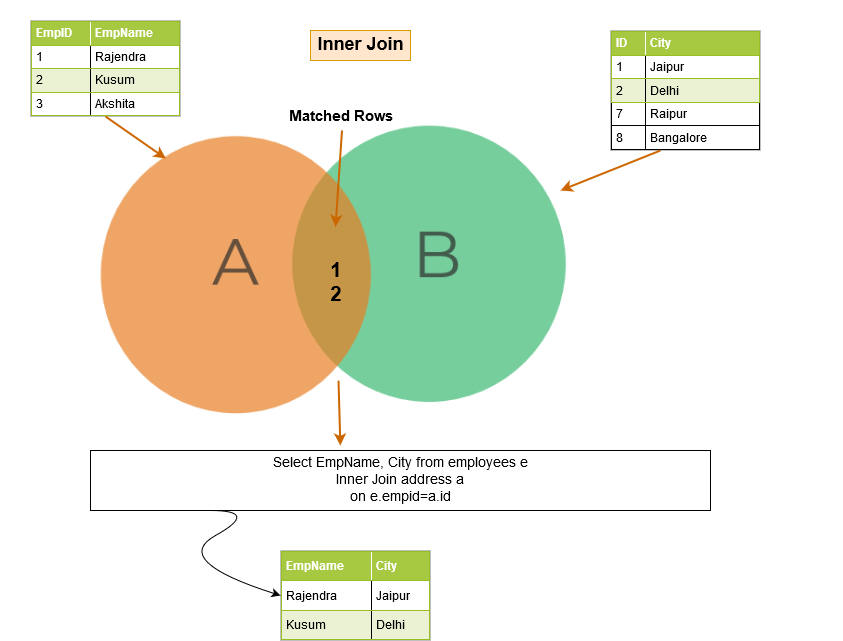


## JOIN types

There are a few different types of JOINs, each which specifies a different way for the database to handle data that doesn’t match the join condition. These Venn diagrams are a nice way of demonstrating what data is returned in these joins.

| **JOIN VISUAL** | **TYPE** | **DESCRIPTION** |
| --- | --- | --- |
|  | **INNER** | **DEFAULT: returns only the rows where matches were found** |
|  | **LEFT OUTER** | **returns matches and all rows from the left listed table** |
|  | **RIGHT OUTER** | **returns matches and all rows from the right listed table** |
|  | **FULL OUTER** | **returns matches and all rows from both tables** |

innerjoin



Problem: List all orders with customer information.

|  |
| --- |
| **ORDER** |
| Id |
| OrderDate |
| OrderNumber |
| CustomerId |
| TotalAmount |

|  |
| --- |
| **CUSTOMER** |
| Id |
| FirstName |
| LastName |
| City |
| Country |
| Phone |

JOIN with 3 TABLES

**Problem: List all orders with product name, quantity, and price, sorted by order number.**

|  |
| --- |
| **PRODUCT** |
| Id |
| ProductName |
| SupplierId |
| **UnitPrice** |
| Package |
| IsDiscontinued |
|  |
|  |

|  |
| --- |
| **ORDERITEM** |
| Id |
| OrderId |
| ProductId |
| UnitPrice |
| Quantity |
|  |

|  |
| --- |
| **ORDER** |
| Id |
| OrderDate |
| **OrderNumber** |
| CustomerId |
| TotalAmount |

**SELECT O.OrderNumber, CONVERT(date,O.OrderDate) AS Date,**

**P.ProductName, I.Quantity, I.UnitPrice**

**FROM [Order] O**

**JOIN OrderItem I ON O.Id = I.OrderId**

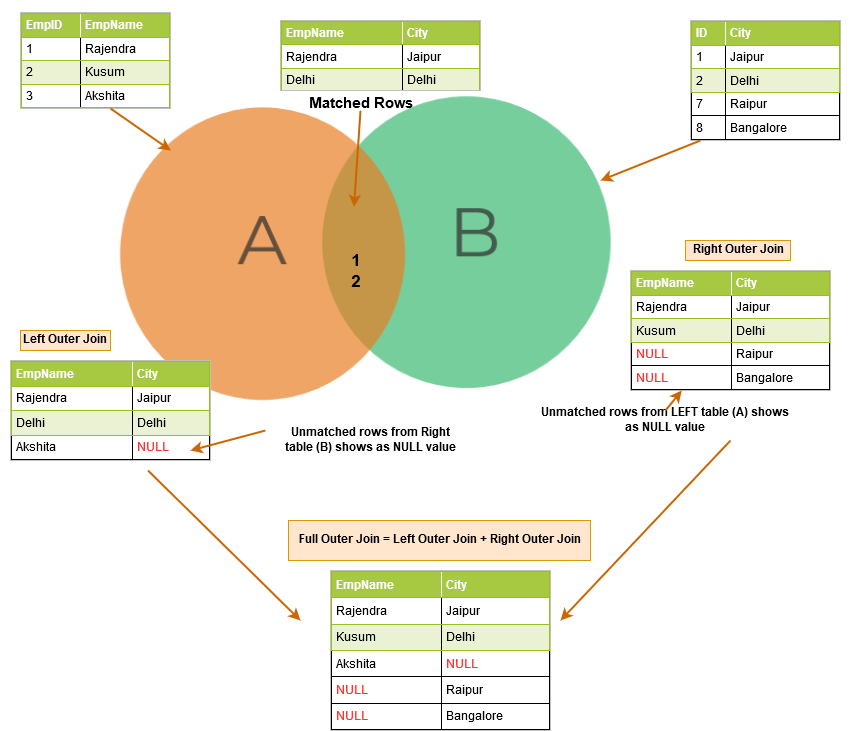
**JOIN Product P ON P.Id = I.ProductId**

**ORDER BY O.OrderNumber**

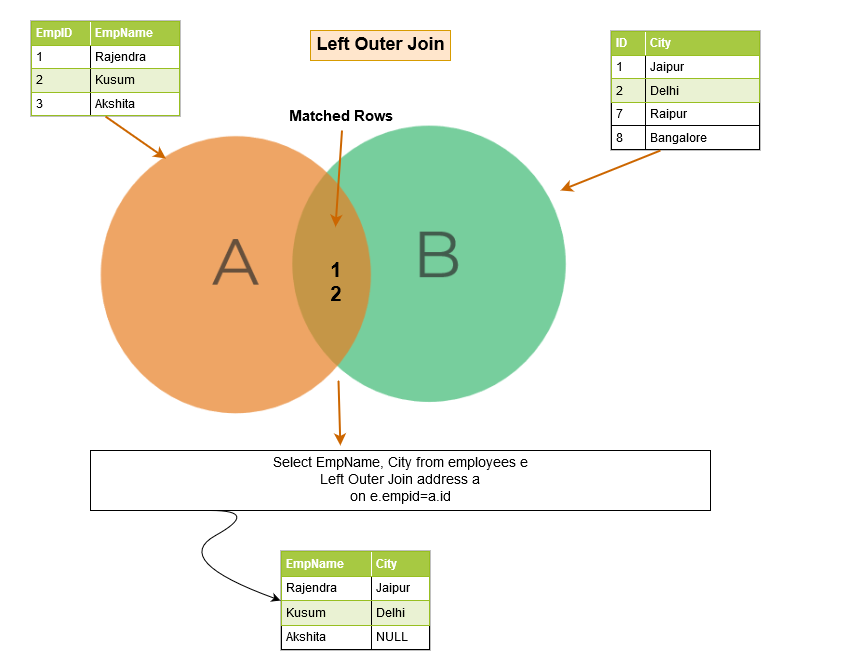
### OUTER JOINSQL outer join

As we explained earlier, the inner join returns the matching rows from both of the tables. When using a SQL outer join, it not only lists the matching rows, but it also returns the unmatched rows from the other tables. The unmatched row depends on the left, right or full keywords.

The below image describes at a high-level the left, right and full outer join.



## Left OuterJoin:exampleIntroduction to SQL Server LEFT JOIN clause



The LEFT JOIN is a clause of the SELECT statement. The LEFT JOIN clause allows you to [query data](https://www.sqlservertutorial.net/sql-server-basics/sql-server-select/) from multiple tables.

The LEFT JOIN returns all rows from the left table and the matching rows from the right table. If no matching rows are found in the right table, NULL are used.

The following illustrates how to join two tables T1 and T2 using the LEFT JOIN clause:

SELECT

select\_list

FROM

T1

LEFT JOIN T2 ON

join\_predicate;

Code language: SQL (Structured Query Language) (sql)

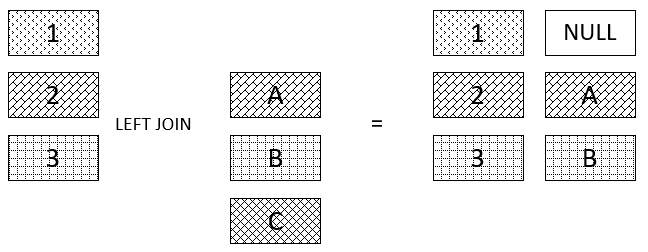
In this syntax, T1 and T2 are the left and right tables, respectively.

For each row from the T1 table, the query compares it with all the rows from the T2 table. If a pair of rows causes the join predicate to evaluate to TRUE, the column values from these rows will be combined to form a new row which is then included in the result set.

If a row from the left table (T1) does not have any matching row from the T2 table, the query combines column values of the row from the left table with NULL for each column values from the right table.

In short, the LEFT JOIN clause returns all rows from the left table (T1) and matching rows or NULL values from the right table (T2).

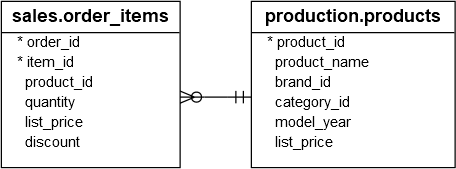
The following illustrates the LEFT JOIN of two tables T1(1, 2, 3) and T2(A, B, C). The LEFT JOIN will match rows from the T1 table with the rows from the T2 table using patterns:



In this illustration, no row from the T2 table matches row 1 from the T1 table; therefore, NULL is used. Rows 2 and 3 from the T1 table match rows A and B from the T2 table, respectively.

## SQL Server LEFT JOIN example

See the following products and order\_items tables:



Each sales order item includes one product. The link between the order\_items and the products tables is the product\_id column.

The following statement uses the LEFT JOIN clause to query data from the products and order\_items tables:

SELECT

product\_name,

order\_id

FROM

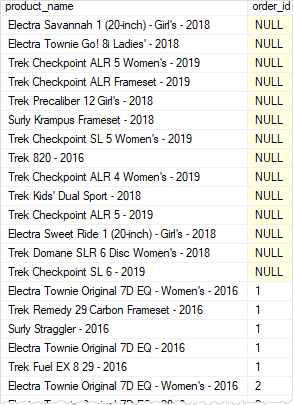
production.products p

LEFT JOIN sales.order\_items o ON o.product\_id = p.product\_id

ORDER BY

order\_id;

Code language: SQL (Structured Query Language) (sql)



As you see clearly from the result set, a list of NULL in the order\_id column indicates that the corresponding products have not been sold to any customer yet.

It is possible to use the [WHERE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-where/) clause to limit the result set. The following query returns the products that do not appear in any sales order:

SELECT

product\_name,

order\_id

FROM

production.products p

LEFT JOIN sales.order\_items o ON o.product\_id = p.product\_id

WHERE order\_id IS NULL

ORDER BY

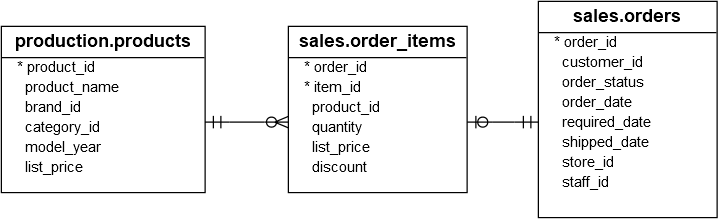
order\_id;

Code language: SQL (Structured Query Language) (sql)



As always, SQL Server processes the WHERE clause after the LEFT JOIN clause.

The following example shows how to join three tables: production.products, sales.orders, and sales.order\_items using the LEFT JOIN clauses:



SELECT

p.product\_name,

o.order\_id,

i.item\_id,

o.order\_date

FROM

production.products p

LEFT JOIN sales.order\_items i

ON i.product\_id = p.product\_id

LEFT JOIN sales.orders o

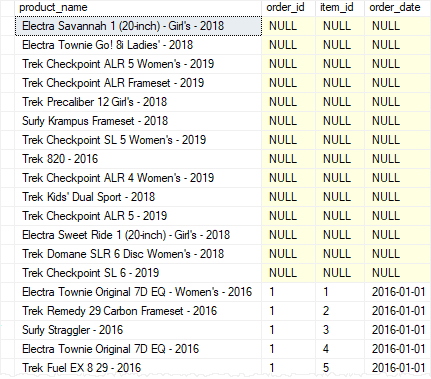
ON o.order\_id = i.order\_id

ORDER BY

order\_id;

Code language: SQL (Structured Query Language) (sql)

Here is the output:



## SQL Server LEFT JOIN: conditions in ON vs. WHERE clause

The following query finds the products that belong to the order id 100:

SELECT

product\_name,

order\_id

FROM

production.products p

LEFT JOIN sales.order\_items o

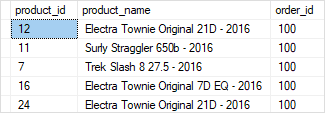
ON o.product\_id = p.product\_id

WHERE order\_id = 100

ORDER BY

order\_id;

Code language: SQL (Structured Query Language) (sql)



Let’s move the condition order\_id = 100 to the ON clause:

SELECT

p.product\_id,

product\_name,

order\_id

FROM

production.products p

LEFT JOIN sales.order\_items o

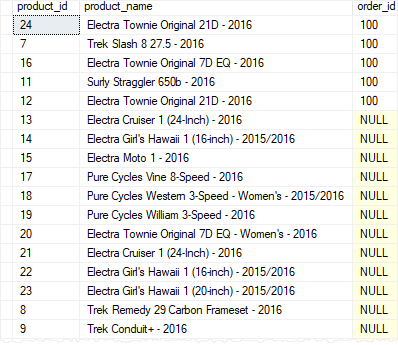
ON o.product\_id = p.product\_id AND

o.order\_id = 100

ORDER BY

order\_id DESC;

Code language: SQL (Structured Query Language) (sql)



The query returned all products, but only the order with id 100 has the associated product’s information.

Note that for the [INNER JOIN](https://www.sqlservertutorial.net/sql-server-basics/sql-server-inner-join/) clause, the condition in the ON clause is functionally equivalent if it is placed in the WHERE clause.

**List all products and their total sales, including those that did not sell.**

**SELECT ProductName, SUM(I.UnitPrice) AS Total**

**FROM Product P**

**LEFT OUTER JOIN OrderItem I ON P.Id = I.ProductId**

**GROUP BY ProductName**

**ORDER BY SUM(I.UnitPrice**

### Example

### [#](https://www.dofactory.com/sql/join-multiple-tables#example)

Problem: List all products that have sold. Include supplier and order numbers. Sort by order number.

**SELECT OrderNumber, CompanyName, ProductName**

**FROM Product P**

**JOIN Supplier S ON S.Id = P.SupplierId**

**JOIN OrderItem I ON P.Id = I.ProductId**

**JOIN [Order] O ON O.Id = I.OrderId**

**ORDER BY OrderNumber**

JOIN with Multiple Tables

Problem: List all orders, sorted by order number, with supplier names, product names, quantities, prices.

|  |
| --- |
| **PRODUCT** |
| Id |
| ProductName |
| SupplierId |
| UnitPrice |
| Package |
| IsDiscontinued |

|  |
| --- |
| **ORDERITEM** |
| Id |
| OrderId |
| ProductId |
| UnitPrice |
| Quantity |

|  |
| --- |
| **ORDER** |
| Id |
| OrderDate |
| OrderNumber |
| CustomerId |
| TotalAmount |

**SELECT O.OrderNumber, S.CompanyName AS Supplier,**

**P.ProductName, I.Quantity, I.UnitPrice**

**FROM [Order] O**

**JOIN OrderItem I ON O.Id = I.OrderId**

**JOIN Product P ON P.Id = I.ProductId**

**JOIN Supplier S ON S.Id = P.SupplierId**

**ORDER BY O.OrderNumber**

LEFT JOIN example

|  |
| --- |
| **ORDER** |
| Id |
| OrderDate |
| OrderNumber |
| CustomerId |
| TotalAmount |

|  |
| --- |
| **CUSTOMER** |
| Id |
| FirstName |
| LastName |
| City |
| Country |
| Phone |

Left Outer Joine

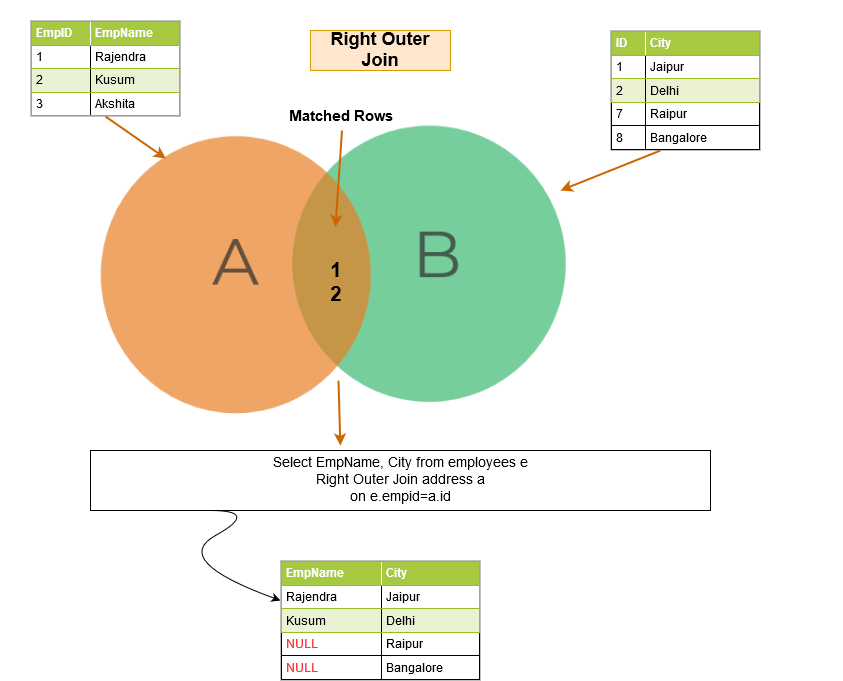
Problem: List all customers and the total amount they spent irrespective whether they placed any orders or not.

**SELECT OrderNumber, TotalAmount, FirstName, LastName, City, Country**

**FROM Customer C**

**LEFT JOIN [Order] O ON O.CustomerId = C.Id**

**ORDER BY TotalAmount**

RIGHT OUTER JOIN

## Introduction to the SQL Server RIGHT JOIN clause

The RIGHT JOIN is a clause of the [SELECT](https://www.sqlservertutorial.net/sql-server-basics/sql-server-select/) statement. The RIGHT JOIN clause combines data from two or more tables.

The RIGHT JOIN clause starts selecting data from the right table and matching it 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.

If a row in the right table does not have any matching rows from the left table, the column of the left table in the result set will have nulls.

The following shows the syntax of the RIGHT JOIN clause:

SELECT

select\_list

FROM

T1

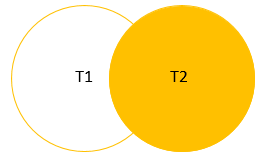
RIGHT JOIN T2 ON join\_predicate;

Code language: SQL (Structured Query Language) (sql)

In this syntax, T1 is the left table and T2 is the right table.

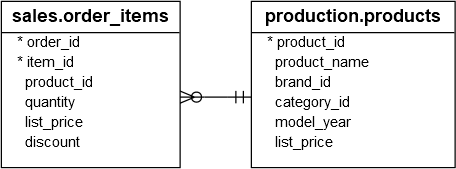
Note that RIGHT JOIN and RIGHT OUTER JOIN is the same. The OUTER keyword is optional.

The following Venn diagram illustrates the RIGHT JOIN operation:



## SQL Server RIGHT JOIN example

We will use the sales.order\_items and production.products table from the [sample database](https://www.sqlservertutorial.net/sql-server-sample-database/) for the demonstration.



The following statement returns all order\_id from the sales.order\_items and product name from the production.products table:

SELECT

product\_name,

order\_id

FROM

sales.order\_items o

RIGHT JOIN production.products p

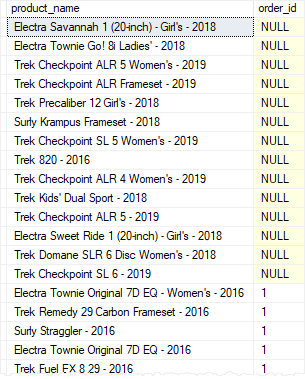
ON o.product\_id = p.product\_id

ORDER BY

order\_id;

Code language: SQL (Structured Query Language) (sql)

Here is the output:



The query returned all rows from the production.products table (right table) and rows from sales.order\_items table (left table). If a product does not have any sales, the order\_id column will have a null.

To get the products that do not have any sales, you add a [WHERE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-where/) clause to the above query to filter out the products that have sales:

SELECT

product\_name,

order\_id

FROM

sales.order\_items o

RIGHT JOIN production.products p

ON o.product\_id = p.product\_id

WHERE

order\_id IS NULL

ORDER BY

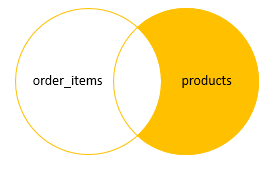
product\_name;

Code language: SQL (Structured Query Language) (sql)

The following picture shows the output:



The following Venn diagram illustrates the above RIGHT JOIN operation:



List all products that have no orders.

**More Examples**

[**#**](https://www.dofactory.com/sql/right-join#examples)

RIGHT JOIN example

|  |
| --- |
| **CUSTOMER** |
| Id |
| FirstName |
| LastName |
| City |
| Country |
| Phone |

|  |
| --- |
| **ORDER** |
| Id |
| OrderDate |
| OrderNumber |
| CustomerId |
| TotalAmount |

Problem: List customers that have not placed orders.

**SELECT FirstName, LastName, City, Country, TotalAmount**

**FROM [Order] O**

**RIGHT JOIN Customer C ON O.CustomerId = C.Id**

**WHERE TotalAmount IS NULL**

RIGHT JOIN, with a JOIN

Problem: List all customers -- with or without orders -- and a count of the orders that include a '2kg box with Konbu' (product with Id = 13). Sort the results by number of orders.

|  |
| --- |
| **CUSTOMER** |
| Id |
| FirstName |
| LastName |
| City |
| Country |
| Phone |

|  |
| --- |
| **ORDER** |
| Id |
| OrderDate |
| OrderNumber |
| CustomerId |
| TotalAmount |

|  |
| --- |
| **ORDERITEM** |
| Id |
| OrderId |
| ProductId |
| UnitPrice |
| Quantity |

**SELECT DISTINCT (C.Id), Firstname, LastName, COUNT(O.Id) AS Orders**

**FROM [Order] O**

**JOIN OrderItem I ON O.Id = I.OrderId AND I.ProductId = 13**

**RIGHT JOIN Customer C ON C.Id = O.CustomerId**

**GROUP BY C.Id, FirstName, LastName**

**ORDER BY COUNT(O.Id)**

FULL JOIN, 2 tables

|  |
| --- |
| **SUPPLIER** |
| Id |
| CompanyName |
| ContactName |
| City |
| Country |
| Phone |
| Fax |

|  |
| --- |
| **CUSTOMER** |
| Id |
| FirstName |
| LastName |
| City |
| Country |
| Phone |

Full Outer JOIN

## Introduction to SQL Server full outer join

The FULL OUTER JOIN is a clause of the [SELECT](https://www.sqlservertutorial.net/sql-server-basics/sql-server-select/) statement. The FULL OUTER JOIN clause returns a result set that includes rows from both left and right tables.

When no matching rows exist for the row in the left table, the columns of the right table will contain [NULL](https://www.sqlservertutorial.net/sql-server-basics/sql-server-null/). Likewise, when no matching rows exist for the row in the right table, the column of the left table will contain NULL.

The following shows the syntax of FULL OUTER JOIN clause when joining two tables T1 and T2:

SELECT

select\_list

FROM

T1

FULL OUTER JOIN T2 ON join\_predicate;

Code language: SQL (Structured Query Language) (sql)

The OUTER keyword is optional so you can skip it as shown in the following query:

SELECT

select\_list

FROM

T1

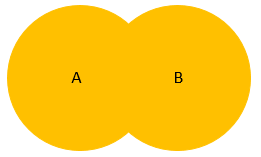
FULL JOIN T2 ON join\_predicate;

Code language: SQL (Structured Query Language) (sql)

In this syntax:

* First, specify the left table T1 in the FROM clause.
* Second, specify the right table T2 and a join predicate.

The following Venn diagram illustrates the FULL OUTER JOIN of two result sets:



## SQL Server full outer join example

Let’s set up some sample table to demonstrate the full outer join.

First, create a new schema named pm which stands for project management:

CREATE SCHEMA pm;

GO

Code language: SQL (Structured Query Language) (sql)

Next, create new tables named projects and members in the pm schema:

CREATE TABLE pm.projects(

id INT PRIMARY KEY IDENTITY,

title VARCHAR(255) NOT NULL

);

CREATE TABLE pm.members(

id INT PRIMARY KEY IDENTITY,

name VARCHAR(120) NOT NULL,

project\_id INT,

FOREIGN KEY (project\_id)

REFERENCES pm.projects(id)

);

Code language: SQL (Structured Query Language) (sql)

Suppose, each member only can participate in one project and each project has zero or more members. If a project is in the initial phase, hence there is no member assigned.

Then, [insert some rows](https://www.sqlservertutorial.net/sql-server-basics/sql-server-insert-multiple-rows/) into the projects and members tables:

INSERT INTO

pm.projects(title)

VALUES

('New CRM for Project Sales'),

('ERP Implementation'),

('Develop Mobile Sales Platform');

INSERT INTO

pm.members(name, project\_id)

VALUES

('John Doe', 1),

('Lily Bush', 1),

('Jane Doe', 2),

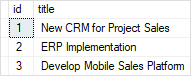
('Jack Daniel', null);

Code language: SQL (Structured Query Language) (sql)

After that, query data from the projects and members tables:

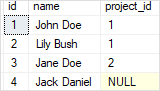
SELECT \* FROM pm.projects;

Code language: SQL (Structured Query Language) (sql)



SELECT \* FROM pm.members;

Code language: SQL (Structured Query Language) (sql)



Finally, use the FULL OUTER JOIN to query data from projects and members tables:

SELECT

m.name member,

p.title project

FROM

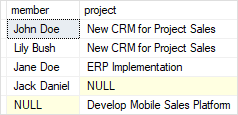
pm.members m

FULL OUTER JOIN pm.projects p

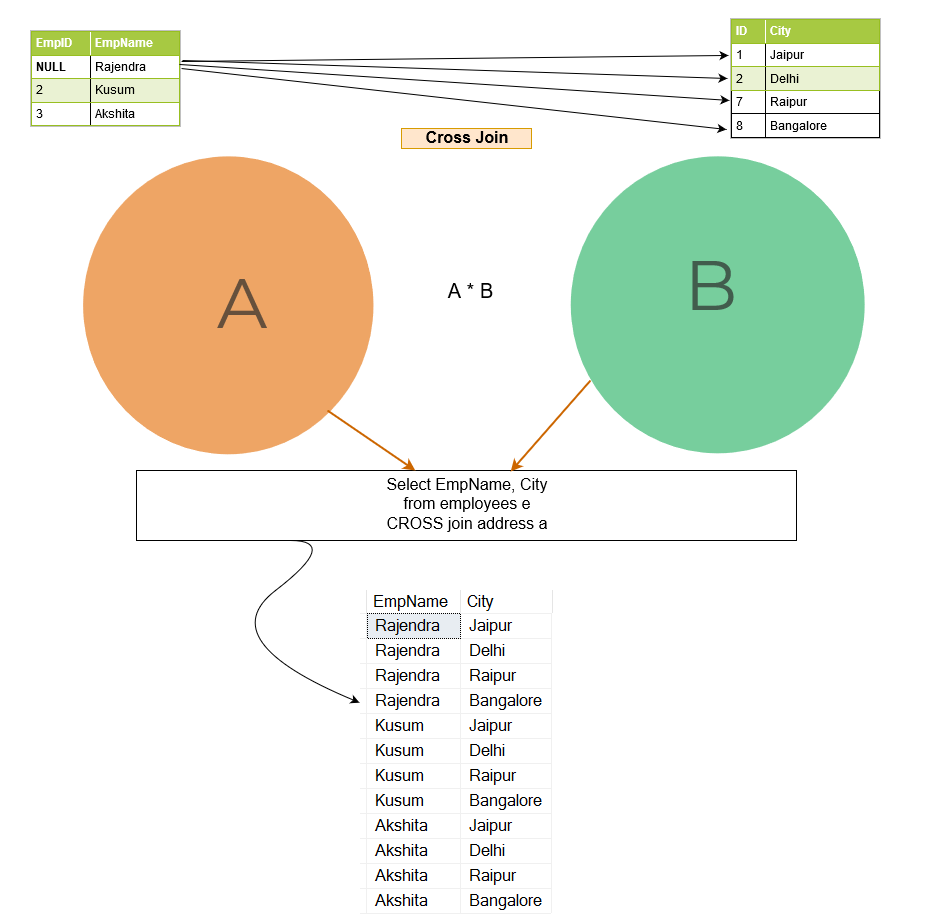
ON p.id = m.project\_id;

Code language: SQL (Structured Query Language) (sql)

Here is the output:



In this example, the query returned members who participate in projects, members who do not participate in any projects, and projects which do not have any members.



To find the members who do not participate in any project and projects which do not have any members, you add a [WHERE](https://www.sqlservertutorial.net/sql-server-basics/sql-server-where/) clause to the above query:

SELECT

m.name member,

p.title project

FROM

pm.members m

FULL OUTER JOIN pm.projects p

ON p.id = m.project\_id

WHERE

m.id IS NULL OR

P.id IS NULL;

Code language: SQL (Structured Query Language) (sql)

The following picture shows the output:

SQL Server full outer join with a WHERE clause example

As clearly shown in the output, Jack Daniel does not participate in any project and Develop Mobile Sales Platform does not have any members.

Problem: Match all customers and suppliers by country.

**SELECT C.FirstName, C.LastName, C.Country AS CustomerCountry,**

**S.Country AS SupplierCountry, S.CompanyName**

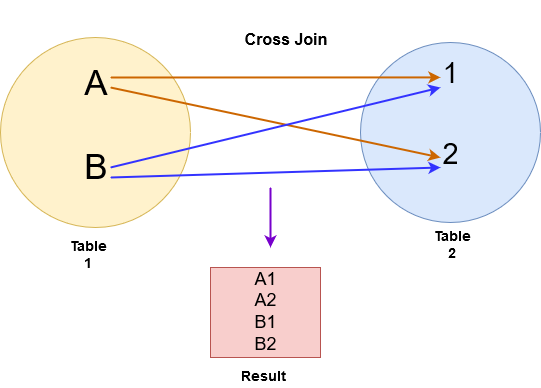
**FROM Customer C**

**FULL JOIN Supplier S ON C.Country = S.Country**

**ORDER BY C.Country, S.Country**

## QL Server CROSS JOIN examples

The following statement returns the combinations of all products and stores. The result set can be used for stocktaking procedure during the month-end and year-end closings:



SELECT

product\_id,

product\_name,

store\_id,

0 AS quantity

FROM

production.products

CROSS JOIN sales.stores

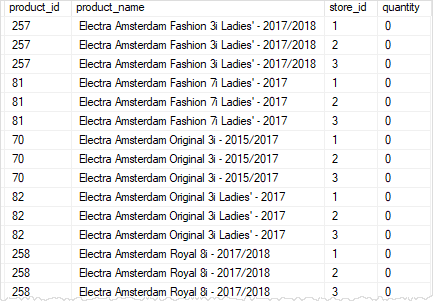
ORDER BY

product\_name,

store\_id;

Code language: SQL (Structured Query Language) (sql)

Here is the partial output:



The following statement finds the products that have no sales across the stores:

SELECT

s.store\_id,

p.product\_id,

ISNULL(sales, 0) sales

FROM

sales.stores s

CROSS JOIN production.products p

LEFT JOIN (

SELECT

s.store\_id,

p.product\_id,

SUM (quantity \* i.list\_price) sales

FROM

sales.orders o

INNER JOIN sales.order\_items i ON i.order\_id = o.order\_id

INNER JOIN sales.stores s ON s.store\_id = o.store\_id

INNER JOIN production.products p ON p.product\_id = i.product\_id

GROUP BY

s.store\_id,

p.product\_id

) c ON c.store\_id = s.store\_id

AND c.product\_id = p.product\_id

WHERE

sales IS NULL

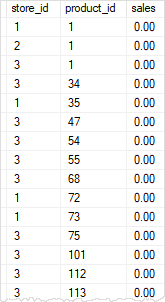
ORDER BY

product\_id,

store\_id;

Code language: SQL (Structured Query Language) (sql)

The following picture shows the partial result set:



In this tutorial, you have learned how to use the SQL Server CROSS JOIN to create Cartesian products of rows from the joined tables

## SQL Server self join syntax

A self join allows you to join a table to itself. It helps query hierarchical data or compare rows within the same table.

A self join uses the [inner join](https://www.sqlservertutorial.net/sql-server-basics/sql-server-inner-join/) or [left join](https://www.sqlservertutorial.net/sql-server-basics/sql-server-left-join/) clause. Because the query that uses the self join references the same table, the [table alias](https://www.sqlservertutorial.net/sql-server-basics/sql-server-alias/) is used to assign different names to the same table within the query.

Note that referencing the same table more than one in a query without using table aliases will result in an error.

The following shows the syntax of joining the table T to itself:

SELECT

select\_list

FROM

T t1

[INNER | LEFT] JOIN T t2 ON

join\_predicate;

Code language: SQL (Structured Query Language) (sql)

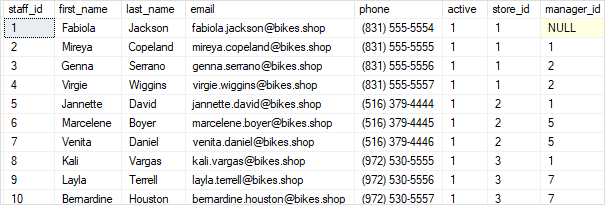
The query references the table T twice. The table aliases t1 and t2 are used to assign the T table different names in the query.

## SQL Server self join examples

Let’s take some examples to understand how the self join works.

### 1) Using self join to query hierarchical data

Consider the following  staffs table from the [sample database](https://www.sqlservertutorial.net/sql-server-sample-database/):



The  staffs table stores the staff information such as id, first name, last name, and email. It also has a column named manager\_id that specifies the direct manager. For example, Mireya reports to Fabiola because the value in the manager\_id of  Mireya is Fabiola.

Fabiola has no manager, so the manager id column has a NULL.

To get who reports to whom, you use the self join as shown in the following query:

SELECT

e.first\_name + ' ' + e.last\_name employee,

m.first\_name + ' ' + m.last\_name manager

FROM

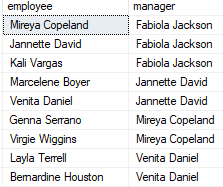
sales.staffs e

INNER JOIN sales.staffs m ON m.staff\_id = e.manager\_id

ORDER BY

manager;

Code language: SQL (Structured Query Language) (sql)



In this example, we referenced to the  staffs table twice: one as e for the employees and the other as m for the managers. The join predicate matches employee and manager relationship using the values in the e.manager\_id and m.staff\_id columns.

The employee column does not have Fabiola Jackson because of the [INNER JOIN](https://www.sqlservertutorial.net/sql-server-basics/sql-server-inner-join/) effect. If you replace the [INNER JOIN](https://www.sqlservertutorial.net/sql-server-basics/sql-server-inner-join/) clause by the [LEFT JOIN](https://www.sqlservertutorial.net/sql-server-basics/sql-server-left-join/) clause as shown in the following query, you will get the result set that includes Fabiola Jackson in the employee column:

SELECT

e.first\_name + ' ' + e.last\_name employee,

m.first\_name + ' ' + m.last\_name manager

FROM

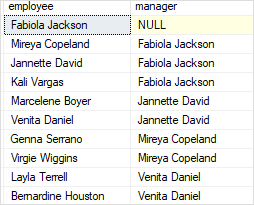
sales.staffs e

LEFT JOIN sales.staffs m ON m.staff\_id = e.manager\_id

ORDER BY

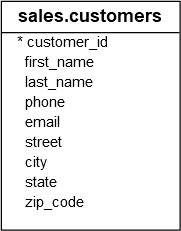
manager;

Code language: SQL (Structured Query Language) (sql)



### 2) Using self join to compare rows within a table

See the following customers table:



The following statement uses the self join to find the customers located in the same city.

SELECT

c1.city,

c1.first\_name + ' ' + c1.last\_name customer\_1,

c2.first\_name + ' ' + c2.last\_name customer\_2

FROM

sales.customers c1

INNER JOIN sales.customers c2 ON c1.customer\_id > c2.customer\_id

AND c1.city = c2.city

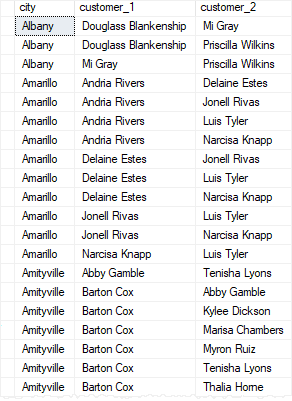
ORDER BY

city,

customer\_1,

customer\_2;

Code language: SQL (Structured Query Language) (sql)



The following condition makes sure that the statement doesn’t compare the same customer:

c1.customer\_id > c2.customer\_id

Code language: SQL (Structured Query Language) (sql)

And the following condition matches the city of the two customers:

AND c1.city = c2.city

Code language: SQL (Structured Query Language) (sql)

Note that if you change the greater than ( > ) operator by the not equal to (<>) operator, you will get more rows:

SELECT

c1.city,

c1.first\_name + ' ' + c1.last\_name customer\_1,

c2.first\_name + ' ' + c2.last\_name customer\_2

FROM

sales.customers c1

INNER JOIN sales.customers c2 ON c1.customer\_id <> c2.customer\_id

AND c1.city = c2.city

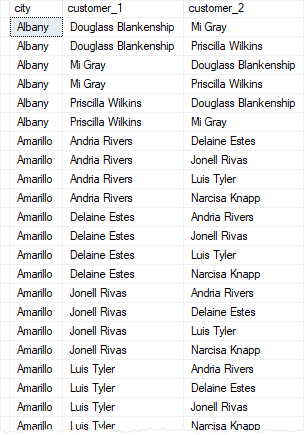
ORDER BY

city,

customer\_1,

customer\_2;

Code language: SQL (Structured Query Language) (sql)



Let’s see the difference between > and <> in the ON clause by limiting to one city to make it easier for comparison.

The following query returns the customers located in Albany:

SELECT

customer\_id, first\_name + ' ' + last\_name c,

city

FROM

sales.customers

WHERE

city = 'Albany'

ORDER BY

c;

Code language: SQL (Structured Query Language) (sql)



This query uses ( >) operator in the ON clause:

SELECT

c1.city,

c1.first\_name + ' ' + c1.last\_name customer\_1,

c2.first\_name + ' ' + c2.last\_name customer\_2

FROM

sales.customers c1

INNER JOIN sales.customers c2 ON c1.customer\_id > c2.customer\_id

AND c1.city = c2.city

WHERE c1.city = 'Albany'

ORDER BY

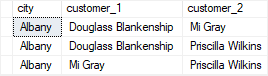
c1.city,

customer\_1,

customer\_2;

Code language: SQL (Structured Query Language) (sql)

The output is:



This query uses ( <>) operator in the ON clause:

SELECT

c1.city,

c1.first\_name + ' ' + c1.last\_name customer\_1,

c2.first\_name + ' ' + c2.last\_name customer\_2

FROM

sales.customers c1

INNER JOIN sales.customers c2 ON c1.customer\_id <> c2.customer\_id

AND c1.city = c2.city

WHERE c1.city = 'Albany'

ORDER BY

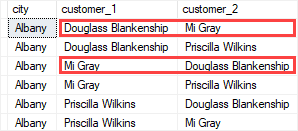
c1.city,

customer\_1,

customer\_2;

Code language: SQL (Structured Query Language) (sql)

Here is the output:

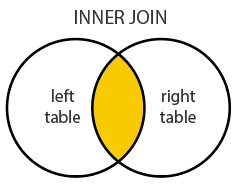


# SQL UPDATE with JOIN

An UPDATE statement can include JOIN operations.

An UPDATE can contain zero, one, or multiple JOIN operations.

The UPDATE affects records that satisfy the JOIN conditions.



### Example

### [#](https://www.dofactory.com/sql/update-join#example)

Increase the unit price by 10% for all products that have been sold before.

**UPDATE P**

**SET P.UnitPrice = P.UnitPrice \* 1.1**

**FROM Product P**

**JOIN OrderItem I ON P.Id = I.ProductId**

Try it live

P and I are table [aliases](https://www.dofactory.com/sql/alias).

Result:  77 records updated

[SQL Self Join](https://www.dofactory.com/sql/self-join)

[SQL Delete Join](https://www.dofactory.com/sql/delete-join)

## Syntax

## [#](https://www.dofactory.com/sql/update-join#syntax)

JOIN syntax.

1. **UPDATE table-name1**
2. **SET column-name1 = value1,**
3. **column-name2 = value2, ...**
4. **FROM table-name1**
5. **JOIN table-name2 ON column-name3 = column-name4**
6. **WHERE condition**

INNER JOIN syntax.

1. **UPDATE table-name1**
2. **SET column-name1 = value1,**
3. **column-name2 = value2, ...**
4. **FROM table-name1**
5. **INNER JOIN table-name2 ON column-name3 = column-name4**
6. **WHERE condition**

JOIN is the same as INNER JOIN; the INNER keyword is optional.  
JOIN, or INNER JOIN, is the most commonly used type of JOIN operation.

## More Examples

## [#](https://www.dofactory.com/sql/update-join#examples)

## UPDATE with LEFT JOIN

|  |
| --- |
| **PRODUCT** |
| Id |
| ProductName |
| SupplierId |
| UnitPrice |
| Package |
| IsDiscontinued |

|  |
| --- |
| **ORDERITEM** |
| Id |
| OrderId |
| ProductId |
| UnitPrice |
| Quantity |

Problem: Discontinue products that have not sold.

**UPDATE P**

**SET IsDiscontinued = 1**

**FROM Product P**

**LEFT JOIN OrderItem I ON P.Id = I.ProductId**

**WHERE I.Id IS NULL**

Try it live

This UPDATE uses a [LEFT JOIN](https://www.dofactory.com/sql/left-join) operation.

Result:  1 record updated.

## UPDATE with 3 JOINs

Problem: For customer Paul Henriot change the unit price to $25 for Queso Calibres in their order.

|  |
| --- |
| **PRODUCT** |
| Id |
| ProductName |
| SupplierId |
| UnitPrice |
| Package |
| IsDiscontinued |

|  |
| --- |
| **ORDERITEM** |
| Id |
| OrderId |
| ProductId |
| UnitPrice |
| Quantity |

|  |
| --- |
| **ORDER** |
| Id |
| OrderDate |
| OrderNumber |
| CustomerId |
| TotalAmount |

**UPDATE I**

**SET I.UnitPrice = 25**

**FROM Customer C**

**JOIN [Order] O ON O.CustomerId = C.Id**

**JOIN OrderItem I ON O.Id = I.OrderId**

**JOIN Product P ON P.Id = I.ProductId**

**WHERE C.FirstName = 'Paul' AND C.LastName = 'Henriot'**

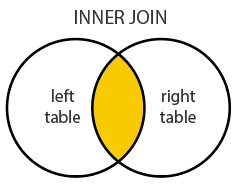
**AND P.ProductName = 'Queso Cabrales'**

# SQL DELETE with JOIN

A DELETE statement can include JOIN operations.

It can contain zero, one, or multiple JOIN operations.

The DELETE removes records that satisfy the JOIN conditions.



### Example

### [#](https://www.dofactory.com/sql/delete-join#example)

Remove products that have not sold.

**DELETE P**

**FROM Product P**

**LEFT JOIN OrderItem I ON P.Id = I.ProductId**

**WHERE I.Id IS NULL**

Try it live

This problem requires a LEFT JOIN.  
P and I are table [aliases](https://www.dofactory.com/sql/alias).

Result:  1 record deleted

[SQL Update Join](https://www.dofactory.com/sql/update-join)

[SQL Subquery](https://www.dofactory.com/sql/subquery)

## Syntax

## [#](https://www.dofactory.com/sql/delete-join#syntax)

JOIN syntax.

1. **DELETE table-name1**
2. **FROM table-name1**
3. **JOIN table-name2 ON column-name3 = column-name4**
4. **WHERE condition**

INNER JOIN syntax.

1. **DELETE table-name1**
2. **FROM table-name1**
3. **INNER JOIN table-name2 ON column-name3 = column-name4**
4. **WHERE condition**

JOIN is the same as INNER JOIN; the INNER keyword is optional.

## More Examples

## [#](https://www.dofactory.com/sql/delete-join#examples)

## DELETE with JOIN

|  |
| --- |
| **PRODUCT** |
| Id |
| ProductName |
| SupplierId |
| UnitPrice |
| Package |
| IsDiscontinued |

|  |
| --- |
| **ORDERITEM** |
| Id |
| OrderId |
| ProductId |
| UnitPrice |
| Quantity |

Problem: For order number 542379 remove the Tofu.

**DELETE OrderItem**

**FROM OrderItem**

**JOIN [Order] on OrderItem.OrderId = [Order].Id**

**JOIN Product ON Product.Id = OrderItem.ProductId**

**WHERE OrderNumber = '542379'**

**AND ProductName = 'Tofu'**