A:
Practice Solutions

Solutions for Practice I: Introduction

The solutions for the “Introduction” lesson practices are discussed below.

Run through the Oracle SQL Developer Demo: Creating a database connection

1. Access the Demo: “Creating a database connection” at:

   http://st-curriculum.oracle.com/tutorial/SQLDeveloper/html/module2/mod02_cp_newdbconn.htm

Starting Oracle SQL Developer

2. Start up Oracle SQL Developer using the “sqldeveloper” desktop icon.

   1. Double-click the “sqldeveloper” desktop icon.

   Note: When you start SQL Developer for the first time, you need to provide the path to the `java.exe` file. This is already done for you as a part of the classroom setup. In any case, if you are prompted, enter the following path:

   `D:\app\Administrator\product\11.1.0\client_1\jdevstudio\jdk\bin`

Creating a New SQL Developer Database Connection

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

4. Create a database connection using the following information:

   a. Connection Name: `myconnection`

   b. Username: `oraxx`, where `xx` is the number of your PC (Ask your instructor to assign you one ora account out of the ora1- ora20 range of accounts.)

   c. Password: `oraxx`

   d. Hostname: Enter the host name of the machine where your database server is running.

   e. Port: 1521

   f. SID: ORCL

   g. Select the Save Password check box.

Testing and Connecting Using the SQL Developer Database Connection

5. Test the new connection.

    1. Click the Test button in the New/Select Database Connection window.

6. If the Status is Success, connect to the database using this new connection.

    1. If the status is Success, click the Connect button.

Browsing the Tables in the Connections Navigator

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

```
COUNTRIES
DEPARTMENTS
EMPLOYEES
JOB_GRADES
JOB_HISTORY
JOBS

LOCATIONS
REGIONS
```

    1. Expand the `myconnection` connection by clicking the plus sign next to it.

    2. Expand the Tables icon by clicking the plus sign next to it.

8. Browse the structure of the `EMPLOYEES` table.

    1. To display the structure of the `EMPLOYEES` table, click `EMPLOYEES`, and by default the Columns tab is the active tab showing the structure of the table.

9. View the data of the `DEPARTMENTS` table.

    1. Click the `DEPARTMENTS` table.

    2. Click the Data tab. The tables data is displayed.

Opening a SQL Worksheet

10. Open a new SQL Worksheet. Examine the shortcut icons available for the SQL Worksheet.

    1. To open a new SQL Worksheet, from the Tools menu, select SQL Worksheet.

    2. Alternatively, right-click `myconnection` and select Open SQL Worksheet.

Solutions for Practice I: Introduction (continued)

3. View the shortcut icons in the SQL Worksheet. Specifically look for the Execute Statement and Run Script icons.

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

Part 1

Test your knowledge:

1. The following `SELECT` statement executes successfully:

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

   True/False

2. The following `SELECT` statement executes successfully:

```
SELECT *
FROM   job_grades;
```

   True/False

3. There are four coding errors in 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 within double quotation marks.**
- **A comma is missing after the `LAST_NAME` column.**

Part 2

Note the following points before you begin with the practices:

- Save all your lab files at the following location: *`D:\labs\SQL1\labs`*.
- Enter your SQL statements in a SQL Worksheet. To save a script in SQL Developer, from the File menu, select Save As or right-click in the SQL Worksheet and select Save file to save your SQL statement as a `lab_<lessonno>_<stepno>.sql` script. To modify an existing script, use File > Open to open the script file, make changes and then make sure you use Save As to save it with a different filename.
- To run the query, click the Execute Statement icon (or press [F9]) in the SQL Worksheet. For DML and DDL statements, click the Run Script icon (or press [F5]).
- After you have executed a saved script, make sure that you do not enter your next query in the same worksheet. Open a new worksheet.

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

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

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

   a. To determine the `DEPARTMENTS` table structure:

```
DESCRIBE departments
```

   b. To view the data contained by the `DEPARTMENTS` table:

```
SELECT *
FROM   departments;
```

5. You need to determine the structure of the `EMPLOYEES` table.

```
DESCRIBE employees
```

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

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

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

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

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

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

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

Part 3

If you have the time, complete the following exercises:

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

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

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

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

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

10. To familiarize yourself with the data in the `EMPLOYEES` table, create a query to display all the data from the `EMPLOYEES` table. Separate each column output with a comma. Name the column as `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;
```

Solutions for Practice 2: Restricting and Sorting Data

The HR department needs your assistance in creating some queries.

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

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

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

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

3. The HR departments needs to find employees with high salary and low salary. 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. Save your SQL statement as `lab_02_03.sql`.

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

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

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

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

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

6. Modify `lab_02_03.sql` to list the last name and salary of employees who earn between $5,000 and $12,000, and are in department 20 or 50. Label the columns `Employee` and `Monthly Salary`, respectively. 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. Create a report to display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions. Use the column's numeric position in the `ORDER BY` clause.

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

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

    a. Enter 12000 when prompted for a value in a dialog box. Click OK.

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

Solutions for Practice 2: Restricting and Sorting Data (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 last_name

    manager_id = 201, sorted by salary

    manager_id = 124, sorted by employee_id

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

If you have the time, complete the following exercises:

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

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

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

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

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

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

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

15. Modify `lab_02_06.sql` to display the last name, salary, and commission for all employees whose commission amount is 20%. 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;
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

If you have the time, complete the following exercises:

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

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

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

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

9. Create a query to display the last name and the number of weeks employed for all employees in department 90. Label the number of weeks column as TENURE. Truncate the number of weeks value to 0 decimal places. Show the records in descending order of the employee's tenure.
   a. **Note:** The TENURE value will differ as it depends on the date when you run the query.

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

Solutions for Practice 4: Using Conversion Functions and Conditional Expressions

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

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

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

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

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

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

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

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

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

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

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

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

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

Solutions for Practice 5: Reporting Aggregated Data Using the Group Functions

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

1. Group functions work across many rows to produce one result per group.
   True/False

2. Group functions include nulls in calculations.
   True/**False**

3. The `WHERE` clause restricts rows before inclusion in a group calculation.
   True/False

The HR department needs the following reports:

4. Find the highest, lowest, sum, and average salary of all employees. Label the columns `Maximum`, `Minimum`, `Sum`, and `Average`, respectively. Round your results to the nearest whole number. Save your SQL statement as `lab_05_04.sql`. Run the query.

```
SELECT ROUND(MAX(salary),0) "Maximum",
       ROUND(MIN(salary),0) "Minimum",
       ROUND(SUM(salary),0) "Sum",
       ROUND(AVG(salary),0) "Average"
FROM   employees;
```

5. Modify the query in `lab_05_04.sql` to display the minimum, maximum, sum, and average salary for each job type. Resave `lab_05_04.sql` as `lab_05_05.sql`. Run the statement in `lab_05_05.sql`.

```
SELECT job_id, ROUND(MAX(salary),0) "Maximum",
               ROUND(MIN(salary),0) "Minimum",
               ROUND(SUM(salary),0) "Sum",
               ROUND(AVG(salary),0) "Average"
FROM   employees
GROUP BY job_id;
```

6. Write a query to display the number of people with the same job.

```
SELECT job_id, COUNT(*)
FROM   employees
GROUP BY job_id;
```

   Generalize the query so that the user in the HR department is prompted for a job title. Save the script to a file named `lab_05_06.sql`. Run the query. Enter `IT_PROG` when prompted and click OK.

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

Solutions for Practice 5: Reporting Aggregated Data Using the Group Functions (continued)

7. Determine the number of managers without listing them. Label the column as `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 as `DIFFERENCE`.

```
SELECT   MAX(salary) - MIN(salary) DIFFERENCE
FROM     employees;
```

If you have the time, complete the following exercises:

9. Create a report to display the manager number and the salary of the lowest-paid employee for that manager. Exclude anyone whose manager is not known. Exclude any groups where the minimum salary is $6,000 or less. Sort the output in descending order of salary.

```
SELECT   manager_id, MIN(salary)
FROM     employees
WHERE    manager_id IS NOT NULL
GROUP BY manager_id
HAVING   MIN(salary) > 6000
ORDER BY MIN(salary) DESC;
```

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

10. Create a query that will display the total number of employees and, of that total, the number of employees hired in 1995, 1996, 1997, and 1998. Create appropriate column headings.

```
SELECT  COUNT(*) total,
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1995,1,0))"1995",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1996,1,0))"1996",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1997,1,0))"1997",
        SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1998,1,0))"1998"
FROM    employees;
```

11. Create a matrix query to display the job, the salary for that job based on the department number, and the total salary for that job, for departments 20, 50, 80, and 90, giving each column an appropriate heading.

```
SELECT   job_id "Job",
         SUM(DECODE(department_id , 20, salary)) "Dept 20",
         SUM(DECODE(department_id , 50, salary)) "Dept 50",
         SUM(DECODE(department_id , 80, salary)) "Dept 80",
         SUM(DECODE(department_id , 90, salary)) "Dept 90",
         SUM(salary) "Total"
FROM     employees
GROUP BY job_id;
```

Solutions for Practice 6: Displaying Data from Multiple Tables

1. Write a query for the HR department to produce the addresses of all the departments. Use the `LOCATIONS` and `COUNTRIES` tables. Show the location ID, street address, city, state or province, and country in the output. Use a `NATURAL JOIN` to produce the results.

```
SELECT location_id, street_address, city, state_province, country_name
FROM   locations
NATURAL JOIN  countries;
```

2. The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all the employees.

```
SELECT last_name, department_id, department_name
FROM   employees
JOIN   departments
USING (department_id);
```

3. The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.

```
SELECT e.last_name, e.job_id, e.department_id, d.department_name
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   locations l
ON     (d.location_id = l.location_id)
WHERE LOWER(l.city) = 'toronto';
```

4. Create a report to display employees' last names and employee number along with their managers' last names and manager number. Label the columns `Employee`, `Emp#`, `Manager`, and `Mgr#`, respectively. Save your SQL statement as `lab_06_04.sql`. Run the query.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM   employees w join employees m
ON     (w.manager_id = m.employee_id);
```

5. Modify `lab_06_04.sql` to display all employees including King, who has no manager. Order the results by the employee number. Save your SQL statement as `lab_06_05.sql`. Run the query in `lab_06_05.sql`.

```
SELECT w.last_name "Employee", w.employee_id "EMP#",
       m.last_name "Manager", m.employee_id  "Mgr#"
FROM   employees w
LEFT   OUTER JOIN employees m
ON     (w.manager_id = m.employee_id)
ORDER BY 2;
```

6. Create a report for the HR department that displays employee last names, department numbers, and all the employees who work in the same department as a given employee. Give each column an appropriate label. Save the script to a file named `lab_06_06.sql`. Run the query.

```
SELECT e.department_id department, e.last_name employee,
       c.last_name colleague
FROM   employees e JOIN employees c
ON     (e.department_id = c.department_id)
WHERE  e.employee_id <> c.employee_id
ORDER BY e.department_id, e.last_name, c.last_name;
```

7. The HR department needs a report on job grades and salaries. To familiarize yourself with the `JOB_GRADES` table, first show the structure of the `JOB_GRADES` table. Then create a query that displays the name, job, department name, salary, and grade for all employees.

```
DESC JOB_GRADES

SELECT e.last_name, e.job_id, d.department_name,
       e.salary, j.grade_level
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
JOIN   job_grades j
ON    (e.salary BETWEEN j.lowest_sal AND j.highest_sal);
```

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

8. The HR department wants to determine the names of all employees who were hired after Davies. Create a query to display the name and hire date of any employee hired after employee Davies.

```
SELECT e.last_name, e.hire_date
FROM   employees e JOIN employees davies
ON     (davies.last_name = 'Davies')
WHERE  davies.hire_date < e.hire_date;
```

9. The HR department needs to find the names and hire dates for all employees who were hired before their managers, along with their managers' names and hire dates. Save the script to a file named `lab_06_09.sql`.

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM   employees w JOIN employees m
ON    (w.manager_id = m.employee_id)
WHERE    w.hire_date <  m.hire_date;
```

Solutions for Practice 7: Using Subqueries to Solve Queries

1. The HR department needs a query that prompts the user for an employee last name. The query then displays the last name and hire date of any employee in the same department as the employee whose name they supply (excluding that employee). For example, if the user enters `Zlotkey`, find all employees who work with Zlotkey (excluding Zlotkey).

```
UNDEFINE Enter_name

SELECT last_name, hire_date
FROM   employees
WHERE  department_id = (SELECT department_id
                        FROM   employees
                        WHERE  last_name = '&&Enter_name')
AND    last_name <> '&Enter_name';
```

2. Create a report that displays the employee number, last name, and salary of all employees who earn more than the average salary. Sort the results in order of ascending salary.

```
SELECT employee_id, last_name, salary
FROM   employees
WHERE  salary > (SELECT AVG(salary)
                 FROM   employees)
ORDER BY salary;
```

3. Write a query that displays the employee number and last name of all employees who work in a department with any employee whose last name contains a "u." Save your SQL statement as `lab_07_03.sql`. Run your query.

```
SELECT employee_id, last_name
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   employees
                         WHERE  last_name like '%u%');
```

4. The HR department needs a report that displays the last name, department number, and job ID of all employees whose department location ID is 1700.

```
SELECT last_name, department_id, job_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   departments
                         WHERE  location_id = 1700);
```

Modify the query so that the user is prompted for a location ID. Save this to a file named `lab_07_04.sql`.

```
SELECT last_name, department_id, job_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   departments
                         WHERE  location_id = &Enter_location);
```

5. Create a report for HR that displays the last name and salary of every employee who reports to King.

```
SELECT last_name, salary
FROM   employees
WHERE  manager_id = (SELECT employee_id
                     FROM   employees
                     WHERE  last_name = 'King');
```

6. Create a report for HR that displays the department number, last name, and job ID for every employee in the Executive department.

```
SELECT department_id, last_name, job_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM   departments
                         WHERE  department_name = 'Executive');
```

If you have the time, complete the following exercise:

7. Modify the query in `lab_07_03.sql` to display the employee number, last name, and salary of all employees who earn more than the average salary and who work in a department with any employee whose last name contains a "u." Resave `lab_07_03.sql` to `lab_07_07.sql`. Run the statement in `lab_07_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);
```

Solutions for Practice 8: Using the Set Operators

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

```
SELECT department_id
FROM   departments
MINUS
SELECT department_id
FROM   employees
WHERE  job_id = 'ST_CLERK';
```

2. The HR department needs a list of countries that have no departments located in them. Display the country ID and the name of the countries. Use the set operators to create this report.

```
SELECT country_id,country_name
FROM countries
MINUS
SELECT l.country_id,c.country_name
FROM locations l JOIN countries c
ON (l.country_id = c.country_id)
JOIN departments d
ON d.location_id=l.location_id;
```

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

```
SELECT distinct job_id, department_id
FROM employees
WHERE department_id = 10
UNION ALL
SELECT DISTINCT job_id, department_id
FROM employees
WHERE department_id = 50
UNION ALL
SELECT DISTINCT job_id, department_id
FROM employees
WHERE department_id = 20
```

Solutions for Practice 8: Using the Set Operators (continued)

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;
```

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;
```

Solutions for Practice 9: Manipulating Data

The HR department wants you to create SQL statements to insert, update, and delete employee data. As a prototype, use the MY_EMPLOYEE table before giving the statements to the HR department.

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

Insert data into the MY_EMPLOYEE table.

1. Run the statement in the lab_09_01.sql script to build the MY_EMPLOYEE table used in this practice.

    a. From File menu, select Open. In the Open dialog box, navigate to D:\labs\sql1\labs folder, double-click lab_09_01.sql.

    b. After the statement is opened in a SQL Worksheet, click the Run Script icon to run the script. You get a Create Table succeeeded message on the Script Output tabbed page.

2. Describe the structure of the MY_EMPLOYEE table to identify the column names.

```
DESCRIBE my_employee
```

3. Create an INSERT statement to add *the first row* of data to the MY_EMPLOYEE table from the following sample data. Do not list the columns in the INSERT clause.

ID	**LAST_NAME**	**FIRST_NAME**	**USERID**	**SALARY**
1	Patel	Ralph	rpatel	895
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audrey	aropebur	1550

```
INSERT INTO my_employee
  VALUES (1, 'Patel', 'Ralph', 'rpatel', 895);
```

Solutions for Practice 9: Manipulating Data (continued)

4. Populate the MY_EMPLOYEE table with the second row of the sample data from the preceding list. This time, list the columns explicitly in the INSERT clause.

```
INSERT INTO my_employee (id, last_name, first_name,
                         userid, salary)
  VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);
```

5. Confirm your additions to the table.

```
SELECT   *
FROM     my_employee;
```

6. Write an insert statement in a dynamic reusable script file to load the remaining rows into the MY_EMPLOYEE table. The script should prompt for all the columns (ID, LAST_NAME, FIRST_NAME, USERID and SALARY). Save this script to a file named lab_09_06.sql.

```
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
        '&p_userid', &p_salary);
```

7. Populate the table with the next two rows of sample data listed in step 3 by running the INSERT statement in the script that you created.

```
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
       '&p_userid', &p_salary);
```

8. Confirm your additions to the table.

```
SELECT   *
FROM my_employee;
```

9. Make the data additions permanent.

```
COMMIT;
```

Solutions for Practice 9: Manipulating Data (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  *
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 the sample data listed in step 3 by using the statements in the script that you created in step 6. Run the statements in the script.

```
INSERT INTO my_employee
VALUES (&p_id, '&p_last_name', '&p_first_name',
   '&p_userid', &p_salary);
```

Solutions for Practice 9: Manipulating Data (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_17;
```

19. Delete all the rows from the `MY_EMPLOYEE` table.

```
DELETE
FROM  my_employee;
```

20. Confirm that the table is empty.

```
SELECT *
FROM   my_employee;
```

21. Discard the most recent `DELETE` operation without discarding the earlier `INSERT` operation.

```
ROLLBACK TO step_17;
```

22. Confirm that the new row is still intact.

```
SELECT *
FROM   my_employee;
```

23. Make the data addition permanent.

```
COMMIT;
```

If you have time, complete the following exercise:

24. Modify the `lab_09_06.sql` script such that the `USERID` is generated automatically by concatenating the first letter of the first name and the first seven characters of the last name. The generated `USERID` must be in lowercase. Hence, the script should not prompt for the `USERID`. Save this script to a file named `lab_09_24.sql`.

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO my_employee
VALUES (&p_id, '&&p_last_name', '&&p_first_name',
   lower(substr('&p_first_name', 1, 1) ||
   substr('&p_last_name', 1, 7)), &p_salary);
SET VERIFY ON
SET ECHO ON
UNDEFINE p_first_name
UNDEFINE p_last_name
```

25. Run the script `lab_09_24.sql` to insert the following record:

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
6	Anthony	Mark	manthony	1230

26. Confirm that the new row was added with the correct `USERID`.

```
SELECT *
FROM my_employee
WHERE ID='6';
```

Solutions for Practice 10: Using DDL Statements to Create and Manage Tables

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

1. Create the `DEPT` table based on the following table instance chart. Save the statement in a script called `lab_10_01.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

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

   a. To confirm that the table was created and to view its structure, issue the following command:

```
DESCRIBE dept
```

2. Populate the `DEPT` table with data from the `DEPARTMENTS` table. Include only those columns that you need.

```
INSERT INTO dept
  SELECT  department_id, department_name
  FROM    departments;
```

3. Create the `EMP` table based on the following table instance chart. Save the statement in a script called `lab_10_03.sql`, and then execute the statement in the script to create the table. Confirm that the table is created.

```
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)
   );
```

   a. To confirm that the table was created and to view its structure:

```
DESCRIBE emp
```

4. Create the `EMPLOYEES2` table based on the structure of the `EMPLOYEES` table. Include only the `EMPLOYEE_ID`, `FIRST_NAME`, `LAST_NAME`, `SALARY`, and `DEPARTMENT_ID` columns. Name the columns in your new table `ID`, `FIRST_NAME`, `LAST_NAME`, `SALARY`, and `DEPT_ID`, respectively.

```
CREATE TABLE employees2 AS
  SELECT  employee_id id, first_name, last_name, salary,
          department_id dept_id
  FROM    employees;
```

Solutions for Practice 10: Using DDL Statements to Create and Manage Tables (continued)

5. Alter the EMPLOYEES2 table status to read-only.

```
ALTER TABLE employees2 READ ONLY
```

6. Try to insert the following row in the EMPLOYEES2 table.

   a. Note, you will get "Update operation not allowed on table" error message. Hence, you will not be allowed to insert any row into the table because it is assigned a read only status.

```
INSERT INTO employees2
VALUES (34, 'Grant','Marcie',5678,10)
```

7. Revert the EMPLOYEES2 table to the read/write status. Now try to insert the same row again.

   a. Now, because the table is assigned a READ WRITE status, you will be allowed to insert a row into the table.

```
ALTER TABLE employees2 READ WRITE

INSERT INTO employees2
VALUES (34, 'Grant','Marcie',5678,10)
```

8. Drop the EMPLOYEES2 table.

   a. **Note:** You can even drop a table that is in the READ ONLY mode. To test this, alter the table again to READ ONLY status and then issue the DROP TABLE command. The table EMPLOYEES2 will be dropped.

```
DROP TABLE employees2;
```

Solutions for Practice 11: Creating Other Schema Objects
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 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 because the DEPT50 view has been created with the WITH CHECK OPTION constraint. This ensures that the DEPTNO column in the view is protected from being changed.

Solutions for Practice 11: Creating Other Schema Objects (continued)
Part 2

7. You need a sequence that can be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1,000. Have your sequence increment by 10. Name the sequence `DEPT_ID_SEQ`.

```
CREATE SEQUENCE dept_id_seq
  START WITH 200
  INCREMENT BY 10
  MAXVALUE 1000;
```

8. To test your sequence, write a script to insert two rows in the `DEPT` table. Name your script `lab_11_08.sql`. Be sure to use the sequence that you created for the ID column. Add two departments: Education and Administration. Confirm your additions. Run the commands in your script.

```
INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Education');

INSERT INTO dept
VALUES (dept_id_seq.nextval, 'Administration');
```

9. Create a nonunique index on the `NAME` column in the `DEPT` table.

```
CREATE INDEX dept_name_idx ON dept (name);
```

10. Create a synonym for your `EMPLOYEES` table. Call it `EMP`.

```
CREATE SYNONYM emp FOR EMPLOYEES;
```