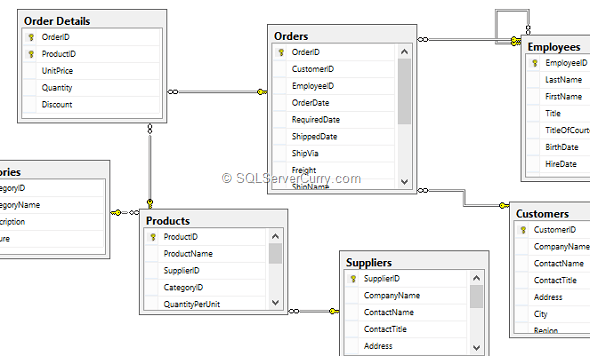
**Types of JOIN in SQL Server - Inner, Self, Outer and Cross JOIN**

When we work with relational databases, the first thing we do is normalize the data. Different forms of normalization like First Normal Form (FNF), Second Normal Form (SNF) and Third Normal Form (TNF) etc divides the data into number of tables for avoiding redundancy. These tables are related with each other using Primary Key constraint and Foreign Key Constraint.  
  
To fetch data from these related tables, we perform various types of queries. One way is to fetch the data using **SQL JOIN**s. There are different types of joins supported by SQL Server as mentioned below -

* INNER JOIN
* SELF JOIN
* OUTER JOIN [LEFT, RIGHT and FULL OUTER JOIN]
* CROSS JOIN

To observe the tables and their relationship, we will use a demo database Northwind provided by Microsoft. We will consider below tables -

* Customers
* Employees
* Categories
* Products
* Suppliers
* Orders
* Order Details

The database diagram looks similar to the following:  
  
  
Now we will use the tables in this diagram to fetch the data using various scenarios. Let's first query individual tables. Open SQL Server Management Studio and click on New Query. Write the following queries and test them -

USE Northwind

GO

SELECT \* FROM Customers

SELECT \* FROM Employees

SELECT \* FROM Orders

SELECT \* FROM [Order Details]

SELECT \* FROM Categories

SELECT \* FROM Products

SELECT \* FROM Suppliers

**SQL INNER JOIN**

Inner Join returns the matching rows from both the tables. For example, if we want to find out all the orders placed by all the customers, we can make use of an Inner Join. The query is as follows:

SELECT

C.ContactName,

C.CompanyName,

C.City,

O.OrderID,

O.OrderDate,

O.RequiredDate

FROM Customers C INNER JOIN Orders O

ON C.CustomerID=O.CustomerID

Find out all the Orders processed by each Employee. The query is as shown below -

SELECT

E.FirstName,

E.LastName,

E.City,

O.OrderID,

O.OrderDate,

O.RequiredDate

FROM Employees E INNER JOIN Orders O

ON E.EmployeeID=O.EmployeeID

You can apply a filter on the data which is fetched by the Join. For example, we will find out all the customers and their orders and filter the result on customer's city. The query is as shown below -

--NonEqui-Join [INNER JOIN with Where Condition]

SELECT

C.ContactName,

C.CompanyName,

C.City,

O.OrderID,

O.OrderDate,

O.RequiredDate

FROM Customers C INNER JOIN Orders O

ON C.CustomerID=O.CustomerID

WHERE C.City='London'

The Inner Join can also be written as shown below -

SELECT

C.ContactName,

C.CompanyName,

C.City,

O.OrderID,

O.OrderDate,

O.RequiredDate

FROM Customers C JOIN Orders O

ON C.CustomerID=O.CustomerID

--OR [Equi JOIN]

SELECT

C.ContactName,

C.CompanyName,

C.City,

O.OrderID,

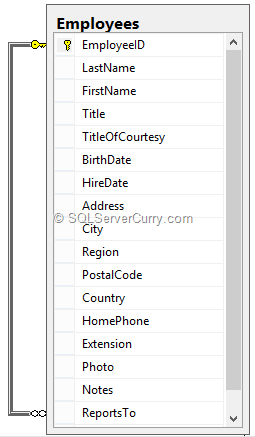
O.OrderDate,

O.RequiredDate

FROM Customers C,Orders O

WHERE C.CustomerID=O.CustomerID

**SQL SELF JOIN**

Joining a table to itself is called as Self Join. We will now see why do we require this type of join. Observe the following Employees table which belongs to the Northwind database -  
  
  
  
  
In the employees table, we have a primary key EmployeeID which is mapped to the employees table's ReportTo column. Assume that ReportsTo column acts as Manager ID column who tells which employees works under whom. In that situation, we will have to make use of Self Join. The query is as shown below -

-- Self JOIN - Joining a Table to itself.

SELECT

Mgr.FirstName + ' ' + Mgr.LastName AS 'Manager',

Emp.FirstName + ' ' + Emp.LastName

as 'Employee Name'

FROM Employees AS Emp INNER JOIN Employees AS Mgr

ON Mgr.EmployeeID=Emp.ReportsTo

**SQL Outer JOIN**

An outer join is divided into three different parts.  
  
**LEFT OUTER JOIN** - This join returns all the rows from the first [Left] table and matching rows from the second [Right] table. For example, If we want to see all the customers who have placed orders and who have not placed any orders. Let’s say we want to display orders which are placed by customers, we can make use of Left Outer Join. The query is as shown below -

--LEFT OUTER JOIN -

SELECT

C.ContactName,

C.CompanyName,

C.City,

O.OrderID,

O.OrderDate,

O.RequiredDate

FROM Customers C LEFT OUTER JOIN Orders O

ON C.CustomerID=O.CustomerID

**RIGHT OUTER JOIN** - This join returns only matching rows from the first [Left] table and all the rows from the second [Right] table. For example, If we want to see all the orders who have been placed by customers and the orders which do not belong to any customer (although unrealistic), but want to display only those customers who have are placed orders, we can make use of Right Outer Join. The query is as shown below -

--RIGHT OUTER JOIN -

SELECT

C.ContactName,

C.CompanyName,

C.City,

O.OrderID,

O.OrderDate,

O.RequiredDate

FROM Customers C RIGHT OUTER JOIN Orders O

ON C.CustomerID=O.CustomerID

**FULL OUTER JOIN** - This join returns matching as well as non matching rows from both tables [Left table as well as Right table]. For example, If we want to see all the customers as well as all the orders, we can make use of FULL Outer Join. The query is as shown below -

--FULL OUTER JOIN -

SELECT

C.ContactName,

C.CompanyName,

C.City,

O.OrderID,

O.OrderDate,

O.RequiredDate

FROM Customers C FULL OUTER JOIN Orders O

ON C.CustomerID=O.CustomerID

You can also perform multi-table join query. Let's look at an example of multi-table join query -

--JOINS with Multiple Tables

SELECT

C.ContactName,

C.CompanyName,

C.City,

O.OrderID,

O.OrderDate,

O.RequiredDate,

OD.Quantity,

OD.UnitPrice,P.ProductName

FROM Customers C INNER JOIN Orders O

ON C.CustomerID=O.CustomerID

INNER JOIN [Order Details] OD

ON O.OrderID=OD.OrderID INNER JOIN Products P

ON OD.ProductID=P.ProductID

**Join Between Table and View**

A Join can be performed in between Views and tables as well. We will first create a view and then perform a join in between a view and a table -

--JOIN Between Table and View

CREATE VIEW LondonCustomers

AS SELECT \* FROM Customers WHERE City='London'

GO

SELECT

CV.ContactName,

CV.CompanyName,

CV.City,

O.OrderID,

O.OrderDate,

O.RequiredDate

FROM LondonCustomers CV INNER JOIN Orders O

ON CV.CustomerID=O.CustomerID

**Join between Table Valued Function as well as Table**

We can also perform a join in between Table Valued Function as well as table. Let's create a Table Valued Function and perform a join in between TVF and a table. The query is as shown below -

--JOIN Between Table and TVF [Table Valued Function]

CREATE FUNCTION BerlinCustomersFunction() RETURNS TABLE

AS

RETURN (SELECT \* FROM Customers WHERE City='Berlin')

GO

SELECT

TVF.ContactName,

TVF.CompanyName,

TVF.City,

O.OrderID,

O.OrderDate,

O.RequiredDate

FROM dbo.BerlinCustomersFunction() TVF INNER JOIN Orders O

ON TVF.CustomerID=O.CustomerID

**CROSS JOIN**

This join returns a Cartesian Product of two or more tables. It joins each row of the first table with all the rows of the second table. Let's take a look at the query as shown below -

--CROSS JOIN

SELECT

ContactName,

CompanyName,

City,

OrderID,

OrderDate,

RequiredDate

FROM Customers CROSS JOIN Orders

**Summary**

In this article, we have seen different types of joins you can perform in SQL Server. We have also seen how to write complex joins (multi-table join). We have seen INNER JOIN, SELF JOIN, OUTER JOIN (LEFT, RIGHT and FULL) and CROSS JOIN. Hope this article has helped in getting up to speed with SQL Joins.