2021-04-21T17:00:00+02:00 | Andrew Bone

What Do the SQL INTERSECT and MINUS Clauses Do?

Do you know the difference between SQL's INTERSECT and MINUS clauses and how to use them? You will find examples and explanations in this article.

SQL `INTERSECT` and `MINUS` are useful clauses for quickly finding the difference between two tables and finding the rows they share.

`INTERSECT` compares the data between tables and returns only the rows of data that exist in **both tables**.

`MINUS` compares the data between tables and returns the rows of data that **exist only in the first table** you specify.

Both SQL `INTERSECT` and `MINUS` (or `EXCEPT`, depending on your SQL dialect) form part of LearnSQL.com's SQL Basics course.

A great way to start learning SQL is with an online SQL course. Find one that gets you using the language right away. LearnSQL.com offers a fantastic SQL Basics course that makes learning SQL quick and easy. Its 129 interactive exercises give you hands-on experience working with databases and SQL statements.

SQL INTERSECT

The SQL `INTERSECT` operator is used to return the results of two or more `SELECT` statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the `INTERSECT` results.

The number and order of the columns must be the same in all of the `SELECT` queries.

The column data types must be the same, or at least compatible with one another. `INTERSECT` filters duplicates and returns only distinct rows that are common between all of the queries.

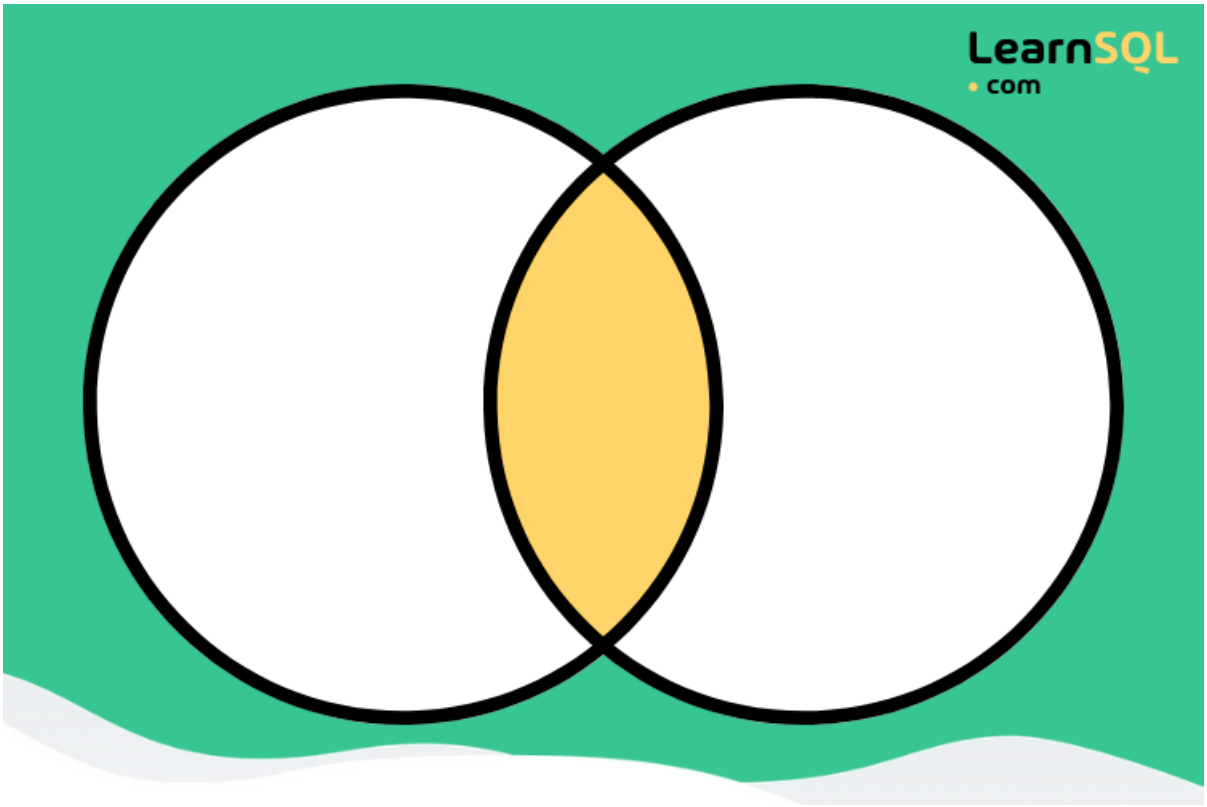
Here is the syntax for the `INTERSECT` operator:

```
SELECT column_1 [, column_2, …, column_n]
FROM table_1 [, table_2, …, table_n]
[WHERE condition]

INTERSECT

SELECT column_1 [, column_2, …, column_n]
FROM table_1 [, table_2, …, table_n]
[WHERE condition]
```

Anything inside the square brackets is entirely optional. The concept of an `INTERSECT` is further explained by the following diagram:



The `INTERSECT` query will return the records in the shaded area. These are the records that exist in both data sets

`INTERSECT` is just one way of merging the results of different SQL queries. If you're interested in learning more, this article covers the different methods for combining the results of SQL queries.

SQL MINUS

The SQL `MINUS` clause is used to combine two `SELECT` statements, but it returns rows from the first `SELECT` statement that are not returned by the second `SELECT` statement. SQL `MINUS` only returns rows that are not available in the second `SELECT` statement.

Each `SELECT` statement within a `MINUS` query must contain the same number of fields in the result sets along with similar data types.

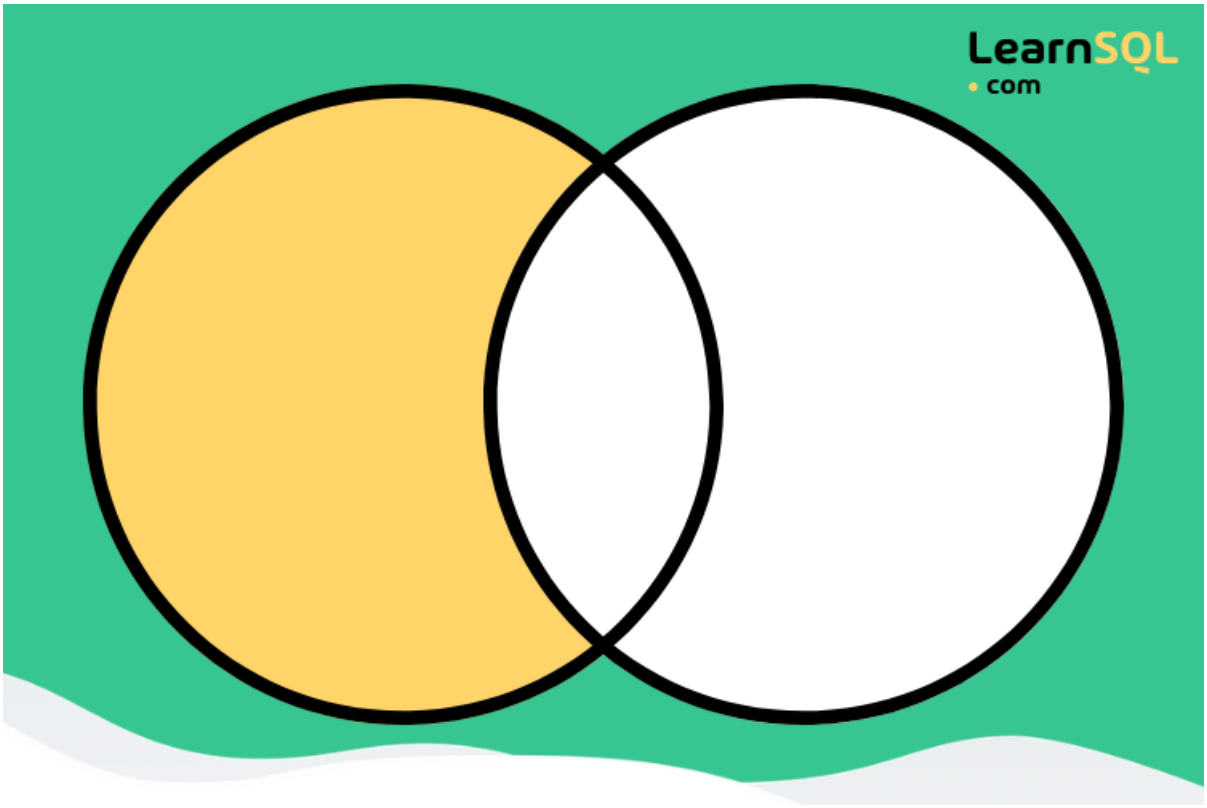
The `MINUS` operator is not supported in all SQL databases. It can be used in databases like MySQL and Oracle. For databases like SQL Server, PostgreSQL, and SQLite, use the `EXCEPT` operator to perform this type of query.

```
SELECT column_1 [, column_2, …, column_n]
FROM table_1 [, table_2, …, table_n]
[WHERE condition]

MINUS

SELECT column_1 [, column_2, …, column_n]
FROM table_1 [, table_2, …, table_n]
[WHERE condition]
```

The SQL code shown in the square brackets is entirely optional. The concept of SQL `MINUS` is further explained by this diagram:



The `MINUS` query will return the records in the red area. These are the records that exist in the first dataset and not the second.

MINUS vs. INTERSECT: Examples

Let's apply the `INTERSECT` and `MINUS` clauses to a practical example. Imagine we have the following tables.

`customers` - Contains details about our customers

id	customer_name	country
1	Infotech Solutions	Germany
2	Corpway Industries	Ireland
3	Fenway Inc	England
4	Fairview Ltd	France

`suppliers` - Contains details about our suppliers.

id	customer_name	country
1	Carbon Way Suppliers	Spain
2	Alloy Inc	France
3	Materials Delivered Ltd	Ireland
4	Concrete Crew	Poland
5	Conglorito Systems	Italy

Now let's write an `INTERSECT` query. We want to find the countries that our suppliers and customers have in common.

```
SELECT country
FROM customers
INTERSECT
SELECT country
FROM suppliers
```

We specify the country column in each `SELECT` clause. Executing this query results in the following data set:

country
France
Ireland

Looking back over the **`customers`** and **`suppliers`** tables, we can see this result is correct. Only the countries of France and Ireland are shared between the tables.

Now let's apply the `MINUS` operator to the same tables. This will allow us to get the countries in our **`customers`** table that are not in our **`suppliers`** table:

```
SELECT country
FROM customers
MINUS
SELECT country
FROM suppliers
```

Executing this query yields the result:

country
England
Germany

There we have it: the countries that are unique to our **`customers`** table. The order of your `SELECT` clauses is very important here, and it is something you must be mindful of when using the `MINUS` operator. Let's reverse the order of our `SELECT` clauses and see what happens.

```
SELECT country
FROM suppliers
EXCEPT
```

```
SELECT country
FROM customers
```

Executing this query returns the following data:

country
Italy
Poland
Spain

As you can see, our result set was wildly different. SQL starts with our `suppliers` table and then removes any countries that exist in the `customers` table.

If you feel overwhelmed, consider trying the SQL Fundamentals track from LearnSQL.com, which will provide you with a solid SQL foundation. It will teach you basic SQL statements like `WHERE`, `GROUP BY`, `ORDER BY`, and `HAVING`. You'll also learn how to `JOIN` tables and add, modify, or remove data from a database.

This was a simple example showing you how the `INTERSECT` and `MINUS` operators can be used to quickly retrieve distinct datasets. Let us look at some more examples that show you how these operators will act in three different scenarios:

> One table is the subset of the other table's data.
>
> Both tables have the same data.
>
> One table in your query contains no data.

Learn the SQL basics by doing interactive courses from our SQL Fundamentals track!

More SQL INTERSECT and MINUS Examples

One Table is a Subset of the Other Table's Data

For this scenario, imagine we have two tables called `employees` and `planning_committee`. As you can see the `planning_committee` table is a

subset of **`employees`**, meaning all of its data is also contained in **`employees`**.

`employees` - All of the employees employed at our company.

employee_id	first_name	last_name
321873	John	Smith
415938	Jane	Ramsey
783273	Andrew	Johnson
832923	Christina	Grey

`planning_committee` - All of the employees on our company's planning committee.

employee_id	first_name	last_name
415938	Jane	Ramsey
783273	Andrew	Johnson

Let's see how the `INTERSECT` clause behaves in this scenario.

```
SELECT employee_id, first_name, last_name
FROM employees
INTERSECT
SELECT employee_id, first_name, last_name
FROM planning_committee
```

The following dataset is returned:

employee_id	first_name	last_name
415938	Jane	Ramsey
783273	Andrew	Johnson

As you can see, only the subset is returned. This is because the **`planning_committee`** table is a subset of the **`employees`** table; thus, the result will be simply the **`planning_committee`** table.

What happens if we use the MINUS clause instead? Imagine we wanted to find all of the employees that were not on the planning committee. This can be achieved by writing the query below:

```
SELECT employee_id, first_name, last_name
FROM employees
MINUS
SELECT employee_id, first_name, last_name
```

```
FROM planning_committee
```

Executing this query yields the following result:

employee_id	first_name	last_name
321873	John	Smith
832923	Christina	Grey

You can see that these employees are not in the **`planning_committee`** table; this is the desired result! Again, the ordering of the tables here is important. If we were to reverse the order of the `SELECT` clauses like so ...

```
SELECT employee_id, first_name, last_name
FROM planning_committee
EXCEPT
SELECT employee_id, first_name, last_name
FROM employees
```

... executing this query would yield a very different result:

employee_id	first_name	last_name

Since all of the data in the **`planning_committee`** table is contained in the **`employees`** table, nothing gets returned. SQL `MINUS` only returns distinct data.

Time to look at our next scenario.

Both Tables Have the Same Data

There may be a situation where two SQL tables have identical data. How do the `INTERSECT` and `MINUS` clauses handle this situation, and what results should you expect? Let's find out!

For this scenario, we will use the following tables:

`payroll` – All employees currently on the payroll at our company.

employee_id	first_name	last_name
321873	John	Smith
415938	Jane	Ramsey
783273	Andrew	Johnson

employee_id	first_name	last_name
832923	Christina	Grey

`employees` - All employees at our company.

employee_id	first_name	last_name
321873	John	Smith
415938	Jane	Ramsey
783273	Andrew	Johnson
832923	Christina	Grey

You can see that all of the employees at our company are currently on the payroll and are getting paid as they should. This results in these tables containing identical data.

Let's look at how the `INTERSECT` clause handles this case:

```
SELECT employee_id, first_name, last_name
FROM employees
INTERSECT
SELECT employee_id, first_name, last_name
FROM payroll
```

Executing this query returns this result:

employee_id	first_name	last_name
321873	John	Smith
415938	Jane	Ramsey
783273	Andrew	Johnson
832923	Christina	Grey

Since all of the data was shared between the two tables, everything gets returned!

Time to see how the `MINUS` clause handles tables that share identical data:

```
SELECT employee_id, first_name, last_name
FROM employees
MINUS
SELECT employee_id, first_name, last_name
FROM payroll
```

Executing this query shows the following result:

employee_id	first_name	last_name

No data is returned! SQL starts by selecting the data in our `employees` table and then subtracts the data that exists in the `payroll` table. In this case, everything gets removed.

This leads to our final scenario. What if one of the tables that makes up part of an `INTERSECT` or `MINUS` clause contains no data?

One Table Contains No Data

For this scenario, we will use the following tables:

`employees` - All of the employees employed at our company.

employee_id	first_name	last_name
321873	John	Smith
415938	Jane	Ramsey
783273	Andrew	Johnson
832923	Christina	Grey

`on_vacation` - All our company employees currently on vacation.

employee_id	first_name	last_name

Let's find out how the `INTERSECT` clause handles an empty table:

```
SELECT employee_id, first_name, last_name
FROM employees
INTERSECT
SELECT employee_id, first_name, last_name
FROM on_vacation
```

We're given the following result after executing this query:

employee_id	first_name	last_name

No results! When we use an empty table as part of the `INTERSECT` clause, we will get an empty dataset. This is because no matches could be found

between the two tables.

How the `MINUS` clause is affected by the inclusion of an empty table is entirely dependent on the order you specify. For example, this query ...

```
SELECT employee_id, first_name, last_name
FROM employees
MINUS
SELECT employee_id, first_name, last_name
FROM on_vacation
```

... yields the following result:

employee_id	first_name	last_name
321873	John	Smith
415938	Jane	Ramsey
783273	Andrew	Johnson
832923	Christina	Grey

The `MINUS` clause here has very little effect, as you are essentially taking away nothing (an empty data set) from the **`employees`** table.

However, if we were to reverse the order of the `SELECT` clauses, like so ...

```
SELECT employee_id, first_name, last_name
FROM on_vacation
MINUS
SELECT employee_id, first_name, last_name
FROM employees
```

... we're faced with a very different result:

employee_id	first_name	last_name
		Smith

Another empty data set! This occurs because the first `SELECT` clause retrieves the data from the **`on_vacation`** table, which in this case is nothing. We then instruct SQL to take away the **`employees`** data from our empty data set. This has no effect, as the data set is already empty!

The best way to learn SQL is through practice. Try out our interactive SQL Basics course.