# Lab Session 04

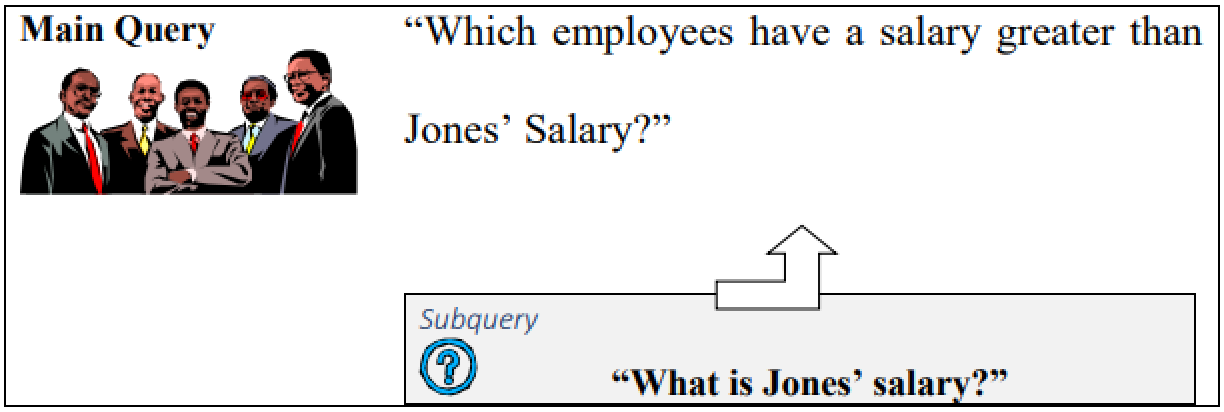
## *(SubQueries, Compound Queries & Joins)*

**By: Ammara Yaseen**

## SubQuery:

In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query.

### Why use subqueries?



The inner query or the subquery returns a value that is used by the outer query or the main query. Using a subquery is equivalent to performing two sequential queries and using the result of the first query as the search value in the second query.

The subquery can be placed in a number of SQL clauses:

- WHERE clause
- HAVING clause
- FROM clause

The syntax of SELECT statement using subqueries is

*SELECT select_list*

*FROM table*

*WHERE expr operator*

*(SELECT select_list FROM table);*

**Note:** In the syntax, operator means comparison operator. Comparison operators fall into two clauses: single-row operators (>, =, >=, <>, <=) and multiple-row operators (IN, ANY, ALL).

**For example**, to display the names of all employees who earn more than employee with number 7566.

***SELECT ename FROM emp WHERE sal > (SELECT sal FROM emp WHERE empno = 7566);***

## Types of Sub queries

**Single-row subquery:** Query that returns only one row from the inner SELECT statement.

**Multiple-row subquery:** Query that returns more than one row form the inner SELECT statement.
**Multiple-column subquery:** Query that returns more than one column from the inner SELECT statement.

***Single-Row Subqueries***

**Examples**

i. To display the employees whose job title is the same as that of employee 7369.

```
SELECT ename, job
        FROM emp
        WHERE job =
                        (SELECT job
                                FROM emp
                                WHERE empno = 7369);
```

ii. To display employees whose job title is the same as that of employee 7369 and whose salary is greater than that of employee 7876.

```
SELECT  ename, job
FROM  emp
WHERE  job =
                (SELECT  job
                FROM  emp
                WHERE  empno = 7369)
AND  sal  >
                (SELECT  sal
                FROM  emp
                WHERE  empno = 7876);
```

iii. We can display data from a main query by using a group function in a subquery to return a single row. e.g. to display the employee name, job title and salary of all employees whose salary is equal to the minimum salary.

```
SELECT ename, job, sal
FROM emp
WHERE sal =
                (SELECT MIN(sal) FROM emp);
```

iv. We can use subqueries not only in the WHERE clause, but also in the HAVING clause. The Oracle server executes the subquery and the results are returned into the HAVING clause of the main query. E.g. to display all departments that have a minimum salary greater than that of department 20.

```
SELECT  deptno, MIN(sal)
FROM  emp
GROUP BY  deptno
HAVING MIN(sal) >
                        (SELECT MIN(sal)
                        FROM emp
                        WHERE  deptno = 20);
```

## Multiple-Row Subqueries

Multiple-row subqueries return more than one row. We use multiple-row operator, instead of a single-row operator, with a multiple-row subquery. The multiple-row operator expects one or more values. Following table illustrates multiple row operators.

| Operator | Meaning |
|---|---|
| IN | Equal to any member in the list |
| ANY | Compare value to each value returned by the subquery |
| ALL | Compare value to every value returned by the subquery |

**Note**: The **NOT** operator can be used with IN, ANY, and ALL operators.

**Examples**

i. Find the employees who earn the same salary as the minimum salary for departments.

```
SELECT  ename, sal, deptno
FROM  emp
WHERE  sal  IN  (SELECT   MIN(sal)
FROM  emp
GROUP  BY  deptno);
```

ii. To display employees whose salary is less than any clerk and who are not clerks.

```
SELECT  empno, ename, job
FROM  emp
WHERE   sal < ANY
(SELECT  sal
FROM  emp
WHERE   job = 'CLERK') AND JOB <> 'CLERK';
```

iii. To display employees whose salary is greater than the average salary of all the departments.

```
SELECT  empno, ename, job
FROM  emp
WHERE  sal > ALL
(SELECT  avg(sal)
FROM  emp
GROUP BY deptno);
```

## Multiple-Column Subqueries

If we want to compare two or more columns, we must write a compound WHERE clause using logical operators. Multiple column subqueries enable us to combine duplicate WHERE conditions into a single WHERE clause.
For example, to display the name of all employees who have done their present job somewhere before in their career.

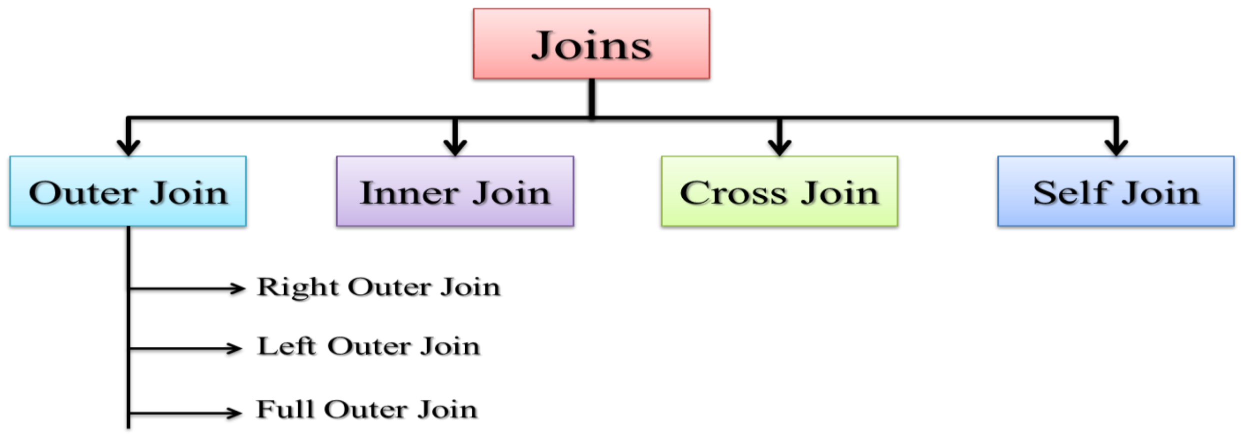
```
SELECT ENAME
FROM EMP
WHERE (EMPNO, JOB)
IN
(SELECT EMPNO, JOB
 FROM JOB_HISTORY)
```

# Joins in Sql

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them.

**For example,** suppose we need a report that displays employee id, name, job and department name. The first three attributes are present in EMP table where as the last one is in DEPT table (see previous lab session).

## Types of Joins



### i. Cross Join OR Cartesian Product:

A Cartesian Product results when all rows in the first table are joined to all rows in the second table. A Cartesian product is formed under following conditions:-

- When a join condition is omitted
- When a join condition is invalid

Consider the following example:-
**SELECT * FROM EMP, DEPT;**
In the above example, if EMP table has 14 rows and DEPT table has 4 rows, then their Cartesian product would generate 14 x 4 = 56 rows.
In fact, the ISO standard provides a special format of the SELECT statement for the Cartesian product:-

SELECT * FROM EMP CROSS JOIN DEPT;

A Cartesian product tends to generate a large number of rows and its result is rarely useful. It is always necessary to include a valid join condition in a WHERE clause. Hence a join is always a subset of a Cartesian product.

## ii. Inner Join/Equi Join

If the join contains an equality condition, it is called equi-join.

**Examples**

i. To retrieve the employee name, their job and department name, we need to extract data from two tables, EMP and DEPT. This type of join is called *equijoin*-that is, values in the DEPTNO column on both tables must be equal. Equijoin is also called *simple join* or *inner join*.

```
SELECT E.ENAME, E.JOB,
D.DNAME FROM EMP E, DEPTD
WHERE E.DEPTNO = D.DEPTNO;
```

The SQL-1999 standard provides the following alternative ways to specify this join:-

```
Select ENAME,JOB,DNAME
FROM EMP NATURAL JOIN DEPT;
```

## iii. Self Join

To find the name of each employee's manager, we need to join the EMP table to itself, or perform a *self join*.

```
SELECT WORKER.ENAME || ‘ works for ‘||
MANAGER.ENAME FROM EMP WORKER, EMP
MANAGER
WHERE WORKER.MGR = MANAGER.EMPNO;
```

## iv. Outer Join

A join between two tables that returns the results of the inner join as well as unmatched rows in the left or right tables is a left or right outer join respectively. A full outer join is a join between two tables that returns the results of a left and right join.

### i. Left Outer Join

```
SELECT E.ENAME, D.DEPTNO,
D.DNAME FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO(+);
```

**NOTE**: The outer join operator appears on only that side that has information missing.

The SQL-1999 standard provides the following alternative way to specify this join:-

```
SELECT E.ENAME, D.DEPTNO, D.DNAME
FROM EMP E LEFT OUTER JOIN
DEPT D ON (E.DEPTNO=D.DEPTNO);
```

### ii. Right Outer Join

```
SELECT E.ENAME, D.DEPTNO,
D.DNAME FROM EMP E, DEPT D
WHERE E.DEPTNO(+) = D.DEPTNO;
```

The SQL-1999 standard provides the following alternative way to specify this join:-

```
SELECT E.ENAME, D.DEPTNO, D.DNAME
FROM EMP E RIGHT OUTER JOIN
DEPT D ON (E.DEPTNO = D.DEPTNO);
```

**NOTE**: In the equi-join condition of EMP and DEPT tables, department OPERATIONS does not appear because no one works in that department. In the outer join condition, the OPERATIONS department also appears.

### iii. Full Outer Join

The SQL-1999 standard provides the following way to specify this join:-

```
SELECT E.ENAME, D.DEPTNO, D.DNAME
FROM EMP E FULL OUTER JOIN
DEPT D ON (E.DEPTNO =
D.DEPTNO);
```

## V. Non EquiJoin

If the join contains inequality condition, it is called non-equijoin. E.g. to retrieve employee name, salary and their grades using *non-equijoins*, we need to extract data from two tables, EMP and SALGRADE.

```
SELECT E.ENAME, E.SAL,
S.GRADE FROM EMP E,
SALGRADE S WHERE E.SAL
BETWEEN S.LOSAL AND S.HISAL;
```

## COMPOUND QUERIES

In SQL, we can use the normal set operators of Union, Intersection and Set Difference to combine the results of two or more component queries into a single result table. Queries containing SET operators are called *compound* queries. The following table shows the different set operators provided in Oracle SQL.

| Operator | Returns |
|---|---|
| UNION | All distinct rows selected by either query |
| UNION ALL | All rows selected by either query including all duplicates |
| INTERSECT | All distinct rows selected by both queries |
| MINUS | All distinct rows that are selected by the first SELECT statement and that are not selected in the second SELECT statement |

## Restrictions on using set Operators

There are restrictions on the tables that can be combined using the set operations, the most important one being that the two tables have to be union-compatible; that is, they have the same structure. This implies that the two tables must contain the same number of columns, and that their corresponding columns contain the same data types and lengths. It is the user's responsibility to ensure that values in corresponding columns come from the same domain. For example, it would not be sensible to combine a column containing the age of staff with the number of rooms in a property, even though both columns may have the same data type i-e NUMBER.

## UNION Operator

The UNION operator returns rows from both queries after eliminating duplicates. By default, the output is sorted in ascending order of the first column of the SELECT clause.

**For example** to display all the jobs that each employee has performed, the following query will be given. (NOTE: If an employee has performed a job multiple times, it will be shown only once)

```
SELECT
EMPNO,JOB FROM
JOB_HISTORY
UNION
SELECT EMPNO,
JOB FROM EMP;
```

## UNION ALL Operator

The UNION ALL operator returns rows from both queries including all duplicates. For example to display the current and previous jobs of all employees, the following query will be given. (NOTE: If an employee has performed a job multiple times, it will be shown separately)

```
SELECT
EMPNO,JOB FROM
JOB_HISTORY
UNION ALL
SELECT EMPNO,
JOB FROM EMP;
```

## INTERSECT Operator

The INTERSECT operator returns all rows that are common to both queries. For example, to display all employees and their jobs those have already performed their present job somewhere else in the past.

```
SELECT
EMPNO,JOB FROM
JOB_HISTORY
INTERSECT
SELECT EMPNO,
JOB FROM EMP;
```

## MINUS Operator

The MINUS operator returns rows from the first query that is not present in the second query. For example to display the ID of those employees whose present job is the first one in their career.

```
SELECT
EMPNO,JOB FROM
JOB_HISTORY
MINUS
SELECT EMPNO,
JOB FROM EMP;
```

# Exercise

1. Write a query to list the name, job name, department name, salary, and grade of the employees according to the department in ascending order.

2. Write a query to list the department name where at least two employees are working.

3. Fetch all the records where salary of employee is less than the lower salary.

4. List department number, Department name for all the departments in which there are no employees in the department.

5. Display employee name, salary, department name where all employees has matching department as well as employee does not have any departments. This query should include non-matching rows.

6. List employee name, salary, department name of all employees whose salary is not within the salary grade 04.

7. Write a query to list the name, job name, annual salary, department id, department name and grade of the employees who earn 60000 in a year or not working as an ANALYST.

8. Write a query to list the employees who are senior to their own manager.

9. List employee name, salary, department name of all employees whose salary is within the salary grade 02.

10. Write a query to list the name, job name, department, salary, and grade of the employees according to the department in ascending order.

11. List all those employees who are working in the same department as their manager (Sub query)

12. Retrieve all employees who are working in department 10 and who earn at least as much as any (i.e. at least one) employee working in department 30.

13. List all employees who are not working in department 30 and who earn more than all employees working in department 30.

14. List all department that have no employee.

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

16. List the job title and total monthly salary for each job except SALES, with a total payroll exceeding $5000. Order the output in descending order of sum of salaries.

17. To display all the employee's name (including KING who has no manager) and their manager name.

18. Create a unique listing of all jobs that in department 30. Include the location of department 30 in the Output.

19. Display the employee name and employee number along with their manager's name Manager Number. Label the columns Employee, Emp# , Manager, and Manager#, respectively.

20. Find the employee name who earn second highest salary.