**Joining tables**

This reading introduces the different types of JOINS supported in MySQL.

**What is a JOIN?**

A join in a database links records of data between one or multiple tables based on a common column between them.

**Why do you need to use a JOIN?**

Sometimes you want to find information about a specific activity or object in the database, where the relevant information exists in more than one table. In this situation, you can use the SQL JOIN clause to query the required data from multiple tables.

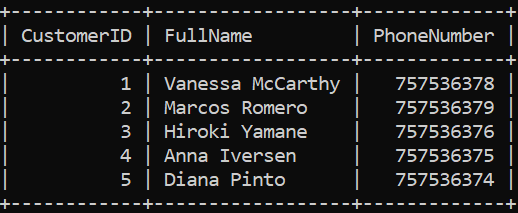
There are four different types of joins supported in MySQL that are covered in this lesson.

* INNER JOIN
* LEFT JOIN
* RIGHT JOIN
* SELF-JOIN

To explain the difference between these types of JOINS, let's look at the Little Lemon restaurant database, which includes two tables.

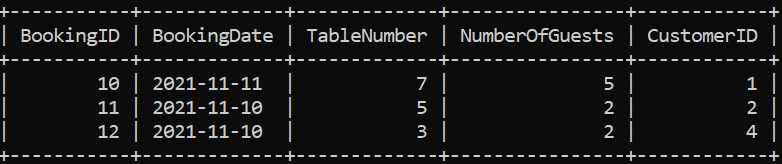
The first is the Customers table with the following columns:

* CustomerID,
* FullName
* and PhoneNumber columns, as shown below:



The second is the bookings table with the columns:

* BookingID,
* BookingDate,
* TableNumber,
* NumberOfGuests
* and CustomerID columns.



You may have noticed that both tables contain the Customer ID column, which represents a common column.

**INNER JOIN**

This type of JOIN returns records of data that have matching values in the joined tables. For example, assume that you want to return the full name and booking ID of customers who made bookings. In this situation, you can use the INNER JOIN clause to extract records of data from the Customers and the Bookings tables based on the matching customer ID value as follows.

1

2

3

 SELECT Customers.FullName, Bookings.BookingID

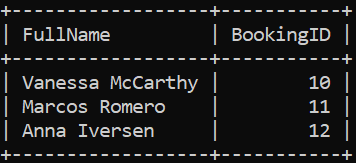
 FROM Customers INNER JOIN Bookings

 ON Customers.CustomerID = Bookings.CustomerID;

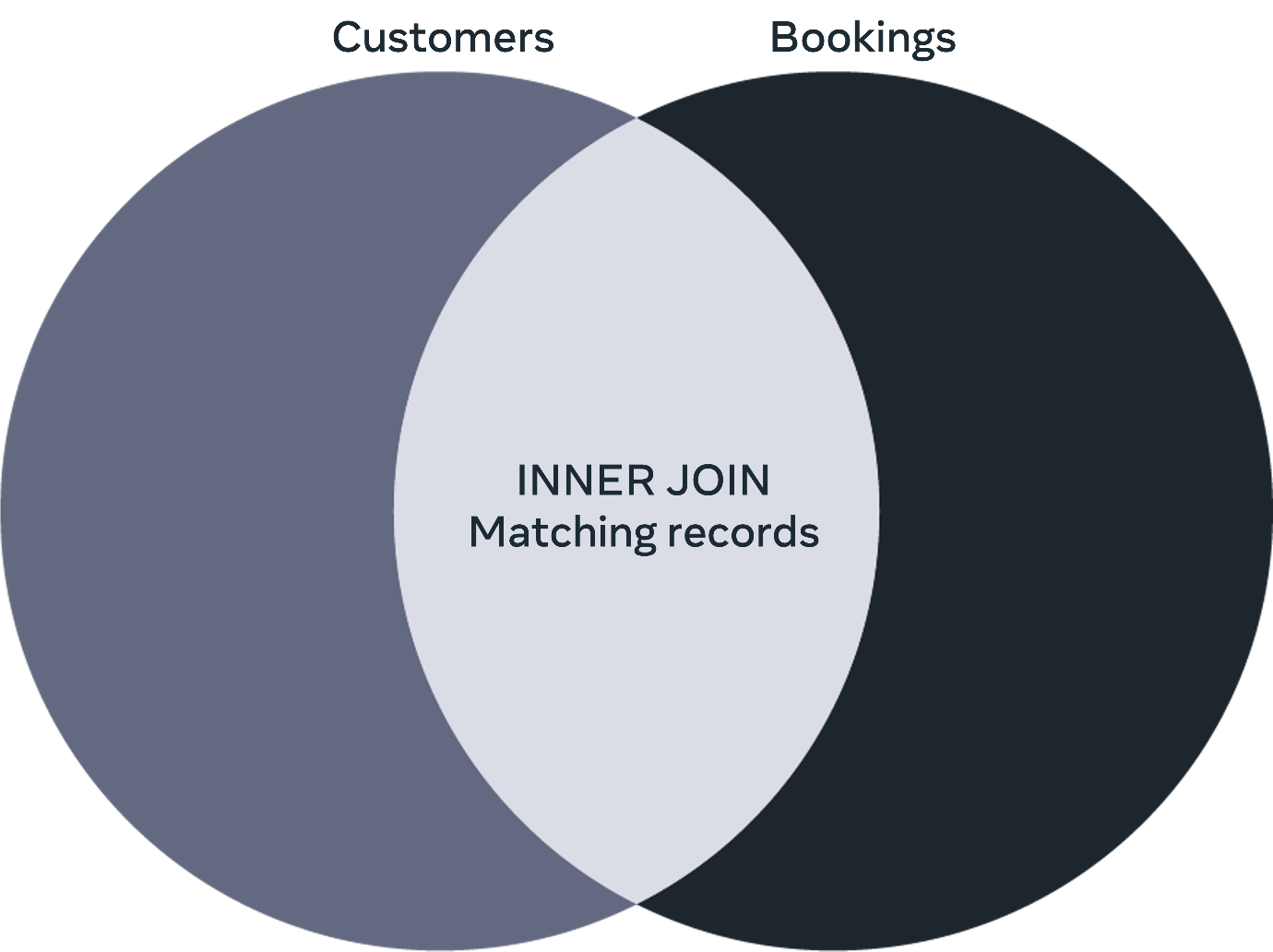




The outuput result shown below



The INNER JOIN is illustrated in the following Venn diagram.



**LEFT JOIN**

You can use the LEFT JOIN clause to extract the full names and the booking IDs from the Customers and the Bookings tables as follows:

1

2

3

SELECT Customers.FullName, Bookings.BookingID

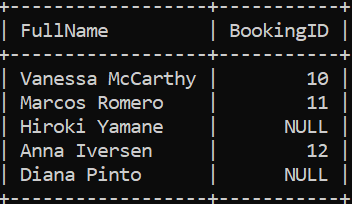
FROM Customers LEFT JOIN Bookings

ON Customers.CustomerID =  Bookings.CustomerID;



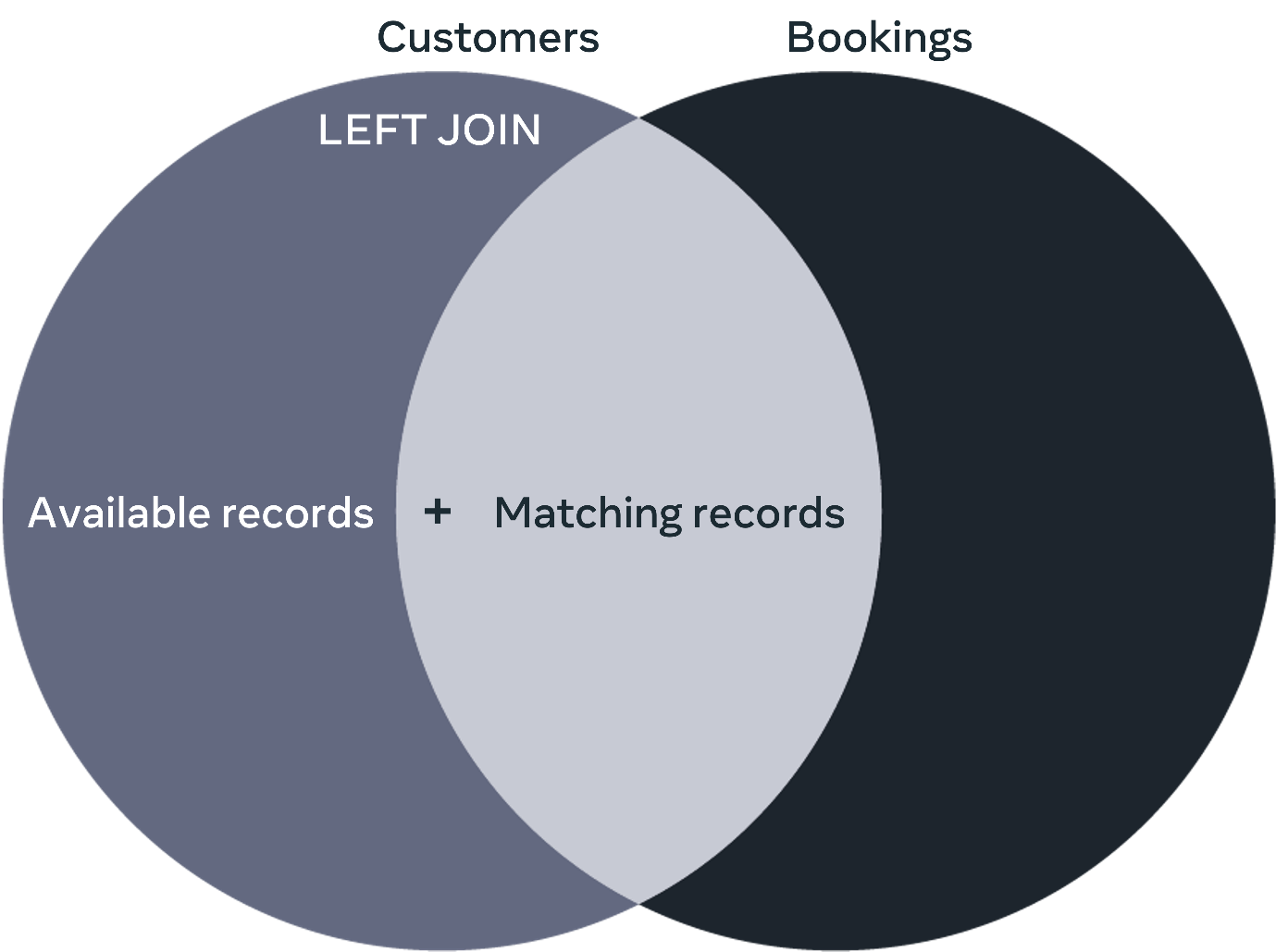


The results of this query are as follows:



The LEFT JOIN returns all common records in a similar way to the INNER JOIN, plus all queried records from the left table regardless of whether there is a match in the right table or not. If there are no matching records in the right table, then null values will be inserted for the bookings IDs.

The LEFT JOIN is illustrated in the following Venn diagram.



**RIGHT JOIN**

You can use the RIGHT JOIN clause to extract the full names and the booking IDs from the Customers and the Bookings tables as follows:

1

2

3

SELECT Customers.FullName, Bookings.BookingID

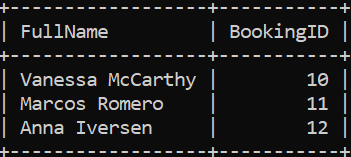
FROM Customers RIGHT JOIN Bookings

ON Customers.CustomerID = Bookings.CustomerID;



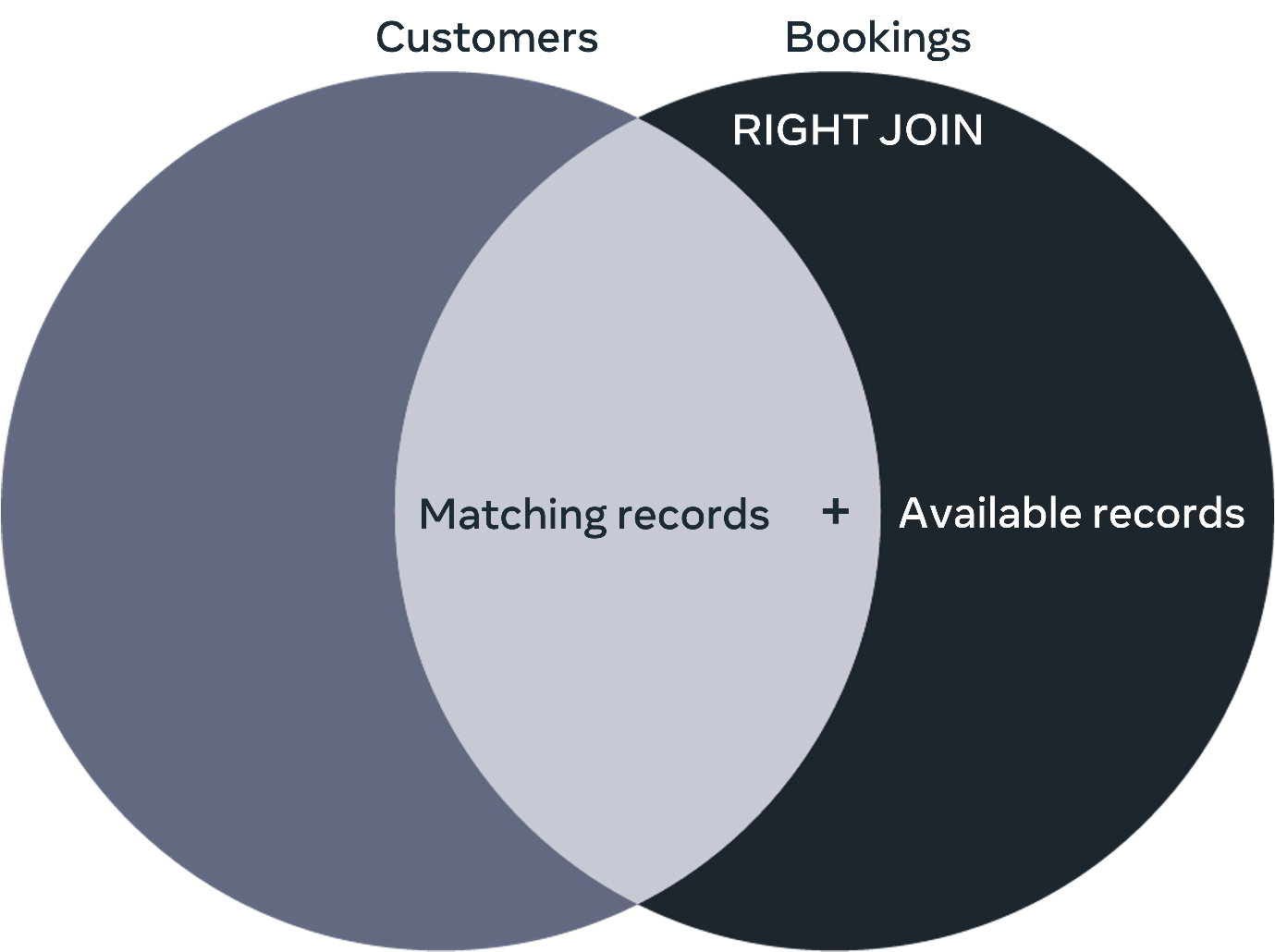


The output of this query is as follows:



The RIGHT JOIN returns all common records in a similar way to the INNER JOIN, plus all queried records from the right table regardless of whether there is a match in the left table or not. If there are no matching records in the left table, then null values will be inserted for the customers full names.

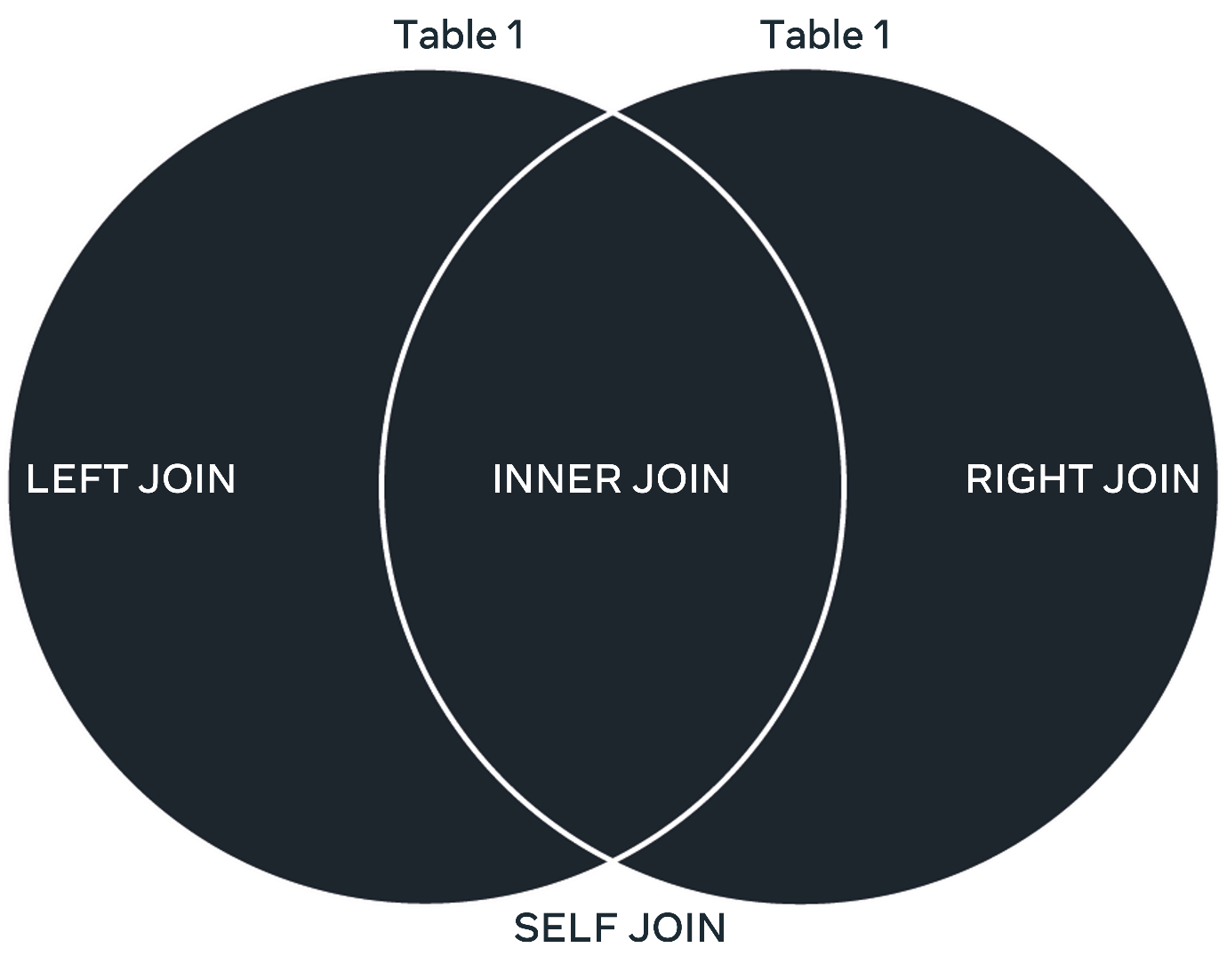
The RIGHT JOIN is illustrated in the following Venn diagram.



**SELF JOIN**

This is a special case where you need to join a table with itself to get specific information existing in the same table.  In this case you may choose the INNER JOIN, LEFT JOIN or RIGHT JOIN presented earlier to query the required data.

The SELF JOIN is illustrated in the following Venn diagram.



**Conclusion**

A join in database links tables together based on a common column between them. In this lesson you will learn about four different types of  joins supported in MySQL that can be used to provide users with different types of information about related records of data. It is important to understand how each of these types works to get the relevant type of information.

**Completed**

Go to next item

Like

Dislike

Report an issue

# MySQL REPLACE statement in depth

In this reading, you will learn about the key differences between the MySQL REPLACE statement and the standard INSERT INTO and UPDATE statements.

## What is the REPLACE statement in MySQL?

The MySQL REPLACE statement is an alternative way to insert and update data in a database table. It is an extension to the SQL Standard, which inserts or updates data in a table. You use it for slightly different purposes than the standard INSERT INTO and UPDATE. (This will be clarified later in this reading when it is explained how each of the three statements works.) You can use two types of syntax to insert or update data with the REPLACE statement. The first syntax is very similar to the standard SQL INSERT INTO statement, where you utilize the REPLACE command instead of the INSERT command as follows:

1

2

REPLACE INTO table\_name (column1name, column2name, ...)

VALUES (value1, value2, ...);





The second syntax is similar to the standard SQL UPDATE statement, where you utilize the REPLACE command with the SET keyword to update data in the table like this.

1

REPLACE INTO table\_name SET column1name = value, column2name = value, ... ;





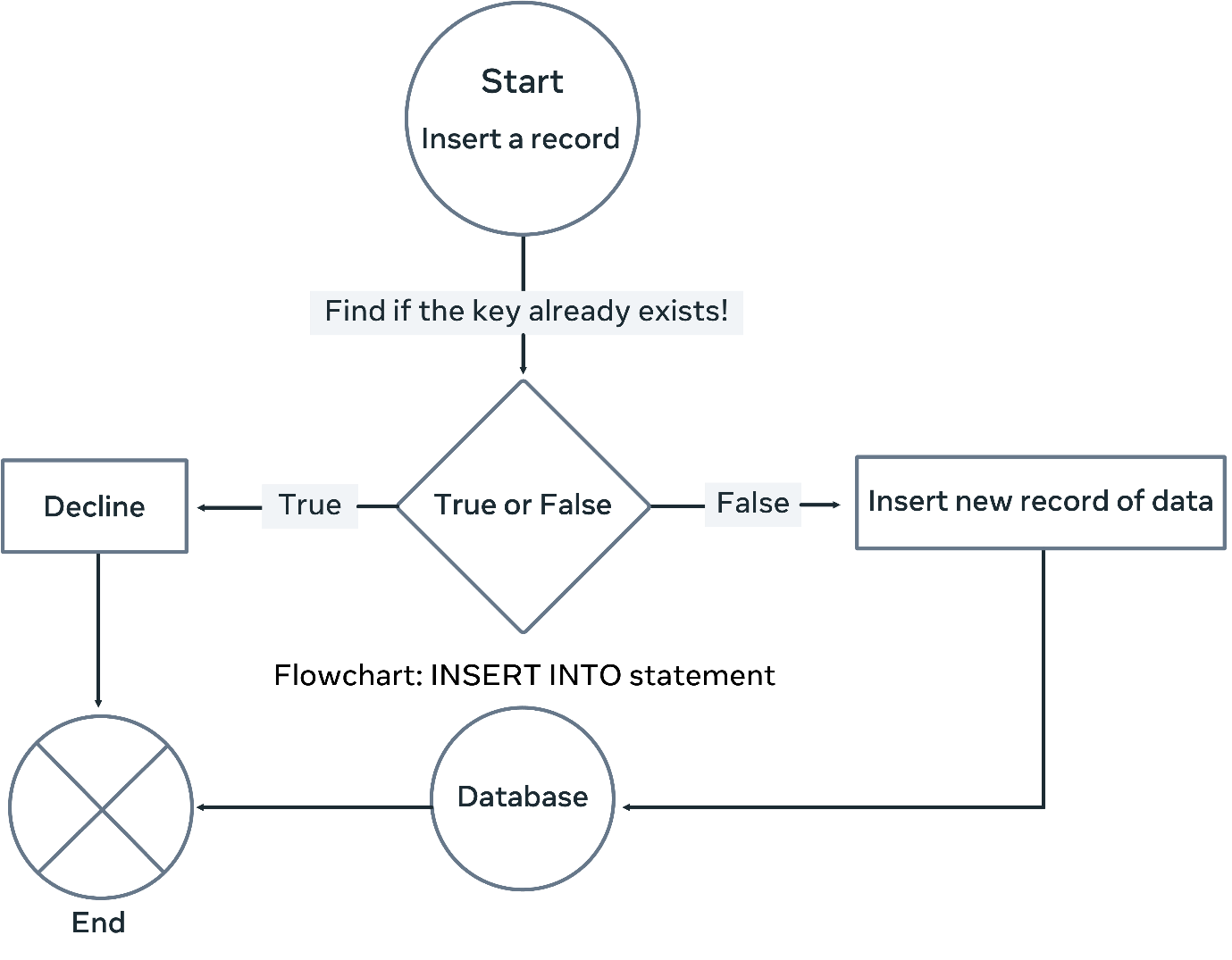
Now, since the INSERT INTO statement inserts data and the UPDATE statement updates data, what is the need for the REPLACE statement?

To answer this question, you need to understand how each of the three statements works.

## How the INSERT INTO statement works

The INSERT INTO statement attempts to insert a new record of data. It checks if the unique key already exists in the table. If YES or TRUE, the insert process is declined, and MySQL generates an error message.

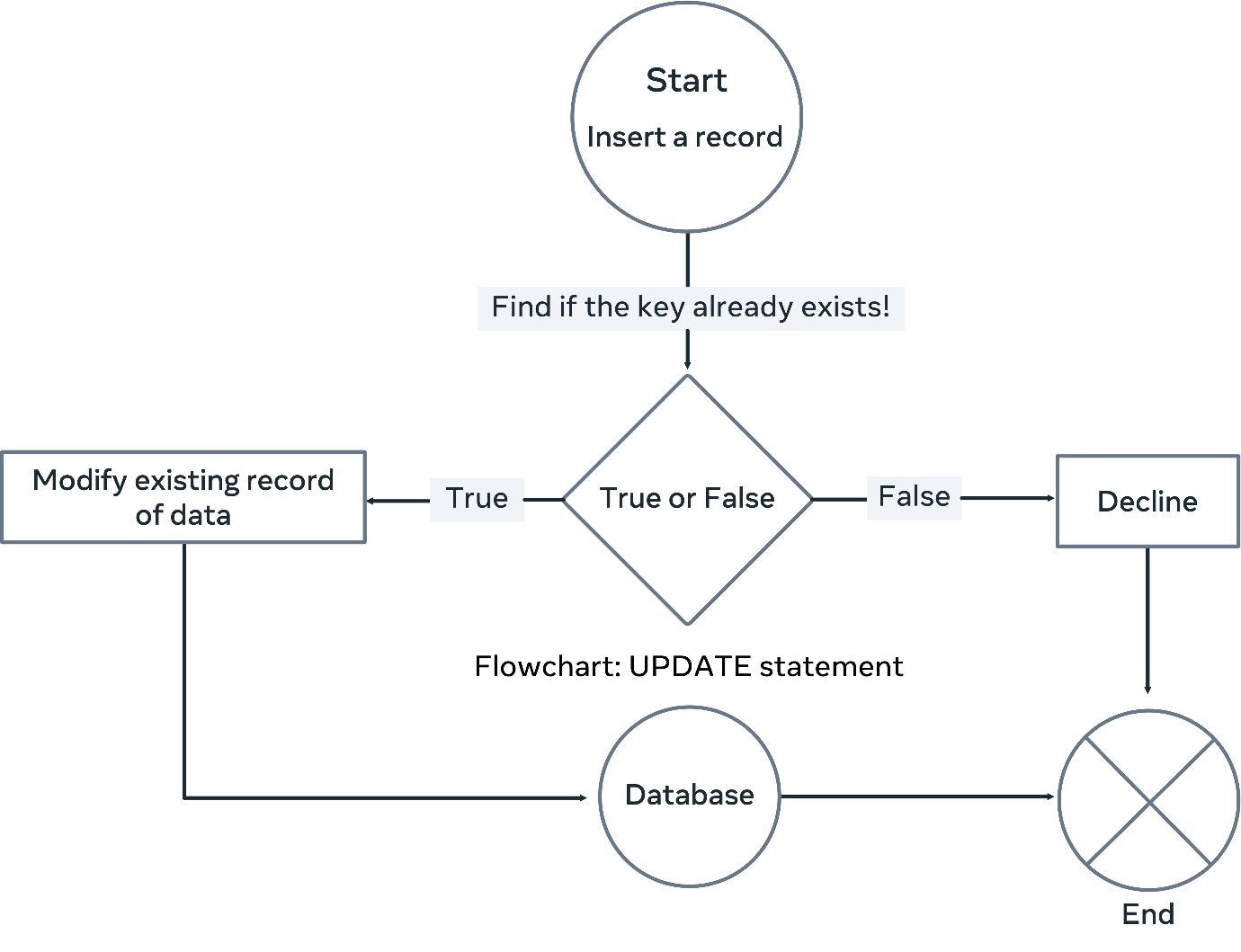
Suppose a value of NO or FALSE is returned. In that case, the insert process is completed, and the new data record is added to the database. A flowchart that demonstrates how the INSERT INTO statement works is illustrated below.



## How the UPDATE statement works

The update statement attempts to modify an existing record with new data. It checks if the unique key already exists in the table. Suppose a value of NO or FALSE is returned. In that case, the update process is declined, and MySQL generates an error message.

Suppose it returns a value of YES or TRUE. In that case, the update process is completed, and the existing data record is modified with the new data. A flowchart demonstrating how the UPDATE statement works illustrated below.



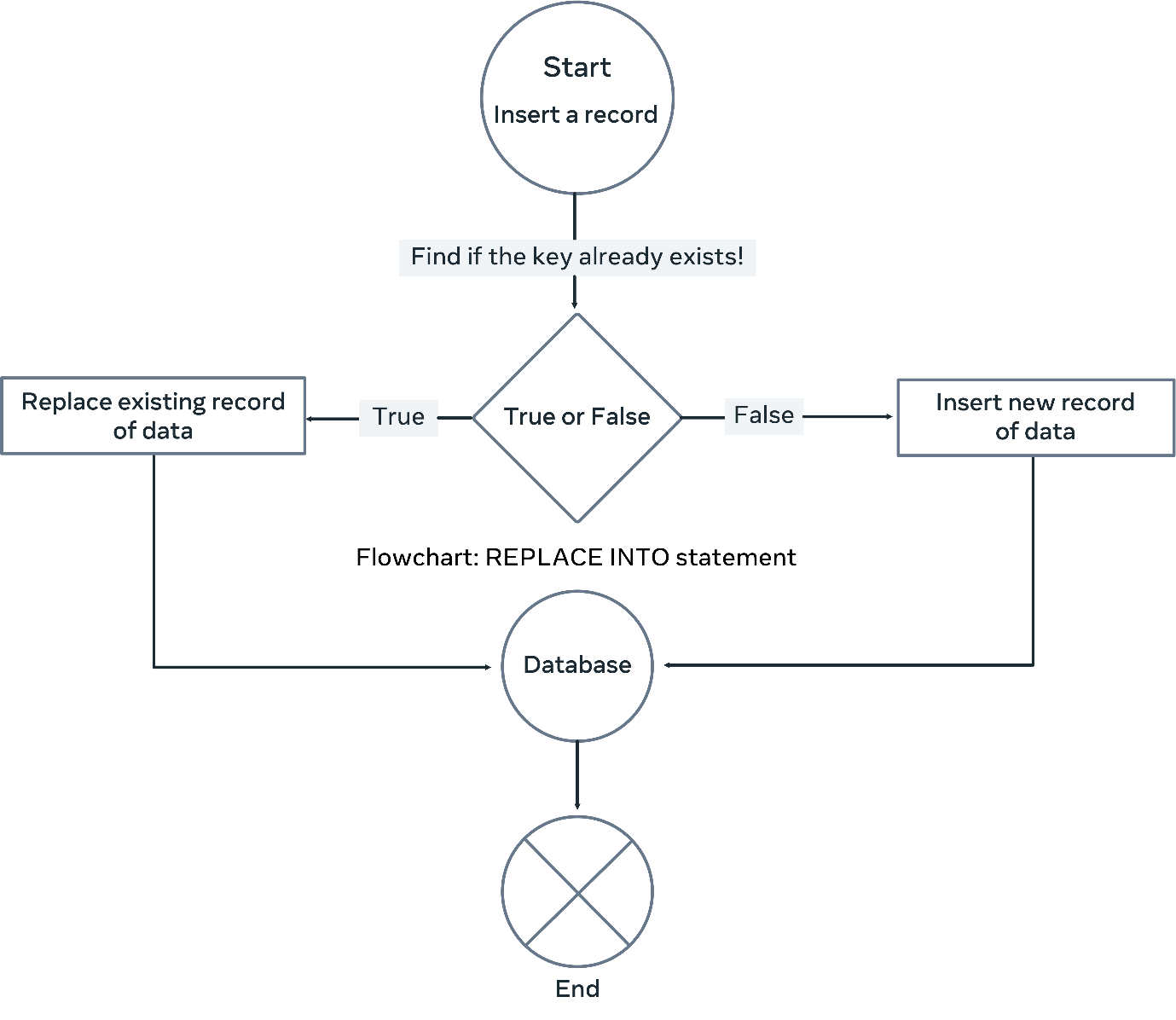
# How the REPLACE INTO statement works

The REPLACE statement checks whether the intended data record's unique key value already exists in the table before inserting it as a new record or updating it.

The REPLACE INTO statement attempts to insert a new record or modify an existing record. In both cases, it checks whether the unique key of the proposed record already exists in the table. Suppose a value of NO or FALSE is returne. In that case, the REPLACE statement inserts the record similar to the INSERT INTO statement.

Suppose the key value already exists in the table (in other words, a duplicate key). In that case, the REPLACE statement deletes the existing record of data and replaces it with a new record of data. This happens regardless of whether you use the first or the second REPLACE statement syntax.

A flowchart outlining how the REPLACE INTO statement works is illustrated below.



Once the REPLACE INTO statement is used to insert or modify data, it determines first whether the new data record already exists in the table. It checks if the PRIMARY or the UNIQUE KEY matches one of the existing records.

If there is no matching key, the REPLACE works like a normal INSERT statement. Otherwise, it deletes the existing record and replaces it with the new one. This is considered a sort of modification or update of an existing record. However, it would be best if you were careful here. Suppose you do not specify a value for a column in the SET clause. In that case, the REPLACE statement uses the default value (if a default value has been set). Otherwise, it's set as NULL.

# Virtual tables in depth

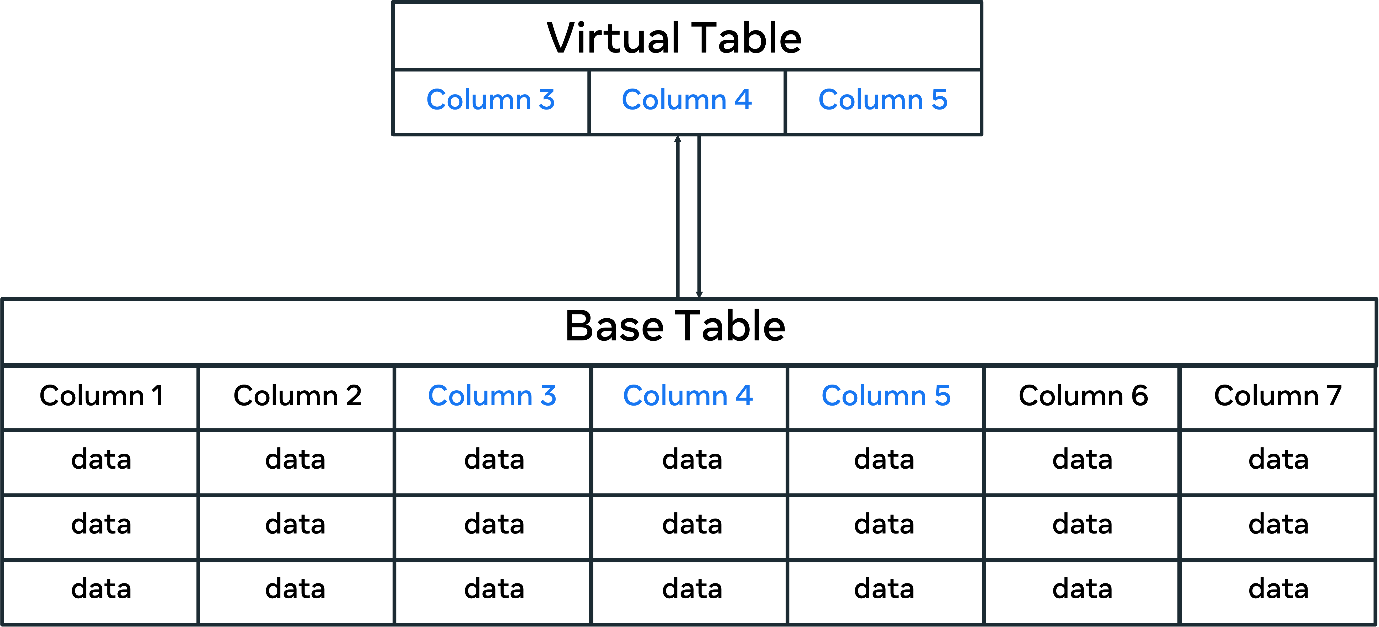
## What is a VIEW in SQL?

In SQL, a VIEW acts as a virtual table that utilizes data stored in existing tables in the database. The virtual table does not store any data itself. Instead, it acts as an interface that provides access to existing data.

## Why do you need to use virtual tables?

You have a database with a base table with 7 columns named: column 1, column 2, column 3, column 4, column 5, column 6 and column 7.

However, you are only interested in viewing and analyzing information in columns 3, column 4 and column 5. In this case, you can create a virtual table that contains the three required columns. This virtual table utilizes the data that exists in the corresponding columns in the base table, as presented in the following illustration.



In this case, all data in the base table could be treated as part of the virtual table. Though physically, they are stored in the base table, not the virtual table.

Along this line, there are several benefits to using virtual tables instead of base tables. These include simplifying access to data, manipulating data and providing security.

Let's explore these benefits in more detail.

## **Simplifying access to data**

You can use virtual tables to simplify how users query and access data in the database. For example, Lucky Shrub keeps information in their database about clients, staff, products and orders.

Lucky Shrub managers want to display the clients' ID, name and location from the Clients table and the orders they placed, including Order ID, Quantity and Cost from the Orders table.

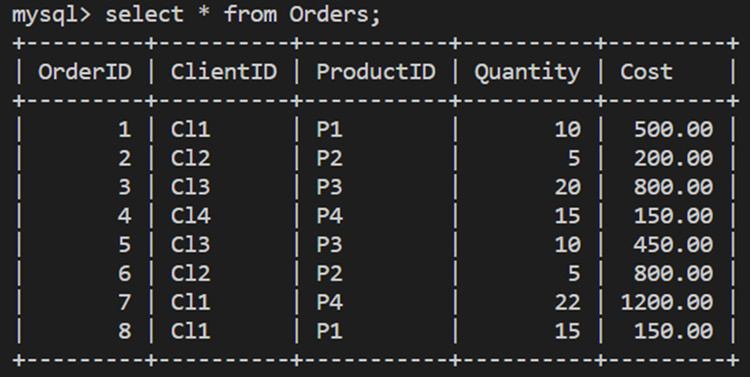
The managers want to display all of this data in one view without typing complicated queries every time they access this data.

To help the managers, you can create a virtual table from the two tables, the clients and the orders. To do this, you can combine the client's ID, name and location from the Clients table with the Order ID, Quantity and Cost from the Orders table.

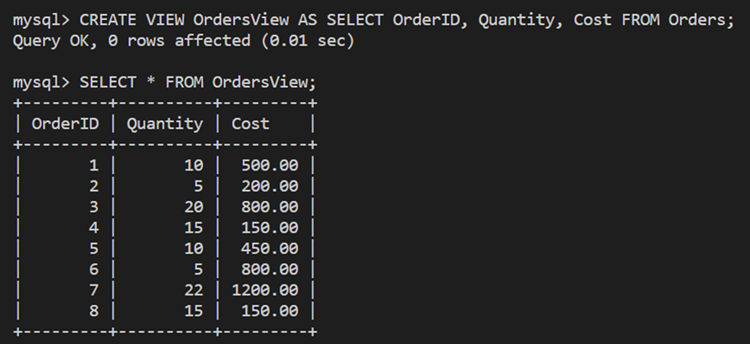
## **Manipulating data in the base table**

The virtual table allows you to manipulate, filter, and even update data in the base table if necessary.

For example, the Orders table below shows information about the Order ID, Client ID, Product ID, Quantity and Cost:



The following virtual table, “OrdersView,” has been created from the Orders table using only three required columns.



Lucky Shrub need to change the cost of the second record (order id = 2) to 250. You can make this change directly in the virtual table. Simply use the following SQL UPDATE statement:

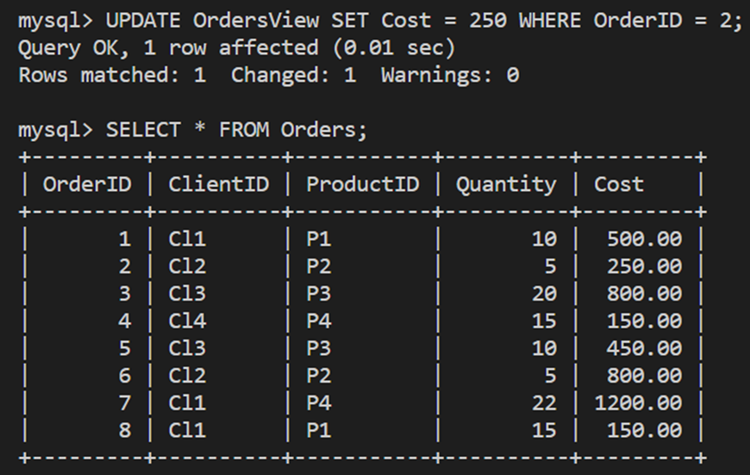
1

UPDATE OrdersView SET Cost = 250 WHERE OrderID = 2;





This updates the cost for order id = 2 in the Orders base table, as shown below.



## Virtual tables support database security

When you design virtual tables, you create interfaces for users to access relevant data in your database, similarly to the base tables. However, with the virtual tables, you can only show the required data and hide what is not needed (including data that you do not want the users to have access to).

In the previous example, you created a virtual table called "OrdersView" from the Orders table with information about the orders ID, quantity and cost only. This restricts the users from viewing specific columns. They only view what you want them to view. In this case, users will not be able to read information about client and product IDs. This may be sensitive information you need to hide from other database users.

## Conclusion

## What is a function in MySQL?

A function is a code that performs an operation and returns a result. Functions are used to manipulate data in a database table.

MySQL has many built-in functions that fall under different categories. These include:

* String manipulation functions
* Date and time functions
* Numeric functions
* Comparison functions
* And control flow functions

At this stage in the lesson, you should be familiar with basic examples of each of these types of functions. So let's take a few moments to explore other examples.

## **Numeric functions**

MySQL numeric functions can be broadly divided into mathematical and aggregate functions. You've reviewed numerous examples of many MySQL aggregate functions. You've also looked at many mathematical functions. In this reading, you'll explore a few more mathematical functions.

Mathematical functions allow you to perform mathematical tasks on numeric data.

Let's look at two meaningful and useful mathematical functions: CEIL and FLOOR.