A step-by-step walkthrough of SQL Inner Join

June 21, 2019 by Gauri Mahajan

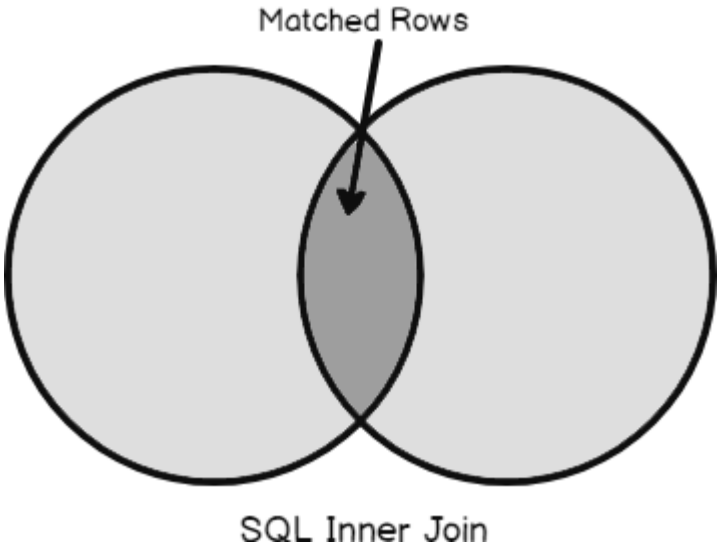
Organizations are generating and analyzing unmatched volumes of data with each passing minute. In this article, we will demonstrate how we can employ **SQL Inner Join** to query and access data from multiple tables that store this incessantly growing data in the SQL databases.

SQL Joins

Before we get started with SQL Inner Join, I would like to call out SQL Join here. **Join** is the widely-used clause in the SQL Server essentially to combine and retrieve data from two or more tables. In a real-world relational database, data is structured in a large number of tables and which is why, there is a constant need to join these multiple tables based on logical relationships between them. There are four basic types of Joins in SQL Server – Inner, Outer (left, right, full), Self and Cross join. To get a quick overview of all these joins, I would recommend going through this link, SQL Join types overview and tutorial.

This article targets all about the Inner Join in SQL Server, so let's head over to it.

Definition of SQL Inner Join



SQL Server Inner Join Syntax

Below is the basic syntax of Inner Join.

SELECT Column_list
FROM TABLE1
INNER JOIN *TABLE2*
ON Table1.ColName = Table2.ColName

Inner Join syntax basically compares rows of Table1 with Table2 to check if anything matches based on the condition provided in the ON clause. When the Join condition is met, it returns matched rows in both tables with the selected columns in the SELECT clause.

SQL Inner Join clause is the same as Join clause and works the same way if we don't specify the type (INNER) while using the Join clause. In short, Inner Join is the default keyword for Join and both can be used interchangeably.

```
SELECT ColName(s)
FROM TABLE1
INNER JOIN TABLE2
ON Table1.ColName = Table2.ColName
```

```
SELECT ColName(s)
FROM TABLE1
JOIN TABLE2
ON Table1.ColName = Table2.ColName
```

that manages different branches of Pizza outlets in a few cities and table 'Foods' that stores food distribution details across these companies. You can execute the code below to create and populate data into these two tables. All this data is hypothetical and you can create in any of your existing databases.

```
CREATE TABLE [dbo].[PizzaCompany]
(
  [CompanyId] [int] IDENTITY(1,1) PRIMARY KEY CLUSTERED,
  [CompanyName] [varchar](50) ,
  [CompanyCity] [varchar](30)
)
SET IDENTITY_INSERT [dbo].[PizzaCompany] ON;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VAL-
UES(1,'Dominos','Los Angeles') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VAL-
UES(2,'Pizza Hut','San Francisco') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VAL-
UES(3,'Papa johns','San Diego') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VAL-
UES(4,'Ah Pizz','Fremont') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VAL-
UES(5,'Nino Pizza','Las Vegas') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VAL-
UES(6,'Pizzeria','Boston') ;
INSERT INTO [dbo].[PizzaCompany] ([CompanyId], [CompanyName], [CompanyCity]) VAL-
UES(7,'chuck e cheese','Chicago') ;

SELECT * FROM PizzaCompany:
```

This is how data in the PizzaCompany table looks like:

Results Messages

	CompanyId	CompanyName	CompanyCity
1	1	Dominos	Los Angeles
2	2	Pizza Hut	San Francisco
3	3	Papa johns	San Diego
4	4	Ah Pizz	Fremont
5	5	Nino Pizza	Las Vegas
6	6	Pizzeria	Boston
7	7	chuck e cheese	Chicago

Let's create and populate Foods table now. CompanyID in this table is the foreign key that is referenc-

```
1,'Large Pizza',5,2)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(
2,'Garlic Knots',6,3)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(
3,'Large Pizza',3,3)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(
4,'Medium Pizza',8,4)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(
5,'Breadsticks',7,1)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(
6,'Medium Pizza',11,1)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(
7,'Small Pizza',9,6)
INSERT INTO [dbo].[Foods] ([ItemId], [ItemName], [UnitsSold], [CompanyId]) VALUES(
8,'Small Pizza',6,7)

SELECT * FROM Foods
```

The following table shows data in the Foods table. This table stores information like units sold per food item and also the pizza outlet (CompanyId) that delivers it.

Results Messages

	ItemId	ItemName	UnitsSold	CompanyID
1	1	Large Pizza	5	2
2	2	Garlic Knots	6	3
3	3	Large Pizza	3	3
4	4	Medium Pizza	8	4
5	5	Breadsticks	7	1
6	6	Medium Pizza	11	1
7	7	Small Pizza	9	6
8	8	Small Pizza	6	7

Now, if we would like to see the items and also the units sold by each pizza company, we can combine these two tables with the help of an inner join clause being used on the field CompanyId (in our case this shares a foreign key relationship).

```
 SELECT pz.CompanyCity, pz.CompanyName, pz.CompanyId AS PizzaCompanyId, f.Compa-
nyID AS FoodsCompanyId, f.ItemName, f.UnitsSold
FROM PizzaCompany pz
INNER JOIN Foods f
```

2	2	Pizza Hut	San Francisco
3	3	Papa johns	San Diego
4	4	Ah Pizz	Fremont
5	5	Nino Pizza	Las Vegas
6	6	Pizzeria	Boston
7	7	chuck e cheese	Chicago

Table PizzaCompany

2	2	Garlic Knots	6	3
3	3	Large Pizza	3	3
4	4	Medium Pizza	8	4
5	5	Breadsticks	7	1
6	6	Medium Pizza	11	1
7	7	Small Pizza	9	6
8	8	Small Pizza	6	7

Table Foods

Inner join result set of the tables without CompanyId = 5 (Unmatched row)

Results Messages

	CompanyCity	CompanyName	PizzaCompanyId	FoodsCompanyId	ItemName	UnitsSold
1	San Francisco	Pizza Hut	2	2	Large Pizza	5
2	San Diego	Papa johns	3	3	Garlic Knots	6
3	San Diego	Papa johns	3	3	Large Pizza	3
4	Fremont	Ah Pizz	4	4	Medium Pizza	8
5	Los Angeles	Dominos	1	1	Breadsticks	7
6	Los Angeles	Dominos	1	1	Medium Pizza	11
7	Boston	Pizzeria	6	6	Small Pizza	9
8	Chicago	chuck e cheese	7	7	Small Pizza	6

With the help of the above result set, we can make out the items and also the count of items delivered by pizza outlets in various cities. For instance, Dominos made a delivery of 7 Breadsticks and 11 Medium Pizza in Los Angeles.

SQL Inner Join on three tables

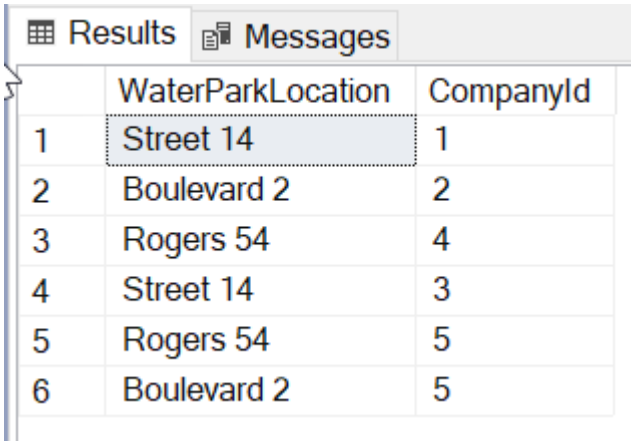
Let's explore more into this join and suppose three waterparks (looks like summer) get opened in the state and these waterparks outsource food from the pizza outlets mentioned in the table PizzaCompany.

I am going to quickly create a table WaterPark and load some arbitrary data into it as shown below.

```
CREATE TABLE [dbo].[WaterPark]
(
  [WaterParkLocation] VARCHAR(50),
 [CompanyId] int,
 FOREIGN KEY(CompanyID) REFERENCES PizzaCompany(CompanyID)
)
```

```
SELECT * FROM WaterPark
```

And below is the output of this table.



Results Messages

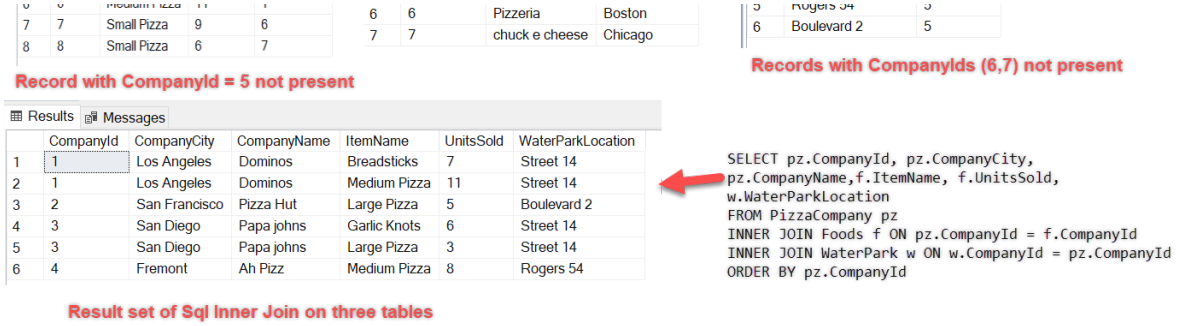
	WaterParkLocation	CompanyId
1	Street 14	1
2	Boulevard 2	2
3	Rogers 54	4
4	Street 14	3
5	Rogers 54	5
6	Boulevard 2	5

As the saying goes, the picture is worth a thousand words. Let's quickly see the database diagram of these three tables with their relationships to understand them better.

Now we are going to include this third table in the SQL Inner Join clause to see how it is going to impact the result set. Per data in the WaterPark table, the three waterparks have been outsourcing food from all the Pizza Companies but Pizzeria (Id=6) and chuck e cheese (Id = 7). Execute the code below to see all the food distribution across the waterparks by the Pizza outlets.

```
SELECT pz.CompanyId, pz.CompanyCity, pz.CompanyName,f.ItemName, f.UnitsSold,
w.WaterParkLocation
FROM PizzaCompany pz
INNER JOIN Foods f ON pz.CompanyId = f.CompanyId
INNER JOIN WaterPark w ON w.CompanyId = pz.CompanyId
ORDER BY pz.CompanyId
```

Based on CompanyId, SQL Inner Join matches rows in both tables, PizzaCompany (Table 1) and Foods (Table 2) and subsequently looks for a match in the WaterPark (Table 3) to return rows. As shown



Let's dig into SQL Inner Join more with a few more T-SQL clauses.

Using WHERE with Inner Join

We can filter records based on a specified condition when SQL Inner Join is used with a WHERE clause. Assume that we would like to get the rows where units sold were more than 6.

In the following query, the WHERE clause is added to extract results with value more than 6 for units sold.

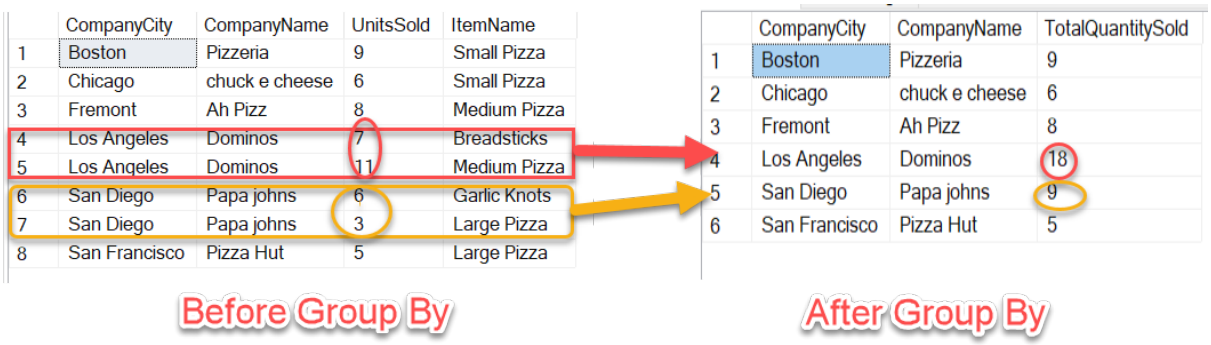
```
SELECT pz.CompanyId, pz.CompanyCity, pz.CompanyName,f.ItemName, f.UnitsSold
FROM PizzaCompany pz
INNER JOIN Foods f ON pz.CompanyId = f.CompanyId
WHERE f.UnitsSold > 6
ORDER BY pz.CompanyCity
```

Execute above code in SSMS to see the below result. Four such records are returned by this query.

CompanyId	CompanyCity	CompanyName	ItemName	UnitsSold
6	Boston	Pizzeria	Small Pizza	9
4	Fremont	Ah Pizz	Medium Pizza	8
1	Los Angeles	Dominos	Breadsticks	7
1	Los Angeles	Dominos	Medium Pizza	11

```
SELECT pz.CompanyCity, pz.CompanyName, SUM(f.UnitsSold) AS TotalQuantitySold
FROM PizzaCompany pz
INNER JOIN Foods f ON pz.CompanyId = f.CompanyId
GROUP BY pz.CompanyCity, pz.CompanyName
ORDER BY pz.CompanyCity
```

Here, we intend to obtain total items sold by each Pizza company present in the City. As you can see below, aggregated result in 'totalquantitysold' column as 18 (7+11) and 9 (6+3) for Los Angeles and San Diego respectively is computed.



	CompanyCity	CompanyName	UnitsSold	ItemName
1	Boston	Pizzeria	9	Small Pizza
2	Chicago	chuck e cheese	6	Small Pizza
3	Fremont	Ah Pizz	8	Medium Pizza
4	Los Angeles	Dominos	7	Breadsticks
5	Los Angeles	Dominos	11	Medium Pizza
6	San Diego	Papa johns	6	Garlic Knots
7	San Diego	Papa johns	3	Large Pizza
8	San Francisco	Pizza Hut	5	Large Pizza

Before Group By

	CompanyCity	CompanyName	TotalQuantitySold
1	Boston	Pizzeria	9
2	Chicago	chuck e cheese	6
3	Fremont	Ah Pizz	8
4	Los Angeles	Dominos	18
5	San Diego	Papa johns	9
6	San Francisco	Pizza Hut	5

After Group By

A brief note on Equi and Theta Join

Before we conclude this article, let's quickly go over terms, a SQL developer may hear sporadically – Equi and Theta Join.

Equi Join

As the name suggests, equi join contains an equality operator '=' either in the Join clause or in the WHERE condition. SQL Inner, Left, Right are all equi joins when '=' operator is being used as a comparison operator. Usually, when there is a mention of SQL Inner Join, it is considered as an Inner equi Join, in an unusual situation only, equality operator is not used.

To make things easier, I am going to refer to AdventureWorksDW2017 sample database and fire a query against existing tables to demonstrate how equi join looks like.

2	281	Michael	Sales Representative	2010-12-29	367000.00
3	282	Linda	Sales Representative	2010-12-29	637000.00
4	283	Jillian	Sales Representative	2010-12-29	565000.00
5	284	Garrett	Sales Representative	2010-12-29	244000.00
6	285	Tsvi	Sales Representative	2010-12-29	669000.00
7	286	Pamela	Sales Representative	2010-12-29	165000.00
8	287	Shu	Sales Representative	2010-12-29	460000.00
9	288	José	Sales Representative	2010-12-29	525000.00
10	289	David	Sales Representative	2010-12-29	226000.00
11	272	Stephen	North American Sales Manager	2010-08-04	7000.00
12	281	Michael	Sales Representative	2010-12-29	556000.00
13	282	Linda	Sales Representative	2010-12-29	781000.00

Theta Join (Non-equi join)

Non-equi join is basically opposite of equi-join and is used when we join on a condition other than '=' operator. This type is rarely used in practice. Below is an example that makes use of theta join with an inequality operator (<) to evaluate profit by estimating cost and selling prices in two tables.

```
SELECT * FROM Table1 T1, Table2 T2 WHERE T1.ProductCost < T2.SalesPrice
```

Conclusion

I hope this article on 'SQL Inner Join' provides a comprehensible approach to one of the important and frequently used clauses – 'Inner join' in the SQL Server to combine multiple tables. In case you have any questions, please feel free to ask in the comments section below.

To continue your learning on SQL Joins, you can refer to below posts:

- SQL OUTER JOIN overview and examples
- SQL Join introduction and overview

See more

Gauri Mahajan

Gauri is a SQL Server Professional and has 6+ years experience of working with global multinational consulting and technology organizations. She is very passionate about working on SQL Server topics like Azure SQL Database, SQL Server Reporting Services, R, Python, Power BI, Database engine, etc. She has years of experience in technical documentation and is fond of technology authoring.

She has a deep experience in designing data and analytics solutions and ensuring its stability, reliability, and performance. She is also certified in SQL Server and have passed certifications like 70-463: Implementing Data Warehouses with Microsoft SQL Server.

View all posts by Gauri Mahajan

Related Posts:

1. Term Extraction Transformation in SSIS

109,384 Views

ALSO ON **SQL SHACK**

A case study of SQL Query tuning in SQL …

2 months ago • 2 comments

This article will describe some tips about query tuning in SQL Server.

Install SQL Server 2019 on Windows Server …

7 months ago • 1 comment

In this article, we will discuss configuring SQL Server Always On …

How to use iterations and conditions …

2 months ago • 2 comments

In this article, we will show how to use the Iterations and Conditions activities …

Don't fe perform

5 months a

This artic tips for S(performa