**11- Ensuring Quality Query Results**

**QUERY-1**

−Create a list of all tables whose first two characters in the name of the table is JO

–The tables must be owned by the current Oracle User

**SQL QUERY**

SELECT table\_name

FROM user\_tables

WHERE table\_name LIKE 'JO%';

ORDER BY table\_name;

**QUERY-2**

− Create a list that includes the first initial of every employee's first name, a space, and the last name of the employee

**SQL QUERY**

SELECT SUBSTR(first\_name, 1, 1) || ' ' || last\_name AS employee\_name

FROM employees;

**QUERY-3**

− Create a list of every employee's first name concatenated to a space and the employee's last name, and the email of all employees where the email address contains the string 'IN'

**SQL QUERY**

SELECT first\_name || ' ' || last\_name AS full\_name, email

FROM employees

WHERE email LIKE '%IN%';

ORDER BY email;

**QUERY -4**

− Create a list of 'smallest' last name and the 'highest' last name from the employees table .

**SQL QUERY**

SELECT

MIN(last\_name) AS smallest\_last\_name,

MAX(last\_name) AS highest\_last\_name

FROM employees;

**QUERY-5**

− Create a list of weekly salaries from the employees table where the weekly salary is between 700 and 3000

− The salaries should be formatted to include a $- sign and have two decimal points like: $9999.99.

**SQL QUERY**

SELECT

‘$’|| ROUND((salary\*12)/52,2) AS weekly\_salary

FROM

employees

WHERE

( salary \*12)/ 52 BETWEEN 700 AND 3000;

**QUERY-6**

− Create a list of every employee and his related job title sorted by job\_title .  
**SQL QUERY**

SELECT SUBSTR

(first\_name,1,1) || ' ' || last\_name AS employee\_name,

job\_title AS job

FROM

employees e,jobs j

WHERE e.job\_id = j.job\_id

ORDER BY job\_title;

**QUERY-7**

−Create a list of every employee’s job, the salary ranges within the job, and the employee's salary

−List the lowest and highest salary range within each job with a dash to separate the salaries like this: 100 – 200

**SQL QUERY**

SELECT SUBSTR(first\_name,1,1)|| ' ' ||last\_name AS employee\_name, job\_title AS job,min\_salary|| ' - ' ||max\_salary AS salary\_range,salary AS employee’s\_salary

FROM employees e,jobs j

WHERE e.job\_id=j.job\_id

ORDER BY job\_title;

**QUERY-8**

− Using an ANSII join method, create a list of every employee's first initial and last name, and department name − Make sure the tables are joined on all of the foreign keys declared between the two tables

**SQL QUERY**

SELECT SUBSTR(first\_name, 1, 1) || ' ' || last\_name AS employee\_name,department\_name

FROM employees NATURAL JOIN departments;

**QUERY-9**

− Change the previous listing to join only on the department\_id column.

**SQL QUERY**

SELECT

SUBSTR(e.first\_name, 1, 1) || ' ' || e.last\_name AS employee\_name,d.department\_name

FROM employees JOIN departments USING(department\_id);

**QUERY-10**

− Create a list of every employee's last name, and the word nobody or somebody depending on whether or not the employee has a manager

− Use the Oracle DECODE function to create the list

**SQL QUERY**

SELECT DECODE(manager\_id, NULL, 'nobody', 'somebody') AS manager\_status, last\_name,

FROM employees;

**QUERY-11**

− Create a list of every employee's first initial and last name, salary, and a yes or no to show whether or not an employee makes a commission

–Fix this query to produce the result

**SQL QUERY**

SELECT

SUBSTR(first\_name, 1, 1) || ' ' || last\_name AS employee\_name,

salary,

DECODE(commission\_pct, NULL, 'no', 'yes') AS makes\_commission

FROM

employees;

**QUERY-12**

− Create a list of every employee's last name, department name, city, and state\_province

− Include departments without employees

− An outer join is required

**SQL QUERY**

SELECT

e.last\_name,

d.department\_name,

l.city,

l.state\_province

FROM

departments d

LEFT JOIN

employees e ON d.department\_id = e.department\_id

LEFT JOIN

locations l ON d.location\_id = l.location\_id

ORDER BY

d.department\_name, e.last\_name;

**QUERY-13**

− Create a list of every employee's last name, department name, city, and state\_province

− Include departments without employees

− An outer join is required

**SQL query**

SELECT

e.last\_name,

d.department\_name,

l.city,

l.state\_province

FROM

departments d

LEFT JOIN

employees e ON d.department\_id = e.department\_id

LEFT JOIN

locations l ON d.location\_id = l.location\_id

ORDER BY

d.department\_name, e.last\_name;

QUERY-14

−Create a list of every employee's first and last names, and the first occurrence of: commission\_pct, manager\_id, or -1

−If an employee gets commission, display the commission\_pct column; if no commission, then display his manager\_id; if he has neither commission nor manager, then the number -1

**SQL QUERY**

SELECT

first\_name,

last\_name,

COALESCE(TO\_CHAR(commission\_pct), TO\_CHAR(manager\_id), '-1') AS first\_occurrence

FROM

employees;

**QUERY-15**

− Create a list of every employee's last name, salary, and job\_grade for all employees working in departments with a department\_id greater than 50

SELECT

e.last\_name,

e.salary,

j.job\_grade

FROM

employees e

JOIN

job\_grades j ON e.job\_id = j.job\_id

WHERE

e.department\_id > 50;

**QUERY-16**

− Produce a list of every employee's last name and department name

− Include both employees without departments, and departments without employees

**SQL QUERY**

SELECT

e.last\_name,

d.department\_name

FROM

employees e

LEFT JOIN

departments d ON e.department\_id = d.department\_id

UNION ALL

SELECT

NULL AS last\_name,

d.department\_name

FROM

departments d

LEFT JOIN

employees e ON d.department\_id = e.department\_id

WHERE

e.department\_id IS NULL;

**QUERY-17**

−Create a treewalking list of every employee's last name, his manager’s last name, and his position in the company

−The top level manager has position 1, this manager's subordinates position 2, their subordinates position 3, and so on −Start the listing with employee number 100

**SQL QUERY**

WITH RECURSIVE EmployeeHierarchy AS (

SELECT

e.employee\_id,

e.last\_name,

e.manager\_id,

1 AS position

FROM

employees e

WHERE

e.employee\_id = 100

UNION ALL

employees in the previous level

SELECT

e.employee\_id,

e.last\_name,

e.manager\_id,

eh.position + 1 AS position

FROM

employees e

JOIN

EmployeeHierarchy eh ON e.manager\_id = eh.employee\_id

)

SELECT

last\_name,

(SELECT last\_name FROM employees WHERE employee\_id = eh.manager\_id) AS manager\_last\_name,

position

FROM

EmployeeHierarchy eh

ORDER BY

position, last\_name;

**QUERY-18**

− Produce a list of the earliest hire date, the latest hire date, and the number of employees from the employees table

**SQL QUERY**

SELECT MIN(hire\_date) AS earliest\_hire\_date,

MAX(hire\_date) AS latest\_hire\_date,

COUNT(\*) AS number\_of\_employees

FROM employees;

**QUERY-19**

− Create a list of department names and the departmental costs (salaries added up)

− Include only departments whose salary costs are between 15000 and 31000, and sort the listing by the cost

**SQL QUERY**

SELECT

d.department\_name,

d.manager\_id,

e.last\_name AS manager\_name,

AVG(e2.salary) AS average\_salary

FROM

Departments d

JOIN

Employees e ON d.manager\_id = e.employee\_id

JOIN

Employees e2 ON d.department\_id = e2.department\_id

GROUP BY

d.department\_name, d.manager\_id, e.last\_name;

**QUERY-20**

− Show the highest average salary for the departments in the employees table

− Round the result to the nearest whole number

**SQL QUERY**

SELECT

department\_id,

ROUND(AVG(salary)) AS highest\_average\_salary

FROM

Employees

GROUP BY

department\_id

ORDER BY

AVG(salary) DESC

LIMIT 1;

**QUERY-21**

− Create a list of department names and their monthly costs (salaries added up)

**SQL QUERY**

SELECT

d.department\_name,

SUM(e.salary) AS monthly\_cost

FROM

Departments d

JOIN

Employees e ON d.department\_id = e.department\_id

GROUP BY

d.department\_name;

**QUERY-22**

− Create a list of department names, and job\_ids

− Calculate the monthly salary cost for each job\_id within a department, for each department, and for all departments added together

**SQL QUERY**

SELECT d.department\_name,e.job\_id,SUM(e.salary) AS monthly\_cost

FROM Departments d

JOIN Employees e ON d.department\_id = e.department\_id

GROUP BY d.department\_name, e.job\_id

UNION ALL

SELECT 'All Departments' AS department\_name,e.job\_id,

SUM(e.salary) AS monthly\_cost

FROME mployees e

GROUP BY e.job\_id

UNION ALL

SELECT 'All Departments' AS department\_name,'All Jobs' AS job\_id,SUM(e.salary) AS monthly\_cost

FROM Employees e;

**QUERY-23**

−Create a list of department names, and job\_ids

−Calculate the monthly salary cost for each job\_id within a department, for each department, for each group of job\_ids irrespective of the department, and for all departments added together

**SQL QUERY**

SELECT

d.department\_name,

e.job\_id,

SUM(e.salary) AS monthly\_cost

FROM

Departments d

JOIN

Employees e ON d.department\_id = e.department\_id

GROUP BY CUBE (d.department\_name, e.job\_id)

ORDER BY

d.department\_name,

e.job\_id;

**QUERY-24**

− Expand the previous list to also show if the department\_id or job\_id was used to create the subtotals shown in the output (Hint: Cube, Grouping)

**SQL QUERY**

SELECT d.department\_name, e.job\_id, SUM(e.salary) AS monthly\_cost, GROUPING(d.department\_name) AS is\_department\_subtotal,

GROUPING(e.job\_id) AS is\_job\_id\_subtotal

FROM Departments d

JOIN Employees e ON d.department\_id = e.department\_id

GROUP BY CUBE (d.department\_name, e.job\_id)

ORDER BY d.department\_name,

e.job\_id;

**QUERY-25**

− Create a list that includes the monthly salary costs for each job title within a department

− In the same list, display the monthly salary cost per city.

**SQL QUERY**

SELECT d.department\_name,e.job\_id,l.city,SUM(e.salary) AS monthly\_cost,

GROUPING(d.department\_name) AS is\_department\_grouping,

GROUPING(e.job\_id) AS is\_job\_id\_grouping,

GROUPING(l.city) AS is\_city\_grouping

FROM Employees e

JOIN Departments d ON e.department\_id = d.department\_id

JOIN Locations l ON d.location\_id = l.location\_id

GROUP BY GROUPING SETS ((d.department\_name, e.job\_id),

(l.city))

ORDER BY d.department\_name,e.job\_id,l.city;

QUERY-26

−Create a list of employee names as shown and department ids −In the same report, list the department ids and department names. And finally, list the cities

−The rows should not be joined, just listed in the same report.

SQL QUERY

SELECT e.first\_name || ' ' || e.last\_name AS name\_or\_city,

e.department\_id, NULL AS department\_name

FROM Employees e

UNION ALL SELECT NULL AS name\_or\_city, d.department\_id, d.department\_name

FROM Departments d

UNION ALL SELECT l.city AS name\_or\_city,NULL AS department\_id, NULL AS department\_name

FROM Locations l ORDER BY name\_or\_city,department\_id,department\_name;

**QUERY-27**

− Create a list of each employee's first initial and last name, salary, and department name for each employee earning more than the average for his department

**SQL QUERY**

SELECT SUBSTR(e.first\_name, 1, 1) || '. ' || e.last\_name AS employee\_name,e.salary,d.department\_name

FROM Employees e

JOIN Departments d ON e.department\_id = d.department\_id

WHERE e.salary > (SELECT AVG(e2.salary)

FROM Employees e2

WHERE e2.department\_id = e.department)

ORDER BY d.department\_name, e.last\_name;