**Additional resources**

**The following resources are additional reading materials that cover different filtering aspects supported in MySQL:**

* [**How to use the MySQL BETWEEN condition with syntax and examples.**](https://www.techonthenet.com/mysql/between.php)
* [**How to use the MySQL LIKE operator to query data based on a specified pattern.**](https://www.mysqltutorial.org/mysql-like/)
* [**Overview of the SQL AND, OR and NOT Operators.**](https://www.w3schools.com/sql/sql_and_or.asp)

**Additional resources**

**The following resources are additional reading materials that provide an introduction to JOINS in MySQL:**

* [**Overview of the different types of MySQL JOIN clauses that can be leveraged for your database operations.**](https://www.mysqltutorial.org/mysql-join/)
* [**Overview of JOIN types and the process steps for creating JOINS.**](https://www.w3schools.com/sql/sql_join.asp)

**Go to next item**

**Additional resources**

**The following resources are additional reading materials that provide more information about how the GROUP BY and HAVING clauses work in MySQL:**

* [**Overview of the GROUP BY and HAVING clause with relevant examples.**](https://www.datacamp.com/tutorial/group-by-having-clause-sql)
* [**Process steps for using the MySQL HAVING clause to specify a filter condition for groups of rows or aggregates.**](https://www.mysqltutorial.org/mysql-having.aspx)
* [**Introduction to the GROUP BY statement with relevant examples.**](https://www.w3schools.com/mysql/mysql_groupby.asp)

**Additional resources**

**The following resources are additional reading materials that explain the REPLACE INTO statement and how it works in MySQL:**

* [**Overview of the differences between the REPLACE INTO and INSERT INTO statements.**](https://dev.mysql.com/doc/refman/8.0/en/replace.html)
* [**Demonstration of how to use the MySQL REPLACE statement to insert data into database tables.**](https://www.mysqltutorial.org/mysql-replace.aspx)
* [**Overview of how to use the REPLACE INTO statements in MySQL.**](https://www.tutorialspoint.com/mysql/mysql_replace.htm)

**Additional resources**

**The following resources are additional reading materials that provide information about the use of constraints in MySQL.**

* [**Overview of MySQL constraints and common ways in which they're used.**](https://www.w3resource.com/mysql/creating-table-advance/constraint.php)
* [**Demonstration of how to work with NOT NULL**](https://www.mysqltutorial.org/mysql-not-null-constraint/))
* [**Overview of the different types of constraints in MySQL**](https://www.w3schools.com/MySQL/mysql_constraints.asp)

**Additional resources**

**The following resources are additional reading materials about altering tables in MySQL.**

* [**Overview of the MySQL ALTER TABLE statement**](https://www.w3resource.com/mysql/altering-table/altering-table.php)
* [**Demonstration of the MySQL ALTER TABLE statement**](https://www.mysqltutorial.org/mysql-alter-table.aspx)
* [**Examples of the MySQL ALTER TABLE statement**](https://www.w3schools.com/mySQl/mysql_alter.asp)

**Additional resources**

**The following resources are additional reading materials that provide more information about subqueries in MySQL.**

* [**Detailed explanation of MySQL subqueries**](https://www.w3resource.com/sql/subqueries/understanding-sql-subqueries.php)
* [**Overview of how to use MySQL subqueries to write complex SQL queries**](https://www.mysqltutorial.org/mysql-subquery/)
* [**Examples of how to create subqueries in MYSQL**](https://learnsql.com/blog/sql-subquery-examples/)

**Additional resources**

**The following resources are additional reading materials about virtual tables in MySQL.**

* [**Overview of virtual tables in MySQL.**](https://www.w3resource.com/mysql/mysql-views.php)
* [**Demonstration of how to manage views in MySQL.**](https://www.mysqltutorial.org/managing-sql-views.aspx)
* [**Examples of how to create, update and drop virtual tables in MySQL.**](https://www.w3schools.com/mySQl/mysql_view.asp)

**Additional resources**

**The following resources are additional reading materials about functions in MySQL:**

* [**Process steps for using comparison functions and operators in MySQL**](https://www.sqlshack.com/learn-mysql-control-flow-functions/)
* [**Demonstration of using control flow functions to add IF-THEN-ELSE logic to SQL queries**](https://www.mysqltutorial.org/mysql-control-flow-functions/)
* [**Coding examples for using comparison functions and operators in MySQL**](https://prestodb.io/docs/current/functions/comparison.html)

**Additional resources**

**The following resources are additional reading material that provide an introduction to the concept of stored procedures, and how it works in MySQL:**

* [**Overview of procedures and how to call them**](https://www.w3resource.com/mysql/mysql-procedure.php)
* [**Process steps for creating a new stored procedure in MySQ**](https://www.mysqltutorial.org/getting-started-with-mysql-stored-procedures.aspx)**L**
* [**Advantages of developing stored procedures**](https://docs.oracle.com/en/database/oracle/oracle-database/12.2/jjdev/stored-procedure-advantages.html#GUID-B6F864FC-5C87-45DC-BBEC-2CEE469A939C)

**Reference sheet: Using operators and the WHERE clause**

**Use the WHERE clause in SQL SELECT, UPDATE and DELETE statements to filter data.**

**The WHERE clause can specify one or more filter conditions to determine how data is limited or narrowed down from the queried tables.**

**It's also possible to use logical operators and some other operators when formulating filter conditions.**

**The logical AND and OR operators are used as conjunctive operators to combine multiple conditions in the WHERE clause. The logical NOT operator negates the result of evaluating a given condition (or conditions) and filters data accordingly.**

**The IN operator can be an alternative to the OR operator. The IN operator can check multiple values, equivalent to using the OR logical operator to specify various conditions.**

**The BETWEEN operator is a convenient way to perform a range check on numeric and date values. It can be used as an alternative to greater than or equal to (>=) and less than or equal to (<=) comparison operators.**

**Use the LIKE operator to match and filter data based on these patterns. The wildcard characters, percentage (%) and underscore (\_) are used to formulate patterns. The percentage ( % ) wildcard matches *any string of zero or more characters*. The underscore (\_) wildcard matches *any single character*.**

**There are different ways in which the % and \_ characters can be used with the LIKE operator to formulate patterns. Here are some examples to help you to get a better understanding.**

**Let's take the example of a scenario where a pattern match must be performed on the EmployeeName column in the employee's table to filter data.**

**Here are some examples of patterns using the LIKE operator that you might consider.**

**Employees' table**

| **ID** | **EmployeeName** | **Role** | **AnnualSalary** |
| --- | --- | --- | --- |
| **1** | **Mario Gollini** | **Manager** | **70000** |
| **2** | **Adrian Gollini** | **Assistant Manager** | **65000** |
| **3** | **Giorgos Dioudis** | **Head Chef** | **50000** |
| **4** | **Fatma Kaya** | **Assistant Chef** | **45000** |
| **5** | **Elena Salvai** | **Head Waiter** | **40000** |
| **6** | **John Millar** | **Receptionist** | **35000** |
| **7** | **Adam Collins** | **Assistant Manager** | **65000** |

**Examples of patterns based on the Employees' table**

| **Pattern** | **How it works** |
| --- | --- |
| **WHERE EmployeeName LIKE 'a%'** | **Finds EmployeeName values that start with "a". Matches with records that have IDs of 2 and 7.** |
| **WHERE EmployeeNameLIKE '%i'** | **Finds EmployeeName values that end with "i". Matches with the records that have IDs 1, 2, and 5.** |
| **WHERE EmployeeName LIKE '%li%'** | **Finds EmployeeName values that have "li" in any position. Matches with the record that have IDs 1, 2 and 7.** |
| **WHERE EmployeeName LIKE '\_a%'** | **Finds EmployeeName values that have "a" in the second position. Matches with the records that have IDs 1 and 4.** |
| **WHERE EmployeeName LIKE 'a\_\_\_%'** | **Finds EmployeeName values that start with "a" and are at least 4 characters in length. Matches with records that have IDs of 2 and 7.** |
| **WHERE EmployeeName LIKE 'a%i'** | **Finds EmployeeName values that start with "a" and end with "i". Matches with the record that has an ID of 2.** |

**Joining tables**

**Introduction**

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.

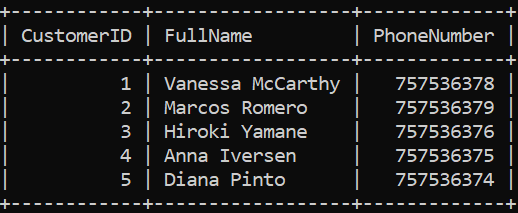
There are four different types of joins supported in MySQL that are covered in this lesson.

* 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.

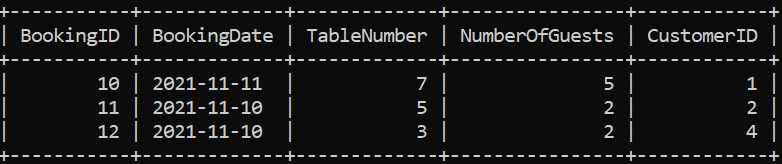
The first is the Customers table with the following columns:

* 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, representing 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

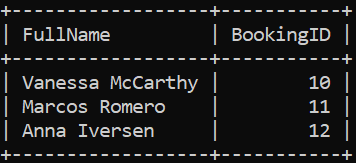
3

 SELECT Customers.FullName, Bookings.BookingID

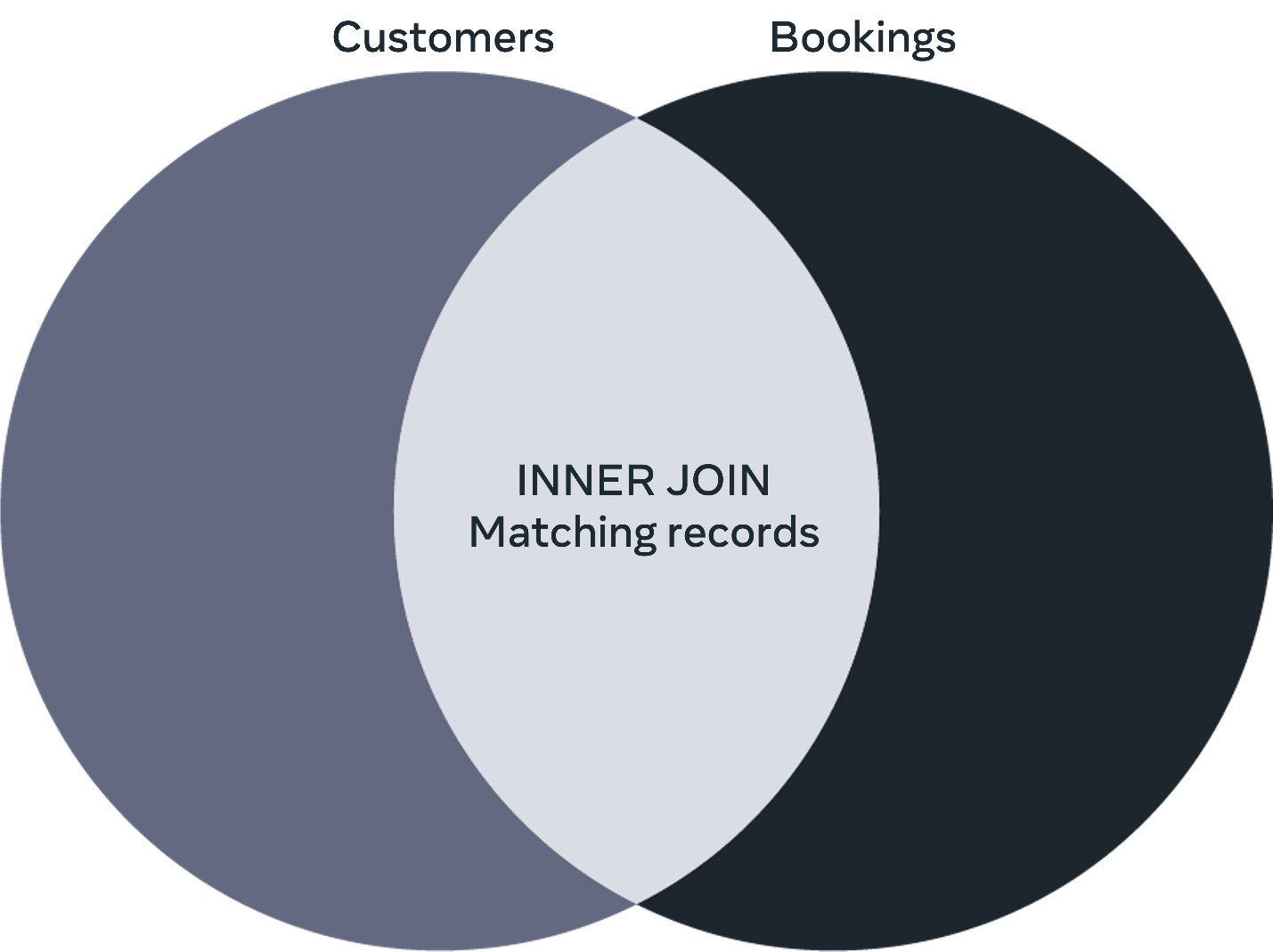
 FROM Customers INNER JOIN Bookings

 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

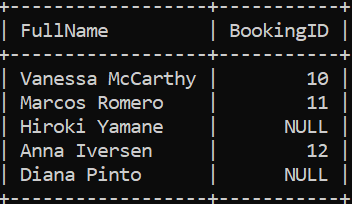
3

SELECT Customers.FullName, Bookings.BookingID

FROM Customers LEFT JOIN Bookings

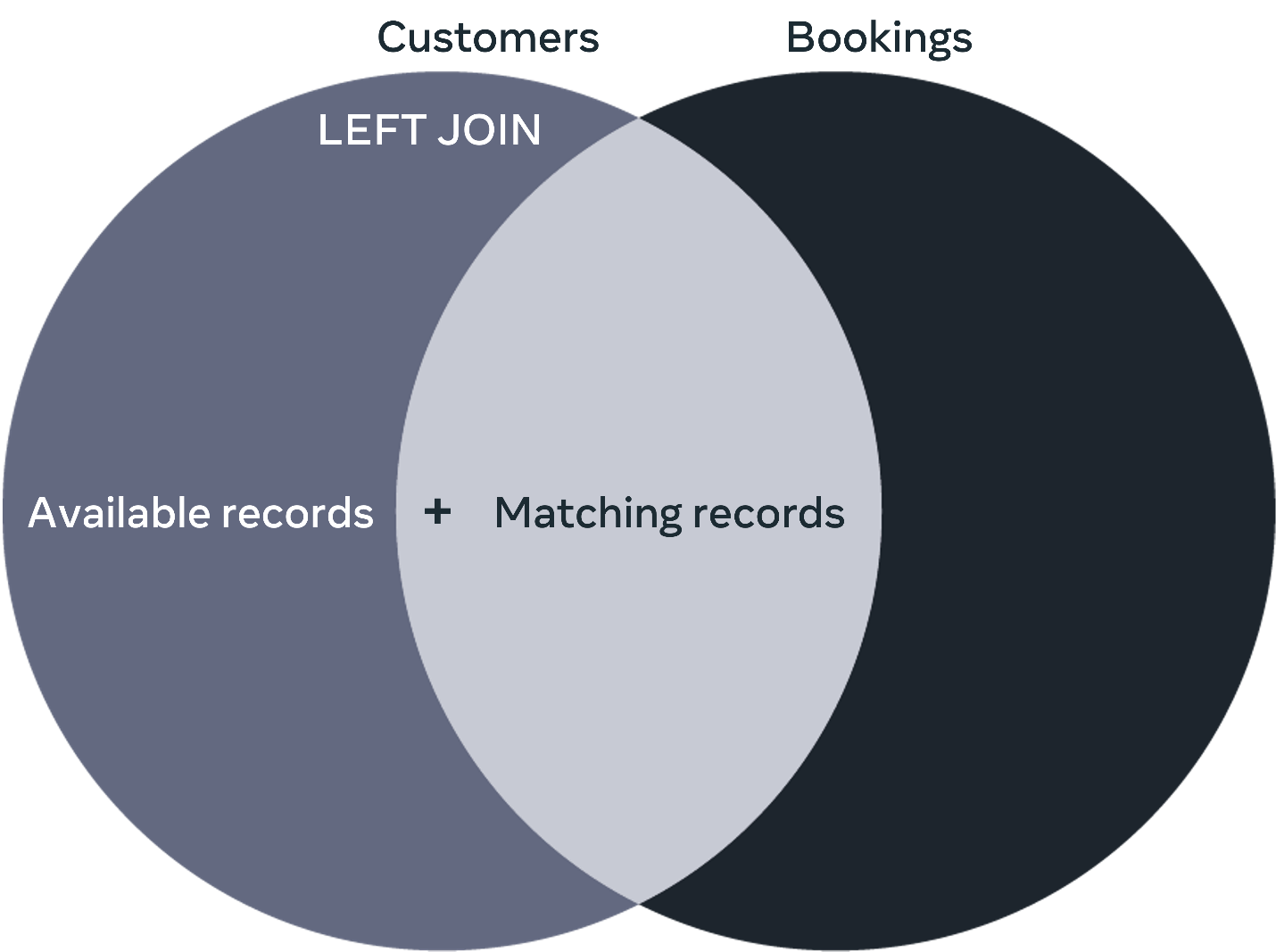
ON Customers.CustomerID =  Bookings.CustomerID;

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

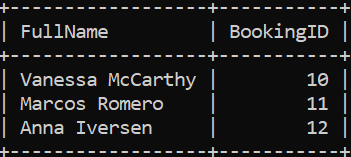
3

SELECT Customers.FullName, Bookings.BookingID

FROM Customers RIGHT JOIN Bookings

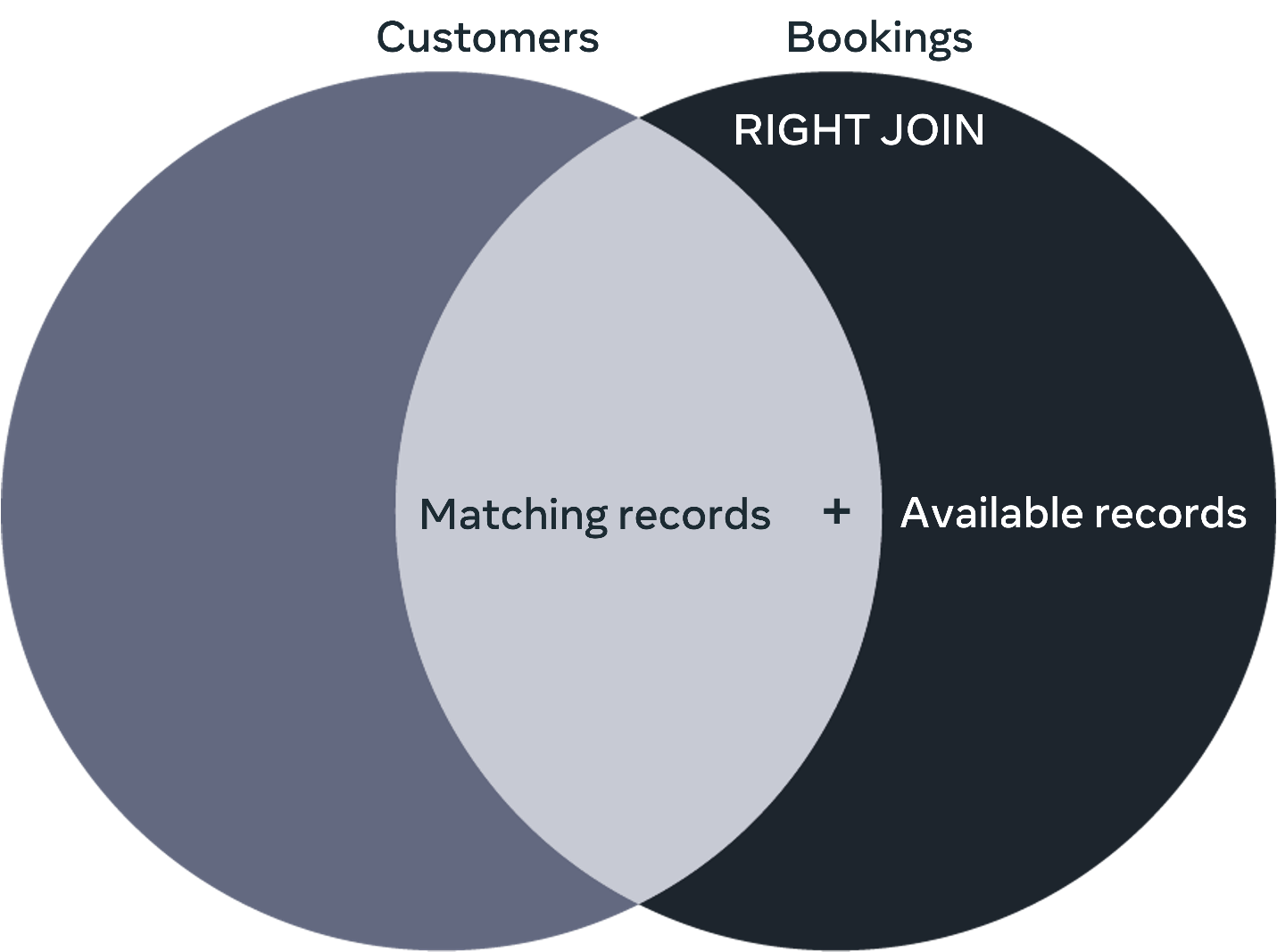
ON Customers.CustomerID = Bookings.CustomerID;

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.

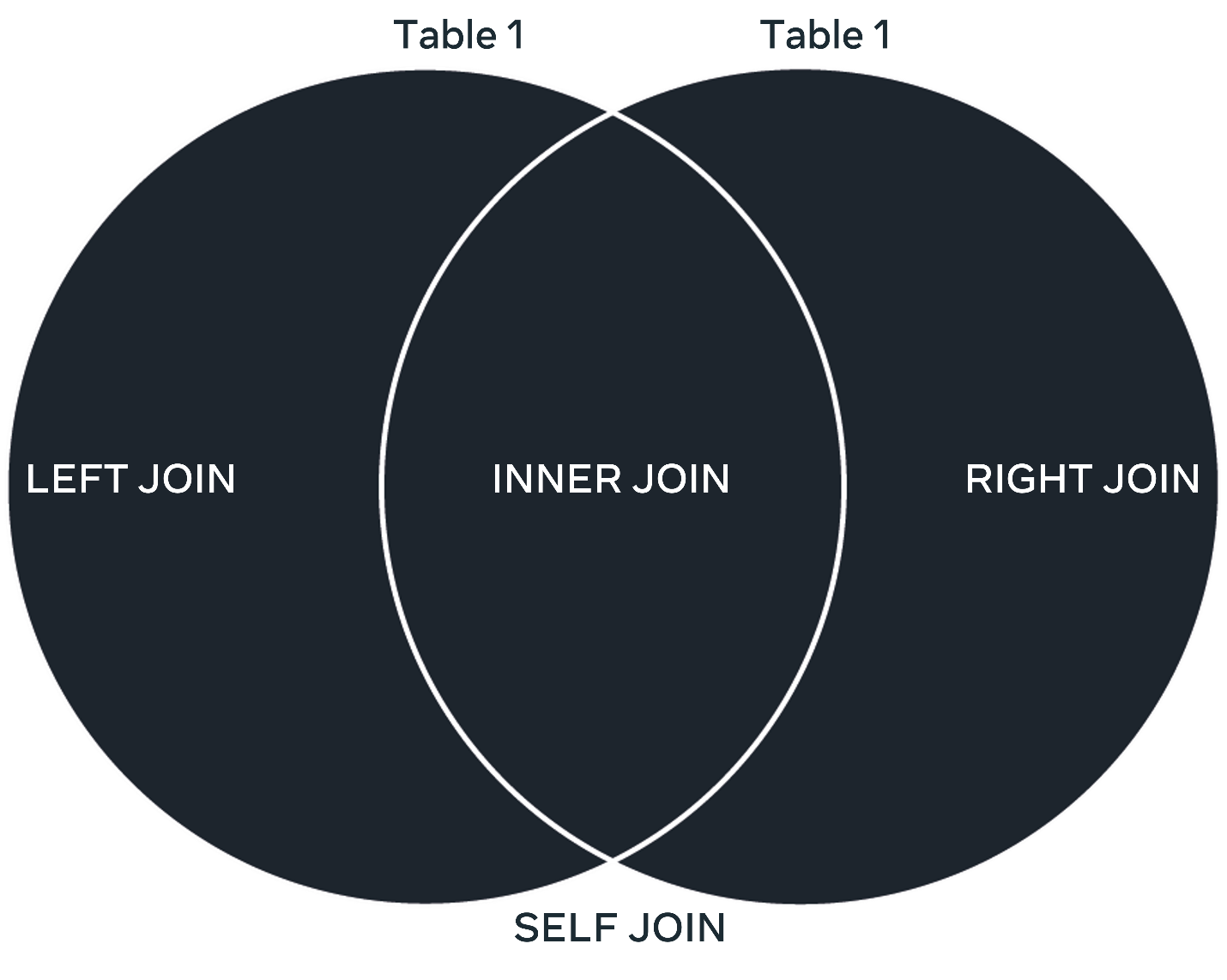
The RIGHT JOIN is illustrated in the following Venn diagram.



**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 database JOIN 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 information about related data records. It is important to understand how each of these types works to get the relevant type of information.

**Reference sheet: Operators and clauses**

**GROUP BY clause**

Use the GROUP BY clause in a SELECT statement to group rows in a table(s) based on a given column(s) into summary rows or subgroups.

It is placed after the FROM clause. If there is a WHERE clause in your SELECT statement, it should be placed after the WHERE clause. After the GROUP BY keyword, place a list of comma-separated column names by which you want to group the data.

**HAVING clause**

If you also want to filter your grouped data, use the HAVING clause. You should be aware that the WHERE clause cannot filter grouped data. The HAVING clause should appear after the GROUP BY clause. In the HAVING clause, you can specify the filter condition(s) that needs to be applied to your grouped data.

**ANY operator**

The ANY operator lets you perform a comparison between a single column value and a range of other values. The range of values comes from the execution of a subquery.

The syntax of a statement that uses an ANY operator is as follows:

1

2

3

4

5

6

SELECT column\_name(s)

FROM table\_name

WHERE column\_name comparison operator ANY

  (SELECT column\_name

  FROM table\_name

  WHERE condition);

The ANY operator returns a boolean value following a comparison operation. It returns a TRUE value if ANY subquery values meet the given condition. In other words, the condition will be TRUE if the operation is true for any of the values in the range.

In this syntax, the ANY operator should be preceded by a column name and a comparison operator that operates on the column name against the set of values.

Standard comparison operators like =, <>, !=, >, >=, <, or <= can be used here.

**ALL operator**

Use the ALL operator for the same purpose as the ANY operator. However, the way it works is a little bit different. It returns a boolean value as a result of performing a comparison operation. It returns TRUE only if ALL subquery values meet the given condition. In other words, the condition will be TRUE only if the operation is true for all values in the range.

The syntax for the ALL operator is as follows:

1

2

3

4

SELECT column\_name(s)

FROM table\_name

WHERE column\_name operator ALL

  (SELECT column\_name FROM table\_name WHERE condition);

The syntax is interpreted in the same way as with ANY and the standard comparison operators can be used with it.

Go to next item

**Completed**

Like

Dislike

Report an issue

**MySQL REPLACE statement in depth**

In this reading, you will learn 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, ... ;

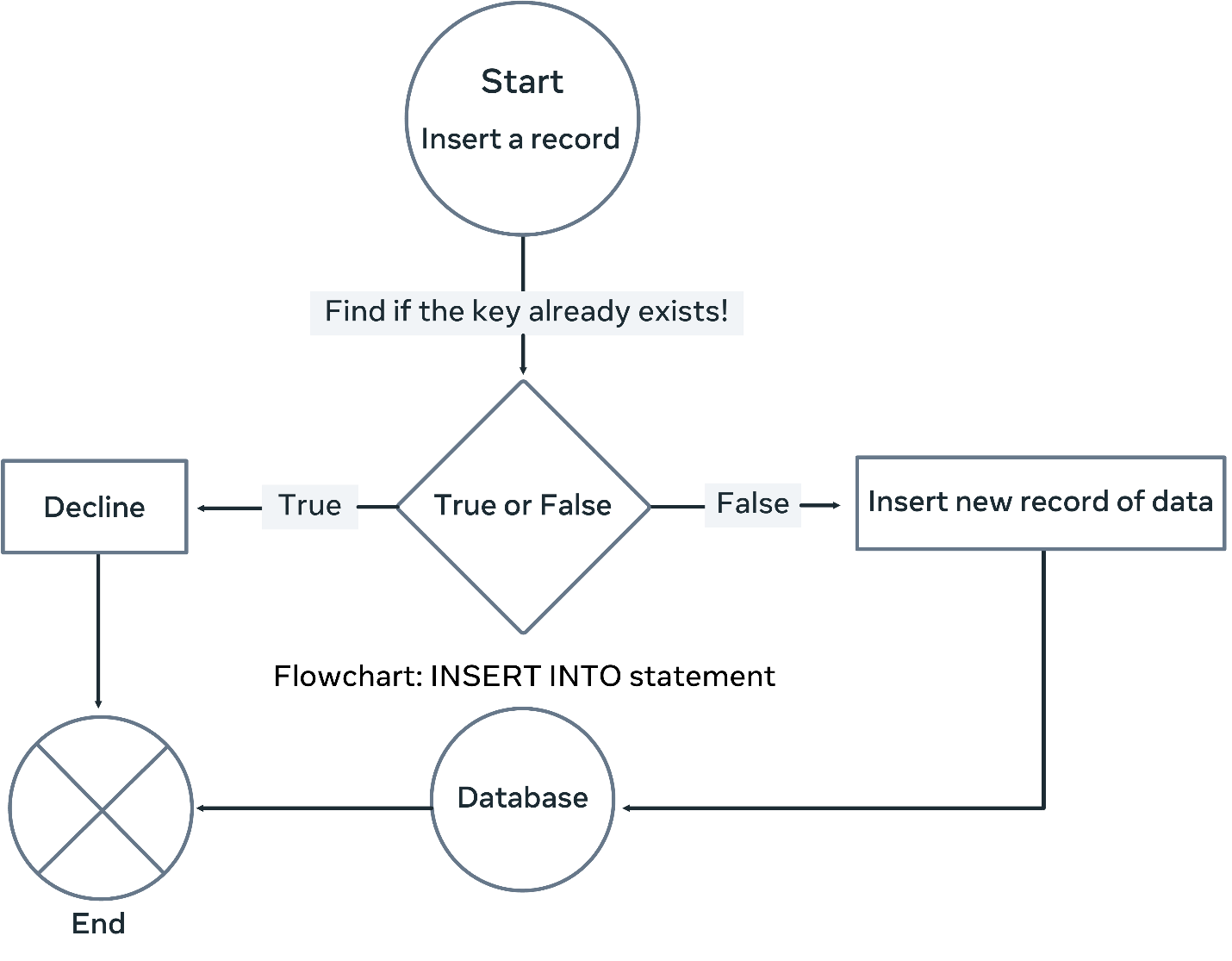
Now, since the INSERT INTO statement inserts data and the UPDATE statement updates data, what is the need for the REPLACE statement?

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.

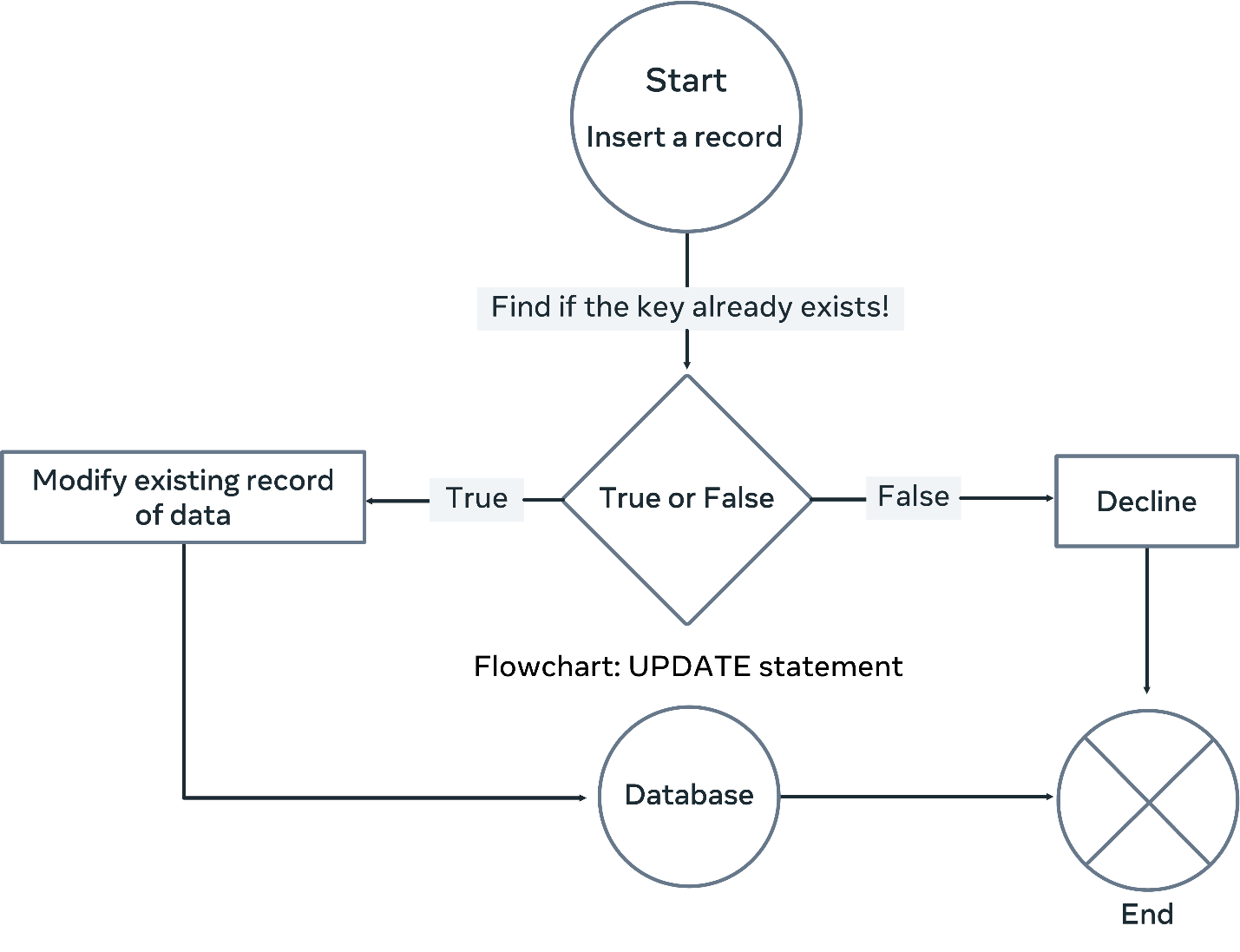
Suppose a value of NO or FALSE is returned. In that case, the insert process is completed, and the new data record is added to the database. A flowchart that demonstrates how the INSERT INTO statement works is illustrated below.



**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.



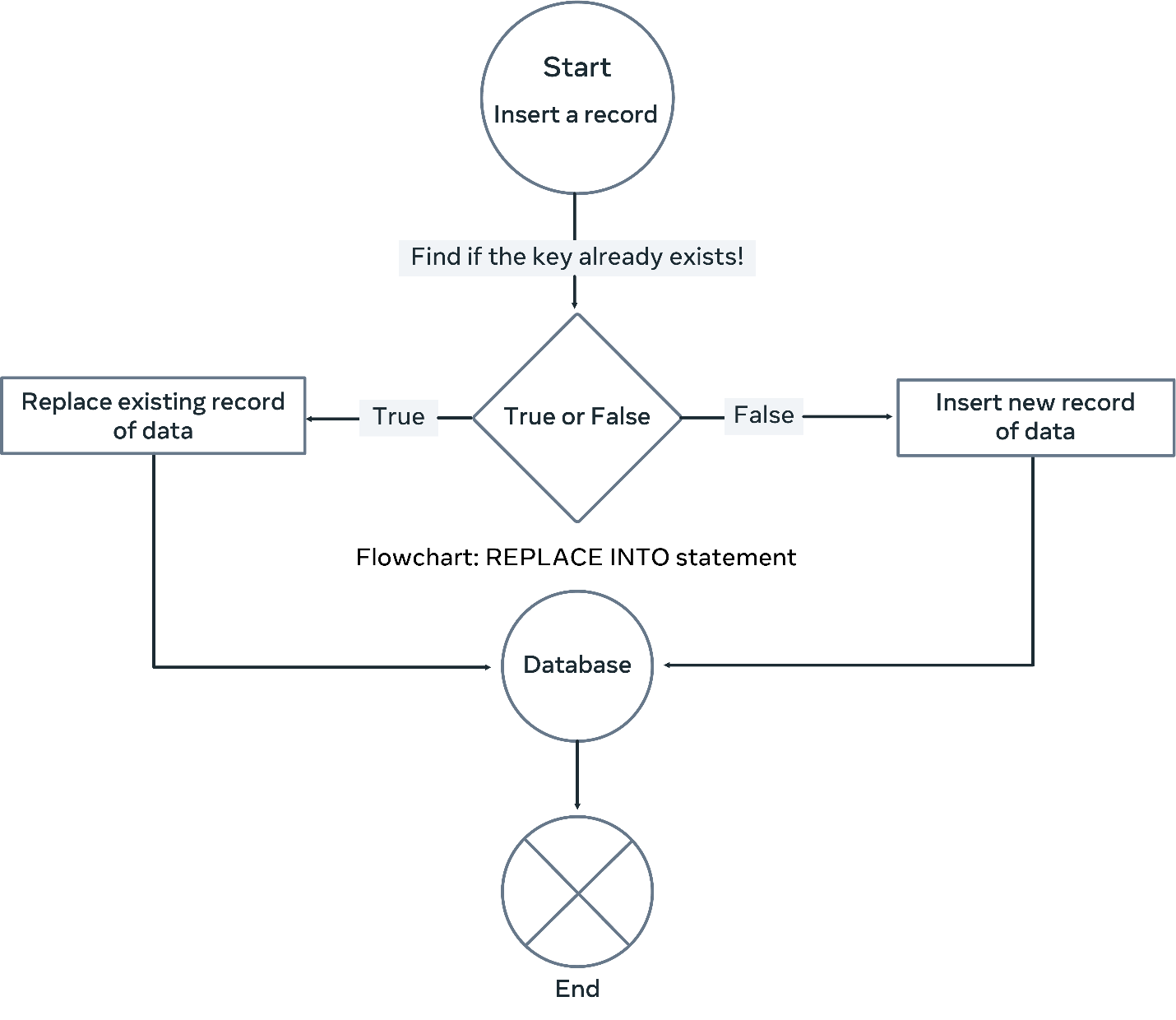
**How the REPLACE INTO statement works**

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 returned. 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.

**Conclusion**

The MySQL REPLACE statement is an alternative way to insert and update data in a table in a database. It is helpful in many cases, especially when you want to delete and replace existing records with new ones.

**Types of MySQL constraints**

In this reading, you will learn about the three main types of constraints that can be applied in MySQL.

**What are constraints in MySQL?**

Constraints in MySQL databases are used to define the rules that maintain data in tables to ensure data validity, accuracy, consistency and reliability.

Applying constraints on data in a relational database controls the type of data stored in a table. This will force MySQL to decline processes that violate the specified rules.

For example, you must be 18 years old in many countries to apply for a driving license. In this case, you can apply the SQL CHECK constraints on the relevant column in the related table in the database to restrict registering any person under 18.

Another example is that in most universities nowadays, each student will be registered with an email address containing the university domain name. You can apply the UNIQUE constraint to ensure that the university does not provide two students with the same email address.

In these cases, if there is any conflict between the constraints and the data insertion or update process, the process violating the constraint rule gets aborted.

**Types of constraints**

There are three main types of constraints that can be applied in MySQL.

**Key Constraints**

There are different types of keys in a relational database. For example, each table must have a primary key that maintains table integrity. The primary key ensures no duplications of records in the same table. Also, it allows identifying each data record using the primary key value. Therefore, it must be unique in each row of the table, and it is not allowed to contain null values.

For example, each citizen living in Denmark must have a unique personal number, which can be used to access different types of state services.

**Domain Constraints**

Domain constraints refer to special rules defined for the values that can be stored for a certain column. To apply this, you must specify what data values are allowed and which ones should be rejected.

For example, you can define a valid range number for users to rate a streaming service that offers a wide variety of TV shows and movies. This range could be a number between 3 and 10, in which case the user will not be able to insert a value that is more than 10 and less than 3.

**Referential Integrity Constraints**

In a relational database, tables are connected via a foreign key in one table linked to a primary key (or a unique key) in another table.

This implies that the value of the foreign key column in the ‘referencing’ table must also exist in the referenced table. Otherwise, you will end up with a problem as the “connection” between the records of the tables will cease.

Therefore, maintaining referential integrity requires that a foreign key value must have a matching primary key value to link the records of different related tables.

**Conclusion**

It is very important to apply constraints on data in a relational database to maintain data in tables to ensure data validity, accuracy, consistency and reliability.

**Reference sheet: ALTER TABLE statement and its uses**

MySQL. You will repeatedly need it when developing your database.

You will find that the table you initially created in the database needs adjusting. Maybe it has some missing columns or constraints, or it may include some irrelevant ones, or you may have misnamed the table or the column, so you need to fix it.

Use the ALTER TABLE statement to solve all these problems, allowing you to change an existing table structure. It will enable you to add columns, delete columns, change column type, change column constraints, and even rename columns and the table itself.

**Commands to use with the ALTER Statement**

**ADD**

To add a column to an existing table, you can use the ADD command followed by the column name and data type. The syntax is as follows:

1

ALTER TABLE table\_name ADD column\_name datatype;

Example:

1

ALTER TABLE Employees ADD Email VARCHAR(255);

**MODIFY**

To modify a column, you can use the MODIFYcommand followed by the column that you want to modify.

Syntax:

1

ALTER TABLE table\_name MODIFY COLUMN column\_name datatype;

Example:

1

ALTER TABLE Employees MODIFY COLUMN Email VARCHAR(50);

**Adding a FOREIGN KEY to an existing table using ADD**

To create a link between one table and another table, you can use the FOREIGN KEY and REFERENCES commands along with the ADD command.

Syntax:

1

ALTER TABLE table\_name ADD FOREIGN KEY (primary\_key\_column\_name) REFERENCES link\_table\_name(reference\_column\_name);

Example:

1

ALTER TABLE Orders ADD FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID);

**DROP COLUMN**

To remove/delete an existing column from a table, you can use the DROP COLUMN command followed by the column that you want to modify.

Syntax:

1

ALTER TABLE table\_name DROP COLUMN column\_name;

Example:

1

ALTER TABLE Employees DROP COLUMN Email;

**CHANGE**

You can use the CHANGE command with the ALTER statement to rename a column.

Syntax:

1

ALTER TABLE table\_name CHANGE from\_column to\_column datatype;

Example:

1

ALTER TABLE Employees CHANGE Email BusinessEmail VARCHAR(50);

**RENAME**

The RENAME command can be used to change a table name, followed by the new name that needs to be given to the table.

Syntax:

1

ALTER TABLE table\_name RENAME new\_table\_name;

Example:

1

ALTER TABLE OrderStatus RENAME OrderDeliveryStatus;

**Reference sheet: Subqueries**

**What is a subquery?**

A subquery is **a query within another query**. In other words, an **inner query.**In a subquery, a SQL SELECT query is placed as a part of another query called an outer query. Think of the inner query as the child query and the outer query as the parent query.

Place a subquery in the SELECT, FROM, or WHERE clause. You will find it used in the WHERE clause most of the time. Also, always place a subquery within a pair of parentheses. There can be subqueries that return a single value, a single row, a single column, or multiple rows of one or more columns. When writing subqueries, it is possible to use comparison operators such as =, >, >=, <, <=, != (or <>). In addition, you can also use ALL, ANY and SOME operators followed by a comparison operator.

**Types of sub-queries**

**Subqueries in the FROM clause**

You already know that in SQL, in the FROM clause, you specify the table name in which the data resides. When you use a sub-query in the FROM clause, the result returned from a subquery is used as a temporary table.

This table is referred to as a derived table as well.

Here’s what the syntax of a sub-query in the FROM clause looks like:

1

SELECT some\_column, some\_column FROM (Subquery) AS alias;

**Subqueries in INSERT statements**

You already know that you can use INSERT INTO SELECT statements. That means to use SELECT statements in an INSERT statement when you want to retrieve some data and insert them into another table.

The SELECT statement here can contain a subquery.

Here’s what the syntax of a subquery in an INSERT statement looks like:

1

INSERT INTO table1 SELECT column1,column2,column3 FROM table2 WHERE some\_column some\_operator(Subquery);

**Subqueries in UPDATE statements**

An update statement can also use a subquery. In the UPDATE statement’s WHERE clause, you can use a subquery to filter out the records you want to update.

Here’s what the syntax of a subquery in an UPDATE statement looks like:

1

UPDATE table1 SET column1 = some\_value WHERE some\_column some\_operator(Sub-query);

**Subqueries in DELETE statements**

You can also use subqueries in your DELETE statement. Once again, in the WHERE clause of the DELETE statement, you can use a subquery to filter out the records you want to delete.

Here’s what the syntax of a subquery in a DELETE statement looks like:

1

DELETE FROM table1 WHERE some\_column some\_operator(Subquery);

**Virtual tables in depth**

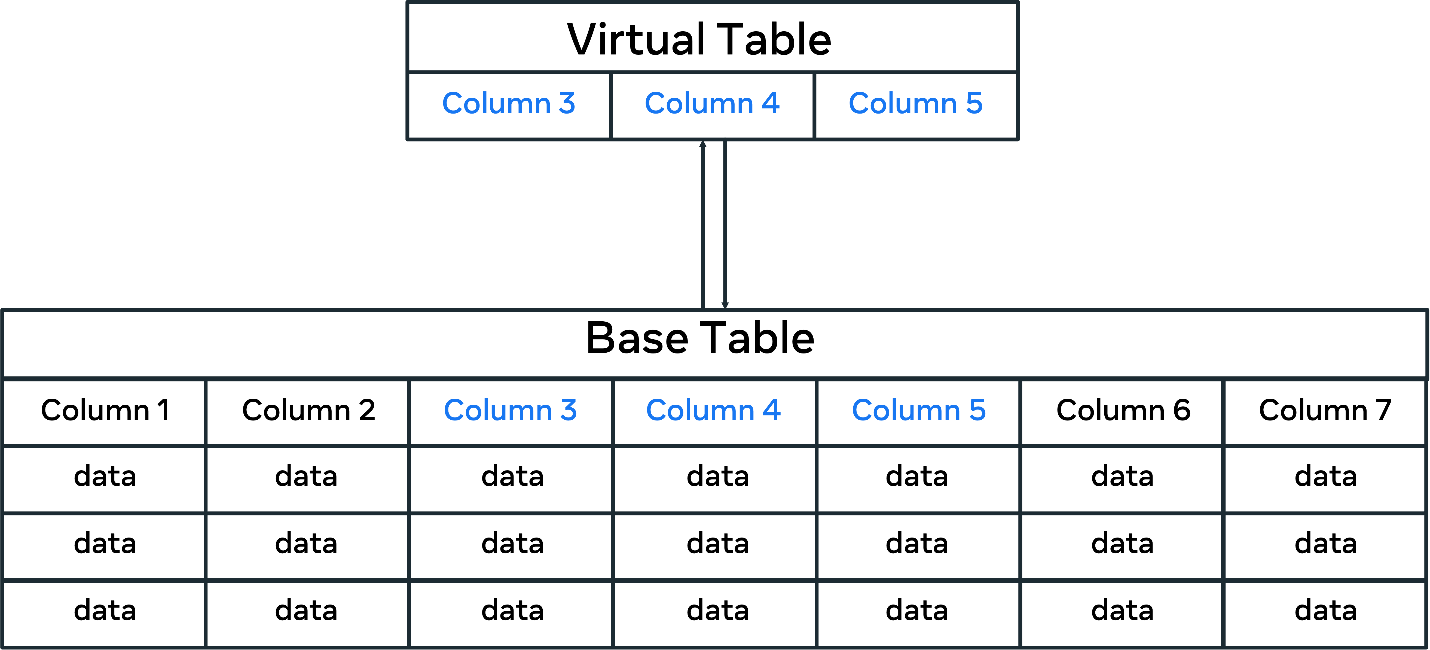
**What is a VIEW in SQL?**

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.

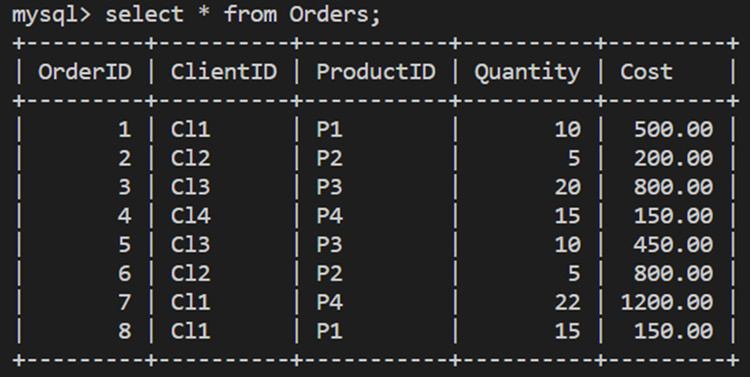
The managers want to display all of this data in one view without typing complicated queries every time they access this data.

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.

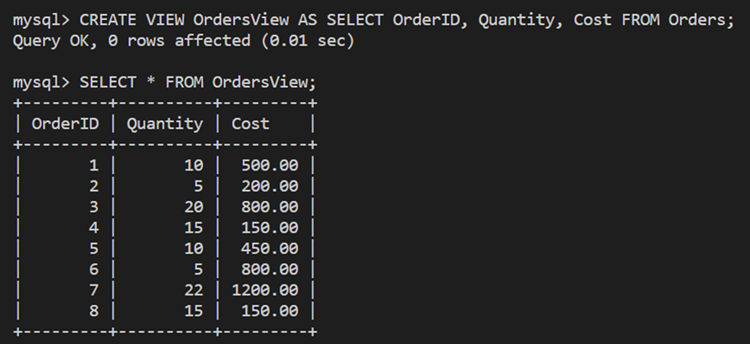
**Manipulating data in the base table**

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.

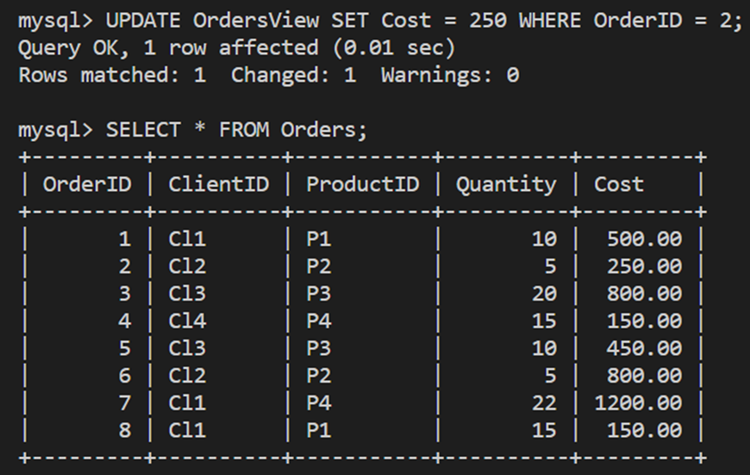


Lucky Shrub need to change the cost of the second record (order id = 2) to 250. You can make this change directly in the virtual table. Simply use the following SQL UPDATE statement:

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**

Virtual tables utilize data stored in existing tables in the database to focus on specific types of information or to hide data that you do not want users to see. Also, you can use virtual tables to gain a comprehensive customized view of existing data in your database. However, you need to create views from multiple tables to do this.

**Reference sheet: Functions in MySQL**

**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.

The CEIL function returns the smallest integer value, which is *not less than* the passed value. For example, passing a value of 1.23 to the CEIL function returns a value of 2.

It returns NULL if the value passed is NULL. The syntax of the CEIL function is as follows:

1

SELECT CEIL(VALUE);

**Example**: The following code returns 16 as it is the smallest integer value that is greater than or equal to 15.50.

1

SELECT CEIL(15.50);

The FLOOR function does the opposite of CEIL. It returns the largest integer value not greater than the passed value. It returns NULL if the passed value is NULL.

Its syntax is the same as the CEIL function. It's just a matter of replacing the CEIL function with the FLOOR function.

1

SELECT FLOOR(VALUE);

**Example**: The following code returns 15 as it is the largest integer value that is less than or equal to 15.50.

1

SELECT FLOOR(15.50);

**String functions**

Now let's look at some more String functions like FORMAT, LENGTH and REPLACE.

The FORMAT function formats the number passed into a format like '#,###,###.##', rounded to the specified number of decimal places. It returns the result as a string.

Here's the syntax:

1

SELECT FORMAT(number\_to\_be\_formatted, number\_of\_decimal\_places);

**Example**: The following code returns 3,750.75 as it formats the number (3750.753, 2) as "#,###.##" and rounds it within two decimal places.

1

FORMAT(3750.753, 2)

If the number of decimal places is 0, the result has no decimal point or fractional part. If the number to be formatted or the number of decimal places is NULL, then the function returns NULL.

**Date functions**

You’ve explored MySQL date functions such as CURRENT\_DATE, CURRENT\_TIME, DATE\_FORMAT and DATEDIFF. Now let’s explore a few more essential date functions with some examples.

The ADDDATE function is used to perform arithmetic with dates. It comes in two forms.

1. **ADDDATE**(date, INTERVAL expr unit)
2. **ADDDATE**(date, days)

The first argument specifies the starting date or datetime value in the first form. The second argument is an expression that determines the interval value to be added to the starting date.

It has three parts:

1. **INTERVAL**is a keyword
2. **expr**represents a quantity
3. and **unit**represents the unit for interpreting the quantity; such as HOUR, DAY, or WEEK

The syntax for the first form is as follows:

1

SELECT ADDDATE(date, INTERVAL expr unit);

In the second form, the first argument is the same and the second argument is the integer, or number of days, to be added to the given date in the first argument.

**Example**: the following code returns 2020-05-15 because it adds 5 days to the specified date.

1

2

SELECT ADDDATE("2020-05-10", INTERVAL 5 DAY);

The syntax for the second form is as follows:

1

SELECT ADDDATE(date, days);

**Example**: The following code returns 2020-06-25 because it adds 5 days to the specified date.

1

SELECT ADDDATE("2020-06-15", 10);

The QUARTER function is also a very versatile function that returns the quarter of the year for the given date. The return value is in the range 1 to 4, or NULL if date is NULL.

The syntax is as follows:

1

SELECT QUARTER(date\_value);

The DATE, MONTH, YEAR and DAY functions extract the date, month, year and 'day of the month' parts respectively, of a given date or date time expression.

**Example:** The following code returns a value of 3. The QUARTER() function returns the quarter of the year for the given date value.

1

 SELECT QUARTER("2020-09-15");

**Comparison functions**

You may know of a few comparison functions of MySQL like GREATEST, LEAST and ISNULL.

COALESCE is another comparison function that takes several arguments and returns the first non-NULL argument. In case all arguments are NULL, the COALESCE function returns NULL. You can think of this function as a NULL checking function.

The syntax for this function is as follows:

1

SELECT COALESCE (value1, value2);

This function is particularly useful when you replace a column value with a NULL value with some other value.

**Example**: The following code returns Coursera, because the COALESCE() function returns the first non-null value in a list.

1

SELECT COALESCE(NULL, 'Coursera', NULL, 'Database');

**Control flow functions**

MySQL also has a function that lets you evaluate conditions and decide how the query should be executed accordingly. You should be familiar with the CASE function in MySQL and IFNULL, another function. It accepts two arguments and returns the first argument if it is not NULL. Otherwise, the IFNULL function returns the second argument. The two arguments can be literal values or expressions.

The syntax is as follows:

1

SELECT IFNULL(evaluated\_expression, alternative\_value);

**Example**: The following code returns Coursera as the value because the evaluated expression is NULL.

1

SELECT IFNULL(NULL, "Coursera");

There’s also the NULLIF function that takes in two values or expressions. If they're equal then it returns NULL. Otherwise, it returns the first value or expression.

This is an example of the syntax:

1

SELECT NULLIF(expression1, expression2);

**Example**: The following NULLIF() function returns 10, as it compares the two expressions (10 and 15) and returns NULL if they are equal. Otherwise, the first expression is returned.

1

SELECT NULLIF(10, 15);

**MySQL stored procedures in depth**

In this reading, you explore stored procedures at a more detailed level by learning when and how to use parameters.

**What is a stored procedure?**

The stored procedure in MySQL is a set of SQL instructions wrapped within the CREATE PROCEDURE statement to achieve a particular objective.

For example:

1

CREATE PROCEDURE GetItalianCustomers() SELECT \* FROM Customers WHERE Country = “Italy”;

This stored procedure returns all customers from Italy by just calling the identifier of the procedure as follows:

1

CALL GetItalianCustomers();

**Why use stored procedures?**

The main idea behind creating stored procedures is to create reusable code that can be invoked and executed efficiently.

So, instead of typing the same code repeatedly, you can save your blocks of code in a stored procedure as a prepared query that you can use whenever needed.

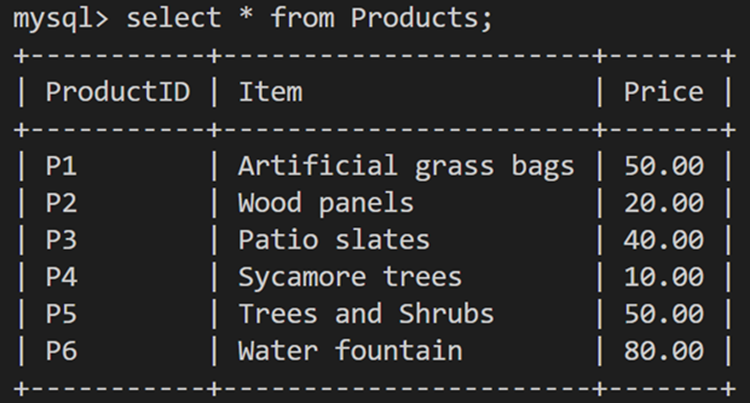
You can create as many procedures as you need. Each procedure can have a unique name, zero or several parameters and contain blocks of code, including SQL statements, variables and conditions.

Stored procedures make the code more consistent, reusable and easier to use and maintain.

**When to create a stored procedure with parameters?**

Each procedure must achieve a clear objective. For example, suppose you expect users to input a value when calling a procedure. In that case, you need to specify a parameter with a suitable data type. Otherwise, the procedure will be created with a parameter of zero.

For example, the Products table from the Lucky Shrub database includes 3 columns: product id, item name and product price. The table is populated with relevant data, as shown below.



You can create a basic stored procedure with empty (or zero) parameters and one single SQL statement by typing the following query.

1

CREATE PROCEDURE GetProductsDetails() SELECT \* FROM Products;

This procedure returns all records from the Products table. No argument (user input value) is expected to be passed to the procedure to execute it. That is why it has no parameter within the parentheses.

Other times, you may want to create a stored procedure with one or more parameters to process the query based on a user-specified value. For example, you can create a stored procedure to retrieve data from the products table based on the user input of a specific price. To do this, you can write a procedure with one parameter as follows:

1

2

CREATE PROCEDURE GetProductsBasedOnPrice (inputPrice INT)

SELECT \* FROM Products WHERE Price <= inputPrice;

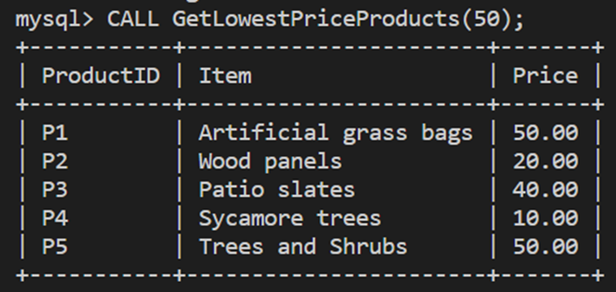
In this statement, you declare a parameter with an INT datatype to pass an integer value into the stored procedure to make the query act based on this value.

In this case, you can return all data from the Products table where the product price is less than or equal to the user input value. Here's an example of the code:

1

CALL GetProductsBasedOnPrice (50);

This lists all products with prices less than or equal to $50 as shown on the screen below.



**Conclusion**

Stored procedures are SQL code instructions encapsulated in a CREATE PROCEDURE statement to achieve a particular objective. You execute procedures by calling its name and passing the parameters (if there are any!). It is easy to maintain, can be reused as often as needed, is consistent, and even helps secure your database by preventing SQL injections.