devart

A Brief Guide to SQL Server JOINs

with Examples & Best Practices

devart

The fundamentals

SQL Server JOINs are used to retrieve data from two or more tables based on logical relationships between them. The result dataset is represented as a separate table.

Deep knowledge of JOINs is a must for an advanced user. The easiest way to work with them is to use dbForge SQL Complete, an add-in for SSMS and Visual Studio that accelerates routine SQL coding with smart completion, formatting, and refactoring.

With its help, you can handle even the most complex JOINs easily. You don't need to memorize multiple column names or aliases. SQL Complete instantly suggests a full JOIN clause depending on foreign keys or conditions based on column names. These suggestions are available after the JOIN and ON keywords.

Types of SQL JOINs

Each basic type of JOIN defines the way tables are related in a query.

Now let's have a more detailed overview of each type.

INNER JOIN

creates a result table by combining rows that have matching values in two or more tables.

SELF JOIN

joins the table to itself and allows comparing rows within the same table.

CROSS JOIN

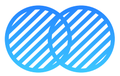
creates a result table containing a paired combination of each row of the first table with each row of the second table.

LEFT OUTER JOIN

delivers a result table that includes unmatched rows from the table that is specified before the LEFT OUTER JOIN clause.

RIGHT OUTER JOIN

delivers a result table that comprises all records from the right table and only matching rows from the left table.



FULL OUTER JOIN

returns a result that includes rows from both left and right tables.

INNER JOIN

INNER JOIN returns only those records or rows that have matching values and is used to retrieve data that appears in both tables.

SYNTAX

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

EXAMPLE

```
SELECT c.CustomerID,
c.CustomerName,
so.ServiceID,
so.ServiceName
FROM Customers c
INNER JOIN ServiceOrders so
ON c.CustomerID = so.CustomerID
```

SELF JOIN

SELF JOIN allows you to join a table to itself. This implies that each row of the table is combined with itself and with every other row of the table. It is most useful for extracting hierarchical data or comparing rows within the same table.

SYNTAX

```
SELECT columns
FROM  table1 a, table1 b
WHERE a.common_field = b.common_field;
```

EXAMPLE

```
SELECT *
FROM ServiceOrders soleft
INNER JOIN ServiceOrders soright
ON soleft.CustomerID < soright.CustomerID
```

CROSS JOIN

CROSS JOIN joins every row from the first table with every row from the second table and returns all combinations of rows. Imagine that you need to find all combinations of size and color. In that case, a CROSS JOIN will come in handy. It does not need any joining condition.

SYNTAX

```
SELECT columns
FROM  table1
CROSS JOIN table2
```

EXAMPLE

```
SELECT c.CustomerName,
so.ServiceName
FROM ServiceOrders so
CROSS JOIN Customers c
```

LEFT OUTER JOIN

LEFT OUTER JOIN delivers the output of the matching rows between two tables. In case no records from the left table match with the records from the right table, it shows those records with null values.

SYNTAX

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

EXAMPLE

```
SELECT c.CustomerID,
c.CustomerName,
so.ServiceID,
o.ServiceName
FROM Customers c
LEFT OUTER JOIN ServiceOrders so
ON c.CustomerID = so.CustomerID
```

RIGHT OUTER JOIN

Similarly to the previous case, RIGHT OUTER JOIN delivers a result table that comprises all records from the right table and only matching rows from the left table. The non-matching rows will have null values.

SYNTAX

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

EXAMPLE

```
SELECT c.CustomerID,
c.CustomerName,
so.ServiceID,
o.ServiceName
FROM Customers c
RIGHT OUTER JOIN ServiceOrders so
ON c.CustomerID = so.CustomerID
```

FULL OUTER JOIN

If you use a FULL OUTER JOIN, it will retrieve both the matching and non-matching rows from both left and right tables.
The non-matching rows will have null values.

SYNTAX

```
SELECT columns
FROM  table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

EXAMPLE

```
SELECT  c.CustomerID,
c.CustomerName,
od.OrderID,
od.OrderDate
FROM Customers c
FULL OUTER JOIN Orders od
ON c.CustomerID = so.CustomerID
```

JOIN algorithms

While logical JOIN operators (e.g., INNER JOIN) define what needs to be done in a query, different physical operators (a.k.a. algorithms) define how it needs to be done. There are three JOIN algorithms in SQL Server.

Nested Loop

The nested loop algorithm is the simplest one, and it is typically used to join smaller tables. It compares each row of one table with each row of the other table looking for rows that meet the JOIN predicate. The nested loop is not the best algorithm for large queries.

Merge

The merge algorithm has both inputs sorted on the merge columns. A merge join can be either a regular or a many-to-many operation, with the latter using a temporary table to store rows. This is generally a very fast algorithm, but it can be expensive if sorting operations are involved.

Hash

The hash algorithm has two inputs: the build input and the probe input. These roles are assigned by the query optimizer, with the build input being the smaller one. This algorithm is used for various set-matching operations. Please note that if both inputs are large, the operation will be expensive.

Best practices

Now as you know the difference between JOIN types,
learn the following tips to make your work with JOINs most effective.

- Use explicit JOINs (i.e. JOIN and ON keywords)
- Choose the appropriate JOIN type carefully
- Apply table aliases when joining multiple tables or tables with long names
- Use the [table alias].[column] name format for columns in queries
- Apply column aliases (the names assigned to the columns in the result dataset)
- If necessary, combine several JOINs for maximum flexibility

Apart from easy handling of JOINs, what else do you get with dbForge SQL Complete?

Download
dbForge SQL Complete
for a **FREE** 14-day trial!

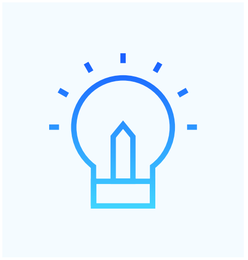
DOWNLOAD

- Double your coding speed with code completion
- Get instant context-based suggestions of database objects
- Beautify your SQL code and unify coding standards with the SQL Formatter
- Eliminate repetitive coding with predefined and custom snippets
- Take advantage of safe and fast refactoring
- Boost code quality with a smart T-SQL Debugger
- Make your work even more efficient with a set of productivity tools

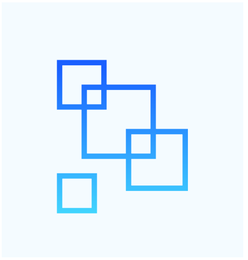
Helpful resources

As a bonus, here are some helpful video tutorials and articles that will help you master SQL JOINs faster.

How to use JOINs in the SELECT statements



Everything you should know about SQL Server JOINs



Different types of JOINs in SQL Server

A comprehensive guide to INNER JOIN

A comprehensive guide to CROSS JOIN