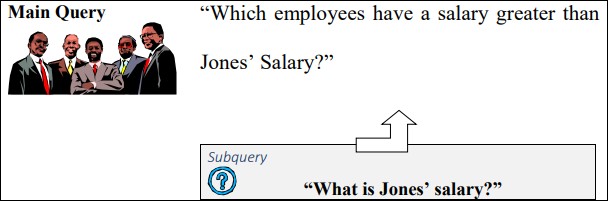
Lab Session 05

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

# 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).

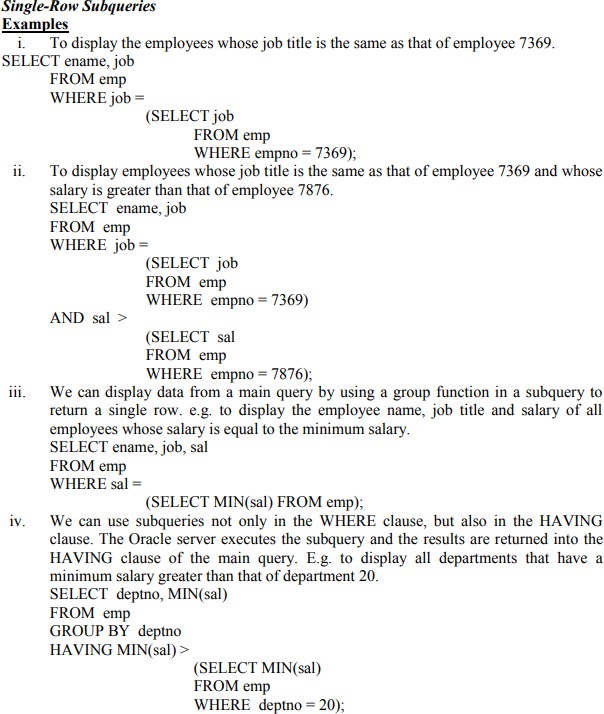
**For example**, to display the names of all employees who earn more than employee with number 7566.

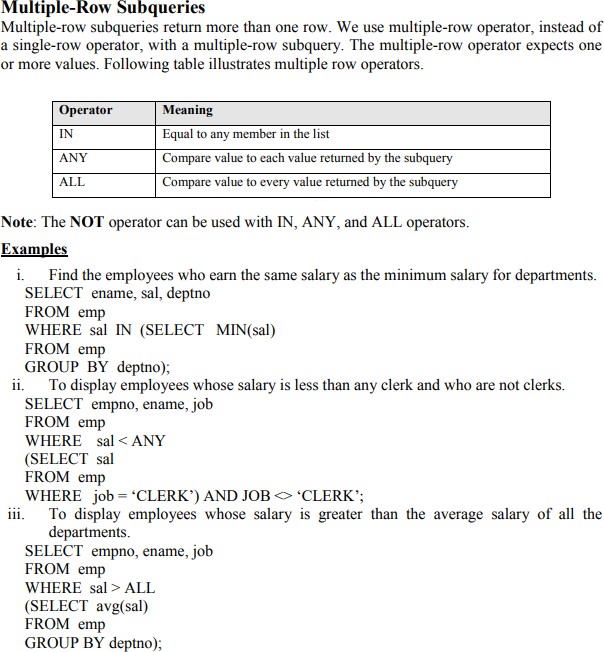
***SELECT ename FROM emp WHERE sal > (SELECT sal FROM emp WHERE empno = 7566);***

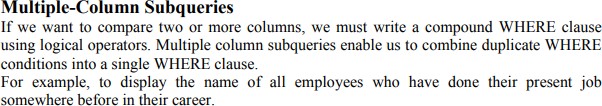
**Types of Sub queries**

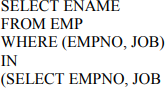
**Single-row subquery:** Query that returns only one row from the inner SELECT statement.

**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.









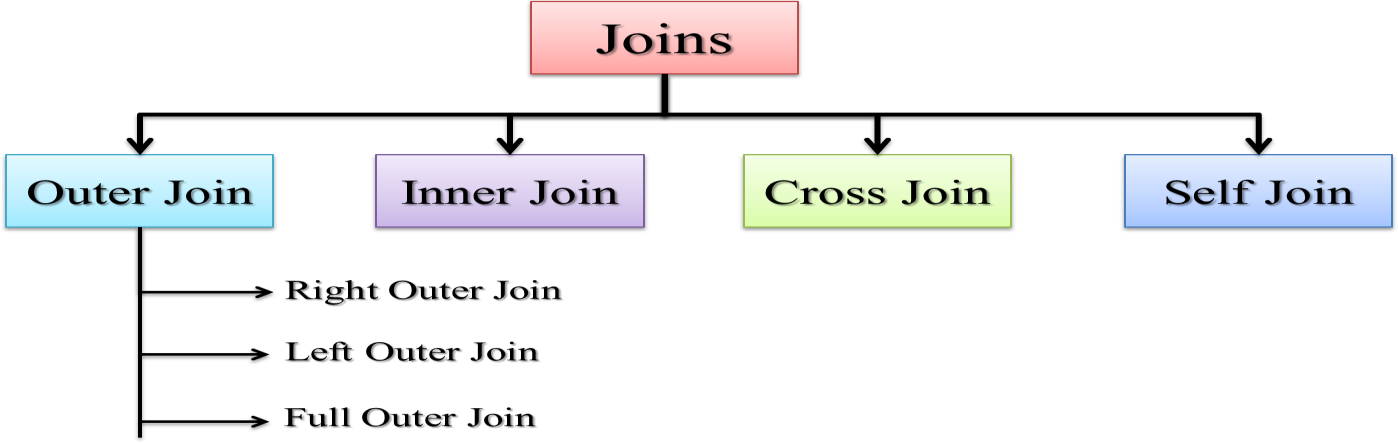
C:\Users\ammara.yaseen\Desktop\Capture.PNG

# 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



### 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.

### 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;

## 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;

## 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.

### 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);

### 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.

### 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);

### 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;