SQL Server JOINS

In real life, we store our data in multiple logical tables that are linked together by a common key value in relational databases like SQL Server, Oracle, MySQL, and others. As a result, we constantly need to get data from two or more tables into the desired output based on some conditions. We can quickly achieve this type of data in SQL Server using the SQL JOIN clause. This article gives a complete overview of JOIN and its different types with an example.

The join clause allows us to **retrieve data from two or more related tables** into a meaningful result set. We can join the table using a **SELECT** statement and a **join condition**. It indicates how SQL Server can use data from one table to select rows from another table. In general, tables are related to each other using **foreign key** constraints.

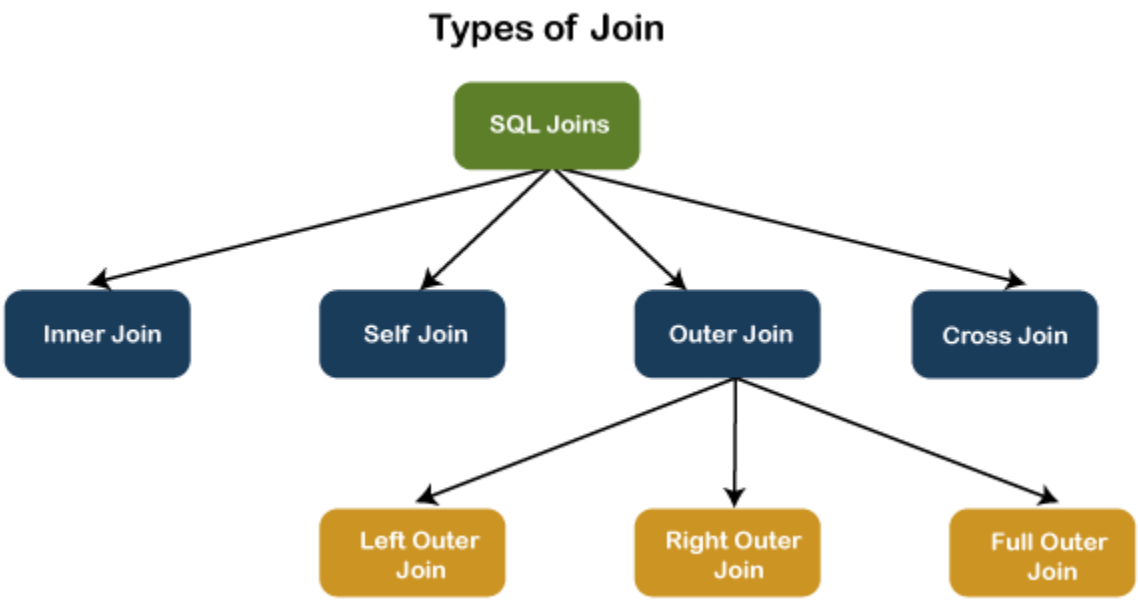
In a JOIN query, a condition indicates how two tables are related:

- Choose columns from each table that should be used in the join. A join condition indicates a foreign key from one table and its corresponding key in the other table.
- Specify the logical operator to compare values from the columns like =, <, or >.

Types of JOINS in SQL Server

SQL Server mainly supports **four types of JOINS**, and each join type defines how two tables are related in a query. The following are types of join supports in SQL Server:

1. INNER JOIN
2. SELF JOIN
3. CROSS JOIN
4. OUTER JOIN

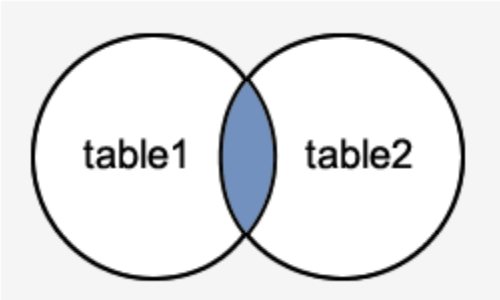


Let us discuss each of these joins in detail.

INNER JOIN

This JOIN returns all records from multiple tables that satisfy the specified join condition. It is the simple and most popular form of join and assumes as a **default join**. If we omit the INNER keyword with the JOIN query, we will get the same output.

The following visual representation explains how INNER JOIN returns the matching records from **table1** and **table2:**



INNER JOIN Syntax

The following syntax illustrates the use of INNER JOIN in SQL Server:

```
SELECT columns
```

```
FROM table1
INNER JOIN table2 ON condition1
INNER JOIN table3 ON condition2
```

INNER JOIN Example

Let us first create two tables "**Student**" and "**Fee**" using the following statement:

```
CREATE TABLE Student (
  id int PRIMARY KEY IDENTITY,
  admission_no varchar(45) NOT NULL,
  first_name varchar(45) NOT NULL,
  last_name varchar(45) NOT NULL,
  age int,
  city varchar(25) NOT NULL
);

CREATE TABLE Fee (
  admission_no varchar(45) NOT NULL,
  course varchar(45) NOT NULL,
  amount_paid int,
);
```

Next, we will insert some records into these tables using the below statements:

```
INSERT INTO Student (admission_no, first_name, last_name, age, city)
VALUES (3354,'Luisa', 'Evans', 13, 'Texas'),
(2135, 'Paul', 'Ward', 15, 'Alaska'),
(4321, 'Peter', 'Bennett', 14, 'California'),
(4213,'Carlos', 'Patterson', 17, 'New York'),
(5112, 'Rose', 'Huges', 16, 'Florida'),
(6113, 'Marielia', 'Simmons', 15, 'Arizona'),
(7555,'Antonio', 'Butler', 14, 'New York'),
(8345, 'Diego', 'Cox', 13, 'California');
```

```
INSERT INTO Fee (admission_no, course, amount_paid)
VALUES (3354,'Java', 20000),
(7555, 'Android', 22000),
(4321, 'Python', 18000),
(8345,'SQL', 15000),
(5112, 'Machine Learning', 30000);
```

Execute the **SELECT** statement to verify the records:

Table: Student

id	admission_no	first_name	last_name	age	city
1	3354	Luisa	Evans	13	Texas
2	2135	Paul	Ward	15	Alaska
3	4321	Peter	Bennett	14	California
4	4213	Carlos	Patterson	17	New York
5	5112	Rose	Huges	16	Florida
6	6113	Marielia	Simmons	15	Arizona
7	7555	Antonio	Butler	14	New York
8	8345	Diego	Cox	13	California

Table: Fee

admission_no	course	amount_paid
3354	Java	20000
7555	Android	22000
4321	Python	18000
8345	SQL	15000
5112	Machine Learning	30000

We can demonstrate the INNER JOIN using the following command:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fe
e.amount_paid
FROM Student
INNER JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:

admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
4321	Peter	Bennett	Python	18000
5112	Rose	Huges	Machine Learning	30000
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

In this example, we have used the **admission_no column** as a join condition to get the data from both tables. Depending on this table, we can see the information of the students who have paid their fee.

SELF JOIN

A table is joined to itself using the SELF JOIN. It means that **each table row is combined with itself** and with every other table row. The SELF JOIN can be thought of as a JOIN of two copies of the same tables. We can do this with the help of **table name aliases** to assign a specific name to each table's instance. The table aliases enable us to use the **table's temporary** name that we are going to use in the query. It's a useful way to extract hierarchical data and comparing rows inside a single table.

SELF JOIN Syntax

The following expression illustrates the syntax of SELF JOIN in SQL Server. It works the same as the syntax of joining two different tables. Here, we use aliases names for tables because both the table name are the same.

```
SELECT T1.col_name, T2.col_name...
FROM table1 T1, table1 T2
WHERE join_condition;
```

Example

We can demonstrate the SELF JOIN using the following command:

```
SELECT S1.first_name, S2.last_name, S2.city
FROM Student S1, Student S2
WHERE S1.id <> S2.iD AND S1.city = S2.city
ORDER BY S2.city;
```

This command gives the below result:

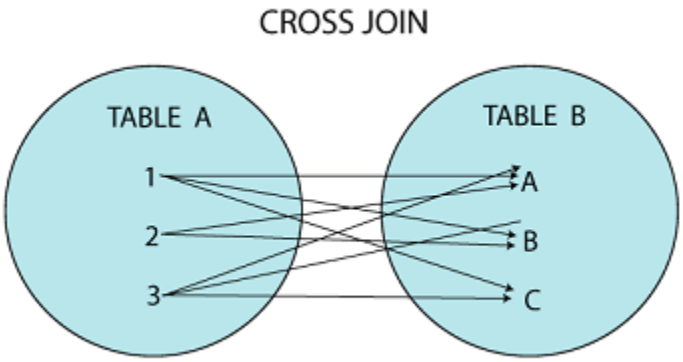
first_name	last_name	City
Peter	Cox	California
Diego	Bennett	California
Carlos	Butler	New York
Antonio	Patterson	New York

In this example, we have used the **id and city column** as a join condition to get the data from both tables.

CROSS JOIN

CROSS JOIN in SQL Server combines all of the possibilities of two or more tables and returns a result that includes every row from all contributing tables. It's also known as **CARTESIAN JOIN** because it produces the **Cartesian product** of all linked tables. The Cartesian product represents all rows present in the first table multiplied by all rows present in the second table.

The below visual representation illustrates the CROSS JOIN. It will give all the records from **table1** and **table2** where each row is the combination of rows of both tables:



CROSS JOIN Syntax

The following syntax illustrates the use of CROSS JOIN in SQL Server:

1. **SELECT** column_lists
2. **FROM** table1
3. CROSS JOIN table2;

Example

We can demonstrate the CROSS JOIN using the following command:

```
1. SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course,
    Fee.amount_paid
2. FROM Student
3. CROSS JOIN Fee
4. WHERE Student.admission_no = Fee.admission_no;
```

This command gives the below result:

admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
4321	Peter	Bennett	Python	18000
5112	Rose	Huges	Machine Learning	30000
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

OUTER JOIN

OUTER JOIN in SQL Server **returns all records from both tables** that satisfy the join condition. In other words, this join will not return only the matching record but also return all unmatched rows from one or both tables.

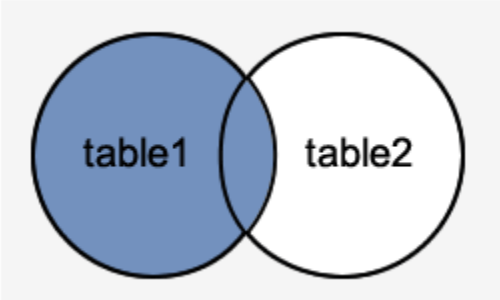
We can categories the OUTER JOIN further into three types:

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

LEFT OUTER JOIN

The LEFT OUTER JOIN **retrieves all the records from the left table and matching rows from the right table**. It will return **NULL** when no matching record is found in the right side table. Since OUTER is an optional keyword, it is also known as LEFT JOIN.

The below visual representation illustrates the LEFT OUTER JOIN:



LEFT OUTER JOIN Syntax

The following syntax illustrates the use of LEFT OUTER JOIN in SQL Server:

```
SELECT column_lists
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Example

We can demonstrate the LEFT OUTER JOIN using the following command:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course,
 Fee.amount_paid
FROM Student
LEFT OUTER JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:

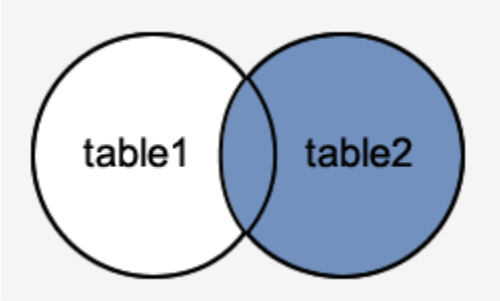
admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
2135	Paul	Ward	NULL	NULL
4321	Peter	Bennett	Python	18000
4213	Carlos	Patterson	NULL	NULL
5112	Rose	Huges	Machine Learning	30000
6113	Marielia	Simmons	NULL	NULL
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

This output shows that the unmatched row's values are replaced with NULLs in the respective columns.

RIGHT OUTER JOIN

The RIGHT OUTER JOIN **retrieves all the records from the right-hand table and matched rows from the left-hand table**. It will return **NULL** when no matching record is found in the left-hand table. Since OUTER is an optional keyword, it is also known as RIGHT JOIN.

The below visual representation illustrates the RIGHT OUTER JOIN:



RIGHT OUTER JOIN Syntax

The following syntax illustrates the use of RIGHT OUTER JOIN in SQL Server:

```
SELECT column_lists
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Example

The following example explains how to use the RIGHT OUTER JOIN to get records from both tables:

```
SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course,
 Fee.amount_paid
FROM Student
RIGHT OUTER JOIN Fee
ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:

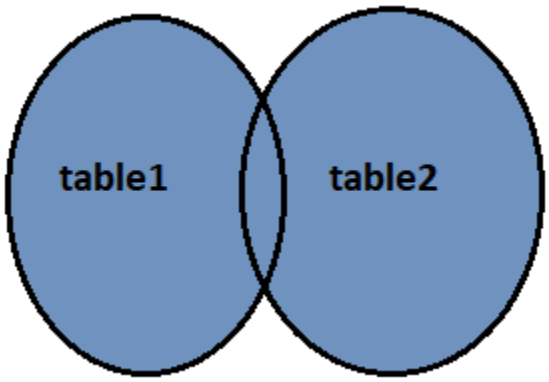
admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
7555	Antonio	Butler	Android	22000
4321	Peter	Bennett	Python	18000
8345	Diego	Cox	SQL	15000
5112	Rose	Huges	Machine Learning	30000

In this output, we can see that no column has NULL values because all rows in the Fee table are available in the Student table based on the specified condition.

FULL OUTER JOIN

The FULL OUTER JOIN in SQL Server **returns a result that includes all rows from both tables**. The columns of the right-hand table return NULL when no matching records are found in the left-hand table. And if no matching records are found in the right-hand table, the left-hand table column returns NULL.

The below visual representation illustrates the FULL OUTER JOIN:



FULL OUTER JOIN Syntax

The following syntax illustrates the use of FULL OUTER JOIN in SQL Server:

```
1. SELECT column_lists
2. FROM table1
3. FULL [OUTER] JOIN table2
4. ON table1.column = table2.column;
```

Example

The following example explains how to use the FULL OUTER JOIN to get records from both tables:

```
1. SELECT Student.admission_no, Student.first_name, Student.last_name, Fee.course,
    Fee.amount_paid
2. FROM Student
3. FULL OUTER JOIN Fee
4. ON Student.admission_no = Fee.admission_no;
```

This command gives the below result:

admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
2135	Paul	Ward	NULL	NULL
4321	Peter	Bennett	Python	18000
4213	Carlos	Patterson	NULL	NULL
5112	Rose	Huges	Machine Learning	30000
6113	Marielia	Simmons	NULL	NULL
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

In this output, we can see that the column has NULL values when no matching records are found in the left-hand and right-hand table based on the specified condition.