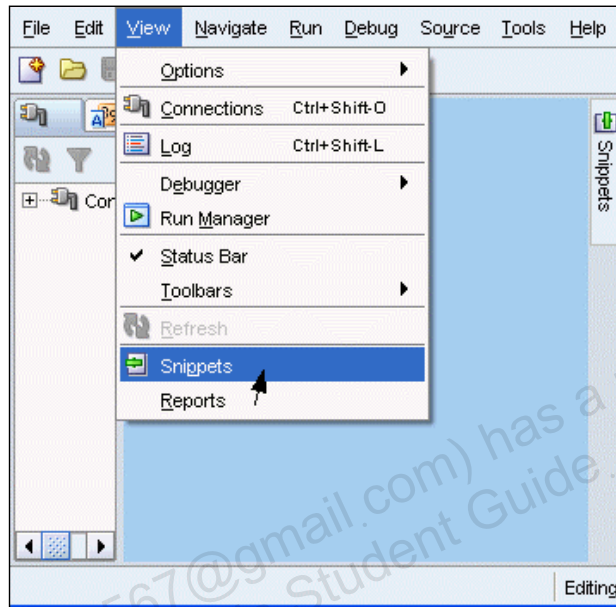
You may want to beautify the indentation, spacing, capitalization, and line separation of the SQL code. SQL Developer has the feature of formatting the SQL code.

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.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

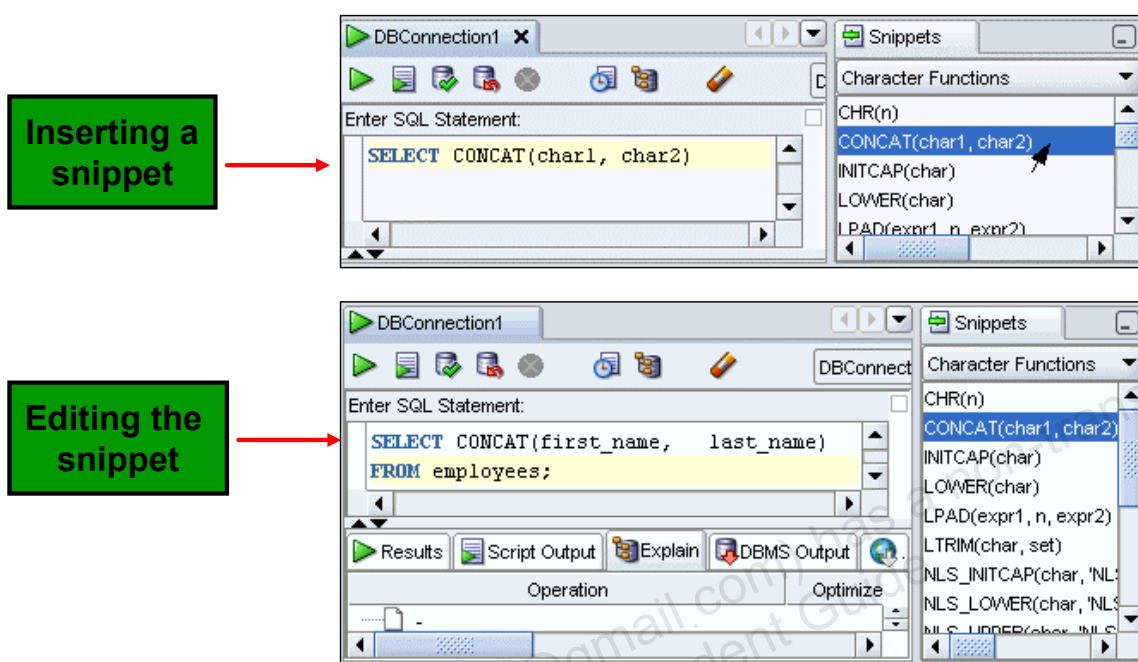
Using Snippets

You may want to use certain code fragments when you are using the SQL Worksheet or creating or editing 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 and drop snippets into the editor window.

To display Snippets, select **View > Snippets**.

The Snippets window is displayed on the right side. 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



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Snippets: Example

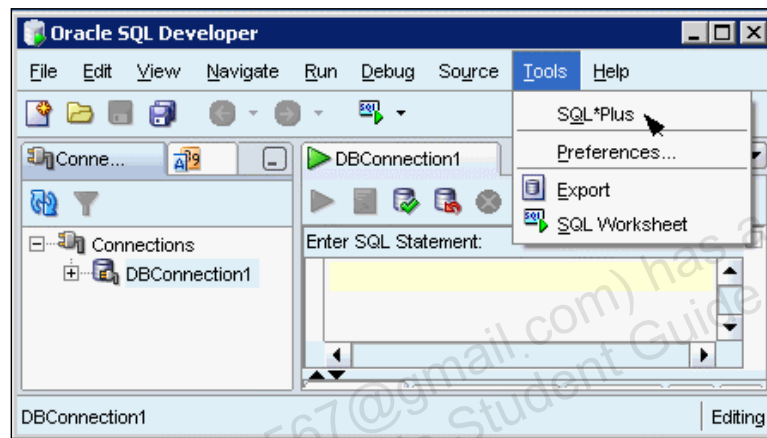
To insert a snippet into your code in SQL Worksheet or in a PL/SQL function or procedure, drag the snippet from the Snippets window and drop it into the desired place in your code. Then, you can edit the syntax so that the SQL function is valid in the current context. To see a brief description of a 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 such as in the following:

```
SELECT CONCAT(first_name, last_name)
FROM employees;
```

Using SQL*Plus

- SQL Worksheet does not support all SQL*Plus statements.
- You can invoke the SQL*Plus command-line interface from SQL Developer.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using SQL*Plus

SQL Worksheet supports some SQL*Plus statements. SQL*Plus statements must be interpreted by the SQL Worksheet before being passed to the database; any SQL*Plus statements that are not supported by the SQL Worksheet are ignored and not passed to the database. For example, some of the SQL*Plus statements that are not supported by SQL Worksheet are:

- append
- archive
- attribute
- break
- change
- clear

For the complete list of SQL*Plus statements that are supported and not supported by SQL Worksheet, refer to SQL Developer online Help.

To display the SQL*Plus command window, select **SQL*Plus** from the Tools menu.

To use this feature, the system on which you are using SQL Developer must have an Oracle Home directory or folder, with a SQL*Plus executable under that location. If the location of the SQL*Plus executable is not already stored in your SQL Developer preferences, you are asked to specify its location.

Unauthorized reproduction or distribution prohibited. Copyright© 2013, Oracle and/or its affiliates.

[illegible]

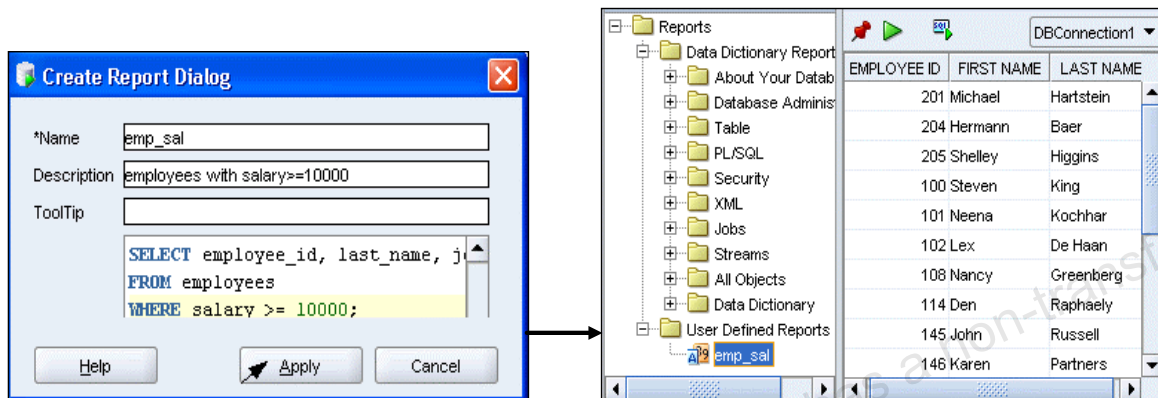
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

SQL Developer provides many reports about the database and its objects. These reports can be grouped into the following categories:

- To display reports, click the Reports tab on the left side of the window. Individual reports are displayed in tabbed panes on the right side of the window; and for each report, you can select (in a drop-down control) 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.

Creating a User-Defined Report

Create and save user-defined reports for repeated use.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a User-Defined Report

User-defined reports are any reports that are 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.

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

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

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.