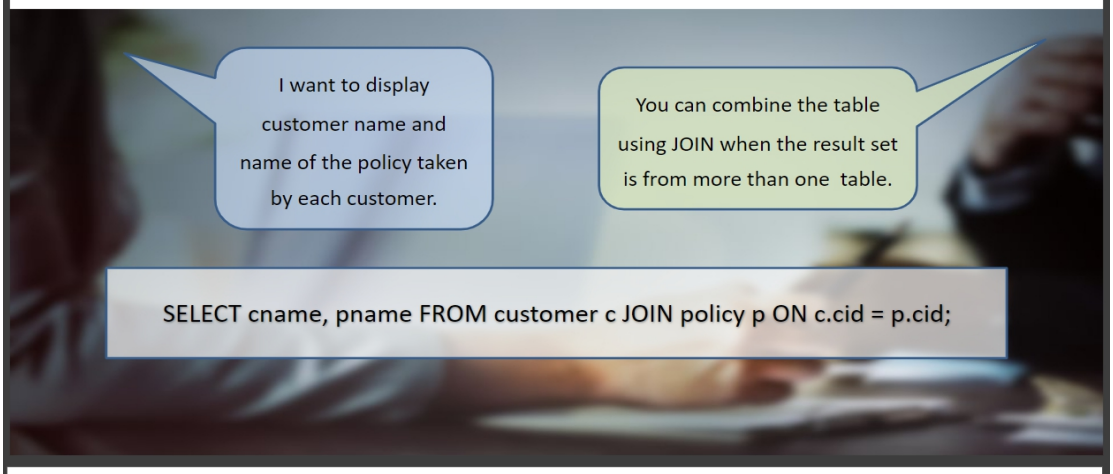
JOINS - INTRODUCTION



JOINS

- A join is a query that combines records from two or more tables.
- A join will be performed whenever multiple tables appear in the FROM clause of the query.
- The select list of the query can select any columns from any of these tables.
- If join condition is omitted or invalid then a **Cartesian product** is formed.

- If any two of these tables have a column name in common, then must qualify these columns throughout the query with table or table alias names to avoid ambiguity.
- Most join queries contain at least one join condition, either in the FROM clause or in the WHERE clause.

CARTESIAN JOINS

Display the details of the customers who have taken the Policy.

CUSTOMER

CID	CNAME
1	Tom
2	John
3	Ram

POLICYENROLLEMENT

EnrollmentID	CID	PID
101	3	MBP
102	1	PP
103	2	PP

```
SELECT c.cid, cname, enrollmentID, p.cid, pid
FROM  customer c, policyenrollment p
Order by c.cid, enrollmentid;
```

OUTPUT

CID	CNAME	ENROLLMENTID	CID	PID
1	Tom	101	3	MBP
1	Tom	102	1	PP
1	Tom	103	2	PP
2	John	101	3	MBP
2	John	102	1	PP
2	John	103	2	PP
3	Ram	101	3	MBP
3	Ram	102	1	PP
3	Ram	103	2	PP

CARTESIAN PRODUCT

- To avoid a Cartesian product, always include a valid join condition.

CUSTOMER

CID	CNAME
1	Tom
2	John
3	Ram

POLICYENROLLEMENT

EnrollmentID	CId	PId
101	3	MBP
102	1	PP
103	2	PP

OUTPUT

CID	CNAME	EnrollmentID	CID	PID
1	Tom	102	1	PP
2	John	103	2	PP
3	Ram	102	3	MBP

```
SELECT c.cid, cname, enrollmentID, p.cid,
pid FROM  customer c, policyenrollment p
WHERE c.cid= p.cid
Order by c.cid;
```

JOIN SYNTAX

- Syntax

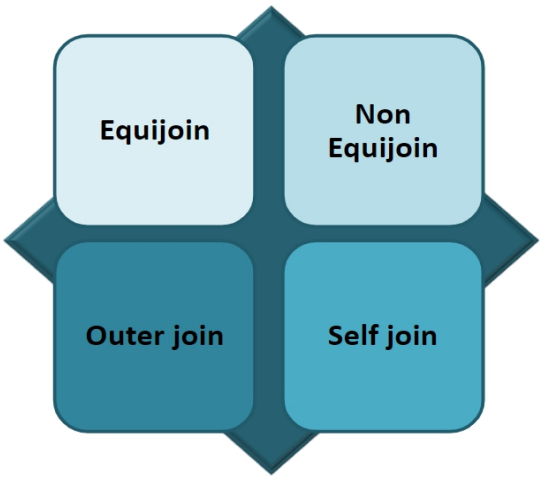
```
SELECT table1.column, table2.column
FROM table1, table2
WHERE table1.column1 = table2.column2;
```

```
SELECT table1.column, table2.column
FROM table1 JOIN table2 ON table1.column1 = table2.column2;
```

ANSI Style join

JOIN and **ON** keyword's are used in the FROM clause of the statement as per **ANSI** standard.

TYPE OF JOINS



EQUIJOIN

What is the join query to display the details of the customers who have taken the Policy?

Equi joins are also called as simple joins or inner joins. When two tables are joined using = operator in the join condition it is called as an equi join.

```
SELECT c.cid, cname, pid FROM customer c, policyenrollment p WHERE c.cid=p.cid;
SELECT c.cid, cname, pid FROM customer c JOIN policyenrollment p ON c.mid=p.mid;
```

c is the alias name for the table customer and p for policyenrollment

Join condition

A non equijoin is a join condition containing something other than an equality operator.

OUTER JOIN

- Equi join or inner join returns rows, only when there is at least one row from both the tables that matches the join condition.
- Outer joins, return all the rows from one of the tables mentioned in the FROM clause even if the condition is not matched.
- Types of outer join
  - Left outer join
  - Right outer join
  - Full outer join

LEFT OUTER JOIN

- The LEFT OUTER JOIN returns all the rows from the left table, with the matching rows of the right table.
- The result is NULL on the right side when there is no match.

Syntax

```
SELECT table1.column, table2.column
FROM table1 LEFT OUTER JOIN table2 ON
table1.column1 = table2.column2;
```

LEFT OUTER JOIN EXAMPLE

•Display the member details along with the book they have taken and the members who have not taken any book

```
SELECT c.cid, cname, pid FROM customer c LEFT OUTER JOIN policyenrollment p ON c.cid=p.cid;
```

CID	CNAME	PID
1	Tom	PP
2	John	PP
3	Ram	MBP
4	Tiny	

RIGHT OUTER JOIN

- The RIGHT OUTER JOIN returns all the rows from the right table, with the matching rows of the left table.
- The result is NULL in the left side when there is no match.

Syntax

```
SELECT table1.column, table2.column
FROM table1 RIGHT OUTER JOIN table2 ON
table1.column1 = table2.column2;
```

RIGHT OUTER JOIN EXAMPLE

•Display the policies that are taken and not taken by customers

```
SELECT p.pid, pname FROM policyenrollment pp RIGHT OUTER JOIN policy p ON pp.pid=p.pid;
```

PID	PNAME
MBP	Money Back Plan
PP	Personal Protect
MBP	Money Back Plan

FULL OUTER JOIN

- The FULL OUTER JOIN returns all rows from both the tables.
- The result is NULL on the left or right side when there is no match.

Syntax

```
SELECT table1.column, table2.column
FROM table1 FULL OUTER JOIN table2 ON
table1.column1 = table2.column2;
```

FULL OUTER JOIN EXAMPLE

•Display the policies that are taken and not taken. Customers who have taken policies and not taken any policies.

```
SELECT c.cid,cname, p.pid, pname FROM policy p FULL OUTER JOIN policyenrollment pp ON p.pid=pp.pid FULL OUTER JOIN customer c ON pp.cid=c.cid;
```

SELF JOIN

▪A self join is a join in which a table is joined with itself (which is also called Unary relationships), especially when the table has a FOREIGN KEY which references it's own PRIMARY KEY.

▪To join a table itself means that each row of the table is combined with itself and with every other row of the table.

▪The self join can be viewed as a join of two copies of the same table.

Query to display the books which are of the same price.

```
SELECT b1.bid,b2.bname,b2.bprice FROM book b1,book b2 WHERE b1.bid!=b2.bid AND
b1.bprice=b2.bprice;
```

NATURAL JOIN

- NATURAL JOIN is a type of EQUI JOIN that compares the common columns of both the tables with each other.
- The associated tables can have one or more pairs of identically named columns.
- The columns must have the same data type.

Check whether common columns exist in both tables before doing a natural join.

```
SELECT cid, cname, pid FROM customer  NATURAL JOIN policyenrollment;
```

JOINS WITH THE USING CLAUSE

The USING clause specifies which columns to test for equality when two tables are joined. It can be used instead of an ON or where clause .

```
SELECT cid, cname, pid FROM customer  JOIN policyenrollment USING (cid);
```

JOINING MULTIPLE TABLES

- SELECT table1.column, table2.column,table3.column FROM table1, table2, table3 WHERE table1.column1 = table2.column2 and table1.column1=table3.column1;
- SELECT table1.column, table2.column, table3.column FROM table1 JOIN table2 ON table1.column1 = table2.column2 JOIN table3 ON table1.column1=table3.column1;

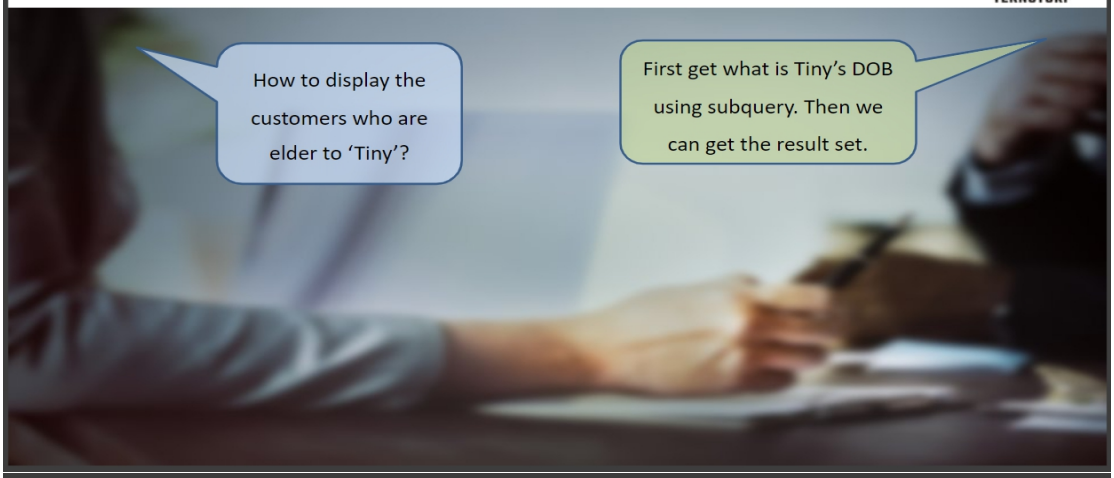
To join n number of tables, you need a minimum of n-1 join conditions. for example, to join three tables, two join conditions are required.

JOINING MULTIPLE TABLES - EXAMPLE

- Query to display policies taken by each customer. Display the policy name, customer name and due date.
- You need to join policy, customer and policyenrollment table. The syntax to join multiple table is:

```
SELECT cname,pname,duedate FROM customer c JOIN policyenrollment p ON c.cid=p.cid
JOIN policy pp ON pp.pid=p.pid;
```

SUBQUERY



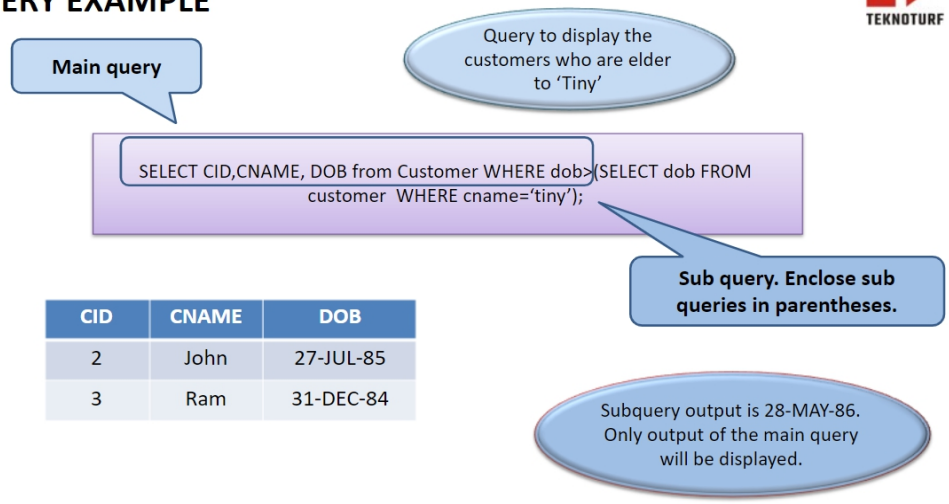
SUBQUERY

- A subquery is a SELECT statement that is embedded in a clause of another SQL statement.
- They can be very useful to select rows from a table with a condition that depends on the data in the same or another table.
- The subquery can be placed in the following SQL clauses:
  - The WHERE clause.
  - The HAVING clause.
  - The FROM clause.

SYNTAX for using SUBQUERY in WHERE clause

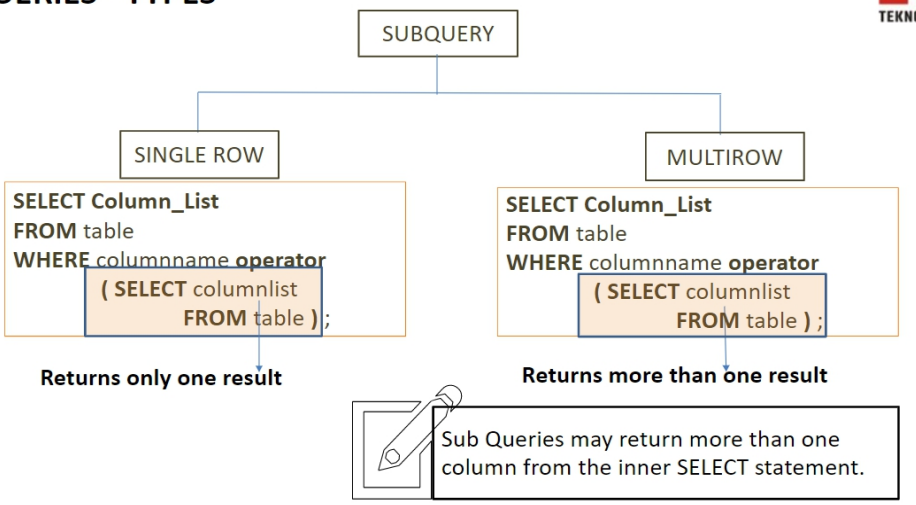
```
SELECT Column_List
FROM table
WHERE columnname operator
    ( SELECT columnlist
      FROM table ) ;
```

SUBQUERY EXAMPLE



CID	CNAME	DOB
2	John	27-JUL-85
3	Ram	31-DEC-84

SUBQUERIES - TYPES



OPERATORS IN SUBQUERY

Single Row Subquery

- Returns only one row
- Operators used are:

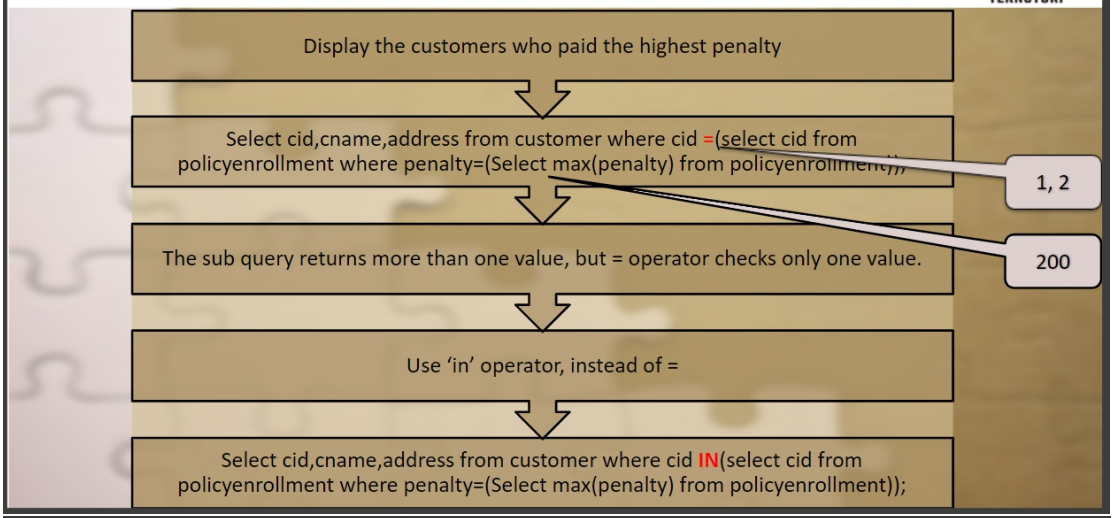
Operator	Meaning
=	Equal to
<	Less than
<=	Less than equal to
>	Greater than
>=	Greater than equal to
<>	Not equal to

Multiple Row Subquery

- Returns more than one row
- Operators used are:

Operator	Meaning
in	Equal to any member in the list
all	Compare to each value returned by the subquery
any	Compare to any value returned by the subquery

MULTIROW SUBQUERY - EXAMPLE



MULTI ROW OPERATORS - IN, ANY AND ALL

Combinations of ANY

- <ANY means - less than any one of the available values.
- >ANY means - more than any one of the available values.
- =ANY is equivalent to IN.

Combinations of ALL

- <ALL means less than all available values.
- >ALL means more than all available values.

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

ANY OPERATOR - EXAMPLE

•Display the customers who have a greater penalty than the penalty paid by customer 4 or 3(cid).

```
SELECT cid,pid,penalty FROM policyenrollment WHERE penalty >ANY(SELECT penalty from
policyenrollment WHERE mid IN(4,3));
```

70, 150

CID	PID	PENALTY
1	PP	200
2	MBP	120
2	PP	200

ALL OPERATOR – EXAMPLE

•Display the customers who have a greater penalty than the penalty paid by both customers whose ID is 4 and 3.

```
SELECT cid,pid,penalty FROM policyenrollment WHERE penalty >=ALL(SELECT penalty FROM
policyenrollment WHERE mid IN(4,3));
```

70, 150

CID	PID	PENALTY
1	PP	200
2	PP	200

SUBQUERY IN THE SELECT LIST

•Query to display the cid, cname and the number of policies they have taken.

```
SELECT cid,cname,(SELECT count(*) FROM policyenrollment p WHERE p.cid=c.cid) as cnt FROM
customer c;
```

Subquery used SELECT clause must to provided with an alias name.

SUBQUERY AS A TABLENAME

•When you insert a select statement into a FROM clause, it becomes a subquery.

•The subquery returns a temporary table in the database server's memory and then it is used by the outer query for further processing.

•Display the policies that have been taken the maximum number of times.

```
SELECT pid,count(*) maxcount FROM policyenrollment GROUP BY pid
    HAVING COUNT(*)=(SELECT MAX(cnt) maxcnt FROM(SELECT
     pid,count(*) cnt FROM policyenrollment GROUP BY pid));
```

Count(*) will count all the rows

Correlated Subquery

•Correlated subquery is a subquery that uses values from the outer query.

•The subquery is evaluated once for each row processed by the outer query.

•Query to display the policies of the same minimum amount to be paid per month

```
SELECT p.pid,p.pname,p.MinAmount FROM policy p WHERE
p.Minamount=(SELECT p1.minamount FROM policy p1 WHERE
    p1.minamount=p.minamount AND p1.pid!=p.pid);
```

SUBQUERY - INSERT

•All the records in the policyenrollment need to be copied into the policyenrollment_history table.

```
INSERT INTO policyenrollment_history SELECT * FROM policyenrollment;
```

Ensure you have created the **policyenrollment_history** table before inserting.

SUBQUERY- CREATE

•Is there any way to copy the records without creating the policyenrollment_history table?

```
CREATE TABLE policyenrollment_history AS SELECT * FROM policyenrollment;
```

SUBQUERY- DELETE

TEKNOTURF

•What is the query to delete all the policyenrollment made by 'John'?

DELETE FROM policyenrollment WHERE cid=(SELECT cid FROM customer WHERE cname='John');