Table joins

DE3 Module 2

qa.com

Contents

Introduction

When we are working with a relational database, we frequently need to compose SQL queries that combine data from multiple tables. This is most commonly achieved through **table joins**.

A table join is an operation that selectively combines rows from two input tables into a single output table.

A table join (hereafter, simply "join") is a binary operation, meaning that it takes place between exactly two operands: a Left Table and a Right Table.

1.1 Example join

Let's see the output of a **left join** between two simple tables. The table **OrderCustomer** has been left-joined to table **OrderValue** by matching on the predicate `OrderCustomer.OrderID = OrderValue.OrderNum`.

OrderCustomer

(Left Table)

OrderID	Customer
1000	Alice
2000	Bob
2000	Cathy
	Diego
3000	Emily

OrderValue

(Right Table)

OrderNum	TotalValue
1000	£31.14
1000	£15.92
2000	£6.53
4000	£58.97
	£9.32

Output

OrderID	Customer	OrderNum	TotalValue
1000	Alice	1000	31.14
1000	Alice	1000	15.92
2000	Bob	2000	6.53
2000	Cathy	2000	6.53
	Diego		
3000	Emily		

NULL

Figure 1: Example left join.

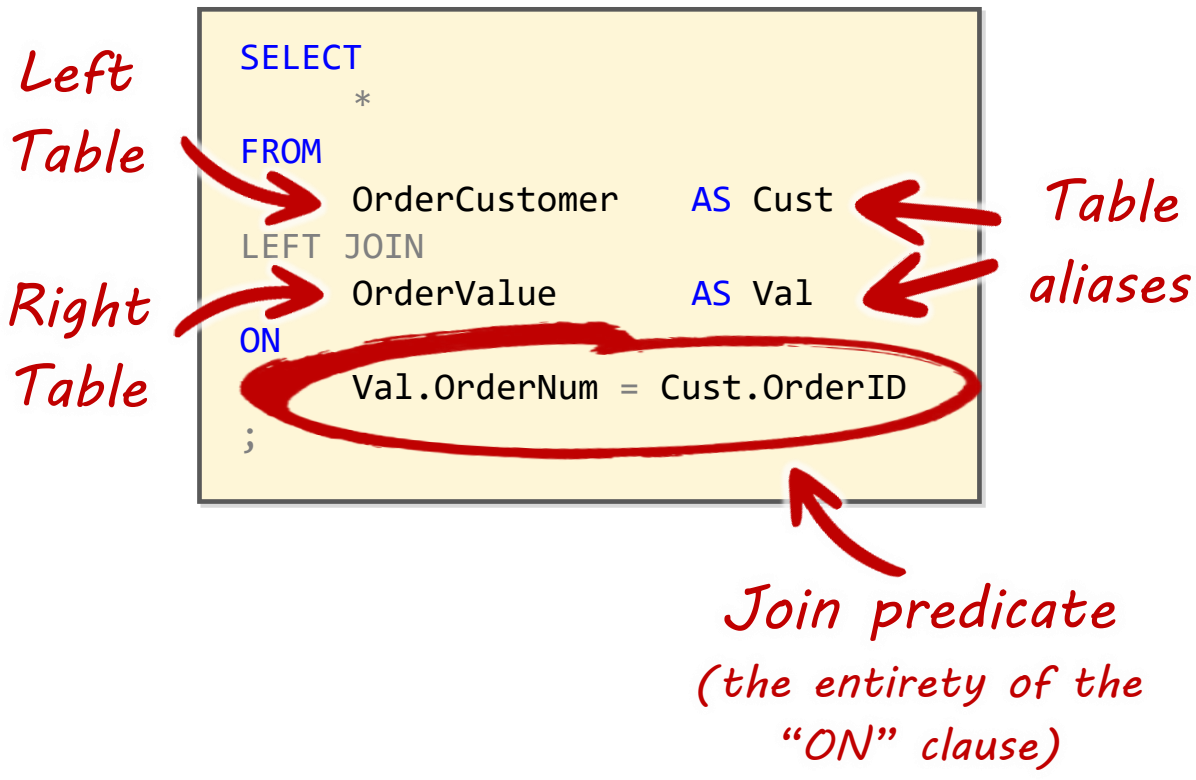
In the **Output** table, we can see that some source rows have been duplicated, while others have been omitted. By the end of this guide, we

will fully understand this output, and we will have all the information we need to predict the output of any join.

1.2 Learning outcomes

- Know the difference between the four main types of join: left join, right join, inner join, and full join.
- Memorise the join algorithm, which enables us to predict the output of any join.
- Understand how non-matching rows are handled for each type of join.
- Understand the behaviour of NULLs in joins.
- Understand that a single row of the Left Table may match with multiple rows from the Right Table, and vice versa.

1.3 Terminology



2 Types of join

The four basic types of join are:

- **Left join**, a.k.a. left outer join.
- **Right join**, a.k.a. right outer join.
- **Inner join**, a.k.a. join.
- **Full join**, a.k.a. full outer join.

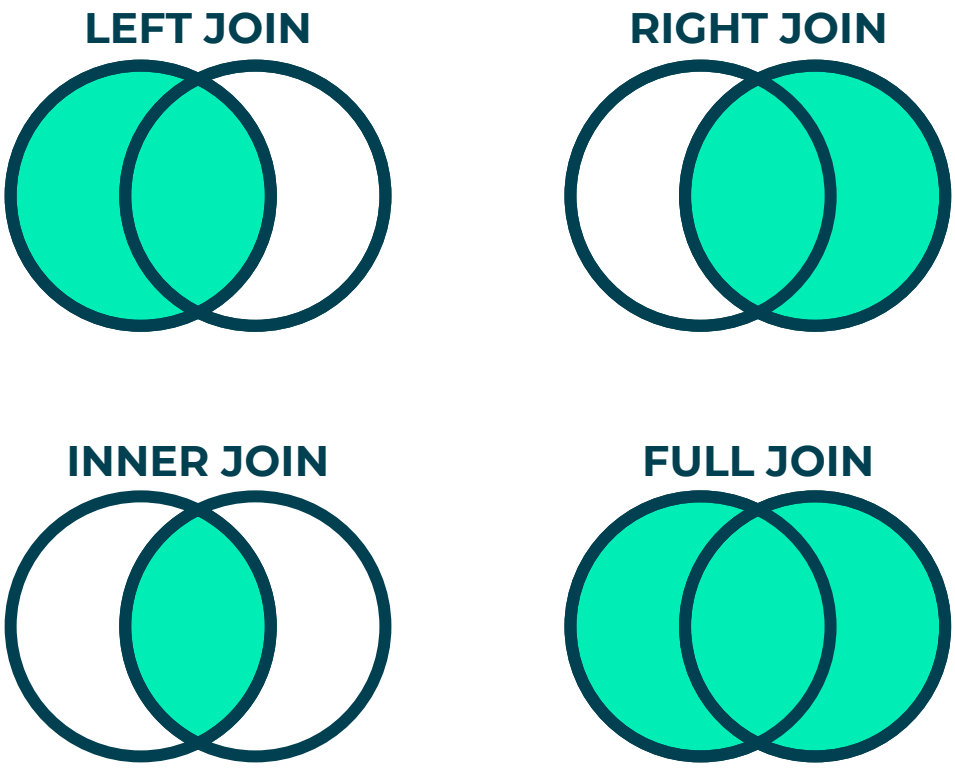


Figure 2: Visual representations of the four basic types of join.

The diagram gives us a hint as to what each type of join does. It is a useful memory aid; however, it doesn't tell us how to predict the exact output of a join.

To predict the output of a join, we need to know the algorithmic definition of that join. That is, we need to know the set of rules that determine how the rows of the two inputs tables get paired with each other and put into the output table.

3 The join algorithm

The steps below allow us to understand what rows will be returned by any join. We will call this the **join algorithm**.

1. For each row of the Left Table, iterate through every row of the Right Table.
2. For each {Left Table row, Right Table row} pair, ask: is the join predicate TRUE or FALSE?
3. If TRUE, add this pair of rows to the output table.
4. Finally, if the type of join is **left**, **right**, or **outer**, also include any **unmatched** rows that must appear in the output table due to this type of join. E.g., for a left join, we append any unmatched rows from the Left Table.

If we carry out steps 1–3 only, we have performed an **inner join**.

For a **left join**, **right join**, or **full join**, we continue to step 4. This step may involve appending some extra rows to the output table from either the Left Table, the Right Table, or both. This step is described in detail in §3.1.

The join algorithm allows us to correctly predict the output of a join. We will go through this algorithm step by step in §4.

3.1 The difference between the types of join

Steps 1–3 of the join algorithm specify an **inner join**. The inner join is the most restrictive type of join—as suggested by **Figure 2**. It also forms the foundation of the other three joins: the outer joins.

To obtain a **left join**, we start with an inner join, and we append to the output any **unmatched** rows from the Left Table. An example of an unmatched row can be seen in **Figure 1**: the OrderID 3000 row of the Left Table does not match with any rows of the Right Table.

Analogously, to obtain a **right join**, we start with an inner join, and we append to the output any unmatched rows from the Right Table.

To obtain a **full join**, we start with an inner join, and we append to the output any unmatched rows from the Left Table **and** any unmatched rows from the Right Table.

When an outer join appends unmatched rows from the {Left/Right} table, the absence of data from the {Right/Left} table is indicated by NULL. This can be seen in the bottom two rows of the **Output** table of **Figure 1**.

Table 1 summarises the difference between inner join and {left, right, full} join.

	Returns [output of inner join] + …
Left join	All unmatched rows from the Left Table.
Right join	All unmatched rows from the Right Table.
Inner join	No extra rows.
Full join	All unmatched rows from both the Left and Right Tables.

Table 1: A summary of the extra rows returned by {left, right, full} join compared to inner join.

4 Joining: step by step

We will now revisit the example join given in the introduction and work through the join algorithm step by step.

As before, table **OrderCustomer** will be left-joined to table **OrderValue** by matching on the predicate `OrderCustomer.OrderID = OrderValue.OrderNum`.

OrderCustomer

(Left Table)

OrderID	Customer
1000	Alice
2000	Bob
2000	Cathy
	Diego
3000	Emily

OrderValue

(Right Table)

OrderNum	TotalValue
1000	£31.14
1000	£15.92
2000	£6.53
4000	£58.97
	£9.32

NULL

This is equivalent to the following SQL query.

```
SELECT
      *
FROM
      OrderCustomer    AS Cust
LEFT JOIN
      OrderValue       AS Val
ON
      Val.OrderNum = Cust.OrderID
;
```

Let's first work through steps 1–3 of the join algorithm.

1. For each row of the Left Table, iterate through every row of the Right Table.
2. For each {Left Table row, Right Table row} pair, ask: is the join predicate TRUE or FALSE?
3. If TRUE, add this pair of rows to the output table.

It is recommended that you work through the above steps yourself. When you are done, verify that you obtain the same results as in **Table 2** below.

Cust row	Val row	Cust value	Val value	Join predicate
1	1	1000	1000	TRUE
1	2	1000	1000	TRUE
1	3	1000	2000	FALSE
1	4	1000	4000	FALSE
1	5	1000	NULL	FALSE
2	1	2000	1000	FALSE
2	2	2000	1000	FALSE
2	3	2000	2000	TRUE
2	4	2000	4000	FALSE
2	5	2000	NULL	FALSE
3	1	2000	1000	FALSE
3	2	2000	1000	FALSE
3	3	2000	2000	TRUE
3	4	2000	4000	FALSE
3	5	2000	NULL	FALSE
4	1	NULL	1000	FALSE
4	2	NULL	1000	FALSE
4	3	NULL	2000	FALSE
4	4	NULL	4000	FALSE
4	5	NULL	NULL	FALSE*
5	1	3000	1000	FALSE
5	2	3000	1000	FALSE
5	3	3000	2000	FALSE
5	4	3000	4000	FALSE
5	5	3000	NULL	FALSE

Table 2: The results of working through steps 1–3 when left-joining **OrderCustomer** to **OrderValue**.

Notice that row 1 of **OrderCustomer** matches with rows 1 and 2 of **OrderValue**, and row 3 of **OrderValue** matches with rows 2 and 3 of **OrderCustomer**. Every pairwise match goes into the output table.

* Also notice that a NULL value does not match with anything, including another NULL. This is an important property of SQL's three-valued logic.

4.1 Inner join

Having enumerated all possible pairs of {Left Table row, Right Table row}, we see that only four pairs match according to the join predicate `OrderCustomer.OrderID = OrderValue.OrderNum`. Putting these pairs of rows together, we obtain the output of **inner join**.

OutputInnerJoin

OrderID	Customer	OrderNum	TotalValue
1000	Alice	1000	31.14
1000	Alice	1000	15.92
2000	Bob	2000	6.53
2000	Cathy	2000	6.53

4.2 Left join

To obtain the output of left join, we start with the output of inner join. Then, we look for any rows of the Left Table that do not match with any rows of the Right Table, and we append these rows to the output.

In our example, we find two such rows.

OrderID	Customer
	Diego
3000	Emily

Appending these rows to the output of inner join, we obtain the output of **left join**. The missing Right Table values are indicated by NULL.

OutputLeftJoin

OrderID	Customer	OrderNum	TotalValue
1000	Alice	1000	31.14
1000	Alice	1000	15.92
2000	Bob	2000	6.53
2000	Cathy	2000	6.53
	Diego		
3000	Emily		

4.3 Right join

To obtain the output of right join, we start with the output of inner join. Then, we look for any rows of the Right Table that do not match with any rows of the Left Table, and we append these rows to the output.

In our example, we find two such rows.

OrderNum	TotalValue
4000	£58.97
	£9.32

Appending these rows to the output of inner join, we obtain the output of **right join**.

OutputRightJoin

OrderID	Customer	OrderNum	TotalValue
1000	Alice	1000	31.14
1000	Alice	1000	15.92
2000	Bob	2000	6.53
2000	Cathy	2000	6.53
		4000	58.97
			9.32

4.4 Full join

Finally, to obtain the output of full join, we again start with the output of inner join. Then, we look for any rows of the Left Table that do not match with any rows of the Right Table, **and** any rows of the Right Table that do not match with any rows of the Left Table, and we append **all** of these rows to the output.

In our example, we have seen that there are two such rows for the Left Table and two for the Right Table.

OrderID	Customer
	Diego
3000	Emily

OrderNum	TotalValue
4000	£58.97
	£9.32

Appending these rows to the output of inner join, we obtain the output of **full join**.

OutputFullJoin

OrderID	Customer	OrderNum	TotalValue
1000	Alice	1000	31.14
1000	Alice	1000	15.92
2000	Bob	2000	6.53
2000	Cathy	2000	6.53
	Diego		
3000	Emily		
		4000	58.97
			9.32

5 Other notes

5.1 Left join vs. right join

A right join is identical to a left join, only with the sense of Left Table and Right table reversed. You can use whichever makes more sense to you in the given context.

In SQL, you can convert between a left join and a right join by swapping the positions of the tables and substituting “LEFT JOIN” for “RIGHT JOIN” or vice versa.

Left join

```
SELECT
      *
FROM
      OrderCustomer    AS Cust
LEFT JOIN
      OrderValue       AS Val
ON
      Val.OrderNum = Cust.OrderID
;
```

Equivalent right join

```
SELECT
      *
FROM
      OrderValue       AS Val
RIGHT JOIN
      OrderCustomer    AS Cust
ON
      Cust.OrderID = Val.OrderNum
;
```

Note: there was no strict need to swap the operands in the join predicate `Val.OrderNum = Cust.OrderID`. This is merely a style preference.

5.2 The order of Left Table and Right Table

For **inner join**, the order of the Left Table and Right Table **does not matter**. Inner join is said to be a *commutative* operation. For a given join predicate, `Table1 INNER JOIN Table2` always returns the same output as `Table2 INNER JOIN Table1`. Consequently, in a chain of any number of inner joins, the order of joins makes no difference.

For each of the three types of outer join—**left join**, **right join**, **full join**—the order of the Left Table and Right Table **usually matters**. Outer joins are *noncommutative*. For example, for a given join predicate, `Table1 LEFT JOIN Table2` is not identical to `Table2 LEFT JOIN Table1`, and will return different output except in special cases. We saw this in §4.

5.3 Joining multiple tables

A join is a binary operation between two tables. If we need to join more than two tables, we can string together more `JOIN` clauses.

```sql
SELECT
      *
FROM
      theatre_group    AS TG
LEFT JOIN
      region           AS R
ON
      TG.region_id = R.region_id
LEFT JOIN
      performer        AS P
ON
      TG.director = P.membership_id
;
```

Joins are processed from **left to right**. Above, we should imagine that **theatre_group** is first joined to **region** to create an intermediate table, and this intermediate table is then joined to **performer**.

5.4 The order of joins

When we are considering a chain of multiple joins, a natural question arises: does the order of joins matter?

The unhelpful answer is "sometimes yes, sometimes no". I do not know all the situations in which the order of joins makes a difference, and so, unfortunately, I cannot offer a definitive checklist.

There is a good Stack Overflow post that covers the case of chained left joins: Does the join order matter in SQL? - Stack Overflow (see top reply). The post explains that `Table1 LEFT JOIN Table2 LEFT JOIN Table3` is sometimes equivalent to `Table1 LEFT JOIN Table3 LEFT JOIN Table2`, and sometimes it is not, and it demonstrates this fact with examples.

Fortunately, the conditions under which left joins can be reordered non-destructively are actually fairly common in real-world scenarios. As a rough rule of thumb, if both of the following conditions are met then a chain of **left joins** can *probably* be safely reordered with no effect the output:

1. Each join predicate references only its immediate Left Table and Right Table.
2. No join predicate tests for NULL, nor coerces NULL to a value.

There are undoubtedly exceptions to this rule. However, since the above conditions are satisfied very often in practice, the rule is still useful.

5.5 Anti joins

The three anti outer joins—**anti left join**, **anti right join**, and **anti full join**—are very similar to their corresponding ordinary joins, except that the intersection (matching rows) between the two tables is omitted.

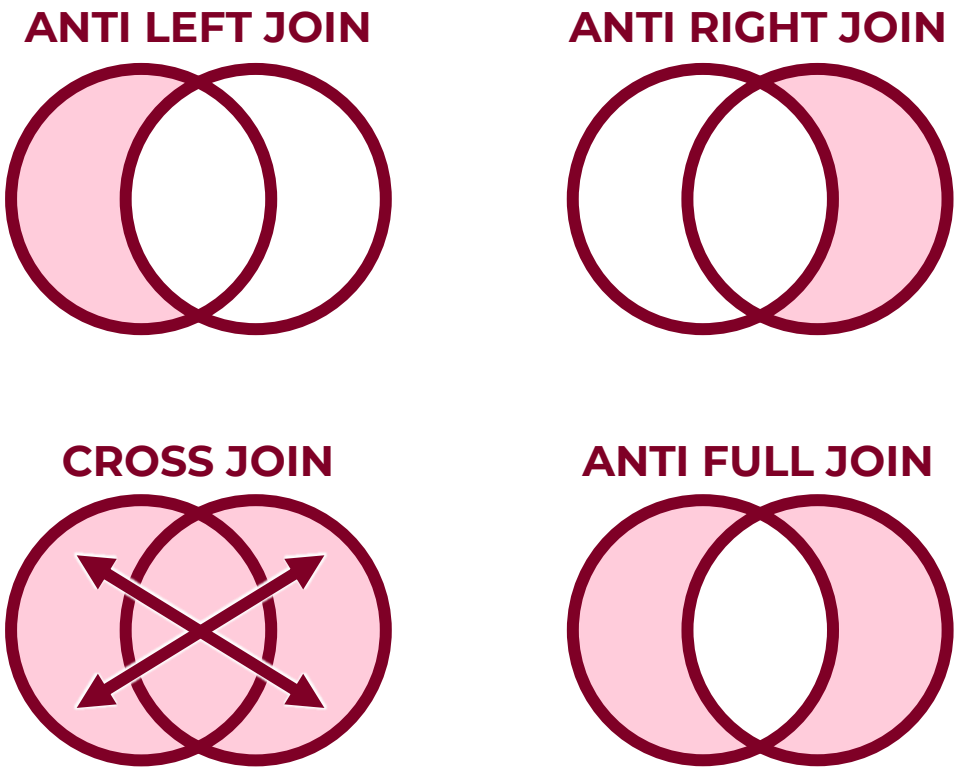


Figure 3: Visual representations of the four types of anti join.

To produce an anti outer join in SQL, we start with the corresponding ordinary join and then add the omission condition in the WHERE clause.

An example of this is shown in **Figure 4**.

Left join

```
SELECT
      *
FROM
      OrderCustomer
      AS Cust
LEFT JOIN
      OrderValue
      AS Val
ON
      Val.OrderNum
      = Cust.OrderID
;
```

Anti left join

```
SELECT
      *
FROM
      OrderCustomer
      AS Cust
LEFT JOIN
      OrderValue
      AS Val
ON
      Val.OrderNum
      = Cust.OrderID
WHERE
      Val.OrderNum
      IS NULL
;
```

Figure 4: An example of how to convert a left join into an anti left join.

The `WHERE` clause filters the output to just the **unmatched rows** from the Left Table.

Remember: in the output of a left join, the unmatched rows from the Left Table correspond to NULL in the Right Table. Therefore, if we filter the join column(s) of the Right Table to NULL, we are effectively filtering to the unmatched rows of the Left Table.

Alternatively, can think of an anti join as an addition to the join algorithm:

5. **Anti join rule:** delete all rows from steps 1–3.

Example SQL for the **anti right join** and **anti full join** is shown in "SQL join diagrams v#.#.#.pdf".

5.6 Cross join

Cross join is the simplest join of all. It can be described in a one-step algorithm:

1. Pair every row of the Left Table with every row of the Right Table.

Letter

Letter
A
B

Number

Number
1
2
3

OutputCrossJoin

Letter	Number
A	1
A	2
A	3
B	1
B	2
B	3

A cross join requires no join predicate.

{Left, right, inner, full} join will each produce an output identical to that of cross join if we join on a predicate that is always TRUE, such as `1 = 1`.

qa.com