Solutions To Workshop Exercises

Chapter 1: SQL and Data

In this chapter, you learned about data, how data is organized in tables, and how the relationships among the tables are depicted in a schema diagram. Based on your newly acquired knowledge, design a schema diagram based on the fictional ACME Construction Company. Draw on your own work experience to design the following components.

1. Draw boxes for these three tables: EMPLOYEE, POSITION, and DEPARTMENT.

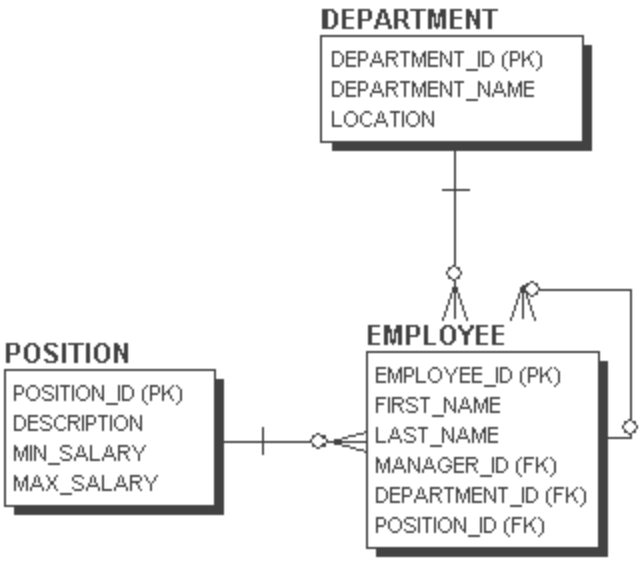
Solution: See the solution for Exercise 3.

2. Create at least three columns for each of the tables and designate a primary key for each table.

Solution: See the solution for Exercise 3.

3. Create relationships among the tables that make sense to you. At least one table should have a self-referencing relationship. Hint: Be sure to include the necessary foreign key columns.

Solution:



4. Think about which columns should not allow NULL values.
Solution: By definition all the primary key columns do not allow null values. In the DEPARTMENT table the DEPARTMENT_NAME column should probably not allow null values.

In the EMPLOYEE table the FIRST_NAME and LAST_NAME columns are two more candidates for NOT NULL columns because all employees should have names.

The foreign key columns DEPARTMENT_ID and POSITION_ID must be NOT NULL as the relationships in the above diagram indicates. The diagram states that for an individual row in the EMPLOYEE table always a row must exist in the POSITION table and the DEPARTMENT table.

The MANAGER_ID column on the other hand must allow nulls as indicated with the optional recursive relationship. If this was not an optional relationship, you would not be able to enter the president of company for instance, because it requires an existing entry for the

President's manager. Therefore, the top of the org chart hierarchy (e.g., the president) has a null value in the MANAGER_ID column.

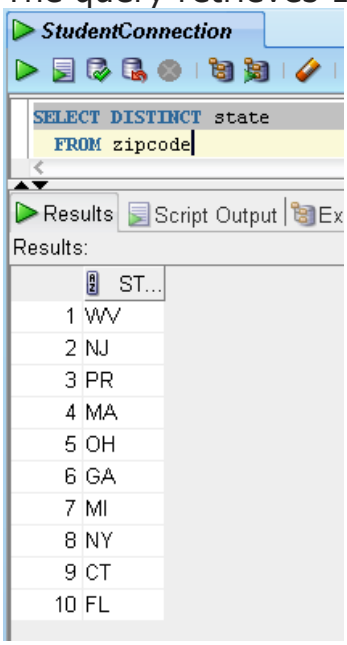
It would be wise to not allow null values for the DESCRIPTION column of the POSITION table as a description should always be entered when a position is created.

Chapter 2: SQL: The Basics

1. Use SQL Developer to retrieve all the STATE values from the ZIPCODE table, without repeating the values.

Solution:

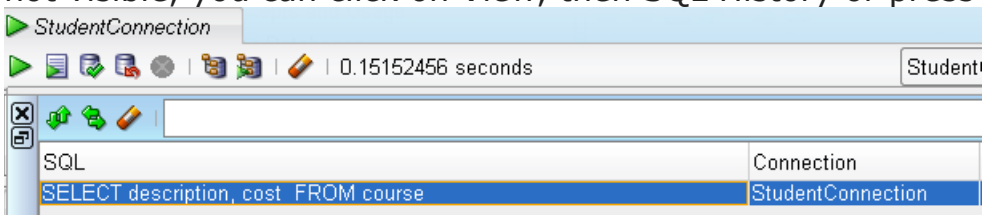
The query retrieves 10 distinct state values.



2. Recall one of the statements you used in Lab 2.1, using the SQL History tab.

Solution:

The SQL commands are saved even after you exit SQL Developer and you use the SQL History tab to retrieve the statement. If the tab is not visible, you can click on View, then SQL History or press F8.



3. What happens if you try to logon to SQL*Plus with the uppercase version of the password learn?

Solution:
You will see an error message indicating that the password is invalid. By default, the Oracle database 11g has case-sensitive passwords.

```
C:\Documents and Settings\Administrator>sqlplus student/LEARN

SQL*Plus: Release 11.1.0.6.0 - Production on Sat Mar 28 11:18:38 2009

Copyright (c) 1982, 2007, Oracle.  All rights reserved.

ERROR:
ORA-01017: invalid username/password; logon denied


Enter user-name:
```

4. Execute the following statements in SQL*Plus and record your observations.

```
SET NULL 'NULL'
SELECT DISTINCT cost
  FROM course
```

Solution:
The SET NULL SQL*Plus command substitutes the display of any null values with another value, such as the literal 'null' in this example. The column's null value is not changed in the database; this command only changes the SQL*Plus display value.

```
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Producti
With the Partitioning, OLAP, Data Mining and Real Application Testir

SQL> SET NULL 'NULL'
SQL> SELECT DISTINCT cost
  2    FROM course;

      COST
----------
      1595
NULL
      1095
      1195

SQL>
```

PD:

Chapter 3: The WHERE and ORDER BY Clauses

1. Create a SQL statement that retrieves data from the COURSE table for courses that cost 1195 and whose descriptions start with Intro, sorted by their prerequisites.

Solution:

```
SELECT *
  FROM course
 WHERE cost = 1195
   AND description like 'Intro%'
 ORDER BY prerequisite;
```

2. Create another SQL Statement that retrieves data from the STUDENT table for students whose last names begin with A,B, or C, and who work for Competrol Real Estate, sorted by their last names.

Solution:

```
SELECT *
  FROM student
 WHERE (last_name like 'A%'
        OR last_name like 'B%'
        OR last_name like 'C%')
   AND employer = 'Competrol Real Estate';
```

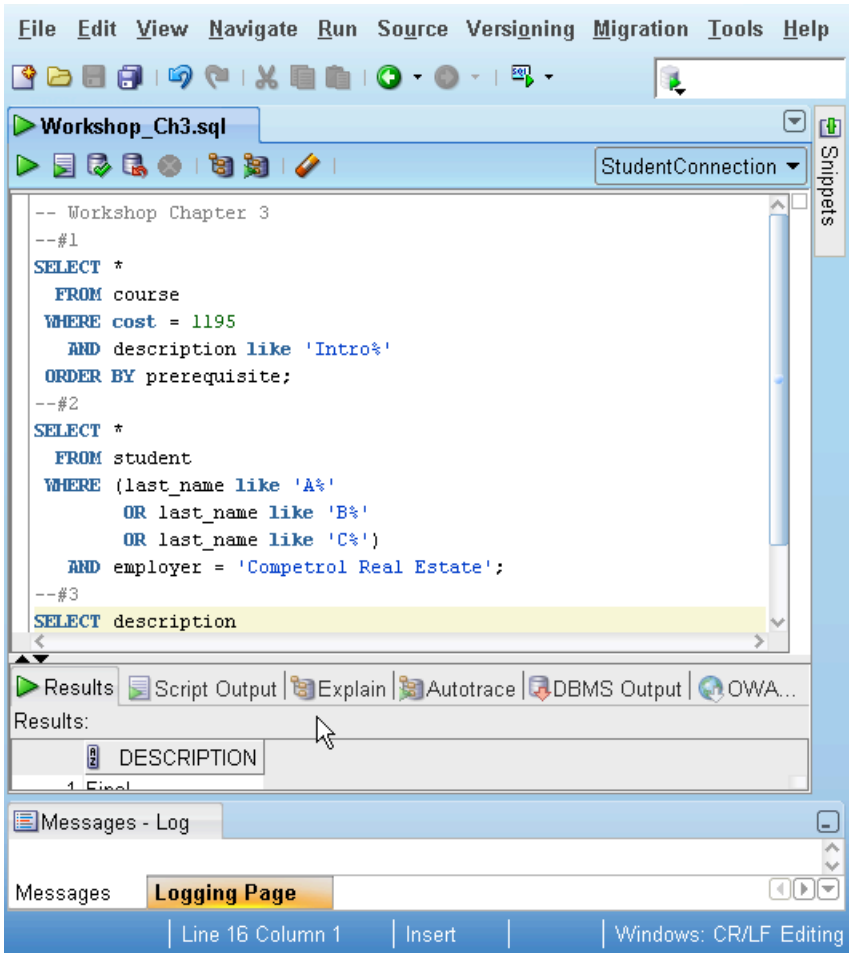
3. Write a SQL statement that retrieves all the descriptions from the GRADE_TYPE table, for rows that were modified by the user MCAFFREY.

Solution:

```
SELECT description
  FROM grade_type
 WHERE modified_by = 'MCAFFREY';
```

4. Save all three SQL statements in a file called Workshop_Ch3.sql

Solution:



Chapter 4: Character, Number, and Miscellaneous Functions

1. Write the SELECT statement that returns the following output

```
ONE_LINE
-------------------------------------------
Instructor: R. Chow..... Phone: 212-555-1212
Instructor: M. Frantzen.. Phone: 212-555-1212
Instructor: F. Hanks..... Phone: 212-555-1212
Instructor: C. Lowry..... Phone: 212-555-1212
Instructor: A. Morris.... Phone: 212-555-1212
Instructor: G. Pertez.... Phone: 212-555-1212
Instructor: N. Schorin... Phone: 212-555-1212
Instructor: T. Smythe.... Phone: 212-555-1212
Instructor: I. Willig.... Phone: 212-555-1212
```

```
Instructor: T. Wojick.... Phone: 212-555-1212
```

Solution:

```
SELECT RPAD('Instructor: '
  ||SUBSTR(first_name,1,1)
  ||'. '
  ||last_name, 25,'.')
  || 'Phone: '
  || SUBSTR(phone,1,3)
  ||'-'
  ||SUBSTR(phone, 4,3)
  ||'-'
  ||SUBSTR(phone, 7) ONE_LINE
 FROM instructor
ORDER BY last_name
```

2. Rewrite the following query to replace all occurrences of the string Unix with Linux.

```
SELECT 'I develop software on the Unix platform'
  FROM dual
```

Solution:

```
SELECT REPLACE('I develop software on the Unix platform',
       'Unix', 'Linux')
  FROM dual
REPLACE('IDEVELOPSOFTWAREONTHEUNIXPLATFO
----------------------------------------
I develop software on the Linux platform

1 row selected.
```

3. Determine which student does not have the first letter of her or his last name capitalized. Show the STUDENT_ID and LAST_NAME columns.

Solution:

```
SELECT student_id, last_name
  FROM student
 WHERE SUBSTR(last_name,1,1) = SUBSTR(LOWER(last_name),1,1)
STUDENT_ID LAST_NAME
---------- -------------
       206 annunziato
```

```
1 row selected.
```

In Chapter 16, "Regular Expressions and Hierarchical Queries," you will learn about regular expressions which can accomplish the same result.

4. Check whether any of the phone numbers in the INSTRUCTOR table have been entered in the (###)###-#### format.

Solution:

```
SELECT phone
  FROM instructor
 WHERE TRANSLATE(
       phone, '0123456789',
              '##########') = '(###)###-####'
```

```
no rows selected
```

You can optionally include an extra space after the '9', which will ignore any extra spaces added to the phone number. For example, numbers in the format (212) 555-1111 and (212)555-1111 would be listed in the result set. In the case of the INSTRUCTOR table, all the phone numbers do not follow this format; therefore no rows are shown in the output.

```
SELECT phone, instructor_id
  FROM instructor
 WHERE TRANSLATE(
         phone, '0123456789 ',
                '##########') = '(###)###-####'
```

```
no rows selected
```

Refer to Chapter 16, "Regular Expressions and Hierarchical Queries," for more advanced format checking using regular expressions.

5. Explain the functionality of the following query.

```
SELECT section_id, capacity,
       CASE WHEN MOD (capacity, 2) <> 0 THEN 'Odd capacity'
       ELSE 'Even capacity'
       END "Odd or Even"
  FROM section
 WHERE section_id IN (101, 146, 147)
```

Solution: The query shows for sections 101, 146, and 147 the SECTION_ID and CAPACITY in the first two columns. The last column displays either the words 'Even capacity' or 'Odd capacity' depending on the value of the capacity.

```
SECTION_ID   CAPACITY Odd or Even
---------- ---------- -------------
       101         10 Even capacity
       146         25 Odd capacity
       147         15 Odd capacity

3 rows selected.
```

Chapter 5: Date and Conversion Functions

1. Display all the sections where classes start at 10:30 AM.

Solution:

```
SELECT section_id, TO_CHAR(start_date_time, 'HH24:MI')
  FROM section
 WHERE TO_CHAR(start_date_time, 'HH24:MI') ='10:30'
SECTION_ID TO_CH
---------- -----
        85 10:30
        95 10:30
       104 10:30
       109 10:30
       116 10:30
       122 10:30

6 rows selected.
```

2. Write a query that accomplishes the following result. The output shows you all the days of the week where sections 99, 89, and 105 start. Note the order of the days.

```
DAY SECTION_ID
--- ----------
Mon         99
Tue         89
Wed        105

3 rows selected.
```

Solution:

```
SELECT TO_CHAR(start_date_time, 'Dy') day, section_id
  FROM section
 WHERE section_id IN (99,89,105)
```

```
 ORDER BY TO_CHAR(start_date_time, 'D')
```

The 'Dy' format displays the days as Mon, Tues, and so forth. The 'D' format listed in the ORDER BY clause will return '1' for Sunday, '2' for Monday, '3' for Tuesday, etc. The 'D' format in the ORDER BY clause is preferable if you want to order not alphabetically but by the day of the week. In this particular example, the alphabetical ordering of the days coincides with the sequence of days of the week, but there may be instances where this is not the case. Therefore, the 'D' format is preferable if that's the desired ordering.

3. Select the distinct course costs for all the courses. If a course cost is unknown, substitute a zero. Format the output with a leading $ sign, and separate the thousands with a comma. Display two digits after the decimal point. The query's output should look like the following result.

```
COST
-----------
      $0.00
  $1,095.00
  $1,195.00
  $1,595.00

4 rows selected.
```

Solution:

```
SELECT DISTINCT TO_CHAR(NVL(cost, 0), '$99,990.99') cost
  FROM course
 ORDER BY cost
```

or:

```
SELECT DISTINCT TO_CHAR(COALESCE(cost, 0), '$99,990.99')
cost
  FROM course
ORDER BY cost
```

4. List all the rows of the GRADE_TYPE table that were created in the year 1998.

Solution:

```
SELECT *
  FROM grade_type
 WHERE created_date >=TO_DATE('01-JAN-1998', 'DD-MON-YYYY')
   AND created_date < TO_DATE('01-JAN-1999', 'DD-MON-YYYY')
```

5. What, if anything, is wrong with the following SQL statement?

```
SELECT zip + 100
 FROM zipcode
```

Solution: The query executes, but doesn't make sense because you don't do calculations on the ZIP column, that's one of the reasons why the ZIP column in VARCHAR2 data type format and it stores leading zeros. Additionally, any calculation should not rely on an implicit conversion; it is better to use the TO_NUMBER function.

6. For the students enrolled on January 30, 2007, display the columns STUDENT_ID and ENROLL_DATE.

Solution: There are many possible solutions, here are just a few listed:

```
SELECT student_id, enroll_date
  FROM enrollment
 WHERE enroll_date> = TO_DATE('30-JAN-2007', 'DD-MON-YYYY')
   AND enroll_date < TO_DATE('31-JAN-2007', 'DD-MON-YYYY')
```

or:

```
SELECT student_id, enroll_date
  FROM enrollment
 WHERE enroll_date >= DATE '2007-01-30'
   AND enroll_date < DATE '2007-01-31'
```

or:

```
SELECT student_id, enroll_date
  FROM enrollment
 WHERE TRUNC(enroll_date) = TO_DATE('30-JAN-2007', 'DD-MON-
YYYY')
```

Resulting output:

```
STUDENT_ID ENROLL_DA
---------- ---------
       102 30-JAN-07
       102 30-JAN-07
       103 30-JAN-07
       104 30-JAN-07
       105 30-JAN-07
       106 30-JAN-07
       106 30-JAN-07
       107 30-JAN-07
       108 30-JAN-07
       109 30-JAN-07
       109 30-JAN-07

11 rows selected.
```

Chapter 6: Aggregate Functions, GROUP BY and HAVING

1. List the order in which the WHERE, GROUP BY, and HAVING clauses are executed by the database in the following SQL statement.

```
SELECT section_id, COUNT(*), final_grade
  FROM enrollment
 WHERE TRUNC(enroll_date) > TO_DATE('2/16/2003', 'MM/DD/YYYY')
 GROUP BY section_id, final_grade HAVING COUNT(*) > 5
```

Solution: First the WHERE clause is executed, then the GROUP BY, and lastly the HAVING clause is applied.

2. Display a count of all the different course costs in the COURSE table.

Solution:

```
SELECT cost, COUNT(*)
  FROM course
 GROUP BY cost
     COST  COUNT(*)
--------- ---------
     1095         3
     1195        25
     1595         1
                  1

4 rows selected.
```

Note the NULL value in the result set, and notice the difference if you write the SQL statement using COUNT(cost)instead.

```
SELECT cost, COUNT(cost)
  FROM course
 GROUP BY cost
     COST COUNT(COST)
--------- -----------
     1095           3
     1195          25
     1595           1
                    0

4 rows selected.
```

3. Determine the number of students living in zip code 10025.

Solution:

```
SELECT COUNT(*)
  FROM student
 WHERE zip = '10025'
 COUNT(*)
---------
        3

1 row selected.
```

4. Show all the different companies for which students work. Display only companies in which more than four students are employed.

Solution:

```
SELECT employer, COUNT(*)
  FROM student
 GROUP BY employer
HAVING COUNT(*) > 4
EMPLOYER              COUNT(*)
-------------------- ---------
Amer.Legal Systems           10
Crane Co.                     6
Electronic Engineers         15
New York Pop                  8

4 rows selected.
```

5. List how many sections each instructor teaches.

Solution:

```
SELECT instructor_id, COUNT(*)
  FROM section
 GROUP BY instructor_id
INSTRUCTOR_ID  COUNT(*)
------------- ---------
          101         9
          102        10
          103        10
          104        10
          105        10
          106        10
          107        10
          108         9
```

```
8 rows selected.
```

6. What problem does the following statement solve?

```
SELECT COUNT(*), start_date_time, location
  FROM section
 GROUP BY start_date_time, location
HAVING COUNT(*) > 1
```

Solution:
List the date, time, and location of sections that meet at the same time, date, and location.

```
COUNT(*) START_DAT LOCATION
-------- --------- --------
       2 09-APR-07 L214
       2 16-APR-07 L509

2 rows selected.
```

7. Determine the highest grade achieved for the midterm within each section.

Solution:

```
SELECT section_id, MAX(numeric_grade)
  FROM grade
 WHERE grade_type_code = 'MT'
 GROUP BY section_id
SECTION_ID MAX(NUMERIC_GRADE)
---------- ------------------
        80                 76
        81                 88
...
       154                 92
       156                 99

56 rows selected.
```

8. Suppose you have a table called CUSTOMER_ORDER, which contains 5,993 rows with an order total of $10,993,333.98 based on the orders from 4,500 customers. Given this scenario, how many row(s) does the following query return?

```
SELECT SUM(order_amount) AS "Order Total"
  FROM customer_order
```

Solution: Aggregate functions always return a single row. The result of the SUM function will return 10,993.333.98.

Chapter 7: Equijoins

1. Select the course description, section number, and location for sections meeting in location L211.

Solution:

```
SELECT description, section_no, location
  FROM course c, section s
 WHERE c.course_no = s.course_no
   AND location = 'L211'
```

Using the ANSI JOIN syntax and the ON clause, it can also be written as:

```
SELECT description, section_no, location
  FROM course c JOIN section s
    ON c.course_no = s.course_no
 WHERE location = 'L211'
```

Or with the USING clause:

```
SELECT description, section_no, location
  FROM course c JOIN section s
 USING (course_no)
 WHERE location = 'L211'
```

```
DESCRIPTION                    SECTION_NO LOCAT
------------------------------ ---------- -----
Project Management                      1 L211
Java Developer I                        4 L211
Intermediate Java Programming           2 L211

3 rows selected.
```

2. Show the course description, section number, starting date and time of the courses Joseph German is taking.

Solution:

```
SELECT description, section_no, start_date_time
  FROM course c, section s, enrollment e, student st
 WHERE c.course_no = s.course_no
   AND s.section_id = e.section_id
   AND e.student_id = st.student_id
   AND st.last_name = 'German'
   AND first_name = 'Joseph'
```

```
DESCRIPTION                SECTION_NO START_DAT
-------------------------- ---------- ---------
Intro to Java Programming           2 24-JUL-07

1 row selected.
```

As always you can express this with the ANSI join syntax as follows which may look like this:

```
SELECT c.description, s.section_no,
       TO_CHAR(s.start_date_time, 'DD-MON-YYYY HH24:MI:SS')
  FROM course c JOIN section s
    ON (c.course_no = s.course_no)
  JOIN enrollment e
    ON (s.section_id = e.section_id)
  JOIN student st
    ON (e.student_id = st.student_id)
 WHERE st.last_name = 'German'
   AND st.first_name = 'Joseph'
```

Or you can write as follows with the USING clause.

```
SELECT c.description, s.section_no,
       TO_CHAR(s.start_date_time, 'DD-MON-YYYY HH24:MI:SS')
  FROM course c JOIN section s
 USING (course_no)
  JOIN enrollment e
 USING (section_id)
  JOIN student st
 USING (student_id)
 WHERE st.last_name = 'German'
   AND st.first_name = 'Joseph'
```

3. List the instructor ID, last name of the instructor, and section ID of sections where class participation contributes to 25 percent of the total grade. Order the result by the instructor's last name.

Solution:

```
SELECT i.instructor_id, s.section_id, last_name
  FROM instructor i, section s, grade_type_weight w
 WHERE i.instructor_id = s.instructor_id
   AND s.section_id = w.section_id
   AND percent_of_final_GRADE = 25
   AND grade_type_code = 'PA'
 ORDER BY last_name
INSTRUCTOR_ID SECTION_ID LAST_NAME
------------- ---------- ----------
```

```
       107        115 Frantzen
       101        133 Hanks
       108        155 Lowry
       105        129 Morris
       105        144 Morris
       104         82 Pertez
       106        137 Smythe
       102        149 Wojick
       102         88 Wojick

9 rows selected.
```

Or as an ANSI join with the USING clause:

```
SELECT instructor_id, section_id, last_name
  FROM instructor JOIN section
 USING (instructor_id)
  JOIN grade_type_weight
 USING (section_id)
 WHERE percent_of_final_grade = 25
   AND grade_type_code = 'PA'
 ORDER BY 3
```

4. Display the first and last names of students who received 99 or more points on the class project.

Solution:

```
SELECT first_name, last_name, numeric_grade
  FROM student s, enrollment e, grade g
 WHERE s.student_id = e.student_id
   AND e.student_id = g.student_id
   AND e.section_id = g.section_id
   AND numeric_grade >= 99
   AND grade_type_code = 'PJ'
FIRST_NAME LAST_NAME       NUMERIC_GRADE
---------- --------------- -------------
May        Jodoin                     99
Joel       Brendler                   99

2 rows selected.
```

Or as expressed with an ANSI join:

```
SELECT first_name, last_name, numeric_grade
  FROM student JOIN enrollment
 USING (student_id)
  JOIN grade
 USING (student_id, section_id)
```

```
 WHERE numeric_grade >= 99
   AND grade_type_code = 'PJ'
```

5. Select the grades for quizzes of students living in zip code 10956.

Solution:

```
SELECT s.student_id, s.last_name, s.first_name,
       g.numeric_grade, s.zip
  FROM student s, enrollment e, grade g
 WHERE s.student_id = e.student_id
   AND e.student_id = g.student_id
   AND e.section_id = g.section_id
   AND g.grade_type_code = 'QZ'
   AND s.zip = '10956'
```

```
STUDENT_ID LAST_NAME   FIRST_NAME  NUMERIC_GRADE ZIP
---------- ----------- ----------- ------------- -----
       193 Jamerncy    Al                     91 10956
       193 Jamerncy    Al                     90 10956

2 rows selected.
```

Alternatively, you can also join the GRADE table directly to the STUDENT table. (For more information on skipping this table, see Lab 7.2 and the paragraph titled "SKIPPING THE PRIMARY/FOREIGN KEY PATH"

```
SELECT s.student_id, s.last_name, s.first_name,
       g.numeric_grade, s.zip
  FROM student s, grade g
 WHERE g.grade_type_code = 'QZ'
   AND g.student_id = s.student_id
   AND s.zip = '10956'
```

Or expressed in ANSI join syntax with three tables:

```
SELECT s.student_id, s.last_name, s.first_name,
       g.numeric_grade, s.zip
  FROM student s JOIN enrollment e
    ON (s.student_id = e.student_id)
  JOIN grade g
    ON (e.student_id = g.student_id
   AND e.section_id = g.section_id)
 WHERE g.grade_type_code = 'QZ'
   AND s.zip = '10956'
```

6. List the course number, section number, and instructor first and last names of classes with course number 350 as a prerequisite.

Solution:

```
SELECT c.course_no, section_no, first_name,
       last_name
  FROM course c, section s, instructor i
 WHERE c.course_no = s.course_no
   AND s.instructor_id = i.instructor_id
   AND prerequisite = 350
COURSE_NO SECTION_NO FIRST_NAME LAST_NAME
--------- ---------- ---------- -----------
      450          1 Fernand    Hanks

1 row selected.
```

The solution can also be achieved using one of the ANSI join syntax variants:

```
SELECT c.course_no, section_no, first_name,
       last_name
  FROM course c JOIN section s
    ON (c.course_no = s.course_no)
  JOIN instructor i
    ON (s.instructor_id = i.instructor_id)
 WHERE prerequisite = 350
```

7. What problem do the following two SELECT statements solve?

```
SELECT stud.student_id, i.instructor_id,
       stud.zip, i.zip
  FROM student stud, instructor i
 WHERE stud.zip = i.zip

SELECT stud.student_id, i.instructor_id,
       stud.zip, i.zip
  FROM student stud, enrollment e, section sec,
       instructor i
 WHERE stud.student_id = e.student_id
   AND e.section_id = sec.section_id
   AND sec.instructor_id = i.instructor_id
   AND stud.zip = i.zip
```

Solution: The two queries identify students that live in the same zip code as instructors.

The first statement determines those instructors who live in the same zip code as students. It builds a Cartesian product, because there are

multiple occurrences of the same zip code in both the INSTRUCTOR and STUDENT tables. The result looks like this:

```
STUDENT_ID INSTRUCTOR_ID ZIP   ZIP
---------- ------------- ----- -----
       223           102 10025 10025
       399           102 10025 10025
       163           102 10025 10025
       223           103 10025 10025
       399           103 10025 10025
       163           103 10025 10025
       223           106 10025 10025
       399           106 10025 10025
       163           106 10025 10025
       223           108 10025 10025
       399           108 10025 10025
       163           108 10025 10025

12 rows selected.
```

The second statement shows the instructors who live in the same zip code as the student they teach. The result is as follows:

```
STUDENT_ID INSTRUCTOR_ID ZIP   ZIP
---------- ------------- ----- -----
       223           103 10025 10025
       163           106 10025 10025

2 rows selected.
```

Chapter 8: Subqueries

1. Using a subquery construct, determine which sections the student Henry Masser is enrolled in.

Solution:

```
SELECT section_id
  FROM enrollment
 WHERE student_id IN
       (SELECT student_id
          FROM student
         WHERE last_name = 'Masser'
           AND first_name = 'Henry')

no rows selected
```

Note: Henry Masser is not enrolled in any section at all. The query returns no rows.

2. What problem does the following SELECT statement solve?

```
SELECT zip
  FROM zipcode z
 WHERE NOT EXISTS
       (SELECT '*'
          FROM student
         WHERE z.zip = zip)
   AND NOT EXISTS
       (SELECT '*'
          FROM instructor
         WHERE z.zip = zip)
```

Solution: The query determines the zip codes not found in either the STUDENT table or the INSTRUCTOR table.

3. Display the course number and description of courses with no enrollment. Also include courses which have no section assigned.

Solution:

```
SELECT course_no, description
  FROM course c
 WHERE NOT EXISTS
       (SELECT NULL
          FROM section s
         WHERE c.course_no = s.course_no)
   OR course_no IN
       (SELECT course_no
          FROM section s2
         WHERE NOT EXISTS
               (SELECT NULL
                  FROM enrollment e
                 WHERE s2.section_id = e.section_id))
```

```
COURSE_NO DESCRIPTION
--------- --------------------------------
       25 Intro to Programming
       80 Programming Techniques
...
      350 Java Developer II
      430 Java Developer III

16 rows selected.
```

4. Can the ANY and ALL operators be used on the DATE data type? Write a simple query to prove your answer.

Solution: Yes, the ANY and ALL operators work on the DATE data type. There are many different possible sample queries. Here is one for each operator. The queries produce the correct result with no error.

```
SELECT 'Hello'
  FROM dual
 WHERE TO_DATE('12-MAR-2009', 'DD-MON-YYYY') < ANY
       (TO_DATE('13-MAR-2009', 'DD-MON-YYYY'),
        TO_DATE('14-MAR-2009', 'DD-MON-YYYY'))
'HELL
-----
Hello

1 row selected.

SELECT 'Hello again'
   FROM dual
  WHERE TO_DATE('12-MAR-2096', 'DD-MON-YYYY') >ALL
        (SELECT created_date
           FROM grade)
'HELLOAGAIN
-----------
Hello again

1 row selected.
```

5. If you have a choice to write either a correlated subquery or a simple subquery, which one would you choose? Why?

Solution:
The correlated subquery using the NOT EXISTS operator tests for NULL values which the NOT IN operator does not. Another consideration is the number of records returned by the outer query and the inner query. If the outer query returns a large number of records, the correlated subquery must test for each of these outer rows, which is very time-consuming. If the inner query returns only a very few records, the simple subquery is typically best. To determine the most efficient statement, test against realistic data volumes and properly indexed tables. For more information about this topic see Chapter 18, "SQL Optimization."

6. Determine the top three zip codes where most of the students live.

Solution:

```
SELECT s.*, ROWNUM ranking
  FROM (SELECT zip, COUNT(*)
          FROM student
         GROUP BY zip
         ORDER BY 2 DESC) s
 WHERE ROWNUM <= 3
ZIP    COUNT(*)    RANKING
----- --------- ---------
07024         9         1
07010         6         2
11368         6         3

3 rows selected.
```

Note, if you execute the inline view query, you notice that there are actually three zip codes with six students enrolled each. Below is a partial listing of the query.

```
SELECT zip, COUNT(*)
  FROM student
 GROUP BY zip
 ORDER BY 2 DESC
ZIP    COUNT(*)
----- ---------
07024         9
07010         6
11373         6
11368         6
07042         5
...
06605         1
06798         1

145 rows selected.
```

The zip code 07024 has the largest number of students. Three zip codes follow with an equal number of enrollments. But only two are included in the query, because the ROWNUM pseudocolumn picks a maximum of three rows. In Chapter 17, "Exploring Data Warehousing Features," you will learn more about top-n queries.

Chapter 9: Set Operators

1. List all the zip codes in the ZIPCODE table that are not used in the STUDENT or INSTRUCTOR tables. Write two different solutions, using set operators for both.

Solution:

```
SELECT zip
  FROM zipcode
MINUS
SELECT zip
  FROM student
MINUS
SELECT zip
  FROM instructor
```

or:

```
SELECT zip
  FROM zipcode
MINUS
(SELECT zip
   FROM student
 UNION
 SELECT zip
   FROM instructor)
```

```
ZIP
-----
00914
06401
...
30342
33431

79 rows selected.
```

2. Write a SQL statement, using a set operator, to show which students enrolled in a section on the same day they registered.

Solution:

```
SELECT student_id, TRUNC(registration_date)
  FROM student
INTERSECT
SELECT student_id, TRUNC(enroll_date)
  FROM enrollment
```

```
no rows selected
```

3. Find the students who are not enrolled in any classes. Write three solutions: a set operation, a subquery, and a correlated subquery.

Solution:

```
SELECT student_id
  FROM student
MINUS
SELECT student_id
  FROM enrollment

SELECT student_id
  FROM student
 WHERE student_id NOT IN
       (SELECT student_id
          FROM enrollment)

SELECT student_id
  FROM student s
 WHERE NOT EXISTS
       (SELECT 'x'
          FROM enrollment e
         WHERE s.student_id = e.student_id)
```

```
STUDENT_ID
----------
       284
       285
...
       397
       399

103 rows selected.
```

4. Show the students who have received grades for their class. Write four solutions: a set operation, a subquery, a correlated subquery, and a join.

Solution:

```
SELECT section_id, student_id
  FROM enrollment
INTERSECT
SELECT section_id, student_id
  FROM grade

SELECT section_id, student_id
  FROM enrollment
```

```
 WHERE (student_id, section_id) IN
       (SELECT student_id, section_id
          FROM grade)

SELECT section_id, student_id
  FROM enrollment e
 WHERE EXISTS
       (SELECT 1
          FROM grade g
         WHERE e.section_id = g.section_id
           AND e.student_id = g.student_id)

SELECT DISTINCT e.section_id, e.student_id
  FROM enrollment e, grade g
 WHERE e.section_id = g.section_id
   AND e.student_id = g.student_id
SECTION_ID STUDENT_ID
---------- ----------
        80        128
        81        103
...
       156        214
       156        215

226 rows selected.
```

Chapter 10: Complex Joins

1. Write a query that shows all the instructors who live in the same zip code.

Solution:

```
SELECT DISTINCT i1.first_name, i1.last_name, i1.zip
  FROM instructor i1, instructor i2
 WHERE i1.zip = i2.zip
   AND i1.instructor_id <> i2.instructor_id
 ORDER BY i1.zip
FIRST_NAME LAST_NAME      ZIP
---------- -------------- -----
Rick       Chow           10015
Fernand    Hanks          10015
Anita      Morris         10015
Charles    Lowry          10025
Nina       Schorin        10025
```

```
Todd        Smythe          10025
Tom         Wojick          10025

7 rows selected.
```

Or the query can also be written as an ANSI join as follows:

```
SELECT DISTINCT i1.first_name, i1.last_name, i1.zip
  FROM instructor i1 JOIN instructor i2
    ON (i1.zip = i2.zip)
 WHERE i1.instructor_id <> i2.instructor_id
 ORDER BY i1.zip
```

Note: You can also move the WHERE conditions into the ON clause and it will yield the same result as they are all AND conditions that need to be met for the records to be returned in the result set.

2. Are any of the rooms overbooked? Determine whether any sections meet at the same date, time, and location.

Solution:

```
SELECT DISTINCT s.section_id,
       TO_CHAR(s.start_date_time, 'DD-MON-YYYY HH24:MI'),
       s.location
  FROM section s, section b
 WHERE s.location = b.location
   AND s.start_date_time = b.start_date_time
   AND s.section_id <> b.section_id
 ORDER BY 2, 3
SECTION_ID TO_CHAR(S.START_D LOCAT
---------- ---------------- -----
       128 09-APR-2007 09:30 L214
       132 09-APR-2007 09:30 L214
       101 16-APR-2007 09:30 L509
       140 16-APR-2007 09:30 L509

4 rows selected.
```

Instead of using the primary key to compare if this is the same row or not, you could use the ROWID pseudocolumn instead (see Chapter 13, "Indexes, Sequences, and Views" for more on ROWIDs). The ROWID can be useful if no primary key on table exists.

```
 WHERE s.location = b.location
   AND s.start_date_time = b.start_date_time
   AND s.rowid <> b.rowid
```

Alternatively, the query can be written as follows:

```
SELECT section_id, TO_CHAR(start_date_time,
```

```
       'DD-MON-YYYY HH24:MI'), location
  FROM section
 WHERE (location, start_date_time) IN
       (SELECT location, start_date_time
          FROM section
         GROUP BY start_date_time, location
        HAVING COUNT(*) > 1)
```

3. Determine whether there is any scheduling conflict between instructors: Are any instructors scheduled to teach one or more sections at the same date and time? Order the result by the INSTRUCTOR_ID and the starting date and time of the sections.

Solution:

```
SELECT DISTINCT s1.instructor_id,
       TO_CHAR(s1.start_date_time, 'DD-MON-YYYY HH24:MI'),
       s1.section_id
  FROM section s1, section s2
 WHERE s1.instructor_id = s2.instructor_id
   AND s1.start_date_time = s2.start_date_time
   AND s1.section_id <> s2.section_id
```

```
INSTRUCTOR_ID TO_CHAR(S1.START_ SECTION_ID
------------- ---------------- ----------
          101 16-APR-2007 09:30        101
          101 16-APR-2007 09:30        140
          102 04-MAY-2007 09:30         88
          102 04-MAY-2007 09:30        149
          103 14-JUL-2007 09:30        107
          103 14-JUL-2007 09:30        119
          103 15-MAY-2007 09:30         89
          103 15-MAY-2007 09:30        150
          103 24-JUL-2007 09:30         81
          103 24-JUL-2007 09:30        127
          103 24-JUL-2007 09:30        142
          104 12-JUN-2007 09:30         90
          104 12-JUN-2007 09:30        151
          105 07-MAY-2007 09:30         97
          105 07-MAY-2007 09:30        129
          107 07-MAY-2007 09:30         99
          107 07-MAY-2007 09:30        115
          107 21-MAY-2007 09:30        138
          107 21-MAY-2007 09:30        154
          108 09-JUN-2007 09:30        100
          108 09-JUN-2007 09:30        139

21 rows selected.
```

Alternatively, you can write the SQL statement as follows:

```
SELECT instructor_id, start_date_time, section_id
  FROM section
 WHERE (instructor_id, start_date_time) IN
       (SELECT instructor_id, start_date_time
          FROM section
         GROUP BY instructor_id, start_date_time
        HAVING COUNT(*) > 1)
```

4. Show the course number, description, course cost, and section ID for courses that cost 1195 or more. Include courses that have no corresponding section.

Solution:

```
SELECT c.course_no, description, section_id, cost
  FROM course c LEFT OUTER JOIN section s
    ON (c.course_no = s.course_no)
 WHERE cost >= 1195
 ORDER BY 1
```

```
COURSE_NO DESCRIPTION                SECTION_ID       COST
--------- ------------------------- ---------- ----------
       10 Technology Concepts               80       1195
...
       80 Programming Techniques                     1595
      100 Hands-On Windows                 141       1195
...
      124 Advanced Java Programming         126       1195
      124 Advanced Java Programming         127       1195
...
      430 Java Developer III                         1195

71 rows selected.
```

Note courses 80 and 430 do not have a corresponding section assigned.

Or you can write the query using the traditional syntax, with the comma between the tables in the FROM clause.

```
SELECT c.course_no, description, section_id, cost
  FROM course c, section s
 WHERE c.course_no = s.course_no(+)
   AND cost >= 1195
 ORDER BY 1
```

5. Write a query that lists the section numbers and students IDs of students enrolled in classes held in location 'L210'. Include sections for which no students are enrolled.

Solution:

```
SELECT s.section_id, e.section_id, e.student_id
  FROM section s LEFT OUTER JOIN enrollment e
    ON s.section_id = e.section_id
 WHERE location = 'L210'
SECTION_ID SECTION_ID STUDENT_ID
---------- ---------- ----------
        81         81        103
        81         81        104
        81         81        240
        84         84        158
...
       124
       129
...
       155        155        248
       155        155        241
       155        155        127

31 rows selected.
```

You can also write the query as follows:

```
SELECT s.section_id, e.section_id, e.student_id
  FROM section s, enrollment e
 WHERE location = 'L210'
   AND s.section_id = e.section_id(+)
```

Chapter 11: Insert, Update, and Delete

1. Write and execute two INSERT statements to insert rows in the ZIPCODE table for the following two cities: Newton, MA 02199 and Cleveland, OH 43011. After your INSERT statements are successful, make the changes permanent.

Solution:

```
INSERT INTO zipcode
  (city, state, zip, created_date, created_by,
   modified_date, modified_by)
VALUES
  ('Newton', 'MA', '02199', SYSDATE, USER,
   SYSDATE, USER)
```

```
INSERT INTO zipcode
  (city, state, zip, created_date, created_by,
   modified_date, modified_by)
VALUES
  ('Cleveland', 'OH', '43011', SYSDATE, USER,
   SYSDATE, USER)
COMMIT
```

2. Make yourself a student by writing and executing an INSERT statement to insert a row into the STUDENT table with data about you. Use one of the zip codes you inserted in exercise 1. Insert values into the columns STUDENT_ID (use the value of '900'), FIRST_NAME, LAST_NAME, ZIP, REGISTRATION_DATE (use a date that is five days after today), CREATED_BY, CREATED_DATE, MODIFIED_BY, and MODIFIED_DATE. Issue a COMMIT command when you are done.

Solution:

```
INSERT INTO student
  (student_id, first_name, last_name,
   zip, registration_date,
   created_by, created_date, modified_by, modified_date)
VALUES
  (900, 'Sandy', 'Dellacorte',
   '02199', SYSDATE + 5,
   USER, SYSDATE, USER, SYSDATE)
COMMIT
```

3. Write an UPDATE statement to update the data about you in the STUDENT table. Update the columns SALUTATION, STREET_ADDRESS, PHONE, and EMPLOYER. Be sure to also update the MODIFIED_DATE column and make the changes permanent.

Solution:

```
UPDATE student
   SET salutation = 'Ms.',
       street_address = '60 Winter St.',
       phone = '617-236-2746',
       employer = 'Raytone',
       modified_by = USER,
       modified_date = SYSDATE
 WHERE student_id = 900
COMMIT
```

4. Delete the row you created the STUDENT table and the two rows you created in the ZIPCODE table. Be sure to issue a COMMIT

command afterwards. . You can perform this action by using a SQL command or SQL Developer's Data tab.

Solution:

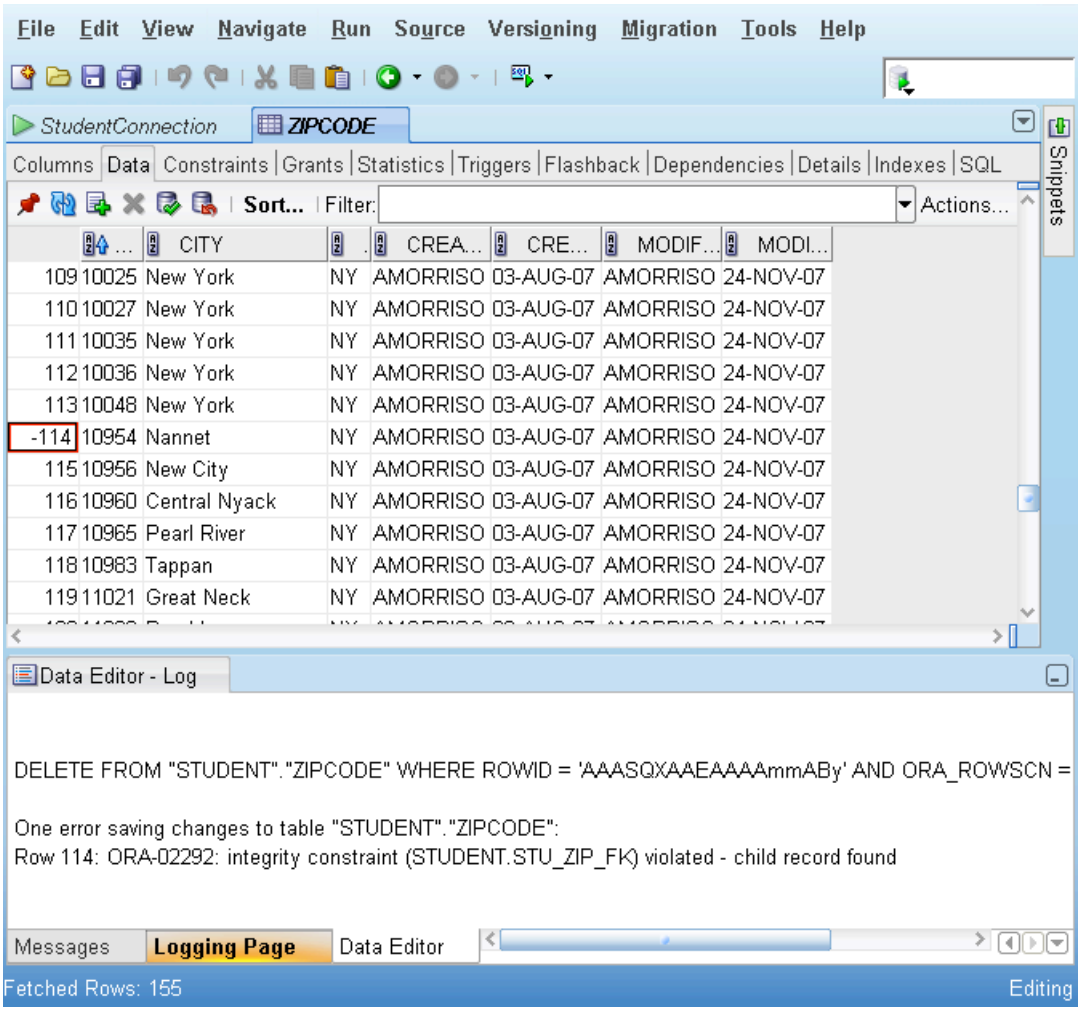
```
DELETE FROM student
 WHERE student_id = 900

DELETE FROM zipcode
 WHERE zip IN ('02199', '43011')

COMMIT
```

5. Delete the zip code 10954 from the ZIPCODE table by using SQL Developer. Commit your change after you delete the row. Describe the results of your actions.

Solution: SQL Developer marks the row for deletion upon pressing the Delete icon. However, as soon as you commit the change, Oracle recognizes that dependent records exist and disallows the deletion of the row. The Data Editor Log tab reports the error as shown below.



If you performed the exercises in this chapter, you will have changed data in most of the tables of the STUDENT schema. If you go back to the previous chapters and reexecute those queries, you might find that the results are different than they were before. Therefore, if you want to reload the tables and data, you can run the rebuildStudent.sql script.

Chapter 12: Create, Alter, and Drop Tables

1. Create a table called TEMP_STUDENT with the following columns and constraints: a column STUDID for student ID that is NOT NULL and is the primary key, a column FIRST_NAME for student first name; a column LAST_NAME for student last name, a column ZIP that is a foreign key to the ZIP column in the ZIPCODE table, a column REGISTRATION_DATE that is NOT NULL and has a CHECK constraint to restrict the registration date to dates after January 1st, 2000.

Solution:

```
CREATE TABLE temp_student
  (studid             NUMBER(8) NOT NULL,
   first_name         VARCHAR2(25),
   last_name          VARCHAR2(25),
   zip                VARCHAR2(5),
   registration_date  DATE NOT NULL,
   CONSTRAINT temp_student_pk PRIMARY KEY(studid),
   CONSTRAINT temp_student_zipcode_fk FOREIGN KEY(zip)
     REFERENCES zipcode(zip),
   CONSTRAINT temp_student_reg_date_ck
     CHECK(registration_date >
           TO_DATE('01-JAN-2000', 'DD-MON-YYYY'))
     )
```

2. Write an INSERT statement that violates one of the constraints for the TEMP_STUDENT table you created in exercise 1. Write another INSERT statement that succeeds when executed, and commit your work.

Solution:

```
INSERT INTO temp_student
  (studid, first_name, last_name,
   zip, registration_date)
VALUES
  (NULL, 'Alex', 'Morrison', '99999', TO_DATE('01-DEC-
1999', 'DD-MON-YYYY'))
INSERT INTO temp_student
      *
ERROR at line 1:
ORA-01400: cannot insert NULL into
("STUDENT"."TEMP_STUDENT"."STUDID")

INSERT INTO temp_student
VALUES (101, 'Alex', 'Morrison', '07656', TO_DATE('01-DEC-
2000', 'DD-MON-YYYY'))
1 row created.
```

3. Alter the TEMP_STUDENT table to add two more columns called EMPLOYER and EMPLOYER_ZIP. The EMPLOYER_ZIP column should have a foreign key constraint that references the ZIP column of the ZIPCODE table. Update the EMPLOYER column, and alter the table once again to make the EMPLOYER column NOT NULL. Drop the TEMP_STUDENT table when you are done with the exercise.

Solution:

```
ALTER TABLE temp_student
  ADD (employer      VARCHAR2(20),
       employer_zip  VARCHAR2(5),
       CONSTRAINT temp_student_fk
         FOREIGN KEY(employer_zip)
         REFERENCES zipcode(zip))

UPDATE temp_student
   SET employer = 'ANM Productions'


ALTER TABLE temp_student
  MODIFY (employer NOT NULL)

DROP TABLE temp_student
```

Chapter 13: Indexes, Sequences, and Views

1. Who can update the SALARY column through the MY_EMPLOYEE view? Hint: The USER function returns the name of the user who is currently logged in.

```
CREATE OR REPLACE VIEW my_employee AS
SELECT employee_id, employee_name, salary, manager
  FROM employee
 WHERE manager = USER
  WITH CHECK OPTION CONSTRAINT my_employee_ck_manager
```

Solution: Only managers can update their respective employee's salaries. The WITH CHECK OPTION constraint ensures that DML statements satisfy the condition in the WHERE clause. This condition enforces that only records are displayed, updated, inserted, and deleted where the value in the MANAGER column is equal to the user currently logged in. A SELECT statement against the MY_EMPLOYEE view for the user with the login ID of JONES could look like this:

```
EMPLOYEE_ID EMPLOYEE_NAME                SALARY MANAG
----------- ------------------------- --------- -----
        150 Gates                         11787 JONES
        251 Sheppard                      11106 JONES
        552 Edwards                        7036 JONES
        353 Philpotts                     11373 JONES
```

2. Which columns in a table should you consider indexing?

Solution: Columns frequently used in the WHERE clause of SQL statements are good candidates for indexes. Be sure to consider the selectivity of the values of the columns, that is, how many distinct values there are in the column. Sometimes it is useful to combine several columns with a low selectivity in a concatenated index. Make sure you properly access the index. You also see more examples on indexes and their impact in Chapter 18, "SQL Optimization." In addition, consider indexing foreign key columns, because they not only are frequently referenced in the WHERE clause of joins, but also improve the locking of records on the child table.

3. Explain the purpose of the Oracle SQL command below.

```
ALTER INDEX crse_crse_fk_i REBUILD
```

Solution: This command rebuilds an existing index named CRSE_CRSE_FK_I without having to drop the old index first and then re-create it.

4. Are NULLs stored in an index? Explain.

Solution: NULLs are not stored in an index. The exception is if it is a concatenated index and the leading column of the index does not contain a NULL value. Another exception is a bitmapped index, which stores null values.

Chapter 14: The Data Dictionary, Scripting, and Reporting

1. Describe the result of the following query.

```
SELECT table_name, column_name, comments
 FROM user_col_comments
```

Solution:
Write a query to display all the column comments in the user's schema.

```
TABLE_NAME COLUMN_NAME     COMMENTS
---------- --------------- --------------------
COURSE     COURSE_NO       The unique ID for a
                           course.
COURSE     DESCRIPTION     The full name for th
```

```
                              is course.
...
ZIPCODE    CREATED_DATE       Audit column - indic
                              ates date of insert.
ZIPCODE    MODIFIED_BY        Audit column - indic
                              ates who made last u
                              pdate.
ZIPCODE    MODIFIED_DATE      Audit column - date
                              of last update.

122 rows selected.
```

The result of your query may vary from the above result depending on the objects you have created. The result shows a list of column comments. It is useful to place comments on columns and/or tables describing the information found within the column or table. The following command creates a column comment for the INSTRUCTOR_ID column on the INSTRUCTOR table.

```
COMMENT ON COLUMN INSTRUCTOR.INSTRUCTOR_ID IS 'The unique
ID for an instructor.'
Comment created.
```

A table comment is stored in the data dictionary view USER_TAB_COMMENTS. The next statement creates a table comment for the instructor table.

```
COMMENT ON TABLE INSTRUCTOR IS 'Profile information for an
instructor.'
Comment created.
```

2. Explain the differences between the views USER_USERS, ALL_USERS, and DBA_USERS.

Solution: The USER_USERS view shows information about the currently logged in user. You see useful information such as the default tablespace name and the temporary tablespace name as well as the date the user was created. The ALL_USERS view shows a list of all the users in the system and the date the user was created. The DBA_USERS view displays all the users in the system. The columns listed include the date the user was created, the default and temporary tablespaces, and the encrypted password.

3. What are the underlying data dictionary views for the public synonyms TABS and COLS?

Solution: The public synonyms are USER_TABLES and USER_TAB_COLUMNS. The queries to determine this solution are:

```
SELECT synonym_name, table_name
  FROM all_synonyms
 WHERE synonym_name IN ('TABS', 'COLS')
```

```
SYNONYM_NAME                   TABLE_NAME
------------------------------ ------------------------------
TABS                           USER_TABLES
COLS                           USER_TAB_COLUMNS

2 rows selected.
```

You can also query the DICT view with this statement.

```
SELECT *
  FROM dict
 WHERE table_name IN ('TABS', 'COLS')
```

```
TABLE_NAME           COMMENTS
-------------------- ------------------------------
COLS                 Synonym for USER_TAB_COLUMNS
TABS                 Synonym for USER_TABLES

2 rows selected.
```

4. Write a dynamic SQL script to drop all views in the STUDENT schema. If there are no views, create some to test your script.

Solution: To create some views issue the following statements.

```
CREATE OR REPLACE VIEW view1_v AS
SELECT *
  FROM student

CREATE OR REPLACE VIEW view2_v AS
SELECT *
  FROM course
```

Now create a file with the following commands. Save the file.

```
SET ECHO OFF
REM File name: drop_view.sql
REM Purpose:   Drop all the views in a user's schema.
REM Created:   17-Mar-2009 AR
REM Version:   1.0
SET PAGESIZE 0
SET LINESIZE 80
SET FEEDBACK OFF
SET TERM OFF
```

```
SPOOL drop_view.out
SELECT 'DROP VIEW '||view_name||' ;'
  FROM user_views;
SPOOL OFF
SET PAGESIZE 24
SET LINESIZE 80
SET FEEDBACK ON
SET TERM ON
SET ECHO ON
@drop_view.out
```

Then execute the file from the SQL*Plus prompt with the @ command.

```
SQL>@drop_view.sql
```

Chapter 15: Security

To complete the exercises below, create a new user called SCHOOL with the password program, and grant CONNECT and RESOURCE privileges to it. Then log in as the STUDENT user.

```
-- Create SCHOOL user
CONN SYSTEM/manager
CREATE USER school IDENTIFIED BY program;
GRANT CONNECT, RESOURCE TO school;
```

1. Create two roles: REGISTRAR and INSTRUCTOR.

Solution: Make sure you have the CREATE ROLE system privilege, otherwise you will not be able to create the roles.

```
CONN student/learn
CREATE ROLE registrar;
CREATE ROLE instructor;
```

2. Create a view called CURRENT_REGS that reflects all students who registered on January 25, 2007. Grant SELECT privileges on the new view to the REGISTRAR role.

Solution:

```
CREATE OR REPLACE VIEW current_regs AS
SELECT first_name, last_name
  FROM student
 WHERE registration_date >= TO_DATE('25-JAN-2007', 'DD-MON-YYYY')
   AND registration_date < TO_DATE('26-JAN-2007', 'DM-MON-YYYY');
GRANT SELECT ON current_regs TO registrar;
```

3. Create a view called roster that reflects all students taught by the instructor Marilyn Frantzen. Grant SELECT privileges on the new view to the INSTRUCTOR role.

Solution:

```
CREATE OR REPLACE VIEW roster AS
SELECT se.course_no course, se.section_id section,
       s.first_name first, s.last_name last,
       e.student_id
  FROM student s, enrollment e, section se, instructor i
 WHERE s.student_id = e.student_id
   AND e.section_id = se.section_id
   AND se.instructor_id = i.instructor_id
   AND i.first_name = 'Marilyn'
   AND i.last_name = 'Frantzen';

GRANT SELECT ON roster TO instructor;
```

4. Grant the REGISTRAR and INSTRUCTOR roles to the new user called SCHOOL.

Solution:

```
GRANT registrar, instructor TO school;
```

5. Log in as the user SCHOOL and select from the two previously created views.

Solution:

```
CONNECT school/program

SELECT *
  FROM student.current_regs;

SELECT *
  FROM student.roster;
```

Chapter 16: Regular Expressions and Hierarchical Queries

1. Name other hierarchical relationships you are familiar with.

Solution: Examples of hierarchical relationships are a parts explosion, also referred to as bill of materials, where you show all the parts that go into the assembly of a final product. Another example is the hierarchy of an organization, showing all the employees and their respective managers. A financial profit and loss statement report can be another example of a tree, where summary accounts are made up of other summary accounts that finally result in posting-level accounts.

2. Change the prerequisite of course number 310 Operating Systems, a root row in the hierarchy, from a null value to 145 Internet Protocols. Write the query to detect the loop in the hierarchy, using the CONNECT_BY_ISCYCLE pseudocolumn.

Solution: Without the change, the hierarchy for course number 310 looks like this:

```
310  Operating Systems
   130  Intro to Unix
      132  Basics of Unix Admin
         134  Advanced Unix Admin
            135  Unix Tips and Techniques
      330  Network Administration
   145  Internet Protocols
```

The UPDATE statement will create a loop in the hierarchy.

```
UPDATE course
   SET prerequisite = 145
 WHERE course_no = 310
```

Essentially, the CONNECT_BY_ISCYCLE returns the value of 1 if a row has a child which is its own ancestor. The next query detects the loop.

```
SELECT *
 FROM (SELECT course_no, prerequisite,
              SYS_CONNECT_BY_PATH(course_no, '/') AS
"Path",
              LEVEL, CONNECT_BY_ISCYCLE AS cycle
         FROM course
       CONNECT BY NOCYCLE PRIOR course_no = prerequisite)
 WHERE cycle = 1
COURSE_NO PREREQUISITE Path            LEVEL CYCLE
--------- ------------ --------------- ----- -----
      145          310 /310/145            2     1
```

```
      310          145 /145/310            2     1

2 rows selected.
```

If the loop is buried deeper in the hierarchy, your query will return multiple rows that all indicate the loop being caused. For simple checking of the hierarchy, it can be helpful to display the course description such as SYS_CONNECT_BY_PATH(description, '*').

Be sure to reset the data back to its original state with this statement.

```
UPDATE course
   SET prerequisite = NULL
 WHERE course_no = 310

COMMIT
```

3. Why doesn't this query return any rows?

```
SELECT *
  FROM instructor
 WHERE REGEXP_LIKE(instructor_id, '[:digit:]')

no rows selected
```

Solution: The REGEXPR_LIKE function does not recognize this pattern as a character class [:digit:] because it is not enclosed with an extra set of square brackets for the character class list.

4. Add a Social Security number column to the STUDENT table or create a separate table with this column. Write a column check constraint that verifies that the social security number is entered in the correct ###-##-#### format.

Solution:

```
ALTER TABLE student
  ADD (ssn VARCHAR2(11))

ALTER TABLE student
    ADD CONSTRAINT stud_ssn_ck CHECK
    (REGEXP_LIKE(ssn,
    '^([[:digit:]]{3}-[[:digit:]]{2}-[[:digit:]]{4})$'))
```

Chapter 17: Exploring Data Warehousing Features

1. Write the question for the following query and answer.

```
SELECT COUNT(DECODE(SIGN(total_capacity-20),
-1, 1, 0, 1)) "<=20",
COUNT(DECODE(SIGN(total_capacity-21),
0, 1, -1, NULL,
DECODE(SIGN(total_capacity-30), -1, 1)))
"21-30",
COUNT(DECODE(SIGN(total_capacity-30), 1, 1)) "31+"
FROM (SELECT SUM(capacity) total_capacity, course_no
FROM SECTION
GROUP BY COURSE_NO)
```

```
<=20          21-30       31+
--------- --------- ---------
        2        10        16

1 row selected.
```

Solution: The question should be similar to one of the following: Determine the total capacity for each course and order them in three columns. List the number of courses with a total capacity of 20 or less in one column, the number of courses with a total capacity between 21 and 30 in another, and lastly show the number of courses with a capacity of over 31 in the third column. The result shows that there are two courses with a total capacity of 20 or less, 10 courses with a capacity between 21 and 30, and 16 courses with a capacity over 31 students.

2. Using an analytical function, determine the top three zip codes where most of the students live.

Solution:

```
SELECT *
FROM (SELECT zip, COUNT(*),
             DENSE_RANK() OVER(ORDER BY
             COUNT(zip) DESC) AS rank
        FROM student
       GROUP BY zip)
WHERE rank <=3
```

```
ZIP     COUNT(*)        RANK
----- ---------- ----------
07024          9          1
```

```
07010          6          2
11373          6          2
11368          6          2
07042          5          3
11355          5          3
11209          5          3
07047          5          3
11375          5          3
11372          5          3

10 rows selected.
```

3. Explain the result of the following query.

```
SELECT 'Q'||TO_CHAR(start_date_time, 'Q') qtr,
       TO_CHAR(start_date_time, 'DY') day, COUNT(*),
       DENSE_RANK() OVER (
         PARTITION BY 'Q'||TO_CHAR(start_date_time, 'Q')
         ORDER BY COUNT(*) DESC) rank_qtr,
       DENSE_RANK() OVER (ORDER BY COUNT(*) DESC) rank_all
  FROM enrollment e, section s
 WHERE s.section_id = e.section_id
 GROUP BY 'Q'||TO_CHAR(start_date_time, 'Q'),
       TO_CHAR(start_date_time, 'DY')
 ORDER BY 1, 4
```

```
QT  DAY COUNT(*)   RANK_QTR   RANK_ALL
--- --- ---------- -------- ----------
Q2  MON         42        1          1
Q2  TUE         35        2          2
Q2  SAT         30        3          3
Q2  SUN         29        4          4
Q2  WED         15        5          6
Q2  FRI         13        6          7
Q2  THU         13        6          7
Q3  SAT         29        1          4
Q3  TUE         20        2          5

9 rows selected
```

Solution: The query generates a listing that shows the starting quarter, day of the week of any sections and within the respective quarter the number of enrollments. The RANK_QTR column indicates ranking of the enrollment number of each quarter and the RANK_ALL column shows the ranking for all time periods.

Chapter 18: SQL Optimization

1. Given the following execution plan, describe the steps and their order of execution.

```
SELECT c.course_no, c.description,
       i.instructor_id
  FROM course c, section s, instructor i
 WHERE prerequisite = 30
   AND c.course_no = s.course_no
   AND s.instructor_id = i.instructor_id
-----------------------------------------------------
| Id | Operation                     | Name           |
-----------------------------------------------------
|  0 | SELECT STATEMENT              |                |
|  1 |  NESTED LOOPS                 |                |
|  2 |   NESTED LOOPS                |                |
|  3 |    TABLE ACCESS BY INDEX ROWID| COURSE         |
|  4 |     INDEX RANGE SCAN          | CRSE_CRSE_FK_I |
|  5 |    TABLE ACCESS BY INDEX ROWID| SECTION        |
|  6 |     INDEX RANGE SCAN          | SECT_CRSE_FK_I |
|  7 |   INDEX UNIQUE SCAN           | INST_PK        |
-----------------------------------------------------
```

Solution: This is a three-table join of the COURSE, SECTION, and INSTRUCTOR tables. The first step executed in the execution plan is the access of the index CRSE_CRSE_FK_I. This index is based on the PREREQUISITE column and, therefore, retrieves the ROWIDs of those records that satisfy the condition WHERE prerequisite = 30. Then the rows with these ROWIDs are retrieved from the COURSE table.
The next step is a nested loop join with the SECTION table. For each of the retrieved COURSE rows, the index SECT_CRSE_FK_I is probed based on join criteria of s.course_no = c.course_no. Lastly, this result is now used for another nested loop join with the INSTRUCTOR index. Note that the query only requires the use of the INSTRUCTOR_ID column, which is also the only column in the SELECT list. Therefore, only a lookup of the value in the index is required, not the INSTRUCTOR table.

2. Describe the steps of the following execution plan.

```
UPDATE enrollment e
   SET final_grade =
       (SELECT NVL(AVG(numeric_grade),0)
```

```
          FROM grade
         WHERE e.student_id = student_id
           AND e.section_id = section_id)
 WHERE student_id = 1000
   AND section_id = 2000
0 rows updated.

-----------------------------------------------
| Id  | Operation                     | Name       |
-----------------------------------------------
|   0 | UPDATE STATEMENT              |            |
|   1 |  UPDATE                       | ENROLLMENT |
|   2 |   INDEX UNIQUE SCAN           | ENR_PK     |
|   3 |   SORT AGGREGATE              |            |
|   4 |    TABLE ACCESS BY INDEX ROWID| GRADE      |
|   5 |     INDEX RANGE SCAN          | GR_PK      |
-----------------------------------------------
```

Solution: This SQL UPDATE statement is a correlated subquery. You can generate explain plans for SQL statements other than SELECT statements. The explain plan shows that the WHERE clause of the UPDATE statement refers to the primary key columns of the ENROLLMENT and GRADE tables to identify the rows and to determine the final grade values.

The inner query utilizes the index GR_PK, the primary key index, and accesses the GRADE table via the ROWID. Because the subquery specifies the AVG function, the step SORT (AGGREGATE) is executed. Note, you will not see a difference in the execution plan between a correlated UPDATE statement and an UPDATE statement with a non-correlated subquery. You need to keep in mind, however, that the correlated update will repeatedly execute the inner query for every row retrieved by the outer query. This is in contrast to the non-correlated subquery that executes the inner query only once. The correlated subquery combined with an UPDATE statement is a very fast way to update data without having to write a program to compute records for each step.

3. The following SQL statement has an error in the hint. Correct the statement so Oracle can use the hint.

```
SELECT /*+ INDEX (student stu_pk) */ *
  FROM student s
 WHERE last_name = 'Smith'
```

Solution: The hint does not specify the table alias. When a table alias is used in the statement, the hint needs to reference the alias

otherwise the hint is ignored. You correct the statement as follows. Note that this is not a very good index choice for this query but just illustrates how you can direct Oracle to use a specific index.

```
SELECT /*+ INDEX (s stu_pk) */ *
  FROM student s
 WHERE last_name = 'Smith'
```