**Learning Objectives**

- Understand the Cartesian product.
- Understand the concepts of joins.
- Understanding the different types of joins.
- Understanding of combining data across multiple joins.

## LO1: Understanding Cartesian product.

### Cartesian Product

The Cartesian Product in SQL is a type of join that returns the combination of every row from one table with every row from another table. It is also called a **Cross Join** and results in an **m × n** combination of rows, where **m** is the number of rows in the first table and **n** is the number of rows in the second table.

**Example:** Let us consider the following tables:

customers

| CustomerId | ContactName | City |
|---|---|---|
| 1 | Ali | Lahore |
| 2 | Ahmed | Faisalabad |
| 3 | Aslam | Karachi |

orders

| OrderId | CustomerId | ShipAddress | ShipPostalCode |
|---|---|---|---|
| 101 | 1 | Lahore | 20022 |
| 102 | 2 | Karachi | 30011 |
| 103 | 2 | Lahore | 15022 |

**Query:**

```
SELECT *
FROM customers, orders;
```

**Result:**

| CustomerId | ContactName | City | CustomerId | OrderId | ShipAddress | ShipPostalCode |
|---|---|---|---|---|---|---|
| 1 | Ali | Lahore | 1 | 101 | Lahore | 20022 |
| 1 | Ali | Lahore | 2 | 102 | Karachi | 30011 |
| 1 | Ali | Lahore | 2 | 103 | Lahore | 15022 |
| 2 | Ahmed | Faisalabad | 1 | 101 | Lahore | 20022 |
| 2 | Ahmed | Faisalabad | 2 | 102 | Karachi | 30011 |
| 2 | Ahmed | Faisalabad | 2 | 103 | Lahore | 15022 |
| **3** | Aslam | Karachi | 1 | 101 | Lahore | 20022 |
| 3 | Aslam | Karachi | 2 | 102 | Karachi | 30011 |
| 3 | Aslam | Karachi | 2 | 103 | Lahore | 15022 |

### Cartesian Product with a Condition

To avoid unnecessary combinations, we can apply a condition to filter only the relevant rows. This is done using the **WHERE** clause or a **JOIN** condition.

**Example:** Filtering Based on Customer ID

**Query:**

```
SELECT *
FROM customers, orders
WHERE customers.CustomerId = orders.CustomerId;
```

**Result:**

| CustomerId | ContactName | city | OrderId | shipAddress | ShipPostalCode |
|---|---|---|---|---|---|
| 1 | Ali | Lahore | 101 | Lahore | 20022 |
| 2 | Ahmed | Faisalabad | 102 | Karachi | 30011 |
| 2 | Ahmed | Faisalabad | 103 | Lahore | 15022 |

## LO2: Understanding the Concept of Joins.

### Joins in SQL Server

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

### Relationship Between Cartesian Product and Joins

In SQL, **joins** are used to combine rows from two or more tables based on a specified condition. The **Cartesian product** (also known as the **cross join**) is the foundation of all types of joins. Joins work by **first**

**creating a Cartesian product** and then filtering the rows based on a specified condition.

**Join = Cartesian Product + Condition**

### Joins Syntax

```
SELECT column_name
FROM table1
INNER JOIN/ CROSS JOIN/ NATURAL JOIN/ LEFT JOIN/ RIGHT JOIN
table2
ON table1.column_name = table2.column_name;
```

## LO3: Understanding the Different types of Joins

SQL server provides the following joins:

- Natural Join
- Inner Join
- Left Outer Join
- Right Outer Join
- Full Outer Join

### Natural Join

A **Natural Join** is used when you want to combine two tables based on all columns with the same name and compatible data types, without explicitly specifying the join condition.

**Example:** Let us consider the below tables first

customers

| CustomerId | ContactName | City |
|---|---|---|
| 1 | Ali | Lahore |
| 2 | Ahmed | Faisalabad |
| 3 | Aslam | Karachi |

orders

| OrderId | CustomerId | ShipAddress | ShipPostalCode |
|---|---|---|---|
| 101 | 1 | Lahore | 200 |
| 102 | 2 | Karachi | 300 |
| 103 | 2 | Lahore | 150 |

**Requirement:** We have to report all customers who ordered.

**Query:**

```
SELECT *
FROM customers
NATURAL JOIN orders;
```

**Result:**

| customerid | contactname | city | orderid | shipaddress | shipostalcode |
|---|---|---|---|---|---|
| 1 | Ali | Lahore | 101 | Lahore | 200 |
| 2 | Ahmed | Faisalabad | 102 | Karachi | 150 |
| 2 | Ahmed | Faisalabad | 103 | Lahore | 150 |

## Inner Join

An **Inner Join** is used to combine two tables based on a specified condition, usually matching values in a common column. Only rows that satisfy the condition (i.e., have matching values in both tables) are included in the result.

**Example:** Let us consider the tables below first

customers

| CustomerId | ContactName | City |
|---|---|---|
| 1 | Ali | Lahore |
| 2 | Ahmed | Faisalabad |
| 3 | Aslam | Karachi |

orders

| orderid | customerid | shippingaddress | Shippostalcode |
|---|---|---|---|
| 101 | 1 | Lahore | 200 |
| 102 | 2 | Karachi | 300 |
| 103 | 2 | Lahore | 150 |
| 104 | 4 | Islamabad | 400 |

**Requirement:** Generate a report listing only those customers who have made at least one purchase, along with their order details.

**Query:**

```
SELECT*
FROM customers
INNER JOIN orders
ON customers.customerid = orders.customerid;
```

**Result:**

| customer_id | contactname | city | orderid | shipaddress | shippostalcode |
|---|---|---|---|---|---|
| 1 | Ali | Lahore | 101 | Lahore | 200 |
| 2 | Ahmed | Faisalabad | 102 | Karachi | 150 |
| 2 | Ahmed | Faisalabad | 103 | Lahore | 150 |

## Left Outer Join

Left Outer join gives the matching rows and the rows which are in left table but not in right table.

**Example:** Let us consider the below tables first

customers

| CustomerId | ContactName | City |
|---|---|---|
| 1 | Ali | Lahore |
| 2 | Ahmed | Faisalabad |
| 3 | Aslam | Karachi |

orders

| orderid | customerid | shipaddress | shippostalcode |
|---|---|---|---|
| 101 | 1 | Lahore | 20022 |
| 102 | 2 | Karachi | 30011 |
| 103 | 2 | Lahore | 15022 |

**Requirement:** We have to report order details of customers and who does not give any orders show null orders also.

**Query:**

```
SELECT *
FROM customers C LEFT JOIN orders O
ON C.customerid = O.customerid
```

**Result:**

| customerid | contactname | city | orderid | shipaddress | shippostalcode |
|---|---|---|---|---|---|
| 1 | Ali | Lahore | 101 | Lahore | 20022 |
| 2 | Ahmed | Faisalabad | 102 | Karachi | 30011 |
| 2 | Ahmed | Faisalabad | 103 | Lahore | 15022 |
| 3 | Aslam | Karachi | NULL | NULL | NULL |

## Right Outer Join

Right Outer join gives the matching rows and the rows which are in right table but not in left table.

**Example:** Let us consider the below tables first

customers

| CustomerId | ContactName | City |
|---|---|---|
| 1 | Ali | Lahore |
| 2 | Ahmed | Faisalabad |
| 3 | Aslam | Karachi |

orders

| orderid | customerid | shipaddress | shippostalcode |
|---|---|---|---|
| 101 | 1 | Lahore | 20022 |
| 102 | 2 | Karachi | 30011 |
| 103 | 2 | Lahore | 15022 |
| 104 | 4 | Islamabad | 40012 |

**Requirement:** We need to report all the details of orders, including information about the customers who placed those orders. Additionally, if an order exists without a corresponding customer in the customers table (due to missing or incorrect data), include these orders as well, showing NULL for the customer-related columns.

**Query:**

```
SELECT *
FROM customers C RIGHT JOIN orders O
```

```
ON C.customerid = O.customerid
```

**Result:**

| customerId | Contactname | city | orderid | Shipaddress | shippostalcode |
|---|---|---|---|---|---|
| 1 | Ali | Lahore | 101 | Lahore | 20022 |
| 2 | Ahmed | Faisalabad | 102 | Karachi | 30011 |
| 2 | Ahmed | Faisalabad | 103 | Lahore | 15022 |
| Null | Null | Null | 104 | Islamabad | 40012 |

## Full Outer Join

**Full Outer Join** combines the results of both Left Outer Join and Right Outer Join:

- It includes **all rows from both tables**, even if there are no matches between them.
- If a match is found between the two tables, it combines the data into a single row

**Example:** Let us consider the below tables first

customers

| CustomerId | ContactName | City |
|---|---|---|
| 1 | Ali | Lahore |
| 2 | Ahmed | Faisalabad |
| 3 | Aslam | Karachi |

orders

| orderid | customerid | shipaddress | shippostalcode |
|---|---|---|---|
| 101 | 1 | Lahore | 20022 |
| 102 | 2 | Karachi | 30011 |
| 103 | 2 | Lahore | 15022 |
| 104 | 4 | Islamabad | 40012 |

**Requirement:** Give all details of customers and orders

**Query:**

```
SELECT *
FROM customers C
```

```
LEFT JOIN orders O ON C.customerId = O.customerId
UNION
SELECT *
FROM customers C
RIGHT JOIN orders O ON C.customerId = O.customerId;
```

**Result:**

| customerId | Contactname | city | orderid | shipaddress | ShipPostalCode |
|---|---|---|---|---|---|
| 1 | Ali | Lahore | 101 | Lahore | 20022 |
| 2 | Ahmed | Faisalabad | 102 | Karachi | 30011 |
| 2 | Ahmed | Faisalabad | 103 | Lahore | 15022 |
| 3 | Aslam | Karachi | Null | Null | Null |
| Null | Null | Null | 104 | Islamabad | 40012 |

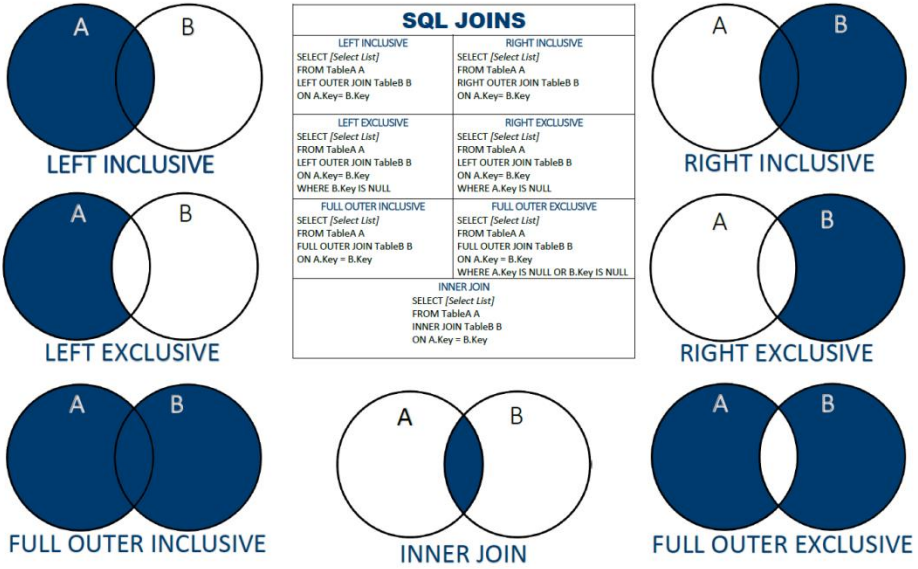


*Figure 1 : Visual Representation of All Joins*

## LO4: Understanding of Combining Data across Multiple Joins

In SQL, complex queries often require combining data from multiple tables using different types of **JOINs**. By effectively using multiple joins, we can extract meaningful relationships and insights from relational databases.

### Concept

When multiple tables are involved, we can use **multiple joins** to connect them based on related columns.

**Example: Multiple Joins in Action**

Let's consider a scenario where we need to fetch order details along with customer and product information.

customers

| CustomerId | ContactName | city |
|---|---|---|
| 1 | Ali | Lahore |
| 2 | Aslam | Karachi |

orders

| orderId | customerId | product_id | amount |
|---|---|---|---|
| 101 | 1 | 10 | 200 |
| 102 | 2 | 11 | 150 |

products

| product_id | product_name | Price |
|---|---|---|
| 10 | Laptop | 500 |
| 11 | Phone | 300 |

**Requirement:**

Retrieve customer names, cities, order amounts, and product names for all placed orders.

**Query:**

```
SELECT *
FROM customers
INNER JOIN orders ON customers.customerid = orders.customerid
```

```
INNER JOIN products ON orders.product_id =
products.product_id;
```

**Result:**

| name | city | amount | product_name |
|---|---|---|---|
| Ali | Lahore | 200 | Laptop |
| Aslam | Karachi | 150 | Phone |

## Tasks:

Perform all JOIN queries on any table using Northwind Schema. You cannot use group by clause in this manual.

1. Write a query that returns all possible combinations of **employees** and **shippers**.
   **Output:** EmployeeID (EmployeeID), EmployeeName (EmployeeName), ShipperID (ShipperID), ShipperName (ShipperName)
2. Generate all possible **customer-order-shipper** combinations but filter only those where the **customer's first three characters matches the shipper's name first three characters.**
   **Output:** CustomerID (CustomerID), CustomerName (CustomerName), ShipperID (ShipperID), ShipperName (ShipperName), First three characters
3. Create a Cartesian product of employees with themselves to identify all possible **employee pairs**. Ensure that each pair appears only once.
   **Output:** Employee1ID (Employee1ID), Employee1Name (Employee1Name), Employee2ID (Employee2ID), Employee2Name (Employee2Name)
4. Display order details showing OrderID, CustomerName, ProductName, Quantity, Employee's Full Name, and Shipper's Name.
   Output: OrderID (OrderID), CustomerName (CustomerName), ProductName (ProductName), Quantity (Quantity), EmployeeName (EmployeeName), ShipperName (ShipperName)
5. Retrieve a list of **all products** that have never been ordered

   **Output:** ProductID (ProductID), ProductName (ProductName), CategoryName (CategoryName), SupplierName (SupplierName)

6. Retrieve a list of **employees** who have **not processed any orders in the last 6 months of an year**.

   **Output:** EmployeeID (EmployeeID), EmployeeName (EmployeeName)

7. Retrieve a list of **all orders** including those that do not have an assigned **employee** or **customer**.
   **Output:** OrderID (OrderID), CustomerID (CustomerID), CustomerName (CustomerName), EmployeeID (EmployeeID), EmployeeName (EmployeeName)
8. Retrieve all customers and their orders. If the customer has never placed an order, display "**No Order Placed**" in the OrderID column.
   **Output:** CustomerID (CustomerID), CustomerName (CustomerName), OrderID (OrderID)
9. Find all orders placed **before 9 AM or after 5 PM**.
   **Output:** OrderID (OrderID), CustomerID (CustomerID), CustomerName (CustomerName), OrderHour (OrderHour)
10. Identify **pairs of employees who have both processed orders for the same product**.
    **Output:** ProductID (ProductID), ProductName (ProductName), Employee1ID (Employee1ID), Employee1Name (Employee1Name), Employee2ID (Employee2ID), Employee2Name (Employee2Name)
11. Find **all suppliers** including those who have no products associated with them.
    **Output:** SupplierID (SupplierID), SupplierName (SupplierName),

12. Retrieve a list of **all customers** who have **never placed an order**.

    **Output:** CustomerID (CustomerID), CustomerName (CustomerName)

13. Retrieve **customer pairs who placed orders on the same date**.

    **Output:** Customer1ID (Customer1ID), Customer1Name (Customer1Name), Customer2ID (Customer2ID), Customer2Name (Customer2Name), OrderDate (OrderDate)

14. Find **pairs of employees who have both served the same customer** at any point.
    **Output:** CustomerID (CustomerID), CustomerName (CustomerName), Employee1ID (Employee1ID), Employee1Name (Employee1Name), Employee2ID (Employee2ID), Employee2Name (Employee2Name)

## What to Submit:

Submit the following file in Zip on Eduko:

- 2024-CS-X.txt