INNOZANT INFOTECH PVT LTD

SQL PRACTICE ASSIGNMENTS

WITH SOLUTIONS

CHAPTER – 1: BASIC SQL SELECT STATEMENT

True / False

Q.1 *i*SQL*Plus commands access the database.

- **True**
- **False**

Ans: FALSE

Q.2 The following `SELECT` statement executes successfully:

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

- **True**
- **False**

Ans: True

Q.3 The following `SELECT` statement executes successfully:

```
SELECT * FROM job_grades;
```

- **True True**
- **False**

Ans: True

Q.4 There are four coding errors in this statement. Can you identify them?

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

Ans:

1. The EMPLOYEES table does not contain a column called sal. The column is called SALARY.
2. The multiplication operator is *, not x, as shown in line 2.
3. The ANNUAL SALARY alias cannot include spaces. The alias should read ANNUAL_SALARY or be enclosed in double quotation marks.
4. A comma is missing after the column, LAST_NAME.

Q.5 Show the structure of the DEPARTMENTS table. Select all data from the DEPARTMENTS table.

Ans: 8 Rows Selected

```
DESCRIBE departments;
SELECT * FROM departments;
```

Q.6 Show the structure of the EMPLOYEES table. Create a query to display the last name, job code, hire date, and employee number for each employee, with employee number appearing first.

Ans:

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

*Q.*7 Create a query to display unique job codes from the EMPLOYEES table.

Ans: 12 rows selected

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

Q.8 Copy the statement from Ans 6 into the *i*SQL*Plus Edit window.
Name the column headings Emp #, Employee, Job, and Hire Date, respectively.
Run your query again.

Ans:

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

Q.9 Display the last name concatenated with the job ID, separated by a comma and space, and name the column Employee and Title.

Ans:

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

Q.10 Create a query to display all the data from the EMPLOYEES table. Separate each column by a comma. Name the column THE_OUTPUT.

Ans:

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

Practice 2 Solutions

Q.1 Create a query to display the last name and salary of employees earning more than $12,000.

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hartstein	13000

Ans:

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

Save this statement in txt file *p2_1.sql* **and run query.**

Q.2 Create a query to display the employee last name and department number for employee number 176.

Ans:

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

Q.3 .Modify *p2_1.sql* to display the last name and salary for all employees whose salary is not in the range of $5,000 and $12,000.

Place your SQL statement in a text file named p2_3.sql.

Ans:

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

Q.4 Display the employee last name, job ID, and start date of employees hired between February 20, 1998, and May 1, 1998.

Order the query in ascending order by start date.

Ans:

```
SELECT last_name, job_id, hire_date FROM employees
WHERE hire_date BETWEEN '20-Feb-1998' AND '01-May-1998'
ORDER BY hire_date;
```

Q.5 Display the last name and department number of all employees in departments 20 and 50 in alphabetical order by name.

Ans:

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

Q.6 Modify *p2_3.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 `p2_3.sql` as `p2_6.sql`.

. Run the statement in `p2_6.sql`.

Ans:

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

Q.7 Display the last name and hire date of every employee who was hired in 1994.

Ans:

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

Q.8 Display the last name and job title of all employees who do not have a manager.

Ans:

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

Q.9 Display the last name, salary, and commission for all employees who earn commissions. Sort data in descending order of salary and commissions.

Ans:

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

Q.10 Display the last names of all employees where the third letter of the name is an *a*.

Ans:

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

Q.11 Display the last name of all employees who have an *a* and an *e* in their last name.

Ans:

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

Q.12 Display the last name, job, and salary for all employees whose job is sales representative or stock clerk and whose salary is not equal to $2,500, $3,500, or $7,000.

Ans:

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

Q.13 Modify `p2_6.sql` to display the last name, salary, and commission for all employees whose commission amount is 20%.

Resave `p2_6.sql` as `p2_13.sql`. Rerun the statement in `p2_13.sql`.

Ans:

.

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

CHAPTER – 3: Single Row Functions

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

Ans:

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

Q.2 For each employee, display the employee number, last_name, salary, and salary increased by 15% and expressed as a whole number. Label the column New Salary. Place your SQL statement in a text file named p3_2.sql.

Ans:

```
SELECT employee_id,last_name,salary,ROUND(salary * 1.15, 0)
"New Salary" FROM employees;
```

Q.3 Run your query in the file p3_2.sql.

Ans:

```
SELECT employee_id, last_name, salary,
ROUND(salary * 1.15, 0) "New Salary" FROM employees;
```

Q.4 Modify your query p3_2.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 p3_4.sql. Run the revised query.

Ans:

```
SELECT employee_id, last_name, salary,
ROUND(salary * 1.15, 0) "New Salary",
ROUND(salary * 1.15, 0) - salary "Increase"
FROM employees;
```

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

Ans:

```
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;
```

Q.6 For each employee, display the employee's last name, and calculate the number of months between today and the date 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.

Ans: **Note:** Your results will differ.

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

Q.7 Write a query that produces the following for each employee:
<employee last name> earns <salary> monthly but wants <3 times salary>.
Label the column Dream Salaries.

Ans:

```
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;
```

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

Ans:

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

Q.9 Display each employee's last name, hire date, and salary review date, which is the first Monday after six months of service. Label the column REVIEW. Format the dates to appear in the format similar to "Monday, the Thirty-First of July, 2000."

Ans:

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

Q.10 Display the last name, hire date, and day of the week on which the employee started. Label the column DAY.
Order the results by the day of the week starting with Monday.

Ans:

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

Q. 11 Create a query that displays the employees' last names and commission amounts. If an employee does not earn commission, put "No Commission."
Label the column COMM.

Ans:

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

Q.12 Create a query that displays the employees' last names and indicates the amounts of their annual salaries with asterisks. Each asterisk signifies a thousand doll ars. Sort the data in descending order of salary.
Label the column EMPLOYEES_AND_THEIR_SALARIES.

Ans:

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

Q.13 Using the DECODE function, write a query that displays the grade of all employees based on the value of the column JOB_ID, as per 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
```

Ans:

```
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;
```

Q.14 Rewrite the statement in the preceding question using the CASE syntax.

Ans:

```
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;
```

CHAPTER-4: Displaying Data From Multiple Tables

1. Write a query to display the last name, department number, and department name for all employees.

Ans:

```
SELECT e.last_name, e.department_id, d.department_name
FROM employees e, departments d
WHERE e.department_id = d.department_id;
```

2. Create a unique listing of all jobs that are in department 30. Include the location of department 90 in the output.

Ans:

```
SELECT DISTINCT job_id, location_id
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND employees.department_id = 80;
```

3. Write a query to display the employee last name, department name, location ID, and city of all employees who earn a commission.

Ans:

```
SELECT e.last_name, d.department_name, d.location_id, l.c ity
FROM employees e, departments d, locations l
WHERE e.department_id = d.department_id AND
d.location_id = l.location_id
AND e.commission_pct IS NOT NULL;
```


4. Display the employee last name and department name for all employees who have an a (lowercase) in their last names. Place your SQL statement in a text file named p4_4.sql.

Ans:

```
SELECT last_name, department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id
AND last_name LIKE '%a%';
```

5. Write a query to display the last name, job, department number, and department name for all employees who work in Toronto.

Ans:

```
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';
```

6. Display the employee last name and employee number along with their manager's last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively.
Place your SQL statement in a text file named p4_6.sql.
Ans:

```
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);
```

7. Modify p4_6.sql to display all employees including King, who has no manager.
Place your SQL statement in a text file named p4_7.sql.
Run the query in p4_7.sql
Ans:

```
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);
```

8. Create a query 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.
Ans:

```
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;
```

9. Show the structure of the JOB_GRADES table.
Create a query that displays the name, job, department name, salary, and grade for all employees.
Ans:

```
DESC JOB_GRADES;
SELECT e.last_name, e.job_id, d.department_name,
e.salary, j.grade_level
FROM employees e, departments d, job_grades j
WHERE e.department_id = d.department_id
AND e.salary BETWEEN j.lowest_sal AND j.highest_sal;
-- OR
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);
```

10. Create a query to display the name and hire date of any employee hired after employee Davies.
Ans:

```
SELECT e.last_name, e.hire_date
FROM employees e, employees davies
WHERE davies.last_name = 'Davies'
AND davies.hire_date < e.hire_date
-- OR
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;
```

11. Display the names and hire dates for all employees who were hired before their managers, along with their manager's names and hire dates. Label the columns Employee, Emp
Hired, Manager, and Mgr Hired, respectively.
Ans:

```
SELECT w.last_name, w.hire_date, m.last_name, m.hire_date
FROM employees w, employees m
WHERE w.manager_id = m.employee_id
AND w.hire_date < m.hire_date;
-- OR
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;
```

CHAPTER-5: Aggregating Data Using 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.
   - **True**
   - **False**

Ans: True

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

Ans: FALSE
Group functions ignore null values. If you want to include null values, use the NVL function.

3. The WHERE clause restricts rows prior to inclusion in a group calculation.
   - **True**
   - **False**

Ans: True

4. Display the highest, lowest, sum, and average salary of all employees. Label the columns Maximum, Minimum, Sum, and Average, respectively.
Round your results to the nearest whole number.
Place your SQL statement in a text file named p5_6.sql.
Ans:

```
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 p5_4.sql to display the minimum, maximum, sum, and average salary for each job type.
Resave p5_4.sql to p5_5.sql. Run the statement in p5_5.sql.
Ans:

```
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.
Ans:

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

7. Determine the number of managers without listing them.
Label the column Number of Managers.
Hint: Use the MANAGER_ID column to determine the number of managers.
Ans:

```
SELECT COUNT(DISTINCT manager_id) "Number of Managers"
FROM employees;
```

8. Write a query that displays the difference between the highest a nd lowest salaries.
Label the column DIFFERENCE.
Ans:

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

9. 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 less than $6,000.
Sort the output in descending order of salary.
Ans:

```
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;
```

10. Write a query to display each department's name, location, number of employees, and the average salary for all employees in that department.
Label the columns Name, Location, Number of People, and Salary, respectively.
Round the average salary to two decimal places.
Ans:

```
SELECT d.department_name "Name", d.location_id "Location",
COUNT(*) "Number of People",
ROUND(AVG(salary),2) "Salary"
FROM employees e, departments d
WHERE e.department_id = d.department_id
GROUP BY d.department_name, d.location_id;
```

11. 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.
Ans:

```
SELECT COUNT(*) total,
SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1995,1,0))"1 995",
SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1996,1,0))"1 996",
SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1997,1,0))"1 997",
SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1998,1,0))"1 998"
FROM employees;
```

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

```
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;
```

CHAPTER-6: Sub Queries

1. Write a query to display the last name and hire date of any employee in the same department as Zlotkey.
Exclude Zlotkey.

Ans:

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

2. Create a query to display the employee numbers and last names of all employees who earn more than the average salary. Sort the results in descending order of salary.

Ans:

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

3. Write a query that displays the employee numbers and last names of all employees who work in a department with any employee whose last name contains a u.
Place your SQL statement in a text file named p6_3.sql. Run your query.

Ans:

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

4. Display the last name, department number, and job ID of all employees whose department location ID is 1700.

Ans:

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

5. Display the last name and salary of every employee who reports to King.

Ans:

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

6. Display the department number, last name, and job ID for every employee in the Executive department.

Ans:

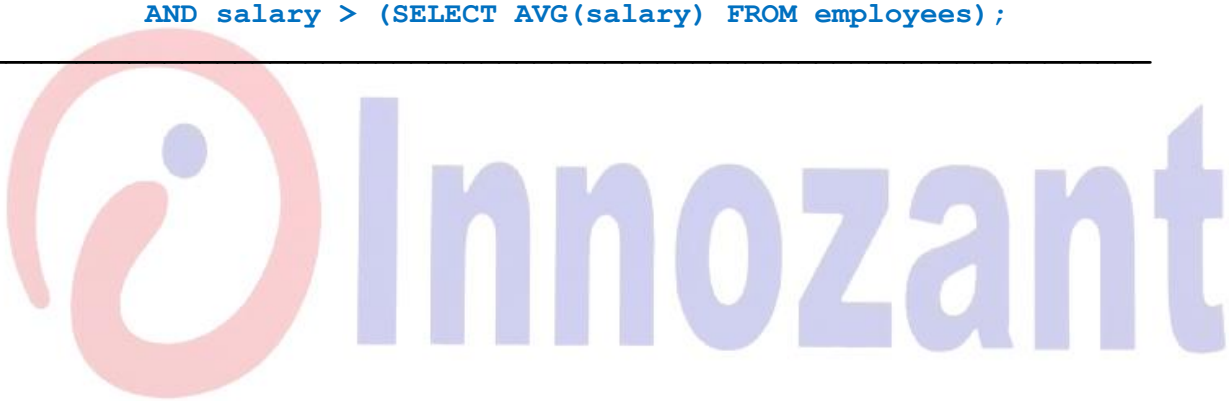
```
SELECT department_id, last_name, job_id
FROM employees
WHERE department_id IN (SELECT department_id
FROM departments
WHERE department_name = 'Executive');
```

7. Modify the query in p6_3.sql to display the employee numbers, last names, and salaries of all employees who earn more than the average salary and who work in a department with any employee with a u in their name.

Resave p6_3.sql to p6_7.sql. Run the statement in p6_7.sql.

Ans:

```
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);
```



CHAPTER-7: Producing Readable Output with iSQL *Plus

Determine whether the following statements are true or false:

1. The following statement is correct:
DEFINE & p_val = 100

- **True**
- **False**

Ans: FALSE

The correct use of DEFINE is DEFINE p_val=100. The & is used within the SQL code.

2. The DEFINE command is a SQL command.

- **True**
- **False**

Ans: FALSE

The DEFINE command is an iSQL*Plus command.

3. Write a script file to display the employee last name, job, and hire date for all employees who started between a given range. Concatenate the name and job together, separated by a space and comma, and label the column Employees. Use the DEFINE command to provide the two ranges.
Use the format MM/DD/YYYY. Save the script file as p7_3.sql.

Ans:

```
SET ECHO OFF
SET VERIFY OFF
DEFINE low_date = 01/01/1998
DEFINE high_date = 01/01/1999
SELECT last_name ||', '|| job_id EMPLOYEES, hire_date
FROM employees
WHERE hire_date BETWEEN TO_DATE('&low_date', 'MM/DD/YYYY')
AND TO_DATE('&high_date', 'MM/DD/YYYY')
/

UNDEFINE low_date
UNDEFINE high_date
SET VERIFY ON
SET ECHO ON
```

4. Write a script to display the employee last name, job, and department name for a given location. The search condition should allow for case-insensitive searches of the department location.
Save the script file as p7_4.sql.

Ans:

```
SET ECHO OFF
SET VERIFY OFF
COLUMN last_name HEADING "EMPLOYEE NAME"
COLUMN department_name HEADING "DEPARTMENT NAME"
SELECT e.last_name, e.job_id, d.department_name
FROM employees e, departments d, locations l
WHERE e.department_id = d.department_id
```

```
AND l.location_id = d.location_id
AND l.city = INITCAP('&p_location')
/
COLUMN last_name CLEAR
COLUMN department_name CLEAR
SET VERIFY ON
SET ECHO ON
```

5. Modify the code in p7_4.sql to create a report containing the department name, employee last name, hire date, salary, and each employee's annual salary for all employees in a given location.
Label the columns DEPARTMENT NAME, EMPLOYEE NAME, START DATE, SALARY, and ANNUAL SALARY, placing the labels on multiple lines.
Resave the script as p7_5.sql and execute the commands in the script.
Ans:

```
SET ECHO OFF
SET FEEDBACK OFF
SET VERIFY OFF
BREAK ON department_name
COLUMN department_name HEADING "DEPARTMENT|NAME"
COLUMN last_name HEADING "EMPLOYEE|NAME"
COLUMN hire_date HEADING "START|DATE"
COLUMN salary HEADING "SALARY" FORMAT $99,990.00
COLUMN asal HEADING "ANNUAL|SALARY" FORMAT $99,990.00
SELECT d.department_name, e.last_name, e.hire_date,
e.salary, e.salary*12 asal
FROM departments d, employees e, locations l
WHERE e.department_id = d.department_id
AND d.location_id = l.location_id
AND l.city = '&p_location'
ORDER BY d.department_name
/
COLUMN department_name CLEAR
COLUMN last_name CLEAR
COLUMN hire_date CLEAR
COLUMN salary CLEAR
COLUMN asal CLEAR
CLEAR BREAK
SET VERIFY ON
SET FEEDBACK ON
SET ECHO ON
```

CHAPTER-8: Manipulating Data

Insert data into the MY_EMPLOYEE table.

1. Run the statement in the p8_1.sql script to build the MY_EMPLOYEE table that will be used for the lab.

Ans:

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

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

Ans:

```
DESCRIBE my_employee;
```

3. Add the first row of data to the MY_EMPLOYEE table from the following sample data. Do not list thecolumns in the INSERT clause.

Ans:

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

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

Ans:

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

5. Confirm your addition to the table.

Ans:

```
SELECT * FROM my_employee;
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

6. Write an insert statement in a text file named loademp.sql to load rows into the MY_EMPLOYEE table. Concatenate the first letter of the first name and the first seven characters of the last name to produce the userid.
Ans:

```
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
```

7. Populate the table with the next two rows of sample data by running the insert statement in the script that you created.
Ans:

```
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
```

8. Confirm your additions to the table.
Ans:

```
SELECT * FROM my_employee;
```

9. Make the data additions permanent.
Ans:

```
COMMIT;
```

Update and delete data in the `MY_EMPLOYEE` table.
10. Change the last name of employee 3 to Drexler.
Ans:

```
UPDATE my_employee
SET last_name = 'Drexler'
WHERE id = 3;
```

11. Change the salary to 1000 for all employees with a salary less than 900.
Ans:

```
UPDATE my_employee
SET salary = 1000
WHERE salary < 900;
```

12. Verify your changes to the table.

Ans:

```
SELECT last_name, salary FROM my_employee;
```

13. Delete Betty Dancs from the MY_EMPLOYEE table.

Ans:

```
DELETE FROM my_employee
WHERE last_name = 'Dancs';
```

14. Confirm your changes to the table.

Ans:

```
SELECT * FROM my_employee;
```

15. Commit all pending changes.

Ans:

```
COMMIT;
```

Control data transaction to the `MY_EMPLOYEE` table.

16. Populate the table with the last row of sample data by modifying the statements in the script that you created in step 6. Run the statements in the script.

Ans:

```
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
```

17. Confirm your addition to the table.

Ans:

```
SELECT * FROM my_employee;
```

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

Ans:

```
SAVEPOINT step_18;
```

19. Empty the entire table.

Ans:

```
DELETE FROM my_employee;
```

20. Confirm that the table is empty.
Ans:

```
SELECT * FROM my_employee;
```

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

```
ROLLBACK TO step_18;
```

22. Confirm that the new row is still intact.
Ans:

```
SELECT * FROM my_employee;
```

23. Make the data addition permanent.
Ans:

```
COMMIT;
```

CHAPTER – 9: Creating and Managing Table

1. Create the DEPT table based on the following table instance chart. Place the syntax in a script called p9_1.sql, then execute the statement in the script to create the table. Confirm that the table is created.
Ans:

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

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

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

3. Create the EMP table based on the following table instance chart. Place the syntax in a script called p9_3.sql, and then execute the statement in the script to create the table. Confirm that the table is created.

Column Name	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Key Type				
Nulls/Unique				
FK Table				
FK Column				
Data type	Number	VARCHAR2	VARCHAR2	Number
Length	7	25	25	7

Ans:

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

4. Modify the EMP table to allow for longer employee last names. Confirm your modification.
Ans:

```
ALTER TABLE emp
MODIFY (last_name VARCHAR2(50));
DESCRIBE emp
```

5. Confirm that both the DEPT and EMP tables are stored in the data dictionary. (Hint: USER_TABLES)

Ans:

```
SELECT table_name FROM user_tables
WHERE table_name IN ('DEPT', 'EMP');
```

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.

Ans:

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

7. Drop the EMP table.

Ans:

```
DROP TABLE emp;
```

8. Rename the EMPLOYEES2 table to EMP.

Ans:

```
RENAME employees2 TO emp;
```

9. Add a comment to the DEPT and EMP table definitions describing the tables. Confirm your additions in the data dictionary.

Ans:

```
COMMENT ON TABLE emp IS 'Employee Information';
COMMENT ON TABLE dept IS 'Department Information';
SELECT * FROM user_tab_comments
WHERE table_name = 'DEPT'
OR table_name = 'EMP';
```

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

Ans:

```
ALTER TABLE emp
DROP COLUMN FIRST_NAME;
DESCRIBE emp
```

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

Ans:

```
ALTER TABLE emp
SET UNUSED (dept_id);
DESCRIBE emp
```

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

```
ALTER TABLE emp
DROP UNUSED COLUMNS;
DESCRIBE emp
```

CHAPTER-10 : Including Constraints

1. Add a table-level PRIMARY KEY constraint to the EMP table on the ID column. The constraint should be named at creation. Name the constraint my_emp_id_pk
Ans:

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

2. Create a PRIMARY KEY constraint to the DEPT table using the ID column. The constraint should be named at creation. Name the constraint my_deptid_pk.
Ans:

```
ALTER TABLE dept
ADD CONSTRAINT my_deptid_pk PRIMARY KEY(id);
```

3. Add a column DEPT_ID to the EMP table. Add a foreign key reference on the EMP table that ensures that the employee is not assigned to a nonexistent department. Name the constraint my_emp_dept_id_fk.
Ans:

```
ALTER TABLE emp
ADD (dept_id NUMBER(7));
ALTER TABLE emp
ADD CONSTRAINT my_emp_dept_id_fk
FOREIGN KEY (dept_id) REFERENCES dept(id);
```

4. Confirm that the constraints were added by querying the USER_CONSTRAINTS view. Note the types and names of the constraints. Save your statement text in a file called p10_4.sql.
Ans:

```
SELECT constraint_name, constraint_type
FROM user_constraints
WHERE table_name IN ('EMP', 'DEPT');
```

5. Display the object names and types from the USER_OBJECTS data dictionary view for the EMP and DEPT tables. Notice that the new tables and a new index were created.
Ans:

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

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

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

CHAPTER – 11: Creating Views

1. Create a view called EMPLOYEES_VU based on the employee numbers, employee names, and department numbers from the EMPLOYEES table.
Change the heading for the employee name to EMPLOYEE.
Ans:

```
CREATE OR REPLACE VIEW employees_vu AS
SELECT employee_id, last_name employee, department_id
FROM employees;
```

2. Display the contents of the EMPLOYEES_VU view.
Ans:

```
SELECT * FROM employees_vu;
```

3. Select the view name and text from the USER_VIEWS data dictionary view.
Note: Another view already exists. The EMP_DETAILS_VIEW was created as part of your schema.
Note: To see more contents of a LONG column, use the iSQL*Plus command SET LONG n, where n is the value of the number of characters of the LONG column that you want to see.
Ans:

```
SET LONG 600
SELECT view_name, text FROM user_views;
```

4. Using your EMPLOYEES_VU view, enter a query to display all employee names and department numbers.
Ans:

```
SELECT employee, department_id
FROM employees_vu;
```

5. Create a view named DEPT50 that contains the employee numbers, employee last names, and department numbers for all employees in department 50. Label the view columns EMPNO, EMPLOYEE, and DEPTNO. Do not allow an employee to be reassigned to another department through the view.
Ans:

```
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;
```

6. Display the structure and contents of the DEPT50 view.
Ans:

```
DESCRIBE dept50
SELECT *
FROM dept50;
```

7. Attempt to reassign Matos to department 80.
Ans:

```
UPDATE dept50
SET deptno = 80
WHERE employee = 'Matos';
```

8. Create a view called SALARY_VU based on the employee last names, department names, salaries, and salary grades for all employees. Use the EMPLOYEES, DEPARTMENTS, and JOB_GRADES tables.
Label the columns Employee, Department, Salary, and Grade, respectively.
Ans:

```
CREATE OR REPLACE VIEW salary_vu
AS
SELECT e.last_name "Employee",d.department_name "Department",
e.salary "Salary",j.grade_level "Grades"
FROM employees e,departments d,job_grades j
WHERE e.department_id = d.department_id
AND e.salary BETWEEN j.lowest_sal and j.highest_sal;
```

CHAPTER – 12: Other Database Obects

1. Create a sequence to be used with the primary key column of the DEPT table. The sequence should start at 200 and have a maximum value of 1000. Have your sequence increment by ten numbers. Name the sequence DEPT_ID_SEQ.
Ans:

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

2. Write a query in a script to display the following information about your sequences: sequence name,maximum value, increment size, and last number. Name the script p12_2.sql. Run the statement in your script.
Ans:

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

3. Write a script to insert two rows into the DEPT table. Name your script p12_3.sql.
Be sure to use the sequence that you created for the ID column. Add two departments named Education and Administration. Confirm your additions. Run the commands in your script.
Ans:

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

4. Create a nonunique index on the foreign key column (DEPT_ID) in the EMP table.
Ans:

```
CREATE INDEX emp_dept_id_idx ON emp (dept_id);
```

5. Display the indexes and uniqueness that exist in the data dictio nary for the EMP table. Save the statement into a script named p12_5.sql.
Ans:

```
SELECT index_name, table_name, uniqueness
FROM user_indexes
WHERE table_name = 'EMP';
```

CHAPTER – 13: Controlling User Access

1. What privilege should a user be given to log on to the Oracle Server? Is this a system or an object privilege?
Ans:

```
The CREATE SESSION system privilege
```

2. What privilege should a user be given to create tables?
Ans:

```
The CREATE TABLE privilege
```

3. If you create a table, who can pass along privileges to other users on your table?
Ans:
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?
Ans:
Create a role containing the system privileges and grant the role to the users

5. What command do you use to change your password?
Ans:

```
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.
Ans:

```
GRANT select
ON departments
TO <user1>;
Team 1 executes the GRANT statement.
GRANT select
ON departments
TO <user2>;
```

WHERE *user1* is the name of team 1 and *user2* is the name of team 2.

7. Query all the rows in your DEPARTMENTS table.
Ans:

```
SELECT * FROM departments;
```

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

Ans:

Team 1 executes this INSERT statement.

```
INSERT INTO departments(department_id, department_name)
VALUES (200, 'Education');
COMMIT;
```

Team 2 executes this INSERT statement.

```
INSERT INTO departments(department_id, department_name)
VALUES (210, 'Administration');
COMMIT;
```

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

Ans:

Team 1 creates a synonym named team2.

```
CREATE SYNONYM team2
FOR <user2>.DEPARTMENTS;
```

Team 2 creates a synonym named team1.

```
CREATE SYNONYM team1
FOR <user1>. DEPARTMENTS;
```

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

Ans:

Team 1 executes this SELECT statement.

```
SELECT *
FROM team2;
```

Team 2 executes this SELECT statement.

```
SELECT *
FROM team1;
```


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

Ans:

```
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.

Ans:

```
SELECT table_name, owner
FROM all_tables
WHERE owner <> <your account>;
```

13. Revoke the SELECT privilege from the other team.

Ans:

Team 1 revokes the privilege.

```
REVOKE select
ON departments
FROM user2;
```

Team 2 revokes the privilege.

```
REVOKE select
ON departments
FROM user1;
```

CHAPTER – 14: SQL Workshop Workshop Overview

1. Create the tables based on the following table instance charts. Choose the appropriate data types and be sure to add integrity constraints.

Table name: MEMBER

Column_ Name	MEMBER_ ID	LAST_ NAME	FIRST_NAM E	ADDRESS	CITY	PHONE	JOIN _DATE
Key Type	PK						
Null/ Unique	NN,U	NN					NN
Default Value							System Date
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	25	25	100	30	15	

Ans:

```
CREATE TABLE member
(member_id NUMBER(10)
CONSTRAINT member_member_id_pk PRIMARY KEY,
last_name VARCHAR2(25)
CONSTRAINT member_last_name_nn NOT NULL,
first_name VARCHAR2(25),
address VARCHAR2(100),
city VARCHAR2(30),
phone VARCHAR2(15),
join_date DATE DEFAULT SYSDATE
CONSTRAINT member_join_date_nn NOT NULL);
```

Table name: TITLE

Column_ Name	TITLE_ID	TITLE	DESCRIPTION	RATING	CATEGORY	RELEASE_ DATE
Key Type	PK					
Null/ Unique	NN,U	NN	NN			
Check				G, PG, R, NC17, NR	DRAMA, COMEDY, ACTION, CHILD, SCIFI, DOCUMEN-TARY	
Data Type	NUMBER	VARCHAR2	VARCHAR2	VARCHAR2	VARCHAR2	DATE
Length	10	60	400	4	20	

Ans:

```
CREATE TABLE title
(title_id NUMBER(10)
CONSTRAINT title_title_id_pk PRIMARY KEY,
title VARCHAR2(60)
```

```
CONSTRAINT title_title_nn NOT NULL,
description VARCHAR2(400)
CONSTRAINT title_description_nn NOT NULL,
rating VARCHAR2(4)
CONSTRAINT title_rating_ck CHECK
(rating IN ('G', 'PG', 'R', 'NC17', 'NR')),
category VARCHAR2(20),
CONSTRAINT title_category_ck CHECK
(category IN ('DRAMA', 'COMEDY', 'ACTION',
'CHILD', 'SCIFI', 'DOCUMENTARY')),
release_date DATE);
```

Table name: TITLE_COPY

Column Name	COPY_ID	TITLE_ID	STATUS
Key Type	PK	PK,FK	
Null/ Unique	NN,U	NN,U	NN
Check			AVAILABLE, DESTROYED, RENTED, RESERVED
FK Ref Table		TITLE	
FK Ref Column		TITLE_ID	
Data Type	NUMBER	NUMBER	VARCHAR2
Length	10	10	15

Ans:

```
CREATE TABLE title_copy
(copy_id NUMBER(10),
title_id NUMBER(10)
CONSTRAINT title_copy_title_if_fk REFERENCES title(title_id),
status VARCHAR2(15)
CONSTRAINT title_copy_status_nn NOT NULL
CONSTRAINT title_copy_status_ck CHECK (status IN
('AVAILABLE', 'DESTROYED','RENTED', 'RESERVED')),
CONSTRAINT title_copy_copy_id_title_id_pk
PRIMARY KEY (copy_id, title_id));
```

Table name: RENTAL

Column Name	BOOK_DATE	MEMBER_ID	COPY_ID	ACT_RET_DATE	EXP_RET_DATE	TITLE_ID
Key Type	PK	PK,FK1	PK,FK2			PK,FK2
Default Value	System Date				System Date + 2 days	
FK Ref Table		MEMBER	TITLE_COPY			TITLE_COPY
FK Ref Column		MEMBER_ID	COPY_ID			TITLE_ID
Data Type	DATE	NUMBER	NUMBER	DATE	DATE	NUMBER
Length		10	10			10

Ans:

```
CREATE TABLE rental
(book_date DATE DEFAULT SYSDATE,
member_id NUMBER(10)
CONSTRAINT rental_member_id_fk
REFERENCES member(member_id),
copy_id NUMBER(10),
act_ret_date DATE,
exp_ret_date DATE DEFAULT SYSDATE + 2,
title_id NUMBER(10),
CONSTRAINT rental_book_date_copy_title_pk
PRIMARY KEY (book_date, member_id,
copy_id,title_id),
CONSTRAINT rental_copy_id_title_id_fk
FOREIGN KEY (copy_id, title_id)
REFERENCES title_copy(copy_id, title_id));
```

Table name: RESERVATION

Column Name	RES_DATE	MEMBER_ID	TITLE_ID
Key Type	PK	PK,FK1	PK,FK2
Null/ Unique	NN,U	NN,U	NN
FK Ref Table		MEMBER	TITLE
FK Ref Column		MEMBER_ID	TITLE_ID
Data Type	DATE	NUMBER	NUMBER
Length		10	10

Ans:

```
CREATE TABLE reservation
(res_date DATE,
member_id NUMBER(10)
CONSTRAINT reservation_member_id
REFERENCES member(member_id),
title_id NUMBER(10)
CONSTRAINT reservation_title_id
REFERENCES title(title_id),
CONSTRAINT reservation_resdate_mem_tit_pk PRIMARY KEY
(res_date, member_id, title_id));
```

2. Verify that the tables and constraints were created properly by checking the data dictionary.
Ans:

```
SELECT table_name
FROM user_tables
WHERE table_name IN ('MEMBER', 'TITLE', 'TITLE_COPY',
'RENTAL', 'RESERVATION');

SELECT constraint_name, constraint_type, table_name
FROM user_constraints
WHERE table_name IN ('MEMBER', 'TITLE', 'TITLE_COPY',
'RENTAL', 'RESERVATION');
```

3. Create sequences to uniquely identify each row in the MEMBER table and the TITLE table.
a. Member number for the MEMBER table: start with 101; do not allow caching of the values. Name the sequence MEMBER_ID_SEQ.
Ans:

```
CREATE SEQUENCE member_id_seq
START WITH 101
NOCACHE;
```

b. Title number for the TITLE table: start with 92; no caching.
Name the sequence TITLE_ID_SEQ.
Ans:

```
CREATE SEQUENCE title_id_seq
START WITH 92
NOCACHE;
```

c. Verify the existence of the sequences in the data dictionary.
Ans:

```
SELECT sequence_name, increment_by, last_number
FROM user_sequences
WHERE sequence_name IN ('MEMBER_ID_SEQ', 'TITLE_ID_SEQ ');
```

4. Add data to the tables. Create a script for each set of data to add.
a. Add movie titles to the TITLE table. Write a script to enter the movie information.
Save the statements in a script named p14_4a.sql.
Use the sequences to uniquely identify each title.
Enter the release dates in the DD-MON-YYYY format.
Remember that single quotation marks in a character field must be specially handled.
Verify your additions.
Ans:

```
SET ECHO OFF
INSERT INTO title(title_id, title, description, rating,
category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Willie and Christmas Too',
'All of Willie''s friends make a Christmas list for
Santa, but Willie has yet to add his own wish list.',
'G', 'CHILD', TO_DATE('05-OCT-1995','DD-MON-YYYY')
/
INSERT INTO title(title_id , title, description, rating,
category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Alien Again', 'Yet another
installment of science fiction history. Can the
heroine save the planet from the alien life form?',
'R', 'SCIFI', TO_DATE( '19-MAY-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
category, release_date)
VALUES (title_id_seq.NEXTVAL, 'The Glob', 'A meteor crashes
near a small American town and unleashes carnivorous
goo in this classic.', 'NR', 'SCIFI',
TO_DATE( '12-AUG-1995','DD-MON-YYYY'))
/
INSERT INTO title(title_id, title, description, rating,
category, release_date)
VALUES (title_id_seq.NEXTVAL, 'My Day Off', 'With a little
luck and a lot ingenuity, a teenager skips school for
a day in New York.', 'PG', 'COMEDY',
TO_DATE( '12-JUL-1995','DD-MON-YYYY'))
/
...
COMMIT
/
SET ECHO ON
SELECT title
FROM title;
```

Title	Description	Rating	Category	Release_date
Willie and Christmas Too	All of Willie's friends make a Christmas list for Santa, but Willie has yet to add his own wish list.	G	CHILD	05-OCT-1995
Alien Again	Yet another installation of science fiction history. Can the heroine save the planet from the alien life form?	R	SCIFI	19-MAY-1995
The Glob	A meteor crashes near a small American town and unleashes carnivorous goo in this classic.	NR	SCIFI	12-AUG-1995
My Day Off	With a little luck and a lot of ingenuity, a teenager skips school for a day in New York	PG	COMEDY	12-JUL-1995
Miracles on Ice	A six-year-old has doubts about Santa Claus, but she discovers that miracles really do exist.	PG	DRAMA	12-SEP-1995
Soda Gang	After discovering a cache of drugs, a young couple find themselves pitted against a vicious gang.	NR	ACTION	01-JUN-1995

b. Add data to the MEMBER table. Place the insert statements in a script named p14_4b.sql. Execute commands in the script. Be sure to use the sequence to add the member numbers.

First_ Name	Last_Name	Address	City	Phone	Join_Date
Carmen	Velasquez	283 King Street	Seattle	206-899-6666	08-MAR-1990
LaDoris	Ngao	5 Modrany	Bratislava	586-355-8882	08-MAR-1990
Midori	Nagayama	68 Via Centrale	Sao Paolo	254-852-5764	17-JUN-1991
Mark	Quick-to-See	6921 King Way	Lagos	63-559-7777	07-APR-1990
Audry	Ropeburn	86 Chu Street	Hong Kong	41-559-87	18-JAN-1991
Molly	Urguhart	3035 Laurier	Quebec	418-542-9988	18-JAN-1991

Ans:

```
SET ECHO OFF
SET VERIFY OFF
INSERT INTO member(member_id, first_name, last_name, address,
city, phone, join_date)
VALUES (member_id_seq.NEXTVAL, '&first_name', '&last_name',
'&address', '&city', '&phone', TO_DATE('&join_date',
'DD-MM-YYYY');
COMMIT;
SET VERIFY ON
SET ECHO ON
```

c. Add the following movie copies in the TITLE_COPY table:
Note: Have the TITLE_ID numbers available for this exercise.

Title	Copy_Id	Status
Willie and Christmas Too	1	AVAILABLE
Alien Again	1	AVAILABLE
	2	RENTED
The Glob	1	AVAILABLE
My Day Off	1	AVAILABLE
	2	AVAILABLE
	3	RENTED
Miracles on Ice	1	AVAILABLE
Soda Gang	1	AVAILABLE

Ans:

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 92, 'AVAILABLE');
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 93, 'AVAILABLE');
```

```
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (2, 93, 'RENTED');
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 94, 'AVAILABLE');
INSERT INTO title_copy(copy_id, title_id, status)
VALUES (1, 95, 'AVAILABLE');
INSERT INTO title_copy(copy_id, title_id,status)
VALUES (2, 95, 'AVAILABLE');
INSERT INTO title_copy(copy_id, title_id,status)
VALUES (3, 95, 'RENTED');
INSERT INTO title_copy(copy_id, title_id,status)
VALUES (1, 96, 'AVAILABLE');
INSERT INTO title_copy(copy_id, title_id,status)
VALUES (1, 97, 'AVAILABLE');
```

d. Add the following rentals to the RENTAL table:
Note: Title number may be different depending on sequence number.

Title_ Id	Copy_ Id	Member_ Id	Book_date	Exp_Ret_Date	Act_Ret_Date
92	1	101	3 days ago	1 day ago	2 days ago
93	2	101	1 day ago	1 day from now	
95	3	102	2 days ago	Today	
97	1	106	4 days ago	2 days ago	2 days ago

Ans:

```
INSERT INTO rental(title_id, copy_id, member_id,
book_date, exp_ret_date, act_ret_date)
VALUES (92, 1, 101, sysdate-3, sysdate-1, sysdate-2);
INSERT INTO rental(title_id, copy_id, member_id,
book_date, exp_ret_date, act_ret_date)
VALUES (93, 2, 101, sysdate-1, sysdate-1, NULL);
INSERT INTO rental(title_id, copy_id, member_id,
book_date, exp_ret_date, act_ret_date)
VALUES (95, 3, 102, sysdate-2, sysdate, NULL);
INSERT INTO rental(title_id, copy_id, member_id,
book_date, exp_ret_date,act_ret_date)
VALUES (97, 1, 106, sysdate-4, sysdate-2, sysdate-2);
COMMIT;
```

5. Create a view named TITLE_AVAIL to show the movie titles and the availability of each copy and its expected return date if rented. Query all rows from the view. Order the results by title.

Ans:

```
CREATE VIEW title_avail AS
SELECT t.title, c.copy_id, c.status, r.exp_ret_date
FROM title t, title_copy c, rental r
WHERE t.title_id = c.title_id
AND c.copy_id = r.copy_id(+)
AND c.title_id = r.title_id(+);
SELECT *
FROM title_avail
ORDER BY title, copy_id;
```

6. Make changes to data in the tables.

a. Add a new title. The movie is "Interstellar Wars," which is rated PG and classified as a scifi movie. The release date is 07-JUL-77.
The description is "Futuristic interstellar action movie. Can the rebels save the humans from the evil empire?"
Be sure to add a title. Copy record for two copies.

Ans:

```
INSERT INTO title(title_id, title, description, rating,
category, release_date)
VALUES (title_id_seq.NEXTVAL, 'Interstellar Wars',
'Futuristic interstellar action movie. Can the
rebels save the humans from the evil Empire?',
'PG', 'SCIFI', '07-JUL-77');
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (1, 98, 'AVAILABLE');
INSERT INTO title_copy (copy_id, title_id, status)
VALUES (2, 98, 'AVAILABLE');
```

b. Enter two reservations. One reservation is for Carmen Velasquez, who wants to rent "Interstellar Wars." The other is for Mark Quick-to-See, who wants to rent "Soda Gang."

Ans:

```
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 101, 98);
INSERT INTO reservation (res_date, member_id, title_id)
VALUES (SYSDATE, 104, 97);
```

c. Customer Carmen Velasquez rents the movie "Interstellar Wars," copy 1. Remove her reservation for the movie. Record the information about the rental. Allow the default value for the expected return date to be used. Verify that the rental was recorded by using the view you created.

Ans:

```
INSERT INTO rental(title_id, copy_id, member_id)
VALUES (98,1,101);
UPDATE title_copy
SET status= 'RENTED'
WHERE title_id = 98
```

```
AND copy_id = 1;

DELETE FROM reservation
WHERE member_id = 101;

SELECT * FROM title_avail
ORDER BY title, copy_id;
```

7. Make a modification to one of the tables.
a. Add a PRICE column to the TITLE table to record the purchase price of the video. The column should have a total length of eight digits and two decimal places. Verify your modifications.
Ans:

```
ALTER TABLE title
ADD (price NUMBER(8,2));
DESCRIBE title
```

b. Create a script named p14_7b.sql that contains update statements that update each video with a price according to the following list. Run the commands in the script.
Note: Have the TITLE_ID numbers available for this exercise.

Title	Price
Willie and Christmas Too	25
Alien Again	35
The Glob	35
My Day Off	35
Miracles on Ice	30
Soda Gang	35
Interstellar Wars	29

Ans:

```
SET ECHO OFF
SET VERIFY OFF
DEFINE price=
DEFINE title_id=UPDATE title
SET price = &price
WHERE title_id = &title_id;
SET VERIFY OFF
SET ECHO OFF
```

c. Ensure that in the future all titles contain a price value. Verify the constraint.
Ans:

```
ALTER TABLE title
MODIFY (price CONSTRAINT title_price_nn NOT NULL);
SELECT constraint_name, constraint_type,
search_condition
FROM user_constraints
WHERE table_name = 'TITLE';
```

8. Create a report titled Customer History Report. This report contains each customer's history of renting videos. Be sure to include the customer name, movie rented, dates of the rental, and duration of rentals. Total the number of rentals for all customers for the reporting period. Save the commands that generate the report in a script file named p14_8.sql.

Ans:

```
SET ECHO OFF
SET VERIFY OFF
TTITLE 'Customer History Report'
BREAK ON member SKIP 1 ON REPORT
SELECT m.first_name||' '||m.last_name MEMBER, t.title,
r.book_date, r.act_ret_date - r.book_date DURATION
FROM member m, title t, rental r
WHERE r.member_id = m.member_id
AND r.title_id = t.title_id
ORDER BY member;
CLEAR BREAK
TTITLE OFF
SET VERIFY ON
SET ECHO ON
```

CHAPTER – 15: Using Set Operators

1. List the department IDs for departments that do not contain the job ID ST_CLERK, using SET operators.

Ans:

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

2. Display the country ID and the name of the countries that have no departments located in them, using SET operators.

Ans:

```
SELECT country_id,country_name FROM countries
MINUS
SELECT l.country_id,c.country_name
FROM locations l, countries c
WHERE l.country_id = c.country_id;
```

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

Ans:

```
COLUMN dummy PRINT
SELECT job_id, department_id, 'x' dummy FROM employees
WHERE department_id = 10
UNION
SELECT job_id, department_id, 'y' FROM employees
WHERE department_id = 50
UNION
SELECT job_id, department_id, 'z' FROM employees
WHERE department_id = 20
ORDER BY 3;

COLUMN dummy NOPRINT
```

4. List the employee IDs and job IDs of those employees, who are currently in the job title that they have held once before during their tenure with the company.

Ans:

```
SELECT employee_id,job_id FROM employees
INTERSECT
SELECT employee_id,job_id FROM job_history;
```

5. Write a compond query that lists the following :
• Last names and department ID of all the employees from the EMPLOYEES table, irrespective of whether they belong to any department
• Department ID and department name of all the departments from the DEPARTMENTS table, irrespective of whether they have employees working in them

Ans:

```
SELECT last_name,department_id,TO_CHAR(null) FROM employees
UNION
SELECT TO_CHAR(null),department_id,department_name FROM
departments;
```

CHAPTER – 16: Oracle 9i Datetime functions

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

```
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
Ans:

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

Singapore
Ans:

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

Egypt
Ans:

```
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.
Ans:

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

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

```
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.
Ans:

```
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.
Ans:

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

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

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

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

Ans:

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

CHAPTER – 17: Enhancement to the GROUP by Clause.

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
Ans:

```
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.
Ans:

```
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
Ans:

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

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.
Ans:

```
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.
Ans:

```
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));
```

CHAPTER – 18: Advanced Subqueries

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.
Ans:

```
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.
Ans:

```
SELECT last_name, department_name, 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.
Ans:

```
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';
```

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

```
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.
Ans:

```
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 alises for the columns retrieved by the query as shown in the sample output.
Ans:

```
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);
```

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

```
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?
Ans:

```
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.

Ans:

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

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

Ans:

```
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 of the employees, and department names of all employees.

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

Ans:

```
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.

Ans:

```
WITH summary AS (
SELECT department_name, SUM(salary) AS dept_total
FROM employees, departments
WHERE employees.department_id = departments.department_id
GROUP BY 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;
```

CHAPTER – 19: Hierarchical Retrieval

1. Look at the following output. Is this output the result of a hierarchical query? Explain why or why not.

a. **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 as 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.

Ans:

```
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.

Ans:

```
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.

Ans:

```
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
```

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 Hann.

Ans:

```
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';
```

CHAPTER-20: Oracle 9i Extensions to DML and DDL Statements

1. Run the cre_sal_history.sql script in the Labs folder to create the SAL_HISTORY table.
Ans:

```
@ \Labs\cre_sal_history.sql;
```

2. Display the structure of the SAL_HISTORY table.
Ans:

```
DESC sal_history;
```

3. Run the cre_mgr_history.sql script in the Labs folder to create the MGR_HISTORY table.
Ans:

```
@ \Labs\cre_mgr_history.sql;
```

4. Display the structure of the MGR_HISTORY table.
Ans:

```
DESC mgr_history;
```

5. Run the cre_special_sal.sql script in the Labs folder to create the SPECIAL_SAL table.
Ans:

```
@ \Labs\cre_special_sal.sql;
```

6. Display the structure of the SPECIAL_SAL table.
Ans:

```
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 SYSDATE into the MGR_HISTORY table.
Ans:

```
INSERT ALL
WHEN SAL > 20000 THEN
INTO special_empsal 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.
Ans:

```
SELECT * FROM special_sal;
```

c. Display the records from the SAL_HISTORY table.
Ans:

```
SELECT * FROM sal_history;
```

d. Display the records from the MGR_HISTORY table.
Ans:

```
SELECT * FROM mgr_history;
```

8. a. Run the cre_sales_source_data.sql script in the Labs folder to create the SALES_SOURCE_DATA table.
Ans:

```
@ \Labs\cre_sales_source_data.sql
```

b. Run the ins_sales_source_data.sql script in the Labs folder to insert records into the SALES_SOURCE_DATA table.
Ans:

```
@ \Labs\ins_sales_source_data.sql
```

c. Display the structure of the SALES_SOURCE_DATA table.
Ans:

```
DESC sales_source_data
```

d. Display the records from the SALES_SOURCE_DATA table.
Ans:

```
SELECT * FROM SALES_SOURCE_DATA;
```

e. Run the cre_sales_info.sql script in the Labs folder to create the SALES_INFO table.
Ans:

```
@ \Labs\cre_sales_info.sql
```

f. Display the structure of the SALES_INFO table.
Ans:

```
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.

Ans:

```
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.
Ans:

```
SELECT * FROM sales_info;
```