**What FULL JOIN Is and When to Use It**

*Learn about FULL JOIN, how to implement it, how it compares with the other types of SQL JOINs, and some of its unique use cases.*

Before we jump into FULL JOINs, let’s quickly recap what an SQL JOIN is. At its core, a JOIN combines data from two or more tables within a database. Tables are usually linked together using unique identifiers in each table, i.e. primary and foreign keys.

To demonstrate a simple SQL JOIN – also known as an INNER JOIN – in action, let’s consider the Products and OrderDetailstables from the well-known Northwind sample database. The Productstable contains a list of all products and the OrderDetailstable contains a list of all recent orders.

Calendar

Description automatically generated

Table

Description automatically generated

Here’s the query:

| SELECT  P.ProductName, O.OrderID, O.Quantity  FROM Products as P  JOIN OrderDetails as O  ON P.ProductID=O.ProductID  ORDER BY Quantity Desc; |
| --- |

And the result:

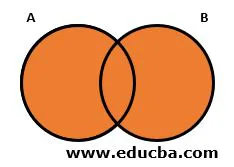
| **ProductName** | **OrderID** | **Quantity** |
| --- | --- | --- |
| Pâté chinois | 10398 | 120 |
| Steeleye Stout | 10286 | 100 |
| Sirop d'érable | 10440 | 90 |

The above example uses ProductID (which is available in both tables) as the key to join the two tables and display each order sorted in descending order by Quantity. Note that a simple JOIN like the one above only returns matching rows from the two tables. In this case, the results would only include recently-ordered products and orders that are linked to a valid ProductID.

**FULL JOIN: An Introduction**

Unlike INNER JOIN, a FULL JOIN returns all the rows from both joined tables, **whether they have a matching row or not**. Hence, a FULL JOIN is also referred to as a FULL OUTER JOIN. A FULL JOIN returns unmatched rows from both tables as well as the overlap between them. When no matching rows exist for a row in the left table, the columns of the right table will have NULLs for those records. Similarly, when no matching rows exist for a row in the right table, the columns of the left table will have NULLs.

Below represents the Venn diagram of the FULL join.



**Syntax:**

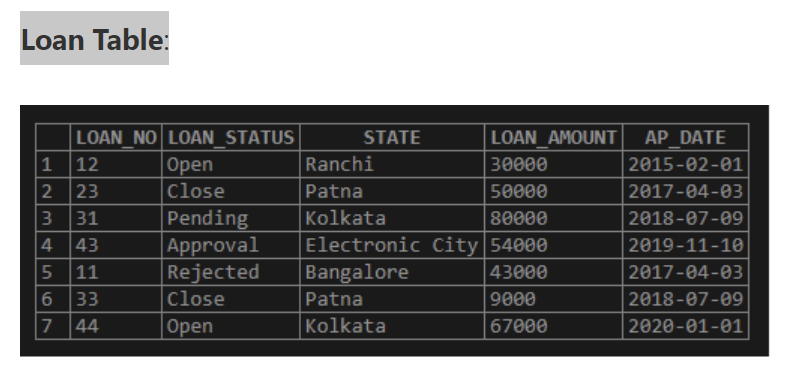
SELECT \* FROM TABLE\_A A  
FULL OUTER JOIN TABLE\_B B  
ON A. Common\_COLUMN =B. Common\_COLUMN  
WHERE <Condition>

The result set contains NULL set values. Below syntax can be used to neglect the NULL values: –

SELECT \* FROM TABLE\_A A  
FULL OUTER JOIN TABLE B B  
ON A. Common\_COLUMN =B. Common\_COLUMN  
WHERE A.Common\_COLUMN IS NOT NULL  
AND A.Common\_COLUMN IS NULL

#### Example #1

Let us consider two tables and apply FULL outer join on the tables:



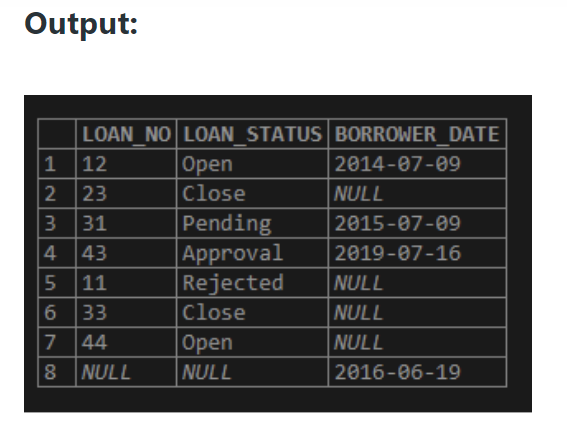
A screenshot of a computer

Description automatically generated with medium confidence

Query to get the loan\_no, status, and borrower date from two tables. **Query:**

**SELECT L.LOAN\_NO, L.LOAN\_STATUS, B.BORROWER\_DATE  
FROM LOAN L FULL OUTER JOIN BORROWER B  
ON L.LOAN\_NO=B.LOAN\_NO**

Let’s check the output of the above table after applying the right join on them.

****

The result set contains NULL set values.

Below syntax can be used to neglect the NULL values: –

SELECT \* FROM TABLE\_A A  
FULL OUTER JOIN TABLE B B  
ON A. Common\_COLUMN =B. Common\_COLUMN  
WHERE A.Common\_COLUMN IS NULL

Let us consider two tables and apply FULL Outer join on the tables:

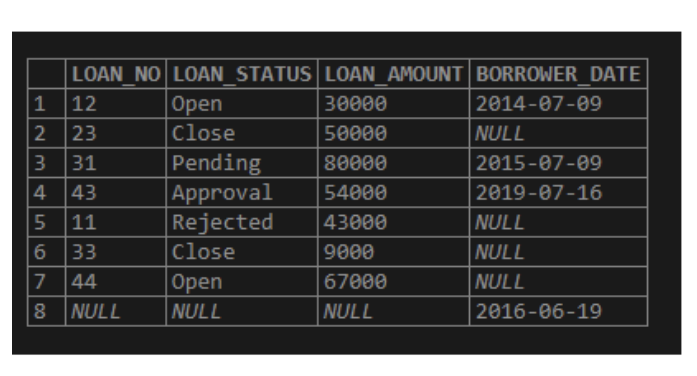
Query to get the loan\_no, status, loan\_aount and borrower date from two tables.

**Query:**

SELECT L.LOAN\_NO, L.LOAN\_STATUS,L.LOAN\_AMOUNT, B.BORROWER\_DATE  
FROM LOAN L FULL OUTER JOIN BORROWER B  
ON L.LOAN\_NO=B.LOAN\_NO

Let’s check the output of the above table after applying the Full outer join on them.

**Output:**



In the above table, LOAN is the right table and the Borrower is the left table. As in FULL OUTER join, we get all rows from both tables. Here we get all the rows from the LOAN table and the Borrower table. Values not present will be NULL.

#### Example #2

Let us consider two tables and apply FULL OUTER join on the tables:

Query to get the loan\_no, status, loan\_aount and borrower date from two tables.

**Query:**

SELECT L.LOAN\_NO, L.LOAN\_STATUS,L.LOAN\_AMOUNT, B.BANK\_ID  
FROM BORROWER B FULL OUTER JOIN LOAN L  
ON L.LOAN\_NO=B.LOAN\_NO

Let’s check the output of the above table after applying the FULL OUTER join on them.

A screenshot of a computer

Description automatically generated with medium confidence

In the above table, LOAN is the right table, and the Borrower is the left table. As in FULL OUTER join, we get all rows from both tables. Here we get all the rows from the LOAN table and the Borrower table. Values not present will be NULL.

In the above table, LOAN is the right table and the Borrower is the left table. As in FULL OUTER join, we get all rows from both tables. Here we get all the rows from the LOAN table and the Borrower table. Values not present will be NULL.

**Conclusion**

To fetch data relevant to the customer requirement we might need to join tables which will be fulfilled by joins. As mentioned earlier joins are used to get data from more than one table. To join more than one table we need at least one column common in both tables. Tables get joined based on the condition specified.

To demonstrate the difference between a simple SQL JOIN and a FULL OUTER JOIN, let’s consider the Projects and Employeestables as shown below. The Projectstable contains a list of all projects undertaken by the company (in-house projects as well as outsourced), while the Employeestable contains a list of all current employees and specifies if they are involved in any company projects.

Table

Description automatically generated

Table

Description automatically generated with medium confidence

If we were to do JOINs between these two tables by matching the tables through ProjectID as the primary key, the results of a simple INNER JOIN and a FULL OUTER JOIN would be quite different. An INNER JOIN would output a table containing only the results where there is a match between the two corresponding entries in both tables. The INNER JOIN result contains employee names and the corresponding project names:

| **EmployeeName** | **ProjectName** |
| --- | --- |
| Albert Ross | HQ Office Renovations |
| Matthias Dias | Automated QA |
| Al Cooper | CRM Upgrade |

On the other hand, a FULL OUTER JOIN will output both tables’ data, irrespective whether it has a match in the other table:

| **EmployeeName** | **ProjectName** |
| --- | --- |
| Albert Ross | HQ Office Renovations |
| Hummer Baird | *NULL* |
| Matthias Dias | Automated QA |
| Al Cooper | CRM Upgrade |
| .... | ... |
| Hamza Imran | *NULL* |
| NULL | ERP Integration |

The result of FULL JOIN contains all employee names, regardless of whether they are assigned to a project, and all project names, even if there are no employees assigned to that project.

**FULL JOIN Syntax**

The basic syntax of a FULL JOIN is similar to other types of JOINs:

| SELECT      left\_table.column1, right\_table.column2,...  FROM left\_table  FULL OUTER JOIN right\_table  ON left\_table.key = right\_table.key; |
| --- |

The OUTER keyword is optional and may be omitted.

**Example**

Now, let’s run through that example of a FULL OUTER JOIN that shows both the EmployeeName and ProjectName again. Here's the query:

| SELECT  Employees.EmployeeName, Projects.ProjectName  FROM Employees  FULL OUTER JOIN Projects  ON Employees.ProjectID=Projects.ProjectID  ORDER BY EmployeeID; |
| --- |

And the result:

| **EmployeeName** | **ProjectName** |
| --- | --- |
| Albert Ross | HQ Office Renovations |
| Hummer Baird | *NULL* |
| Matthias Dias | Automated QA |
| Al Cooper | CRM Upgrade |
| .... | ... |
| Hamza Imran | *NULL* |
| *NULL* | ERP Integration |

From the result table, we see that a FULL JOIN also outputs NULL results from both the  Projects and Employees tables – even though there were no matches on the other table. NULLs in the ProjectName column indicate that employee is not specifically assigned to any project and likely serves other functions in the company. On the other hand, a NULL in the EmployeeName column suggests that the specific project is likely outsourced and not directly managed by any company employee. Alternatively, a NULL value could also point to a potential error or inadvertent data deletion in the database or the system. (We will discuss this more below.)

It’s good to note that some databases, like MySQL, do not support full joins. In that case, you can use the UNION ALL operator to combine the LEFT JOIN and RIGHT JOIN.

Our prior article on [SQL JOINs](https://learnsql.com/blog/sql-joins-types-explained) has a few more examples that you can examine. Our [SQL JOINs](https://learnsql.com/course/joins) course also provides comprehensive examples of FULL JOINs and includes practical exercises to polish your knowledge.

**FULL JOIN Use Cases**

Compared to the other types of SQL JOINs, you will likely use FULL JOIN less frequently. That being said, it is a very handy tool for a few unique situations. Some of its use cases are:

* To retrieve all records from both tables, whether there is a match or not. This is the example we saw above.
* To find mismatched or orphaned data from both tables. An orphaned record is a record whose foreign key value references a non-existent primary key value; this often happens in rapidly scaling or very old systems.
* To run exception reports. This is a form of data analysis comparing one set of data against a desired/expected base data set and highlighting items that don't match.
* Although the application of FULL JOINs is quite unique, they are a great way to find and diagnose potential data integrity issues.

## Types of JOINs: Recap

As we had mentioned above, there are a few more types of JOINs.

An INNER JOIN returns rows when the JOIN condition is satisfied in both the left and right tables. In other words, it returns only the matched records from the tables. This is the most common type of SQL JOIN and is the default when you haven’t specified the type of JOIN.

An OUTER JOIN returns all the rows from one table and some or all of the rows from another table (depending on the type of OUTER JOIN). Aside from the FULL OUTER JOIN, there are two other types:

* A [LEFT OUTER JOIN](https://learnsql.com/blog/what-is-left-join-sql) returns all rows from the **left** table, even if no matching rows were found in the right table. If there are no matching records in the right table, the query will return NULL values for those columns.
* A RIGHT OUTER JOIN returns all rows from the **right** table. If there are no matching records in the left table, NULL values are returned for those columns – the reverse of a LEFT JOIN.

A CROSS JOIN (also called a Cartesian JOIN) returns every possible combination of rows from the tables that have been joined. Since it returns all possible combinations, this is the only JOIN type that does not need a JOIN condition and therefore does not require an ON clause.