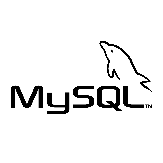
**A logo of a university of engineering

Description automatically generatedDatabase Systems**

Lab Manual 4

**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 |

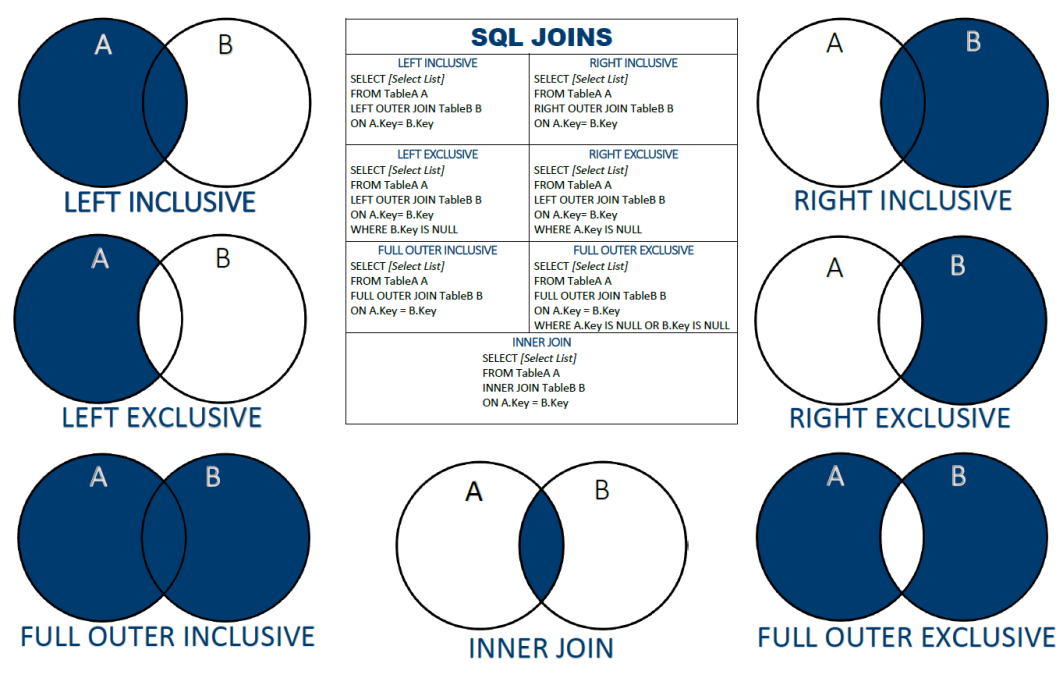


Figure 1 : Visual Representation of All Joins

## ****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.
* Perform self-cross join and see if there is any difference between cross join and self cross join.

SELECT Customers.custid, Customers.companyname, Orders.orderid, Orders.orderdate

FROM Sales.Customers AS C

INNER JOIN Sales.Orders AS O

ON Customers.custid = Orders.custid;

1. **List the customers who made purchases during August 1996.**  
   (CustomerID, ContactName, OrderID, OrderDate, ShipName,ShipAddress)
2. **Show the names of the products that were purchased on August 8, 1997.** (ProductID, ProductName, OrderID, OrderDate)
3. **List all countries where beverages have been delivered.** (Country)
4. Find the SupplierID and SupplierName of suppliers who have provided products, ensuring that each supplier's product belongs to a known category. Display the category details alongside. (CategoryId, CategoryName).
5. **Retrieve a list of all customers along with their order details, including those who have not placed any orders.** (CustomerID, ContactName, OrderID, OrderDate)
6. Retrieve a list of customers along with their order details, but only show orders that were shipped to France or Brazil or Switzerland. (CustomerID, ContactName, OrderID, OrderDate,ShipCountry)
7. Retrieve details of customers from the US and whose quantities are greater than 50. Also show order details. (CustomerID, ContactName, OrderID, OrderDate,Quantity)
8. **Identify customers who have not placed any orders, and return their IDs only.** (CustomerID)
9. **Create a query that lists all customers, but only links them to orders placed on September 4, 1997.** (CustomerID, ContactName, OrderID, OrderDate, TotalAmount)
10. **Retrieve the address, city, and country information for all orders serviced by Anne that were shipped past their due date.** (OrderID, CustomerID, CustomerName, ShipAddress, ShipCity, ShipCountry, ShippedDate, RequiredDate)
11. **Identify all products that have discount along with their product details.** (ProductID, ProductName, CategoryName, UnitPrice, UnitsinStock)
12. **Find all employees who shipped orders by Company “Speedy Express” to city Sevilla. (EmployeeID, FirstName, Title, Address, HireDate)**.
13. **List all orders that were shipped to the same country as the customer's residing country. Display the CustomerID, OrderID, ShipCountry, and Customer's Country.** (CustomerID, OrderID, ShipCountry, Country)
14. **Identify the employees who have handled orders shipped to 'Germany' but are not located in the 'USA'.** (EmployeeID, FirstName, LastName, City, Country)
15. **Find the customers who have placed orders for products from the 'Seafood' category and have had those orders shipped by 'United Package'.** (CustomerID, ContactName, OrderID, CategoryName, ShipVia)
16. **List all pairs of employees who live in the same city but have different titles.** (EmployeeID1, FirstName1, LastName1, Title1, EmployeeID2, FirstName2, LastName2, Title2, City)

## What to Submit:

Submit the following file in Zip on Eduko:

* 2024-CS-X.txt