Experiment No: 5

Title : Implementation of different types of Joins

- Inner Join
- Outer Join
- Natural Join..etc

Objective :

To implement different types of joins

Theory :

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.The join is actually performed by the 'where' clause which combines specified rows of tables.

Syntax:

```
SELECT column 1, column 2, column 3...
FROM table_name1, table_name2
  WHERE table_name1.column name = table_name2.columnname;
```

Types of Joins :

1. Simple Join
2. Self Join
3. Outer Join

Simple Join:

It is the most common type of join. It retrieves the rows from 2 tables having a common column and is further classified into

Equi-join :

A join, which is based on equalities, is called equi-join.

Example:

```
Select * from item, cust where item.id=cust.id;
```

In the above statement, item-id = cust-id performs the join statement. It retrieves rows from both the tables provided they both have the same id as specified by the where clause. Since the where clause uses the comparison operator (=) to perform a join, it is said to be

equijoin. It combines the matched rows of tables. It can be used as follows:

- ✔ To insert records in the target table.
- ✔ To create tables and insert records in this table.
- ✔ To update records in the target table.
- ✔ To create views.

Non Equi-join:

It specifies the relationship between columns belonging to different tables by making use of relational operators other than'='.

Example:

Select * from item, cust where item.id<cust.id;

Table Aliases
Table aliases are used to make multiple table queries shorter and more readable. We give an alias name to the table in the 'from' clause and use it instead of the name throughout the query.

Self join:

Joining a table to itself is known as self-join. It joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table.

Example:

select * from emp x ,emp y where x.salary >= (select avg(salary) from x.emp where x. deptno =y.deptno);

Outer Join:

It extends the result of a simple join. An outer join returns all the rows returned by simple join as well as those rows from one table that do not match any row from the table. The symbol(+) represents outer join.

- Left outer join
- Right outer join
- Full outer join

LAB PRACTICE ASSIGNMENT:

Consider the following schema:

Sailors (sid, sname, rating, age)

Boats (bid, bname, color)

Reserves (sid, bid, day(date))

1. Find all information of sailors who have reserved boat number 101.

```
mysql> SELECT Sailors.*
    -> FROM Sailors
    -> JOIN Reserves ON Sailors.s_id = Reserves.s_id
    -> WHERE Reserves.b_id = 101;
+------+--------+--------+------+
| s_id | s_name | rating | age  |
+------+--------+--------+------+
|    1 | Alice  |      5 |   30 |
+------+--------+--------+------+
1 row in set (0.00 sec)
```

2. Find the name of the boat reserved by Bob.

```
mysql> SELECT Boats.b_name
    -> FROM Sailors
    -> JOIN Reserves ON Sailors.s_id = Reserves.s_id
    -> JOIN Boats ON Reserves.b_id = Boats.b_id
    -> WHERE Sailors.s_name = 'Bob';
+-----------+
| b_name    |
+-----------+
| Blue Wave |
+-----------+
1 row in set (0.01 sec)
```

3. Find the names of sailors who have reserved a red boat, and list in the order of age.

```
mysql> SELECT Sailors.s_name
    -> FROM Sailors
    -> JOIN Reserves ON Sailors.s_id = Reserves.s_id
    -> JOIN Boats ON Reserves.b_id = Boats.b_id
    -> WHERE Boats.color = 'Red'
    -> ORDER BY Sailors.age;
+--------+
| s_name |
+--------+
| Alice  |
+--------+
1 row in set (0.00 sec)
```

4. Find the names of sailors who have reserved at least one boat.

```
mysql> SELECT DISTINCT Sailors.s_name
    -> FROM Sailors
    -> JOIN Reserves ON Sailors.s_id = Reserves.s_id;
+---------+
| s_name  |
+---------+
| Alice   |
| Bob     |
| Charlie |
| David   |
| Eve     |
+---------+
5 rows in set (0.00 sec)
```

5. Find the ids and names of sailors who have reserved two different boats on the same day.

```
mysql> SELECT Sailors.s_id, Sailors.s_name
    -> FROM Sailors
    -> JOIN Reserves ON Sailors.s_id = Reserves.s_id
    -> WHERE (Reserves.s_id, Reserves.day) IN (
    ->     SELECT s_id, day
    ->     FROM Reserves
    ->     GROUP BY s_id, day
    ->     HAVING COUNT(DISTINCT b_id) > 1
    -> );
Empty set (0.00 sec)
```

6. Find the ids of sailors who have reserved a red boat or a green boat.

```
mysql> SELECT DISTINCT Sailors.s_id
    -> FROM Sailors
    -> JOIN Reserves ON Sailors.s_id = Reserves.s_id
    -> JOIN Boats ON Reserves.b_id = Boats.b_id
    -> WHERE Boats.color IN ('Red', 'Green');
+------+
| s_id |
+------+
|    1 |
|    3 |
+------+
2 rows in set (0.00 sec)
```

7. Find the name and the age of the youngest sailor.

```
mysql> SELECT s_name, age
    -> FROM Sailors
    -> ORDER BY age ASC
    -> LIMIT 1;
+--------+------+
| s_name | age  |
+--------+------+
| Eve    |   28 |
+--------+------+
1 row in set (0.00 sec)
```

8. Count the number of different sailor names.

```
mysql> SELECT COUNT(DISTINCT s_name) AS num_different_names
    -> FROM Sailors;
+---------------------+
| num_different_names |
+---------------------+
|                   5 |
+---------------------+
1 row in set (0.00 sec)
```

9. Find the average age of sailors for each rating level.

```
mysql> SELECT rating, AVG(age) AS average_age
    -> FROM Sailors
    -> GROUP BY rating;
+--------+-------------+
| rating | average_age |
+--------+-------------+
|      5 |     29.0000 |
|      7 |     45.0000 |
|      6 |     35.0000 |
|      8 |     40.0000 |
+--------+-------------+
4 rows in set (0.00 sec)
```

10. Find the average age of sailors for each rating level that has at least two sailors.

```
mysql> SELECT rating, AVG(age) AS average_age
    -> FROM Sailors
    -> GROUP BY rating
    -> HAVING COUNT(s_id) >= 2;
+--------+-------------+
| rating | average_age |
+--------+-------------+
|      5 |     29.0000 |
+--------+-------------+
1 row in set (0.00 sec)
```