**JOINS:**

Joins are used to retrieve data from multiple tables at a time. In relational databases we are storing related data in multiple tables like employee details, department details, customer details, orders details, products details,....... etc.

To combined data and retrieve data from those multiple tables then we need joins.

Types of joins:

1.Inner Join

2.Outer Join

a) Left Outer Join

b) Right Outer Join

c) Full Outer Join

3. Cross Join (Or) Cartesian Join

4. Natural Join

**JOINS TABLES:**

EX:

STUDENT

STID SNAME CID

1021 SAI 10

1022 ADAMS 20

1023 JONES 30

COURSE

CID CNAME CFEE

10 ORACLE 2500

20 JAVA 6000

40 PHP 4500

**EQUI JOIN:**

- retrieving data from multiple tables based on "equal operator ( = ) " is called as equi join.

- when we use equi join between two (or) more than two tables there must be common column (or) common field name is no need to be same name (but recommend)

- common column (or) common field datatype must be match.

- when we perform any join operation between tables there is no need to have relationship(optional).(i.e primary key & foreign key relation)

- equi join always retrieving only matching data / matching rows.

SYNTAX:

---------------

WHERE <TABLE NAME1>.<COMMON COLUMN> = <TABLE NAME2>.<COMMON COLUMN>(OR)

WHERE <TN1 ALIAS NAME>.<COMMON COLUMN> = <TN2 ALIAS NAME>.<COMMON COL>;

EX1:

Write a query to retrieve student and the corresponding course details from

student,course tables by using equi join ?

SOL:

SQL> SELECT \* FROM STUDENT,COURSE WHERE CID=CID;

ERROR at line 1:

ORA-00918: column ambiguously defined

NOTE: In above example we get an error is "column ambiguously defined".to over come this error then we should use a table name as an identityto ambiguous column CID like below,

SOL:

SQL> SELECT \* FROM STUDENT,COURSE WHERE STUDENT.CID=COURSE.CID;

(OR)

SQL> SELECT \* FROM STUDENT S,COURSE C WHERE S.CID=C.CID;

**RULE OF JOIN:**

A row in a first table is comparing the given join condition with all rows of second table.

EX2:

WAQ TO RETRIEVE STUDENT,COURSE DETAILS FROM TABLES IF CID IS 20 ?

SOL:

SQL> SELECT \* FROM STUDENT S,COURSE C WHERE S.CID=C.CID AND C.CID=20;

EX3:

WAQ TO RETRIEVE LIST OF EMPLOYEE FROM EMP,DEPT TABLES BY USING EQUI JOIN WHO ARE WORKING IN THE LOCATION IS 'CHICAGO' ?

SOL:

SQL> SELECT \* FROM EMP E,DEPT D WHERE E.DEPTNO=D.DEPTNO AND LOC='CHICAGO';

EX4:

WAQ TO DISPLAY SUM OF SALARIES OF DEPARTMENTS FROM EMP,DEPT TABLES

BY USING EQUI JOIN ?

SOL:

SQL> SELECT DNAME,SUM(SAL) FROM EMP E,DEPT D WHERE E.DEPTNO=D.DEPTNO GROUP BY DNAME;

EX5:

WAQ TO DISPLAY SUM OF SALARIES OF DEPARTMENTS FROM EMP,DEPT TABLES

BY USING EQUI JOIN IF SUM OF SALARIES OF DEPARTMENTS ARE MORE THAN

10000?

SOL:

SQL> SELECT DNAME,SUM(SAL) FROM EMP E,DEPT D WHERE E.DEPTNO=D.DEPTNO GROUP BY DNAME HAVING SUM(SAL)>10000;

**INNER JOIN:**

Inner join is similar to equi join but retrieving data from multiple tables with "on" clause condition.

SYNTAX:

ON <TABLE NAME1>.<COMMON COLUMN> = <TABLE NAME2>.<COMMON COLUMN>;

(OR)

ON<TN1 ALIAS NAME>.<COMMON COLUMN> = <TN2 ALIAS NAME>.<COMMON COLUMN>;

EX1:

WAQ TO RETRIEVE STUDENT,COURSE DETAILS FROM TABLES BY USING INNER JOIN?

SOL:

SQL> SELECT \* FROM STUDENT INNER JOIN COURSE ON STUDENT.CID=COURSE.CID;

EX2:

WAQ TO DISPLAY EMPLOYEE FROM EMP,DEPT TABLES BY USING INNER JOIN WHO ARE WORKING IN THE LOCATION IS "CHICAGO" ?

SOL:

SQL> SELECT \* FROM EMP E INNER JOIN DEPT D ON E.DEPTNO=D.DEPTNO AND LOC='CHICAGO';

(OR)

SQL> SELECT \* FROM EMP E INNER JOIN DEPT D ON E.DEPTNO=D.DEPTNO

WHERE LOC='CHICAGO';

**OUTER JOINS:**

In the above inner join we are retrieving only matching rows but not un matching rows from multiple tables.so to overcome this problem then we use "outer joins" mechanism.

These are again three types:

1. Left outer join

2. Right outer join

3. Full outer join

**LEFT OUTER JOIN:**

Retrieving all rows(matching & un matching) from left side table, but retrieving matching rows from right side table.

SQL> SELECT \* FROM <TN1 S> LEFT OUTER JOIN <TN2 C> ON S.CID=C.CID;

**RIGHT OUTER JOIN:**

Retrieving all rows(matching & un matching) from right side table but retrieving matching rows from left side table.

SQL> SELECT \* FROM STUDENT S RIGHT OUTER JOIN COURSE C ON S.CID=C.CID;

**FULL OUTER JOIN**:

Retrieving matching and also un matching rows from both sides tables.

SELECT \* FROM STUDENT S FULL OUTER JOIN COURSE C ON S.CID=C.CID;

**NON-EQUI JOIN:**

Retrieving data from multiple tables based on any condition except equal operator condition is called as non-equi join. In this join we can use the following operators are <,>,<=,>=,and,between,.........etc.

EX1:

SQL> SELECT \* FROM TEST1 T1,TEST2 T2 WHERE T1.SNO>T2.SNO;

(OR)

SQL> SELECT \* FROM TEST1 T1 JOIN TEST2 T2 ON T1.SNO>T2.SNO;

EX2:

WAQ TO DISPLAY ENAME,SALARY,LOW SALARY,HIGH SALARY FROM EMP,SALGRADE

TABLES WHOSE SALARY BETWEEN LOW SALARY AND HIGH SALARY ?

SOL:

SQL> SELECT ENAME,SAL,LOSAL,HISAL FROM EMP,SALGRADE

WHERE SAL BETWEEN LOSAL AND HISAL;(OR)

SQL> SELECT ENAME,SAL,LOSAL,HISAL FROM EMP,SALGRADE WHERE (SAL>=LOSAL) AND (SAL<=HISAL);

**CROSS JOIN / CARTISEAN JOIN:**

Joining Two (Or) More Than Two Tables Without Any Condition Is Called As "Cross / Cartesian Join.

In cross join, each row of the first table will join joins with each row of the second table. That means a first table is having "m" no.of rows and a second table is having "n" no.of rows then the result is mxn no.of rows.

EX1:

SQL> SELECT \* FROM STUDENT CROSS JOIN COURSE;

(OR)

SQL> SELECT \* FROM STUDENT,COURSE;

**NATURAL JOIN:**

Natural join is similar to equi join. when we use natural join we should have a common column name. this column datatype must be match.

- whenever we are using natural join there is no need to write a joining condition by explicitly because internally oracle server is preparing joining condition based on an "equal operator(=)" with column name automatically.

- by using natural join we avoid duplicate columns while retrieving data from multiple tables.

EX:

SQL> SELECT \* FROM STUDENT S NATURAL JOIN COURSE C;

**SELF JOIN:**

- Joining a table by itself is called as self join. In self join a row in one table joined with the row of same table.

- When we use self join mechanism then we should create alias names on a table. Once we create alias name on a table internally oracle server is creating virtual table(copy) on each alias name.

- we can create any no.of alias names on a single table by each alias name should be different name.

- self join can be implemented at two situations:

1. Comparing a sinlge column values by itself in the table.

2. Comparing two different columns values to each other in the table.

Eg. On Comparing A Single Column Values By Itself:

Q: Waq To Display Employee Who Are Working In The Same Location Of The Employee Is "SCOTT" ?

SOL:

SQL> SELECT T1.ENAME,T1.LOC FROM TEST T1,TEST T2 WHERE T1.LOC=T2.LOC

AND T2.ENAME='SCOTT';

EX.ON COMPARING TWO DIFF.COLUMNS TO EACH OTHER:

EX1:

WAQ TO DISPLAY MANAGERS AND THEIR EMPLOYEES FROM

EMP TABLE?

SSQL> SELECT M.ENAME MANAGER,E.ENAME EMPLOYEES FROM EMP E,EMP M

WHERE M.EMPNO=E.MGR;

**UNION:**

Joins horizontally combine results from different tables. If you instead would like to vertically concatenate columns, you can do so with a union.

SELECT column1, column2 FROM table1

UNION

SELECT column3, column4 FROM table2;

Here are some rules for using the UNION operator:

* **Same number of columns**: The SELECT statements must have the same number of columns.
* **Compatible data types:** The corresponding columns in each SELECT statement must have compatible data types.
* **Column names**: The column names of the second query will determine the column names of the final result set.
* **Execution order**: The UNION operator executes the SELECT statements independently and combine the result sets.
* **Sorting result set**: Use the [ORDER BY](https://www.sqltutorial.org/sql-order-by/) clause in the second query to sort the rows in the final result set.

**Effective Query:**

Effective queries are designed to retrieve the desired data efficiently and accurately while minimizing resource usage and execution time.

**1.show\_amount\_of\_data\_scanned()**

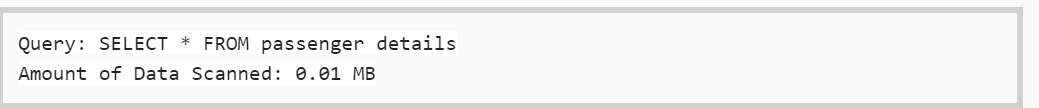
This function is used to display the amount of data that a query scans during its execution. This is particularly useful for understanding the performance and cost implications of your queries, especially in environments where data scanning directly impacts billing (e.g., BigQuery).

* **Purpose**: To provide insights into the volume of data processed by a query.
* **Usage**: Helps in optimizing queries by identifying large data scans and reducing unnecessary data processing.

query = "SELECT \* FROM passenger details`"

show\_amount\_of\_data\_scanned(query);

OUTPUT:



query = "SELECT passenger Id, name, cost FROM passenger details`"

show\_amount\_of\_data\_scanned(query);



**2.show\_time\_to\_run()**

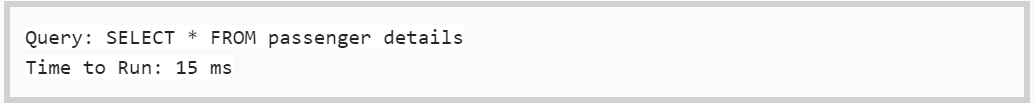
This function prints the time it takes for a query to execute. It is useful for performance monitoring and optimization.

* **Purpose**: To measure the execution time of a query.
* **Usage**: Helps in identifying slow queries and optimizing them for better performance.

**eg**

query = "SELECT \* FROM passenger details`"

show\_time\_to\_run(query)



query = "SELECT Passenger id, name, cost FROM passenger details`"

show\_time\_to\_run(query)



**Some Effective Queries**

**1. Select Only Necessary Columns**

Avoid using SELECT \* as it retrieves all columns, which can be inefficient. Instead, specify only the columns you need:

SELECT name, age FROM users;

**2. Use Indexes**

Indexes can significantly speed up query performance by allowing the database to quickly locate the data:

CREATE INDEX idx\_users\_name ON users(name);

**3. Filter Data Early**

Use WHERE clauses to filter data as early as possible in your query to reduce the amount of data processed:

SELECT name, age FROM users WHERE age > 30;

**4. Avoid Complex Joins**

While joins are necessary, try to avoid overly complex joins that can slow down query performance. Simplify joins where possible:

SELECT u.name, o.order\_date

FROM users u

JOIN orders o ON u.id = o.user\_id;

**5. Use Subqueries and CTEs Wisely**

Common Table Expressions (CTEs) and subqueries can make queries more readable and maintainable, but they should be used judiciously to avoid performance hits:

WITH RecentOrders AS (

    SELECT user\_id, order\_date

    FROM orders

    WHERE order\_date > '2025-01-01')

SELECT u.name, r.order\_date

FROM users u

JOIN RecentOrders r ON u.id = r.user\_id;

**6. Optimize Aggregations**

When using aggregate functions, ensure they are optimized by grouping only necessary columns and using indexes:

SELECT department, COUNT(\*) as employee\_count

FROM employees

GROUP BY department;

**7. Limit Results**

Use LIMIT to restrict the number of rows returned, especially in large datasets:

SELECT name, age FROM users LIMIT 10;

**8. Avoid Redundant Data**

Ensure your database schema is normalized to avoid redundant data, which can slow down queries and increase storage requirements.