SQL Server self join syntax

A self join allows you to join a table to itself. It is useful for querying hierarchical data or comparing 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 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;

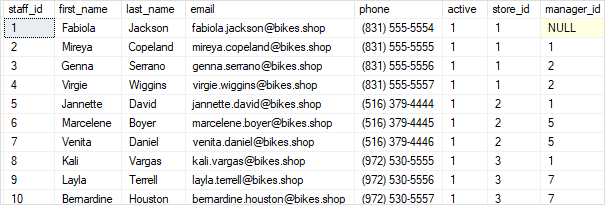
The query references the table T twice. The table aliases t1 and t2 are used to assign the T table different names in the query.

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

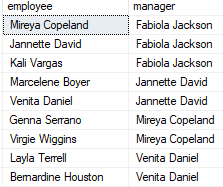
**FROM**

sales.staffs e

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

**ORDER** **BY**

manager;



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

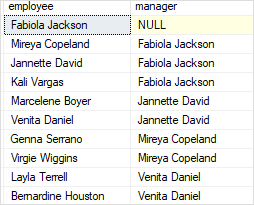
**FROM**

sales.staffs e

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

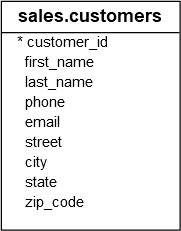
**ORDER** **BY**

manager;



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 who locate 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

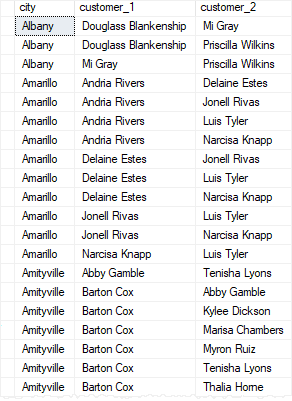
**AND** c1.city = c2.city

**ORDER** **BY**

city,

customer\_1,

customer\_2;



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

c1.customer\_id > c2.customer\_id

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

AND c1.city = c2.city

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

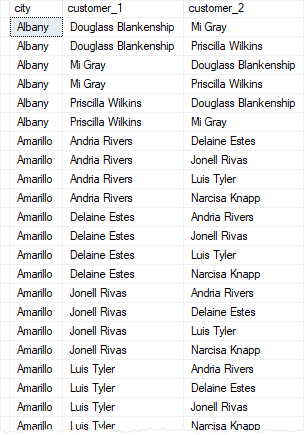
**AND** c1.city = c2.city

**ORDER** **BY**

city,

customer\_1,

customer\_2;



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 who locate in Albany:

**SELECT**

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

city

**FROM**

sales.customers

**WHERE**

city = 'Albany'

**ORDER** **BY**

c;

