SUBQUERY

Subqueries

Subquery can be simply defined as a query within another query. A Subquery or Inner query or a Nested query is a query within another SQL query.

A subquery may occur in :

- **A SELECT clause**
- **A FROM clause**
- **A WHERE clause**

The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.

A subquery is usually added within the WHERE Clause of another SQL SELECT statement.

You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.

A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.

The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

Subqueries

A subquery is a SELECT statement nested inside another statement such as SELECT, INSERT, UPDATE, or DELETE. Typically, you can use a subquery anywhere that you use an expression.
Oracle allows a maximum nesting of 255 subquery levels in a WHERE clause. There is no limit for nesting subqueries expressed in a FROM clause. In practice, the limit of 255 levels is not really a limit at all because it is rare to encounter subqueries nested beyond three or four levels.

The complete syntax of a subquery is:

```
SELECT [DISTINCT] subquery_select_parameter
FROM {table_name | view_name}
        {table_name | view_name} ...
[WHERE search_conditions]
[GROUP BY column_name [,column_name ] ...]
[HAVING search_conditions]
```

```
SELECT first_name, salary, department_id
FROM employees
WHERE salary = (SELECT MIN (salary)
                FROM employees);
```

```
SELECT first_name, department_id
FROM employees
WHERE department_id IN (SELECT department_id
                        FROM departments
                        WHERE LOCATION_ID = 100)
```

Subqueries

Enlist the names of students who have secured marks more than the average of all students in CPP.

```
select name
from students
where mark> (select avg(mark)
from student
Where subject = 'CPP');
```

Find the Students who are living in the same area as 'Deep'

```
select * from students
where area = (select area
from students
where name = 'Deep');
```

Subqueries

Single Row:

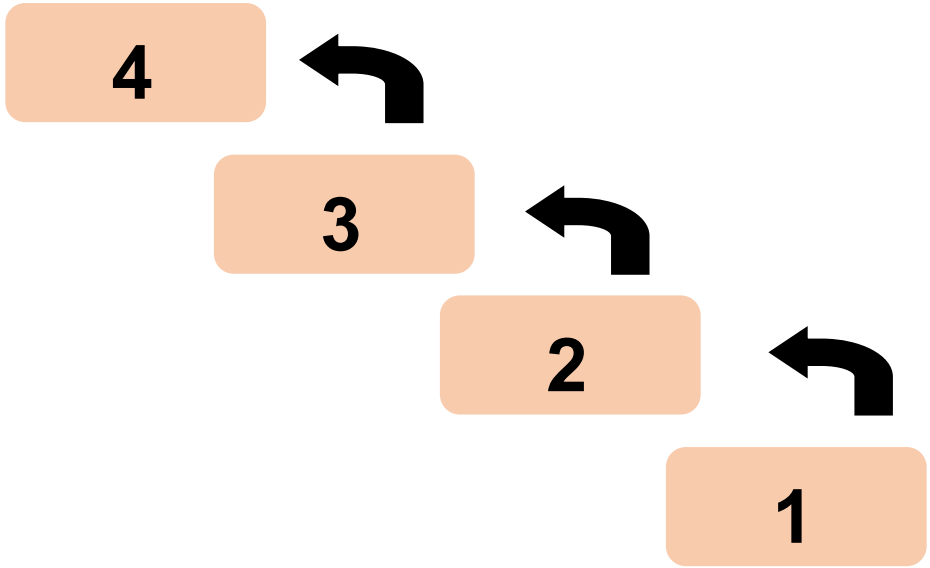
- Sub query which returns single row output.
- A single row subquery returns zero or one row to the outer SQL statement.
- Single row comparison operators i.e. >,= are used with WHERE conditions.

Multiple Row:

- The subquery returns more than one row.
- Multiple row subquery returns one or more rows to the outer SQL statement.
- Multiple row comparison operators like IN, ANY, ALL are used in the comparisons

Subqueries

The basic concept is to pass a single value or many values from the innermost subquery to the next (one level up) query and so on.



When reading or writing SQL sub queries, you should start from the bottom upwards, working out which data is to be passed to the next query up.

SQL Joins

- A Join clause is used to fetch data from two or more data tables, based on join condition
- Join clause is used to combine rows from one (Self-Join) or more tables, based on a related column(s) between them
- In SQL server we have:

Join

An **equi join** is a type of join that combines tables based on matching values in specified columns.

```sql
SELECT *
FROM TableName1, TableName2
WHERE TableName1.ColumnName = TableName2.ColumnName;
-- OR
SELECT *
FROM TableName1
JOIN TableName2
ON TableName1.ColumnName = TableName2.ColumnName;
```

```
SQL> SELECT STUDENT.STUDENTNAME, STUDENT.PALYERID, GAMES.GAMEID, GAMES.NAME
  2  FROM STUDENT,GAMES
  3  WHERE STUDENT.PALYERID=GAMES.GAMEID;

STUDENTNAME            PALYERID     GAMEID NAME
-------------------- ---------- ---------- --------
NINA DOREB                    3          3 CHESS
```

```
SQL> SELECT STUDENT.STUDENTNAME, STUDENT.PALYERID, GAMES.GAMEID, GAMES.NAME
  2  FROM STUDENT
  3   JOIN GAMES
  4  ON STUDENT.PALYERID=GAMES.GAMEID;

STUDENTNAME            PALYERID     GAMEID NAME
-------------------- ---------- ---------- --------
NINA DOREB                    3          3 CHESS
```

Join

A **self JOIN** is a regular join, but the table is joined with itself – this is extremely useful for comparisons within a table.

First Name	Last Name	City
John	Doe	Lahore
Sam	Smith	Karachi
Shawn	Magen	Lahore
Homer	Simpson	Lahore
Bart	Green	Karachi

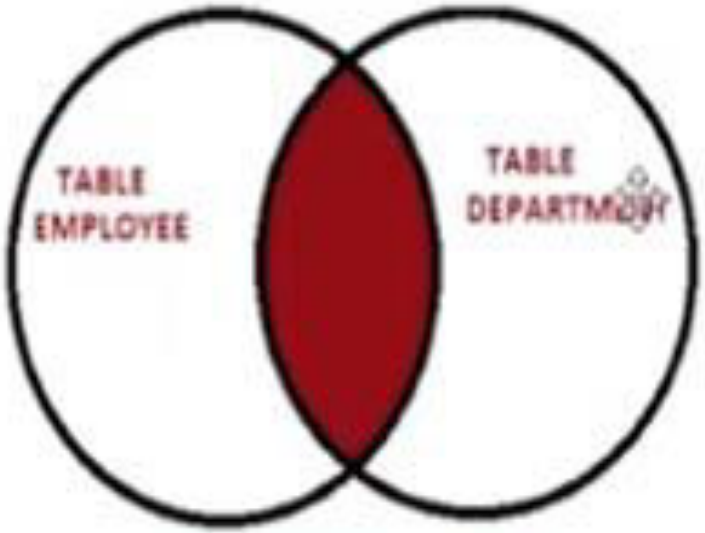
```
SELECT column_names
FROM table1 T1, table1 T2
WHERE condition;
```

```
SELECT A.City,
       B.name AS FirstName1,
       A.name AS FirstName2
  FROM user_info A, user_info B
  WHERE A.name <> B.name
  AND A.city = B.city
  ORDER BY A.city
```

Output

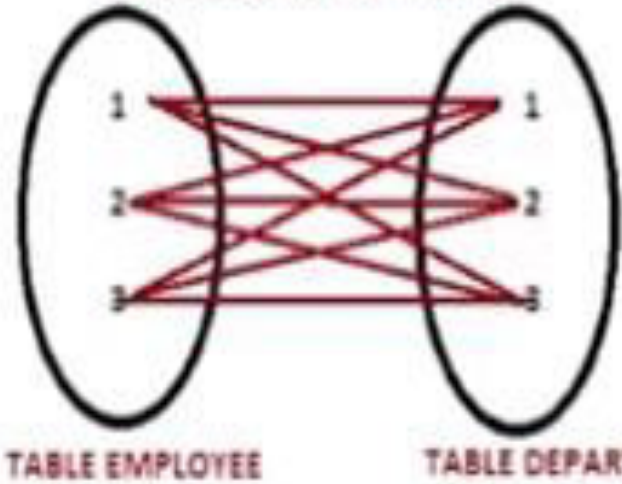
```
City     FirstName1      FirstName2
Karachi  Sam Smith       Bart Green
Karachi  Bart Green      Sam Smith
Lahore   John Doe        Homer Simpson
Lahore   Homer Simpson   John Doe
Lahore   Homer Simpson   Shawn Magen
Lahore   Shawn Magen     John Doe
Lahore   John Doe        Shawn Magen
Lahore   Shawn Magen     Homer Simpson
```

INNER JOIN EXAMPLE

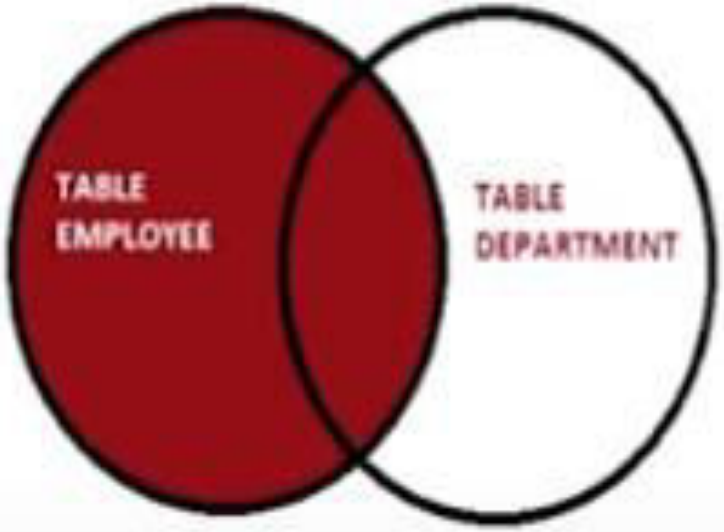


FULL JOIN EXAMPLE

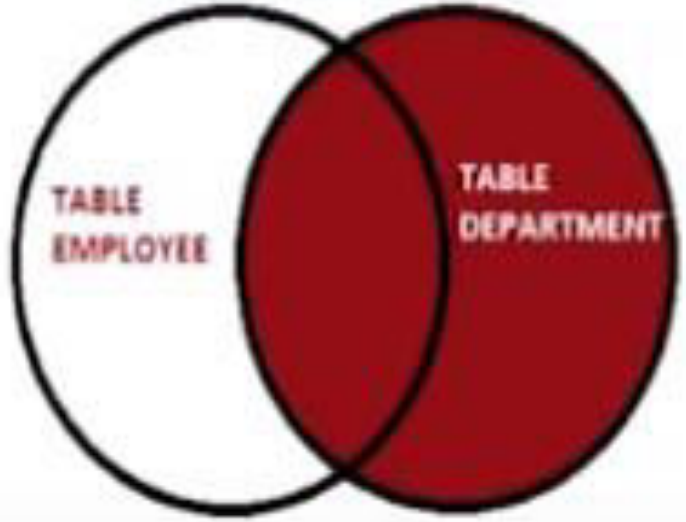
CROSS JOIN EXAMPLE



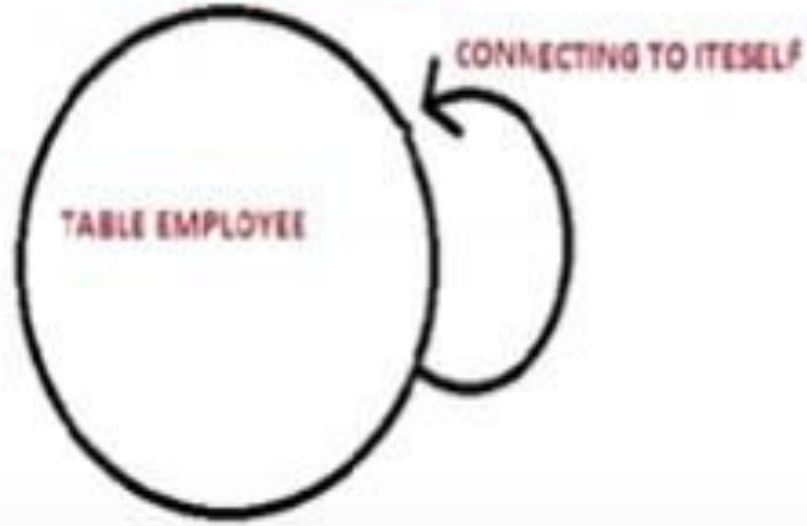
LEFT JOIN EXAMPLE



RIGHT JOIN EXAMPLE



SELF JOIN EXAMPLE



Inner Join

The INNER join fetches records that have matching values in both tables

```
Syntax:

SELECT <column_list>
FROM <table_name1> as t1
INNER JOIN <table_name2> as t2
ON t1.column_name = t2.column_name;
```

LEFT OUTER Join

- The LEFT OUTER Join returns rows to the left (t1) even if there are no rows on the right (t2) of the join clause
- The result is NULL for rows on RIGTH table, when there is no match

```
Syntax:

SELECT <column_list>
FROM <table_name1> as t1
LEFT OUTER JOIN <table_name2> as t2
ON t1.column_name = t2.column_name;
```

RIGHT OUTER Join

- The RIGHT OUTER join returns the rows to the right (t2) relation (table) even if there are no matching rows on the left (t1) relation (table)
- The result is NULL for rows on LEFT table, when there is no match

```
Syntax:

SELECT <column_list>
FROM <table_name1> as t1
RIGHT OUTER JOIN <table_name2> as t2
ON t1.column_name = t2.column_name;
```

FULL OUTER Join

- The FULL OUTER JOIN keyword return all records when there is a match in either left (t1) or right (t2) table records
- If there are rows in "t1" that do not have matches in "t2" or vice-versa, those rows will be listed as well

```
Syntax:

SELECT <column_list>
FROM <table_name1> as t1
FULL OUTER JOIN <table_name2> as t2
ON t1.column_name = t2.column_name;
```