Practice 1: Solutions

To complete question 6 and the subsequent questions, you need to connect to the database using *i*SQL*Plus. To do this, launch the Internet Explorer browser from the desktop of your client. Enter the URL in the *http://machinename:5560/isqlplus/* format and use the *oraxx* account and corresponding *password* and *service identifier* (in the *Tx* format) provided by your instructor to log on to the database.

1. What privilege should a user be given to log on to the Oracle server? Is this a system or an object privilege?
   The `CREATE SESSION` system privilege

2. What privilege should a user be given to create tables?
   The `CREATE TABLE` privilege

3. If you create a table, who can pass along privileges to other users on your table?
   You can, or anyone you have given those privileges to, by using the `WITH GRANT OPTION`

4. You are the DBA. You are creating many users who require the same system privileges. What should you use to make your job easier?
   Create a role containing the system privileges and grant the role to the users.

5. What command do you use to change your password?
   The `ALTER USER` statement

6. Grant another user access to your `DEPARTMENTS` table. Have the user grant you query access to his or her `DEPARTMENTS` table.

   Team 2 executes the GRANT statement.

```
GRANT select
ON    departments
TO    <user1>;
```

   Team 1 executes the GRANT statement.

```
GRANT select
ON    departments
TO    <user2>;
```

   Here, *user1* is the name of Team 1 and *user2* is the name of Team 2.

7. Query all the rows in your `DEPARTMENTS` table.

```
SELECT  *
FROM    departments;
```

Practice 1: Solutions (continued)

8. Add a new row to your DEPARTMENTS table. Team 1 should add Education as department number 500. Team 2 should add Human Resources as department number 510. Query the other team's table.

```
Team 1 executes this INSERT statement.
 INSERT INTO departments(department_id, department_name)
 VALUES (500, 'Education');
 COMMIT;
Team 2 executes this INSERT statement.
 INSERT INTO departments(department_id, department_name)
 VALUES (510, 'Human Resources');
 COMMIT;
```

9. Create a synonym for the other team's DEPARTMENTS table.

```
Team 1 creates a synonym named team2.
  CREATE SYNONYM  team2
  FOR <oraxx>.DEPARTMENTS;
Team 2 creates a synonym named team1.
  CREATE SYNONYM  team1
  FOR <oraxx>. DEPARTMENTS;
```

10. Query all the rows in the other team's DEPARTMENTS table by using your synonym.

```
Team 1 executes this SELECT statement.
  SELECT  *
  FROM    team2;
Team 2 executes this SELECT statement.
  SELECT  *
  FROM    team1;
```

Practice 1: Solutions (continued)

11. Query the USER_TABLES data dictionary to see information about the tables that you own.

```
SELECT  table_name
  FROM    user_tables;
```

12. Query the ALL_TABLES data dictionary view to see information about all the tables that you can access. Exclude tables that you own.

```
SELECT  table_name, owner
  FROM    all_tables
  WHERE   owner  <> 'Oraxx';
```

13. Revoke the SELECT privilege from the other team.

```
Team 1 revokes the privilege.
  REVOKE select
    ON     departments
    FROM   <oraxx>;
Team 2 revokes the privilege.
  REVOKE select
    ON     departments
    FROM   <oraxx>;
```

14. Remove the row you inserted into the DEPARTMENTS table in step 8 and save the changes.

```
Team 1 executes this DELETE statement.
   DELETE FROM departments
   WHERE department_id = 500;
   COMMIT;
Team 2 executes this DELETE statement.
   DELETE FROM departments
   WHERE department_id = 510;
   COMMIT;
```

Practice 2: Solutions

1. Create the DEPT2 table based on the following table instance chart. Place the syntax in a script called lab_02_01.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	NAME
Key Type		
Nulls/Unique		
FK Table		
FK Column		
Data type	NUMBER	VARCHAR2
Length	7	25

```
CREATE TABLE dept2
 (id NUMBER(7),
  name VARCHAR2(25));

DESCRIBE dept2
```

2. Populate the DEPT2 table with data from the DEPARTMENTS table. Include only the columns that you need.

```
INSERT INTO dept2
SELECT  department_id, department_name
FROM    departments;
```

3. Create the EMP2 table based on the following table instance chart. Place the syntax in a script called lab_02_03.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

```
CREATE TABLE  emp2
(id           NUMBER(7),
 last_name     VARCHAR2(25),
 first_name    VARCHAR2(25),
 dept_id       NUMBER(7));

DESCRIBE emp2
```

4. Modify the EMP2 table to allow for longer employee last names. Confirm your modification.

```
ALTER TABLE emp2
MODIFY (last_name   VARCHAR2(50));

DESCRIBE emp2
```

5. Confirm that both the DEPT2 and EMP2 tables are stored in the data dictionary. (**Hint:** USER_TABLES)

```
SELECT   table_name
FROM     user_tables
WHERE    table_name IN ('DEPT2', 'EMP2');
```

6. 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;
```

7. Drop the EMP2 table.

```
DROP TABLE emp2;
```

Practice 2: Solutions (continued)

8. Query the recycle bin to see whether the table is present.

```
SELECT original_name, operation, droptime
FROM recyclebin;
```

9. Undrop the EMP2 table.

```
FLASHBACK TABLE emp2 TO BEFORE DROP;
DESC emp2;
```

10. Drop the FIRST_NAME column from the EMPLOYEES2 table. Confirm your modification by checking the description of the table.

```
ALTER TABLE employees2
DROP COLUMN first_name;

DESCRIBE employees2
```

11. In the EMPLOYEES2 table, mark the DEPT_ID column as UNUSED. Confirm your modification by checking the description of the table.

```
ALTER TABLE employees2
SET   UNUSED (dept_id);

DESCRIBE employees2
```

12. Drop all the UNUSED columns from the EMPLOYEES2 table. Confirm your modification by checking the description of the table.

```
ALTER TABLE employees2
DROP UNUSED COLUMNS;

DESCRIBE employees2
```

Practice 2: Solutions (continued)

13. Add a table-level PRIMARY KEY constraint to the EMP2 table on the ID column. The constraint should be named at creation. Name the constraint my_emp_id_pk.

```
ALTER TABLE    emp2
ADD CONSTRAINT my_emp_id_pk PRIMARY KEY (id);
```

14. Create a PRIMARY KEY constraint to the DEPT2 table using the ID column. The constraint should be named at creation. Name the constraint my_dept_id_pk.

```
ALTER TABLE    dept2
ADD CONSTRAINT my_dept_id_pk PRIMARY KEY(id);
```

15. Add a foreign key reference on the EMP2 table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my_emp_dept_id_fk.

```
ALTER TABLE emp2
ADD CONSTRAINT my_emp_dept_id_fk
FOREIGN KEY (dept_id) REFERENCES dept2(id);
```

16. Confirm that the constraints were added by querying the USER_CONSTRAINTS view. Note the types and names of the constraints.

```
SELECT   constraint_name, constraint_type
FROM     user_constraints
WHERE    table_name IN ('EMP2', 'DEPT2');
```

17. Display the object names and types from the USER_OBJECTS data dictionary view for the EMP2 and DEPT2 tables. Notice that the new tables and a new index were created.

```
SELECT   object_name, object_type
FROM     user_objects
WHERE    object_name LIKE 'EMP%'
OR       object_name LIKE 'DEPT%';
```

If you have time, complete the following exercise:

18. Modify the EMP2 table. Add a COMMISSION column of the NUMBER data type, precision 2, scale 2. Add a constraint to the COMMISSION column that ensures that a commission value is greater than zero.

```
ALTER TABLE emp2
ADD commission NUMBER(2,2)
CONSTRAINT my_emp_comm_ck CHECK (commission > 0);
```

Practice 2: Solutions (continued)

19. Drop the EMP2 and DEPT2 tables so that they cannot be restored. Verify the recycle bin.

```
DROP TABLE emp2 PURGE;
DROP TABLE dept2 PURGE;

SELECT original_name, operation, droptime
FROM recyclebin;
```

20. Create the DEPT_NAMED_INDEX table based on the following table instance chart. Name the index for the PRIMARY KEY column as DEPT_PK_IDX.

Column Name	Deptno	Dname
Primary Key	Yes	
Data Type	Number	VARCHAR2
Length	4	30

```
CREATE TABLE DEPT_NAMED_INDEX
 (deptno NUMBER(4)
 PRIMARY KEY USING INDEX
 (CREATE INDEX dept_pk_idx ON
  DEPT_NAMED_INDEX(deptno)),
  dname VARCHAR2(30));
```

Practice 3: Solutions

1. Run the `lab_03_01.sql` script in the lab folder to create the `SAL_HISTORY` table.

2. Display the structure of the `SAL_HISTORY` table.

```
DESC sal_history
```

3. Run the `lab_03_03.sql` script in the lab folder to create the `MGR_HISTORY` table.

4. Display the structure of the `MGR_HISTORY` table.

```
DESC mgr_history
```

5. Run the `lab_03_05.sql` script in the lab folder to create the `SPECIAL_SAL` table.

6. Display the structure of the `SPECIAL_SAL` table.

```
DESC special_sal
```

7. a. Write a query to do the following:
   - Retrieve the details of the employee ID, hire date, salary, and manager ID of those employees whose employee ID is less than 125 from the `EMPLOYEES` table.
   - If the salary is more than $20,000, insert the details of employee ID and salary into the `SPECIAL_SAL` table.
   - Insert the details of the employee ID, hire date, and salary into the `SAL_HISTORY` table.
   - Insert the details of the employee ID, manager ID, and salary into the `MGR_HISTORY` table.

```
INSERT ALL
WHEN SAL > 20000 THEN
INTO  special_sal VALUES (EMPID, SAL)
ELSE
INTO sal_history VALUES(EMPID,HIREDATE,SAL)
INTO mgr_history VALUES(EMPID,MGR,SAL)
SELECT employee_id EMPID, hire_date HIREDATE,
```

```
salary SAL, manager_id MGR
FROM employees
WHERE employee_id < 125;
```

b. Display the records from the SPECIAL_SAL table.

```
SELECT * FROM  special_sal;
```

c. Display the records from the SAL_HISTORY table.

```
SELECT * FROM  sal_history;
```

d. Display the records from the MGR_HISTORY table.

```
SELECT * FROM mgr_history;
```

8. a. Run the lab_03_08a.sql script in the lab folder to create the SALES_SOURCE_DATA table.

b. Run the lab_03_08b.sql script in the lab folder to insert records into the SALES_SOURCE_DATA table.

c. Display the structure of the SALES_SOURCE_DATA table.

```
DESC sales_source_data
```

d. Display the records from the SALES_SOURCE_DATA table.

```
SELECT * FROM SALES_SOURCE_DATA;
```

e. Run the lab_03_08c.sql script in the lab folder to create the SALES_INFO table.

f. Display the structure of the SALES_INFO table.

```
DESC sales_info
```

g. Write a query to do the following:

- Retrieve the details of the employee ID, week ID, sales on Monday, sales on Tuesday, sales on Wednesday, sales on Thursday, and sales on Friday from the SALES_SOURCE_DATA table.

- Build a transformation such that each record retrieved from the SALES_SOURCE_DATA table is converted into multiple records for the SALES_INFO table.

  Hint: Use a pivoting INSERT statement.

```
INSERT ALL
  INTO sales_info VALUES (employee_id, week_id, sales_MON)
  INTO sales_info VALUES (employee_id, week_id, sales_TUE)
  INTO sales_info VALUES (employee_id, week_id, sales_WED)
  INTO sales_info VALUES (employee_id, week_id,
        sales_THUR)
  INTO sales_info VALUES (employee_id, week_id, sales_FRI)
  SELECT EMPLOYEE_ID, week_id, sales_MON, sales_TUE,
  sales_WED, sales_THUR,sales_FRI  FROM sales_source_data;
```

h. Display the records from the SALES_INFO table.

```
SELECT * FROM sales_info;
```

9. You have the data of past employees stored in a flat file called emp.data. You want to store the names and e-mail IDs of all employees past and present in a table. To do this, first create an external table called EMP_DATA using the emp.dat source file in the emp_dir directory. You can use the script in lab_03_09.sql to do this.

```
CREATE TABLE emp_data
  (first_name  VARCHAR2(20)
  ,last_name   VARCHAR2(20)
  , email     VARCHAR2(30)
  )
ORGANIZATION EXTERNAL
(
 TYPE oracle_loader
 DEFAULT DIRECTORY emp_dir
 ACCESS PARAMETERS
 (
  RECORDS DELIMITED BY NEWLINE CHARACTERSET US7ASCII
  NOBADFILE
  NOLOGFILE
  FIELDS
  ( first_name POSITION ( 1:20) CHAR
  , last_name POSITION (22:41) CHAR
  ,  email   POSITION (43:72) CHAR )
 )
 LOCATION ('emp.dat') ) ;
```

10. Next, run the `lab_03_10.sql` script to create the `EMP_HIST` table.
    a. Increase the size of the e-mail column to 45.
    b. Merge the data in the `EMP_DATA` table created in the last lab into the data in the `EMP_HIST` table. Assume that the data in the external `EMP_DATA` table is the most up-to-date. If a row in the `EMP_DATA` table matches the EMP_HIST table, update the e-mail column of the `EMP_HIST` table to match the `EMP_DATA` table row. If a row in the `EMP_DATA` table does not match, insert it into the `EMP_HIST` table. Rows are considered matching when the employee's first and last names are identical.

```
MERGE INTO EMP_HIST f USING EMP_DATA h
 ON (f.first_name = h.first_name
 AND f.last_name = h.last_name)
WHEN MATCHED THEN
 UPDATE SET f.email = h.email
WHEN NOT MATCHED THEN
 INSERT (f.first_name
    , f.last_name
    , f.email)
 VALUES (h.first_name
    , h.last_name
    , h.email);
```

c. Retrieve the rows from `EMP_HIST` after the merge.

```
SELECT * FROM emp_hist;
```

11. Create the `EMP3` table using the `lab_03_11.sql` script. In the `EMP3` table, change the department for Kochhar to 60 and commit your change. Next, change the department for Kochhar to 50 and commit your change. Track the changes to Kochhar using the Row Versions feature.

```
UPDATE emp3 SET department_id = 60
WHERE last_name = 'Kochchar';
COMMIT;
UPDATE emp3 SET department_id = 50
WHERE last_name = 'Kochchar';
COMMIT;
```

```
SELECT VERSIONS_STARTTIME "START_DATE",
   VERSIONS_ENDTIME "END_DATE",  DEPARTMENT_ID
FROM EMP3
   VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE LAST_NAME ='Kochhar';
```

Practice 4: Solutions

1. Write a query to display the following for those employees whose manager ID is less than 120:
   - Manager ID
   - Job ID and total salary for every job ID for employees who report to the same manager
   - Total salary of those managers
   - Total salary of those managers, irrespective of the job IDs

```
SELECT manager_id,job_id,sum(salary)
FROM   employees
WHERE manager_id < 120
GROUP BY ROLLUP(manager_id,job_id);
```

2. Observe the output from question 1. Write a query using the `GROUPING` function to determine whether the `NULL` values in the columns corresponding to the `GROUP BY` expressions are caused by the `ROLLUP` operation.

```
SELECT manager_id MGR ,job_id JOB,
 sum(salary),GROUPING(manager_id),GROUPING(job_id)
 FROM   employees
 WHERE manager_id < 120
 GROUP BY ROLLUP(manager_id,job_id);
```

3. Write a query to display the following for those employees whose manager ID is less than 120:
   - Manager ID
   - Job and total salaries for every job for employees who report to the same manager
   - Total salary of those managers
   - Cross-tabulation values to display the total salary for every job, irrespective of the manager
   - Total salary irrespective of all job titles

```
SELECT manager_id, job_id, sum(salary)
FROM   employees
WHERE manager_id < 120
GROUP BY CUBE(manager_id, job_id);
```

Practice 4: Solutions (continued)

4. Observe the output from question 3. Write a query using the GROUPING function to determine whether the NULL values in the columns corresponding to the GROUP BY expressions are caused by the CUBE operation.

```
SELECT manager_id MGR ,job_id JOB,
sum(salary),GROUPING(manager_id),GROUPING(job_id)
FROM   employees
WHERE manager_id < 120
GROUP BY CUBE(manager_id,job_id);
```

5. Using GROUPING SETS, write a query to display the following groupings:
   - department_id, manager_id, job_id
   - department_id, job_id
   - manager_id, job_id

   The query should calculate the sum of the salaries for each of these groups.

```
SELECT department_id, manager_id, job_id, SUM(salary)
FROM employees
GROUP BY
GROUPING SETS ((department_id, manager_id, job_id),
(department_id, job_id),(manager_id,job_id));
```

Practice 5: Solutions

1. Alter the session to set the NLS_DATE_FORMAT to DD-MON-YYYY HH24:MI:SS.

```
ALTER SESSION SET NLS_DATE_FORMAT =
'DD-MON-YYYY HH24:MI:SS';
```

2. a. Write queries to display the time zone offsets (TZ_OFFSET) for the following time zones.

US/Pacific-New

```
SELECT TZ_OFFSET ('US/Pacific-New') from dual;
```

Singapore

```
SELECT TZ_OFFSET ('Singapore') from dual;
```

Egypt

```
SELECT TZ_OFFSET ('Egypt') from dual;
```

b. Alter the session to set the TIME_ZONE parameter value to the time zone offset of US/Pacific-New.

```
ALTER SESSION SET TIME_ZONE = '-7:00';
```

c. Display the CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP for this session.
Note: The output may be different based on the date when the command is executed.

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP,
   LOCALTIMESTAMP FROM DUAL;
```

d. Alter the session to set the TIME_ZONE parameter value to the time zone offset of Singapore.

```
ALTER SESSION SET TIME_ZONE = '+8:00';
```

e. Display the CURRENT_DATE, CURRENT_TIMESTAMP, LOCALTIMESTAMP for this session.
Note: The output might be different, based on the date when the command is executed.

```
SELECT CURRENT_DATE, CURRENT_TIMESTAMP,
LOCALTIMESTAMP FROM DUAL;
```

Practice 5: Solutions (continued)

Note: Observe in the preceding practice that CURRENT_DATE, CURRENT_TIMESTAMP, and LOCALTIMESTAMP are all sensitive to the session time zone.

3. Write a query to display the DBTIMEZONE and SESSIONTIMEZONE.

```
SELECT DBTIMEZONE,SESSIONTIMEZONE
FROM DUAL;
```

4. Write a query to extract the YEAR from the HIRE_DATE column of the EMPLOYEES table for those employees who work in department 80.

```
SELECT last_name, EXTRACT (YEAR FROM HIRE_DATE)
FROM employees
WHERE department_id = 80;
```

5. Alter the session to set NLS_DATE_FORMAT to DD-MON-YYYY.

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY';
```

6. Examine and run the lab_05_06.sql script to create the SAMPLE_DATES table and populate it.

   a. Select from the table and view the data.

```
SELECT * FROM sample_dates;
```

   b. Modify the data type of the DATE_COL column and change it to TIMESTAMP. Select from the table to view the data.

```
ALTER TABLE sample_dates MODIFY date_col TIMESTAMP;
SELECT * FROM sample_dates;
```

   c. Try to modify the data type of the DATE_COL column and change it to TIMESTAMP WITH TIME ZONE. What happens?

```
ALTER TABLE sample_dates MODIFY date_col
TIMESTAMP WITH TIME ZONE;
```

You are unable to change the data type of the DATE_COL column as the Oracle server does not permit you to convert from TIMESTAMP to TIMESTAMP WITH TIMEZONE by using the ALTER statement.

Practice 5: Solutions (continued)

7. Create a query to retrieve last names from the EMPLOYEES table and calculate the review status. If the year hired was 1998, then display Needs Review for the review status; otherwise, display not this year! Name the review status column Review. Sort the results by the HIRE_DATE column.
   Hint: Use a CASE expression with the EXTRACT function to calculate the review status.

```
SELECT e.last_name
,     (CASE extract(year from e.hire_date)
       WHEN 1998 THEN 'Needs Review'
       ELSE 'not this year!'
      END )           AS "Review "
FROM   employees e
ORDER BY e.hire_date;
```

8. Create a query to print the last names and the number of years of service for each employee. If the employee has been employed five or more years, then print 5 years of service. If the employee has been employed 10 or more years, then print 10 years of service. If the employee has been employed 15 or more years, then print 15 years of service. If none of these conditions match, then print maybe next year! Sort the results by the HIRE_DATE column. Use the EMPLOYEES table.
   Hint: Use CASE expressions and TO_YMINTERVAL.

```
SELECT e.last_name, hire_date, sysdate,
       (CASE
        WHEN  (sysdate -TO_YMINTERVAL('15-0'))>=
            hire_date THEN    '15 years of service'
        WHEN (sysdate -TO_YMINTERVAL('10-0'))>=
hire_date
          THEN '10 years of service'
        WHEN (sysdate - TO_YMINTERVAL('5-0'))>=
hire_date
           THEN '5 years of service'
             ELSE 'maybe next year!'
          END) AS "Awards"
FROM   employees e;
```

Practice 6: Solutions

1. Write a query to display the last name, department number, and salary of any employee whose department number and salary both match the department number and salary of any employee who earns a commission.

```
SELECT last_name, department_id, salary
FROM   employees
WHERE  (salary, department_id) IN
       (SELECT  salary, department_id
        FROM    employees
       WHERE   commission_pct IS NOT NULL);
```

2. Display the last name, department name, and salary of any employee whose salary and commission match the salary and commission of any employee located in location ID1700.

```
SELECT e.last_name, d.department_name, e.salary
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND   (salary, NVL(commission_pct,0)) IN
            (SELECT salary, NVL(commission_pct,0)
             FROM   employees e, departments  d
             WHERE  e.department_id = d.department_id
             AND    d.location_id = 1700);
```

3. Create a query to display the last name, hire date, and salary for all employees who have the same salary and commission as Kochhar.

   Note: Do not display Kochhar in the result set.

```
SELECT last_name, hire_date, salary
FROM   employees
WHERE  (salary, NVL(commission_pct,0)) IN
             (SELECT salary, NVL(commission_pct,0)
              FROM       employees
              WHERE  last_name = 'Kochhar')
AND last_name != 'Kochhar';
```

Practice 6: Solutions (continued)

4. Create a query to display the employees who earn a salary that is higher than the salary of all the sales managers (`JOB_ID = 'SA_MAN'`). Sort the results on salary from the highest to the lowest.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  salary > ALL
              (SELECT salary
               FROM   employees
WHERE  job_id = 'SA_MAN')
ORDER BY salary DESC;
```

5. Display the details of the employee ID, last name, and department ID of those employees who live in cities whose name begins with *T*.

```
SELECT employee_id, last_name, department_id
FROM   employees
WHERE  department_id IN (SELECT department_id
                         FROM departments
                         WHERE location_id IN
                               (SELECT  location_id
                                FROM locations
                                WHERE city LIKE 'T%'));
```

6. Write a query to find all employees who earn more than the average salary in their departments. Display last name, salary, department ID, and the average salary for the department. Sort by average salary. Use aliases for the columns retrieved by the query as shown in the sample output.

```
SELECT e.last_name ename, e.salary salary,
       e.department_id deptno, AVG(a.salary) dept_avg
FROM   employees e, employees a
WHERE  e.department_id = a.department_id
AND    e.salary > (SELECT AVG(salary)
             FROM   employees
             WHERE  department_id = e.department_id )
GROUP BY e.last_name, e.salary, e.department_id
ORDER BY AVG(a.salary);
```

Practice 6: Solutions (continued)

7. Find all employees who are not supervisors.
   a. First, do this by using the NOT EXISTS operator.

```
SELECT outer.last_name
FROM    employees outer
WHERE  NOT EXISTS (SELECT 'X'
                   FROM employees inner
                    WHERE inner.manager_id =
                          outer.employee_id);
```

   b. Can this be done by using the NOT IN operator? How, or why not?

```
SELECT outer.last_name
FROM   employees outer
WHERE  outer.employee_id
NOT IN (SELECT inner.manager_id
        FROM   employees inner);
```

This alternative solution is not a good one. The subquery picks up a NULL value, so the entire query returns no rows. The reason is that all conditions that compare a NULL value result in NULL. Whenever NULL values are likely to be part of the value set, *do not* use NOT IN as a substitute for NOT EXISTS.

8. Write a query to display the last names of the employees who earn less than the average salary in their departments.

```
SELECT last_name
FROM    employees outer
WHERE outer.salary < (SELECT AVG(inner.salary)
                      FROM employees inner
                      WHERE inner.department_id
                            = outer.department_id);
```

Practice 6: Solutions (continued)

9. Write a query to display the last names of employees who have one or more coworkers in their departments with later hire dates but higher salaries.

```
SELECT  last_name
FROM    employees outer
WHERE EXISTS (SELECT 'X'
              FROM employees inner
              WHERE inner.department_id =
                    outer.department_id
              AND inner.hire_date > outer.hire_date
              AND inner.salary > outer.salary);
```

10. Write a query to display the employee ID, last names, and department names of all employees.

    Note: Use a scalar subquery to retrieve the department name in the `SELECT` statement.

```
 SELECT employee_id, last_name,
       (SELECT department_name
        FROM departments d
        WHERE   e.department_id =
           d.department_id ) department
FROM employees e
ORDER BY department;
```

11. Write a query to display the department names of those departments whose total salary cost is above one-eighth (1/8) of the total salary cost of the whole company. Use the `WITH` clause to write this query. Name the query `SUMMARY`.

```
WITH
summary AS (
  SELECT d.department_name, SUM(e.salary) AS dept_total
  FROM employees e, departments d
  WHERE e.department_id = d.department_id
  GROUP BY d.department_name)
SELECT department_name, dept_total
FROM summary
WHERE dept_total > ( SELECT SUM(dept_total) * 1/8
                     FROM summary  )
ORDER BY dept_total DESC;
```

Practice 7: Solutions

1. Look at the following output examples. Are these outputs the result of a hierarchical query? Explain why or why not.

   Exhibit 1: This is not a hierarchical query; the report simply has a descending sort on `SALARY`.

   Exhibit 2: This is not a hierarchical query; there are two tables involved.

   Exhibit 3: Yes, this is most definitely a hierarchical query because it displays the tree structure representing the management reporting line from the `EMPLOYEES` table.

2. Produce a report showing an organization chart for Mourgos's department. Print last names, salaries, and department IDs.

```
SELECT last_name, salary, department_id
FROM employees
START WITH last_name = 'Mourgos'
CONNECT BY PRIOR employee_id = manager_id;
```

3. Create a report that shows the hierarchy of the managers for the employee Lorentz. Display his immediate manager first.

```
SELECT last_name
FROM employees
WHERE last_name != 'Lorentz'
START WITH last_name = 'Lorentz'
CONNECT BY PRIOR manager_id = employee_id;
```

4. Create an indented report showing the management hierarchy starting from the employee whose `LAST_NAME` is Kochhar. Print the employee's last name, manager ID, and department ID. Give alias names to the columns as shown in the sample output.

```
COLUMN name FORMAT A20
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-
2,'_')
       name,manager_id mgr, department_id deptno
FROM employees
START WITH last_name = 'Kochhar'
CONNECT BY PRIOR employee_id = manager_id
/
COLUMN name CLEAR
```

Practice 7: Solutions (continued)

If you have time, complete the following exercises:

5. Produce a company organization chart that shows the management hierarchy. Start with the person at the top level, exclude all people with a job ID of `IT_PROG`, and exclude De Haan and those employees who report to De Haan.

```
SELECT last_name,employee_id, manager_id
FROM   employees
WHERE  job_id != 'IT_PROG'
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id
AND last_name != 'De Haan';
```

Practice 8: Solutions

1. Write a query to search the EMPLOYEES table for all employees whose first names start with "Ne" or "Na."

```
SELECT first_name, last_name
FROM employees
WHERE REGEXP_LIKE (first_name, '^N(e|a).');
```

2. Create a query that removes the spaces in the STREET_ADDRESS column of the LOCATIONS table in the display.

```
SELECT regexp_replace (street_address, ' ','')
FROM locations;
```

3. Create a query that displays "St" replaced by "Street" in the STREET_ADDRESS column of the LOCATIONS table. Be careful that you do not affect any rows that already have "Street" in them. Display only those rows, which are affected.

```
SELECT REGEXP_REPLACE (street_address, 'St$',
'Street') FROM locations
WHERE REGEXP_LIKE (street_address, 'St');
```