Lab Manual for Database Management System

Lab No. 4

SQL Joins

LAB 4: SQL JOINS

1. INTRODUCTION:

In relational databases, such as SQL Server, Oracle, MySQL, and others, data is stored in multiple tables that are related to each other with a common key value. Accordingly, there is a constant need to extract records from two or more tables into a results table based on some condition. In SQL Server, this can be easily accomplished with the SQL JOIN clause.

JOIN is an SQL clause used to query and access data from multiple tables, based on logical relationships between those tables.

In other words, JOINS indicate how SQL Server should use data from one table to select the rows from another table.

2. OBJECTIVE:

After completing this lab the student should be able to:

a. Understand the concept of joins.
b. Use joins on multiple tables.
c. Put the information from two tables together into one result set.

3. THEORY

A join condition defines the way two tables are related in a query by:

- Specifying the column from each table to be used for the join. A typical join condition specifies a foreign key from one table and its associated key in the other table.

- Specifying a logical operator (for example, = or <>,) to be used in comparing values from the columns

There are various forms of the JOIN clause. These will include:

i. INNER JOIN

ii. OUTER JOIN

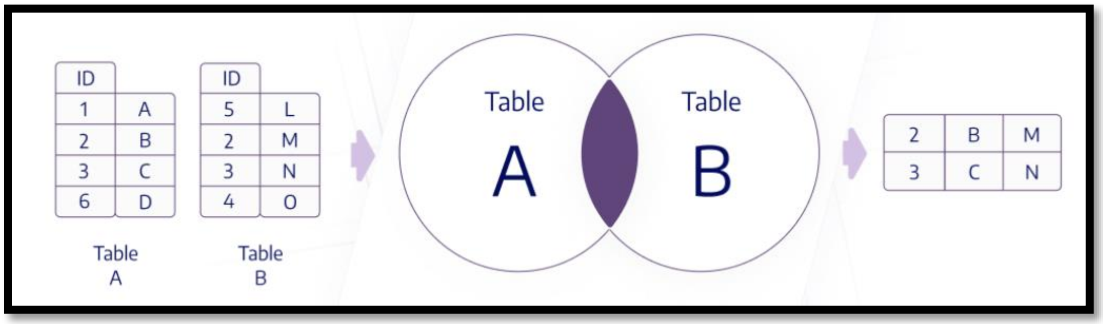
(Left, Right and Full)

iii. CROSS JOIN

iv. SELF JOIN

i. The INNER Join

The **INNER JOIN** is used to fetch records that have matching values in two or more tables. Hence, the result table is created containing matching rows in all these tables.



Syntax →

```
SELECT column1, column2, ...
FROM table_1
INNER JOIN tabel_2
ON table1.column_name = table2.column_name;
```

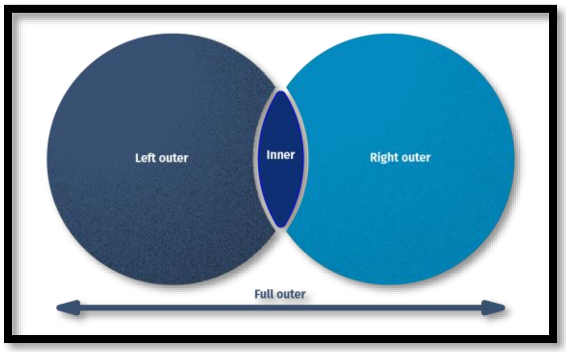
Example

We will join the Products table with the Categories table, by using the CategoryID field from both tables:

```
SELECT ProductID, ProductName, CategoryName
FROM Products
INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID;
```

ii. The OUTER Join

When applying an SQL INNER JOIN, the output returns only matching rows from the stated tables. In contrast, if you use an SQL OUTER JOIN, it will retrieve not only the matching rows but also the unmatched rows as well.



The outer join is further divided into following:

1) ***LEFT OUTER JOIN:*** The LEFT OUTER JOIN gives the output of the matching rows between both tables. In case, no records match from the left table, it shows those records with null values.

Syntax →

```
SELECT column_name(s)
FROM table_1
LEFT JOIN  table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Example

The following SQL statement will select all customers, and any orders they might have:

```
SELECT Customers.ContactName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers. ContactName;
```

2) ***RIGHT OUTER JOIN:*** The RIGHT OUTER JOIN works by the same principle as the LEFT OUTER JOIN. The RIGHT OUTER JOIN selects data from the right table and matches this data with the rows from the left table. The RIGHT JOIN returns a result set that includes all rows in the right table, whether or not they have matching rows from the left table. In case, a row in the right table does not have any matching rows in the left table, the column of the left table in the result set will have nulls.

Syntax →

```
SELECT column_name(s)
FROM table_1
RIGHT JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Example

The following SQL statement will return all employees, and any orders they might have placed:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

3) ***FULL OUTER JOIN:*** The FULL OUTER JOIN returns a result that includes rows from both left and right tables. In case, no matching rows exist for the row in the left table, the columns of the right table will have nulls. Correspondingly, the column of the left table will have nulls if there are no matching rows for the row in the right table.

Syntax →

```
SELECT column_name(s)
FROM table_1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

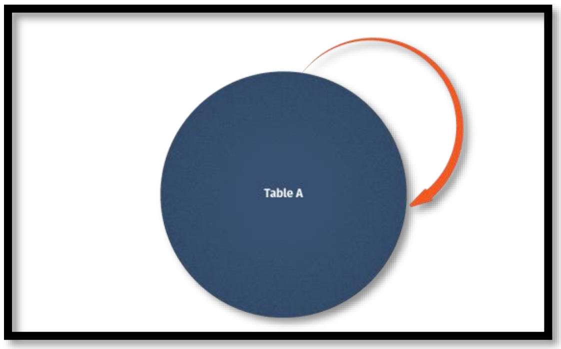
Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.ContactName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.ContactName;
```

iii. SELF JOIN

The SELF JOIN allows you to join a table to itself. This implies 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. The table is not actually copied, but SQL performs the command as though it were. This is accomplished by using table name aliases to give each instance of the table a separate name. It is most useful for extracting hierarchical data or comparing rows within the same table.



Syntax →

```
SELECT column_names(s)
FROM table1 T1, table1 T2
WHERE condition;
```

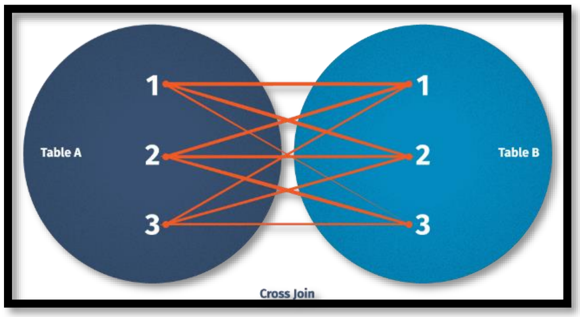
Example

The following SQL statement matches customers that are from the same city:

```
SELECT A.ContactName AS CustomerName1, B.ContactName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

iv. CROSS JOIN

The CROSS JOIN command in SQL, also known as a cartesian join, returns all combinations of rows from each table. Envision that you need to find all combinations of size and color. In that case, a CROSS JOIN will be an asset. Note, that this join does not need any condition to join two tables. In fact, CROSS JOIN joins every row from the first table with every row from the second table and its result comprises all combinations of records in two tables.



Syntax →

```
SELECT column_names(s)
FROM table1
CROSS JOIN Table2;
```

Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.ContactName, Orders.OrderID
FROM Customers
CROSS JOIN Orders;
```

4. ACTIVITY TIME BOXING

Activity Name	Activity Time	Total Time
Instruments Allocation + Setting up Lab	10 mints	10 mints
Walk through Theory & Tasks (Lecture)	60 mints	60 mints
Implementation & Practice time	90 mints	80 mints
Evaluation Time	20 mints	20 mints
	Total Duration	180 mints

5. EXERCISE:

PERFORMANCE TASKS:

1. ***All Example Tasks.***
2. *Get Customer's Company Name for an order using inner join. (Display only orderID from orders and comapnayName from Customers table)*
3. *Display Company Name and Total orders placed by the company. (Table: orders, customer)*
4. *Display the name of Supplier for the product starting with letter C. (Northwind Database, tabel: Supplier & Product).*
5. *Display Product Name and its Category Name where Category Name starts with C. (Tabel: category and product)*

LAB FILE TASKS:

1. *Display Product's name and their supplier's name (ContactName), whose supplier's name does not contain "A" and there should be at most 10 characters in Product's name.*
2. *Create a report showing order ID., OrderDate, EmployeeID, and the firstname and LastName of the associated Employee. (Tables: Employees & Orders)*
3. *Count no of Products's against each Suppliers name. List only those products whose count is less than 5.*
4. *Display Titles and their quantity. Show only those records where quantity is less than 10 (Hint Pubs database, table title and sales)*
5. *Display Book Title and publisher Names (pubs Database, Table Title and Publisher).*
6. *Display name of those publishers who haven't published any book yet.*
7. *Display Authors name whose city is same (Author id should be different).*
8. *Use cross join (Cartesian join) to Display Book Title and Publisher Name.*
9. *Use two full joins to Display Product Name, Category Name and Supplier Name.*
10. *Write a query to list the names of employees that belongs to the same location as the employee named Nancy.*