Co-funded by the
Erasmus+ Programme
of the European Union

Co-funded by the
Erasmus+ Programme
of the European Union

3.1.:

Code4SP Training

Material Package

WP3:

Code4SP Training Materials –

SQL

Prepared by:

Project Information

Project Acronym: Code4SP

Project Title: Coding for Social Promotion

Project Reference: 621417-EPP-1-2020-1-PT-EPPKA3-IPI-SOC-IN

Project website: www.code4sp.eu

Authoring Partner: C.I.P. Citizens in Power

Document Version: 2

Date of Preparation: 11/03/2022

Document History			
Date	Version	Author	Description
11/03/2022	1	CIP	Draft
22/03/2022	2	CEPROF	Revision

Table of Contents

Topic Information

Topic:

4. SQL

Prerequisites:

Basic computer literacy, basic software installed, and basic knowledge of working with files.

Workload:

10 hours

Description:

In this topic, we cover the basics of SQL to get learners acquainted with the programming world and encourage them to gain more expertise on SQL. We explain the attributes, syntax, and other relevant terms that learners may have heard or be familiar with and how these fit into the programming language. We provide a detailed account of the logic and syntax used in SQL, its structure and other basic and essential functions.

Learning outcomes:

- Recognise the concept and usage of Structured Query Language (SQL) in Relational Database Management Systems (RDBMS)
- Formulate basic syntax of SQL queries
- Use SQL to access, manage, and manipulate RDBMS

Material required:

- Computer or laptop
- Internet connection

- Online SQL compiler (https://www.mycompiler.io/new/sql)
- Online text editor (https://www.w3schools.com/sql/default.asp)

Lesson Scenario:

The total time dedicated to this topic is 10 hours, and it is up to the judgement of the trainer/coach to decide how much time will be spent on each subtopic. We suggest using the PPT training presentations explicitly created for this topic to ease the teaching process and increase time efficiency. These presentations encompass the following:

- Progressive development of subtopics and core concepts to retain, and
- Recommended Exercises.

Depending on the trainer's/coach's preferences, the progressive development of the presentations allows the completion of the SQL session within the stipulated time, i.e., 10 hours. The presentations can also be made available to learners for self-studying.

Subtopics:

- SQL Basics
- SQL Databases
- SQL References
- SQL Examples

Additional resources:

- W3Schools - Guide for every SQL keyword and function, and examples for each of them
- Tutorialspoint – Another detailed guide on SQL keywords and functions, and various examples for each them

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

SQL Basics

What is SQL?

SQL stands for Structured Query Language. SQL is a standard programming language specifically designed for storing, retrieving, managing or manipulating data found within a Relational Database Management System (RDBMS). A relational database is a collection of data items with pre-defined relationships between them. These items are organised as a set of tables with columns and rows.

SQL became an ISO standard in 1987.

SQL is the most widely-implemented database language supported by popular relational database systems, like MySQL, SQL Server, and Oracle. SQL was initially developed at IBM in the early 1970s. Originally, it was called SEQUEL (Structured English Query Language) and it was later changed to SQL (pronounced as S-Q-L).

Applications of SQL

SQL is one of the most widely used query languages for databases. Some of its many applications are:

- Allowing users to access data in the relational database management systems,
- Allowing users to describe the data,
- Allowing users to define the data in a database and manipulate that data.

SQL Syntax

SQL statements are straightforward, like plain English, but with a specific syntax.

An SQL statement is composed of a sequence of keywords, identifiers etc., terminated by a semicolon (;).

Example:

```
SELECT emp_name, hire_date, salary FROM employees WHERE salary > 5000;
```

For better readability, you can also write the same statement, as follows:

```
SELECT emp_name, hire_date, salary
FROM employees
WHERE salary > 5000;
```

Use semicolon at the end of an SQL statement — it terminates the statement or submits the information to the database server.

Case Sensitivity in SQL

Consider another SQL statement that retrieves records from the Employees table:

```
SELECT emp_name, hire_date, salary FROM employees;
```

The same statement can also be written as follows:

```
select emp_name, hire_date, salary from employees;
```

SQL keywords are case-insensitive, which means SELECT is the same as select.

However, the database and table names may be case-sensitive depending on the operating system. In general, Unix or Linux platforms are case-sensitive, whereas Windows platforms aren't.

SQL Select

The SELECT statement selects or retrieves data from one or more tables. You can use this statement to retrieve all the rows from a table in one go or retrieve only those rows that satisfy a specific condition or a combination of conditions.

Suppose we have a table named Employees in our database that contains the following records:

```
| emp_id| emp_name| hire_date| salary| dept_id|

+--------+--------------+------------+--------+---------+

|      1 | Ethan Hunt   | 2001-05-01 |   5000 |       4 |

|      2 | Tony Montana | 2002-07-15 |   6500 |      1 |

|      3 | Sarah Connor | 2005-10-18 |   8000 |       5 |

|      4 | Rick Deckard | 2007-01-03 |   7200 |       3 |

|      5 | Martin Blank | 2008-06-24 |   5600 |    NULL |

+--------+--------------+------------+--------+---------+
```

Select All from Table

The following statement will return all the rows from the employees' table.

```
>> SELECT * FROM employees;
```

Select Specific Columns from Table

If you don't require all the data, you can select specific columns, like this:

```
SELECT emp_id, emp_name, hire_date, salary
FROM employees;
```

After executing the above statement, you will get an output like this:

```
| emp_id| emp_name| hire_date| salary|

+--------+--------------+------------+--------+

|      1 | Ethan Hunt   | 1995-10-30 |   5000 |

|      2 | Tony Montana | 1990-07-15 |   6500 |

|      3 | Sarah Connor | 2011-04-13 |   5600 |

|      4 | Rick Deckard | 2005-10-18 |   7200 |

|      5 | Martin Blank | 1996-05-24 |   8000 |

+--------+--------------+------------+--------+
```

SQL Select Distinct

Commented [1]: mipos ennoei auto pou egrapsa pio pano to na epilegoum sigkekrimena columns?

Commented [2]: I diafora tou distinct apo o sketo select in oti to distinct den sou dinei ta duplicated stoixeia.

Commented [3]: aaaaa thank you!

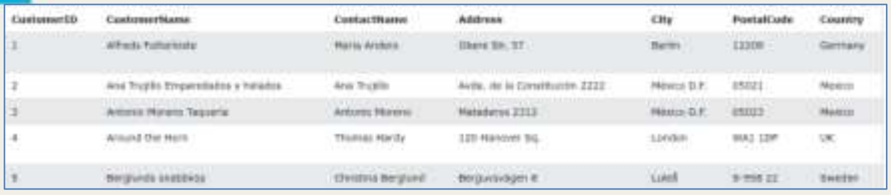
The SELECT DISTINCT statement omits duplicated values when used in a query.

You can find duplicated values inside a table, but sometimes you want to see the “unique” values.

Syntax:

```
SELECT DISTINCT column1, column2, ...
FROM table_name;
```

In the following examples, we will be using the Customers table that contains data about our customers.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Table 1 *- Customers table in SELECT DISTINCT example* ***(Source****:* https://www.w3schools.com/sql/sql_distinct.asp*)*

To *select all values* from the Country column in the Customers table, you would use the following statement:
SELECT Country FROM Customers;

However, this statement will include duplicate values.

If you want to omit the duplicate values from your query, use SELECT DISTINCT:
SELECT DISTINCT Country FROM Customers;

Suppose you wanted to list the number of different customer countries; you would use the following statement:
SELECT COUNT(DISTINCT Country) FROM Customers;

Please note that this example will not work in Firefox since COUNT(DISTINCT column_name) is not supported in MS Access.

To get the equivalent result in Access, use this:

```
>> SELECT Count(*) AS DistinctCountries
   FROM (SELECT DISTINCT Country FROM Customers);
```

SQL Where

Previously, we have learnt how to fetch all the records from a table or columns from a table.

However, in real-world cases, we generally need to select, update or delete only those records which fulfil certain conditions, like users who belong to a particular age group, country, etc.

The WHERE clause is used with the SELECT, UPDATE, and DELETE.

The WHERE clause is used with the SELECT statement to extract only those records that fulfil specified conditions.

The basic syntax is as follows:

```
SELECT column_list
FROM table_name
WHERE condition;
```

Now, let's check out some examples that demonstrate how it works.

Suppose we have a table called Employees in our database with the following records:

```
| emp_id| emp_name| hire_date| salary| dept_id|
+--------+--------------+------------+--------+---------+
|      1 | Ethan Hunt   | 2001-05-01 |   5000 |       4 |
|      2 | Tony Montana | 2002-07-15 |   6500 |       1 |
|      3 | Sarah Connor | 2005-10-18 |   8000 |       5 |
|      4 | Rick Deckard | 2007-01-03 |   7200 |       3 |
+--------+--------------+------------+--------+---------+
```

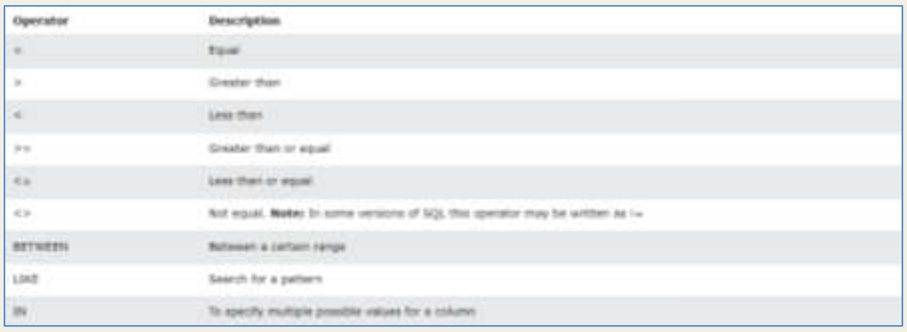
The following SQL statement will return all employees from the Employees' table *whose salary is greater than 7000*:

```
SELECT * FROM employees WHERE salary > 7000;
```

The WHERE clause simply filters out the unwanted data.

Another example would be to *select all* employees with *department id =1*:
SELECT * FROM employees WHERE dept_id=1;

The following table provides the list of operators that can be used with the WHERE clause:



Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Table 2 -*Table of Operators used in WHERE clause (***Source***: https://www.w3schools.com/sql/sql_where.asp)*

SQL And, Or, Not

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition.

The AND operator displays a record if all the conditions that use AND are TRUE.
AND Syntax:
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;

Example: Select all fields from Customers table where country is “Germany” AND City is “Berlin”.

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

The OR operator displays a record if any of the conditions that use OR are TRUE.

OR Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example 1: Select all fields from Customers table where city is “Berlin” or “München”

```
SELECT * FROM Customers.
WHERE City='Berlin' OR City='München';
```

Example 2: Select all fields from Customers table where country is “Germany” or “Spain”.

```
SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';
```

The NOT operator displays a record if the condition(s) is NOT TRUE.

NOT Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Example: Select all fields from Customers table where country is NOT “Germany”.

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

Combining AND, OR and NOT

Example 1: Select all rows from the Customers table where country is Germany and city must be either Berlin or München.

```
SELECT * FROM Customers
WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

Example 2: Select all rows from the Customers table where country is NOT Germany and NOT USA.

```
SELECT * FROM Customers
WHERE NOT Country='Germany' AND NOT Country='USA';
```

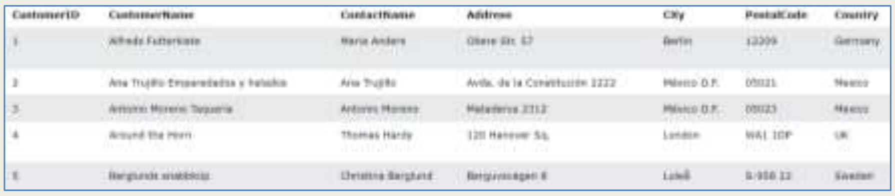
SQL Order By

The ORDER BY keyword sorts the result-set in ascending or descending order.The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

In the following examples, we will use the Customers table shown below:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Table 3 - *Customers Table in ORDER BY example* ***(Source:*** *https://www.w3schools.com/sql/sql_orderby.asp)*

Example 1: Selects all customers from the Customers table and sorts them by the Country column

```
SELECT * FROM Customers
ORDER BY Country;
```

Example 2: Selects all customers from the same table and sorts them in descending order by the Country column

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

Example 3: Selects all customers from the same table and sorts them by Country and Customer Name.

```
SELECT * FROM Customers
ORDER BY Country, CustomerName;
```

Here, the order is initially sorted by Country. However, if there are some rows that have the same country, then they are sorted by Customer Name.

Example 4: Selects all customers from the same table and sorts them in ascending order by Country and descending order by Customer Name

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

SQL Insert Into

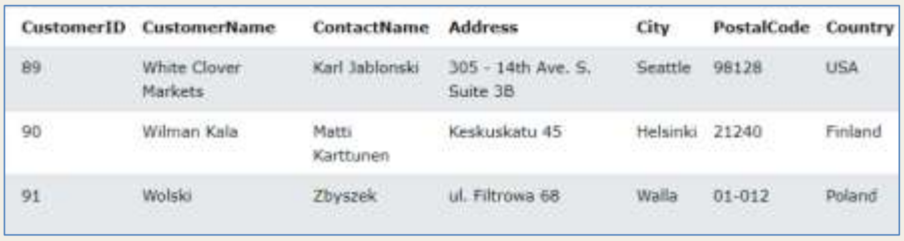
The INSERT INTO statement inserts new records in a table.

For your code to run correctly, specify both the column names and the values that will be inserted.

Syntax:

```
INSERT INTO table_name (column1, column2, column3, ...)
```

VALUES (value1, value2, value3, ...);



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

Table 5 - *INSERT INTO Example* ***(Source:****: https://www.w3schools.com/sql/sql_insert.asp)*

For example, to add a new record in your “Customers” table, use the following:

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

SQL Null Values

A field with a NULL value is a field with no value. If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value has been **left blank** during record creation!

Testing for NULL Values

It is impossible to test for NULL values with comparison operators, such as =, <, or <>.

Instead, we will have to use the **IS NULL** and **IS NOT NULL** operators.

IS NULL Syntax:

SELECT column_names
FROM table_name

WHERE column_name IS NULL;

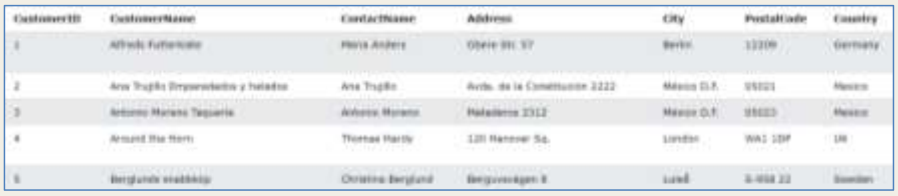
IS NOT NULL Syntax:

SELECT column_names

FROM table_name

WHERE column_name IS NOT NULL;

The following examples use the table below:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

***Table 6 -** Customers Table in NULL Values Example **(Source:** https://www.w3schools.com/sql/sql_null_values.asp)*

Example of IS NULL

Selects all customers with Null values (i.e. empty values) in the Address column:

SELECT CustomerName, ContactName, Address

FROM Customers

WHERE Address IS NULL;

To look for Null values, always use IS NULL.

Example of IS NULL

Selects all customers with NOT Null values (i.e. non-empty values) in the Address column:

SELECT CustomerName, ContactName, Address

FROM Customers

WHERE Address IS NOT NULL;

Co-funded by the Erasmus+ Programme of the European Union

SQL Update

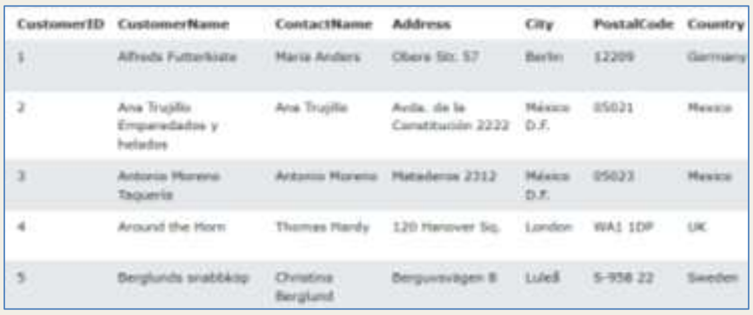
The UPDATE statement modifies the existing records of a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) should be updated.

If you **omit** the WHERE clause, all records in the table will be updated!

Example



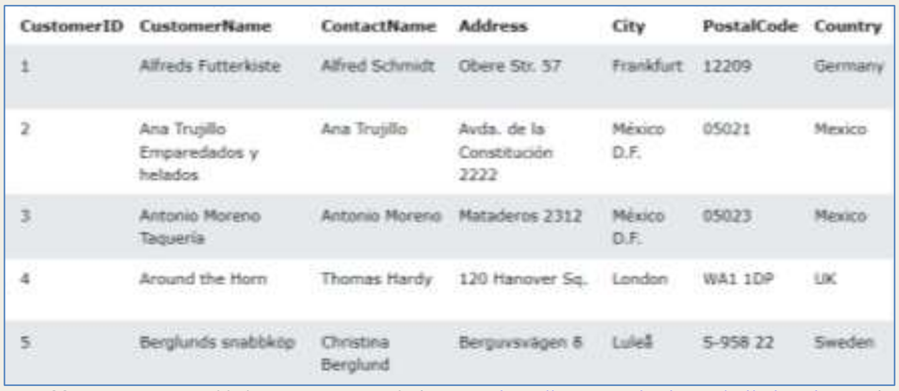
CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

***Table 7 -** Customers table in UPDATE example **(Source:** https://www.w3schools.com/sql/sql_update.asp)*

To update CustomerID=1 from the "Customers" table in the sample database, use the following:

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

And the “Customers” table will now look like this:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

***Table 8** - Customers table in UPDATE example **(Source:** https://www.w3schools.com/sql/sql_update.asp)*

The WHERE clause determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where the country is "Mexico" in the Customers table:

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

SQL Delete

The DELETE statement deletes existing records in a table.

```
DELETE FROM table_name WHERE condition;
```

Keep in mind that you need to be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted.

If you **omit** the WHERE clause, all records in the table will be deleted!

Example:

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

Delete All Records

It is possible to delete all rows in a table without deleting the table itself. This means that the table’s structure, attributes, and indexes will stay intact:

DELETE FROM table_name;

SQL Select Top

The SELECT TOP clause is used to specify the number of records that will be returned.

The SELECT TOP clause is useful on large tables with thousands of records. However, returning a large number of records can impact performance.

Note that not all database systems support the SELECT TOP clause. MySQL supports the LIMIT clause to select a limited number of records, while Oracle uses FETCH FIRST n ROWS ONLY and ROWNUM.

SQL Server/MS Access Syntax:

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

MySQL Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
```

```
LIMIT number;
```

Oracle 12 Syntax:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s)
FETCH FIRST number ROWS ONLY;
```

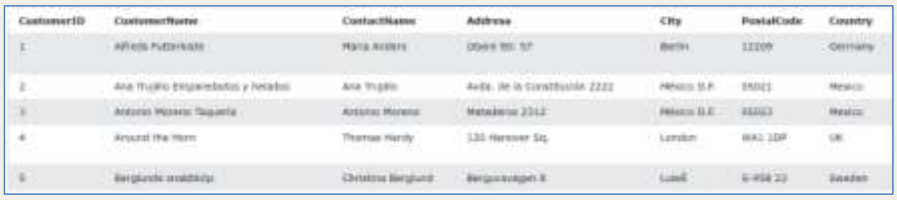
Older Oracle Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number;
```

Older Oracle Syntax with ORDER BY:

```
SELECT *
FROM (SELECT column_name(s) FROM table_name ORDER BY column_name(s))
WHERE ROWNUM <= number;
```

We will use the “Customers” table shown below in the following examples.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Table 9 - *Customers table in SELECT TOP examples* ***(Source:*** *https://www.w3schools.com/sql/sql_top.asp)*

Examples of TOP, LIMIT and FETCH FIRST

To select the first three records from the Customers table in SQL Server/MS Access, use the following:

```
SELECT TOP 3 * FROM Customers;
```

To execute a query with the same result as above in MySQL, use the following statement:

```
SELECT * FROM Customers
LIMIT 3;
```

To execute a query with the same result as the two above in Oracle:

```
SELECT * FROM Customers
FETCH FIRST 3 ROWS ONLY;
```

Examples of TOP PERCENT

To select the first 50% of records found in the Customers table, execute the following statement in SQL Server/MS Access:

```
SELECT TOP 50 PERCENT * FROM Customers;
```

Its equivalent in Oracle is the following:

```
SELECT * FROM Customers
FETCH FIRST 50 PERCENT ROWS ONLY;
```

Examples of ADD a WHERE Clause

In the following example, we are selecting the first three records from the Customers table, where the Country is “Germany” for SQL Server/MS Access:

```
>> SELECT TOP 3 * FROM Customers
   WHERE Country='Germany';
```

Its equivalent in MySQL:

```
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

Its equivalent in Oracle:

```
SELECT * FROM Customers
WHERE Country='Germany'
FETCH FIRST 3 ROWS ONLY;
```

SQL Min and Max

The MIN() function returns the smallest values from selected columns.

MIN() Syntax

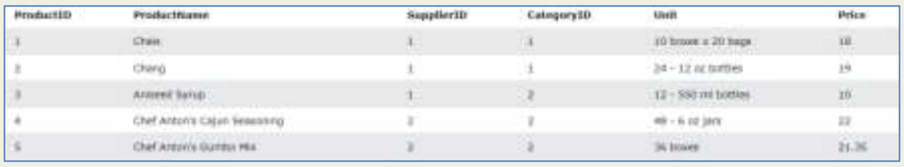
```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

The MAX() function returns the largest value from selected columns.

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

In the following examples, we will use the Products table shown below:



ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

Table 10 - *Products Table in Min and Max Examples* ***(Source:*** https://www.w3schools.com/sql/sql_min_max.asp*)*

Example 1: Finds the price of the cheapest product

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

Example 2: Finds the price of the most expensive product

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

SQL Count, Avg, Sum

The COUNT() function returns the number of rows that match a specified criterion.

COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

The AVG() function returns the average value of a numeric column.

AVG() Syntax

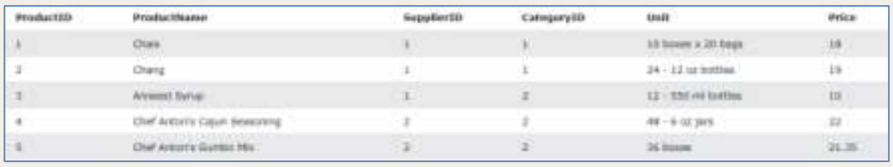
```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

The SUM() function returns the total sum of a numeric column.

SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

We will use the Products table in the following examples below.



ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

Table 11 - *Products table in Count, Avg, Sum Examples* ***(Source:*** *https://www.w3schools.com/sql/sql_count_avg_sum.asp)*

Example of COUNT()

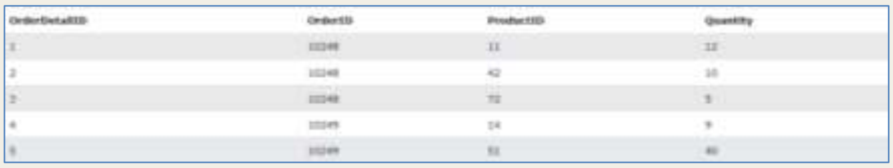
Execute a query to find the number of products:

```
SELECT COUNT(ProductID)
FROM Products;
```

Example of AVG()

Execute a query to find the average price of all products:

```
SELECT AVG(Price)
FROM Products;
```



OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

Table 12 *- OrdersDetails table in Count, Avg, Sum Examples* ***(Source:*** https://www.w3schools.com/sql/sql_count_avg_sum.asp*)*

In the following example, we will use the OrdersDetails table, shown above, to find the sum of “Quantity”:

```
SELECT SUM(Quantity)
FROM OrderDetails;
```

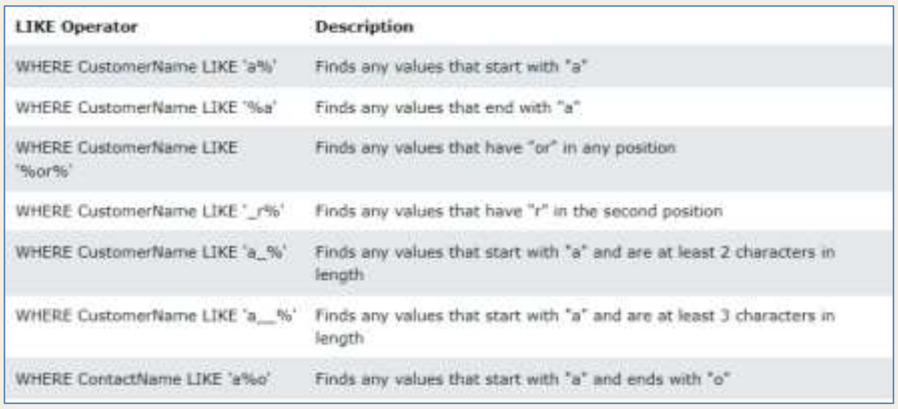
SQL Like

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE column LIKE pattern;
```

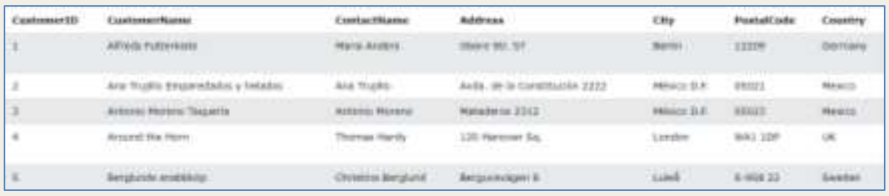
Here are some examples showing different LIKE operators with '%' and '_' wildcards:



LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Table 13 *- LIKE operators (**Source**: https://www.w3schools.com/sql/sql_like.asp)*

We will see some examples that will be using the Customers table shown below.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Table 14 *- Customers table in LIKE examples (**Source:** https://www.w3schools.com/sql/sql_like.asp)*

Example 1: Selects all customers with a Customer Name that starts with ‘a’.

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Example 2: Selects all Customer names ending with ‘a’.

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

Example 3: Selects all Customer names that contain ‘or’ in their name in any position.

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

Example 4: Selects all Customer names that contain ‘r’ in the second position.

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

Example 5: Selects all Customer names that begin with ‘a’ and have at least three characters in length.

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

Example 6: Selects all Customer names that begin with ‘a’ and end with ‘o’.

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%o';
```

Example 7: Selects all Customer names that do not begin with ‘a’.

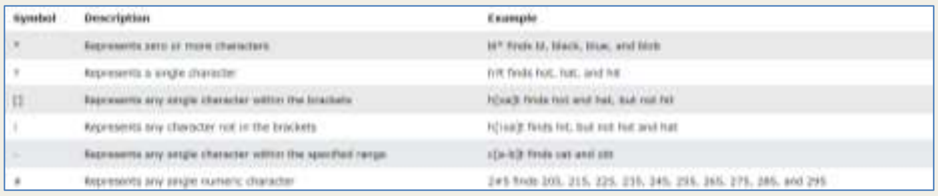
```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

SQL Wildcards

A wildcard character substitutes one or more characters in a string. It is used with the LIKE operator.

The LIKE operator is also used in a WHERE clause to search for a specified pattern in a column, as we have seen in the previous subsection.

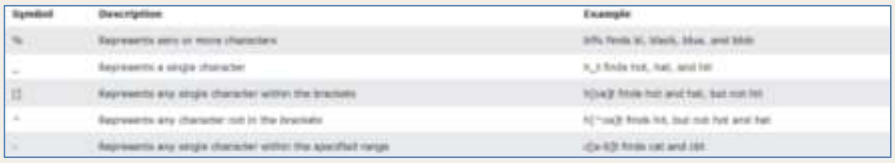
Wildcards in MS Access



Symbol	Description	Example
*	Represents zero or more characters	bl* finds bl, black, blue, and blob
?	Represents a single character	h?t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
!	Represents any character not in the brackets	h[!oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt
#	Represents any single numeric character	2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295

__Table 15__ – Wildcards in Access (__Source:__ https://www.w3schools.com/sql/sql_wildcards.asp)

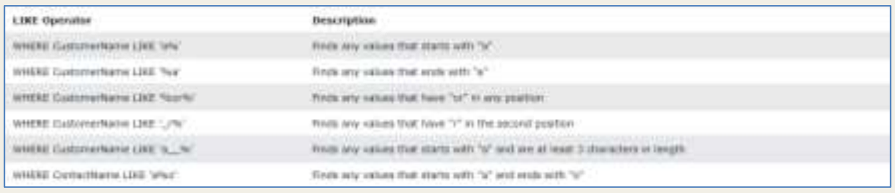
Wildcards in SQL Server



Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt

__Table 16__ – Wildcards in SQL Server (__Source__: https://www.w3schools.com/sql/sql_wildcards.asp)

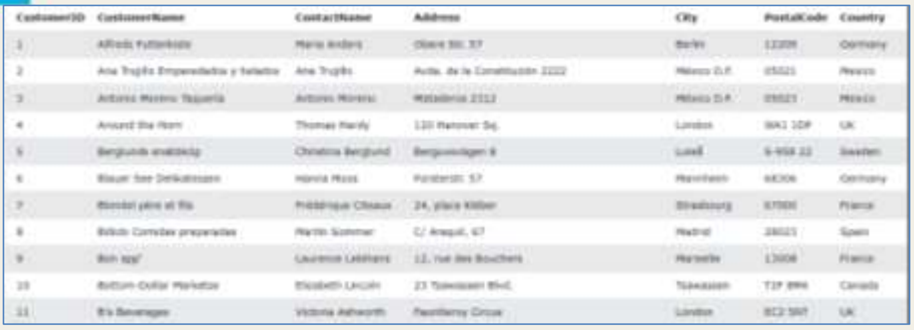
Wildcards can be used in combination; look at some examples at the table below:



LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

__Table 17__ - Examples of Wildcards with % and '_' (__Source__: https://www.w3schools.com/sql/sql_wildcards.asp)

Let's see some examples with the Customers table.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas preparadas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
10	Bottom-Dollar Marketse	Elizabeth Lincoln	23 Tsawassen Blvd.	Tsawassen	T2F 8M4	Canada
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK

Table 18 - *Customers table in WILDCARDS example (**Source**: https://www.w3schools.com/sql/sql_wildcards.asp)*

Examples with the % Wildcard

In this example, we are selecting all customers with a City starting with “ber”:

SELECT * FROM Customers

WHERE City LIKE 'ber%';

In the following example, we are selecting all customers with a City containing “es”:

SELECT * FROM Customers

WHERE City LIKE '%es%';

Examples with the _ Wildcard

Here, we are selecting all customers with a City starting with any character followed by “ondon”:

SELECT * FROM Customers

WHERE City LIKE '_ondon';

In this example, we are again selecting all customers with a City starting with ‘L’, followed by any character, followed by "n", followed by any character, followed by "on":

SELECT * FROM Customers

WHERE City LIKE 'L_n_on';

Examples with the [charlist] Wildcard

Here, we are selecting all customers with a City starting with ‘b’,’ s’, or ‘p’:

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

In the following example, we are selecting all customers with City starting with ‘a’, ‘b’, or ‘c’:

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

Examples with the [!charlist] Wildcard

The exclamation mark shows characters that do not contain a specified string. For example, we want to select all customers with a City that does not start with ‘b’, ‘s’, or ‘p’:

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

As an alternative, we can use the following:

```
SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

SQL In

The IN operator is used to specify multiple values in a WHERE clause. It can be considered as satisfying various conditions.

Syntax 1:

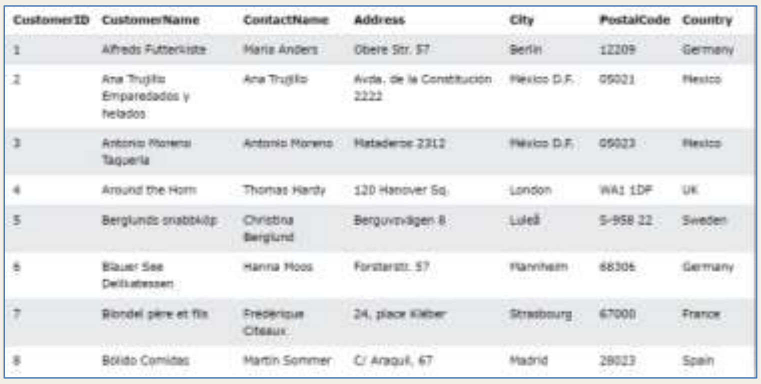
```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

Syntax 2:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

There are two ways to use the IN operator, as you have seen.

Suppose that we have a table called “Customers” containing the following columns: CustomerID, CustomerName, ContactName, Address, City, PostalCode and Country.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
8	Bólido Comidas	Martín Sommer	C/ Araquil, 67	Madrid	28023	Spain

***Table 19** - Customers Table in IN operator example (**Source:** https://www.w3schools.com/sql/sql_in.asp)*

As an example, we want to select all the customers that are located in Germany, France or UK:

```
SELECT * FROM Customers
WHERE Country IN (‘Germany’, ‘France’, ‘UK’)
```

Another example is selecting all the customers that are **not** located in Germany, France or the UK:

```
SELECT * FROM Customers
WHERE Country NOT IN ( ‘Germany’,  ‘France’, ‘UK’)
```

Let's also consider a third example where we want to select customers that are from the same countries as the suppliers:

```
SELECT * FROM Customers
WHERE Country  IN (SELECT Country FROM Suppliers)
```

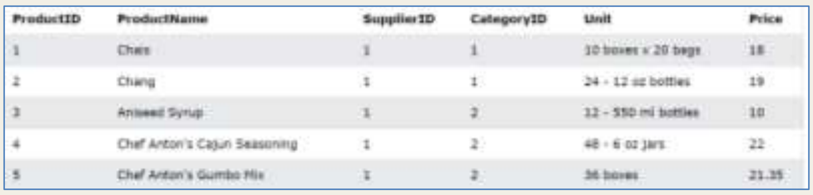
SQL Between

The BETWEEN operator provides a range of values to select from. The values can be text, numbers or dates. The BETWEEN operator includes the starting and end values.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Let's say that we have the following table, which contains information on different products.



ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	1	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	1	2	36 boxes	21.35

***Table 20** - BETWEEN operator example (**Source**: https://www.w3schools.com/sql/sql_between.asp)*

There will be a series of examples with the following operators: BETWEEN, NOT BETWEEN, BETWEEN with IN, BETWEEN and NOT BETWEEN with text values and BETWEEN dates.

BETWEEN Example: Selects all products with a price range of 10 to 20.

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

NOT BETWEEN Example: Shows all the products outside the range that we set in the previous example.

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

BETWEEN with IN Example: Selects all products with a price range of 10 to 20 and does not show products with CategoryID 1, 2, or 3.

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

BETWEEN with text values Example: Selects all products with a ProductName between Carnarvon Tigers and Mozzarella di Giovanni.

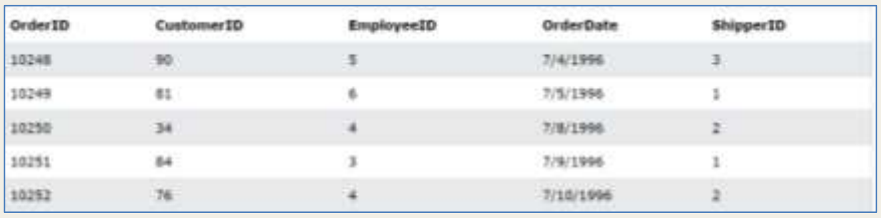
```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;
```

NOT BETWEEN with text values Example: Selects all products with a ProductName NOT between Carnarvon Tigers and Mozzarella di Giovanni.

```
SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di
Giovanni'
ORDER BY ProductName;
```

Co-funded by the
Erasmus+ Programme
of the European Union

Assume that we have the following table that contains information on different Orders:



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	7/4/1996	3
10249	81	6	7/5/1996	1
10250	34	4	7/8/1996	2
10251	84	3	7/9/1996	1
10252	76	4	7/10/1996	2

Table 21 *- BETWEEN operator example (**Source**: https://www.w3schools.com/sql/sql_between.asp)*

BETWEEN Dates Example: Selects all orders with an OrderDate between '01-July-1996' and '31-July-1996'.

There are two ways that this can be done, by either using a hashtag (#) or quote marks (''):

SELECT * FROM Orders
WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
OR
SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';

SQL Aliases

Aliases assign a temporary name to a table or a column within a table. An alias exists only for the duration of a query, and it is often used to make column names more readable. An alias is created by using the keyword **AS**.

Syntax for column alias:
SELECT column_name AS alias_name
FROM table_name;

Syntax for table alias:

SELECT column_name(s)

FROM table_name AS alias_name;

Column Aliases

Let's see an example that creates two aliases, one for each column:

SELECT CustomerID AS ID, CustomerName AS Customer

FROM Customers;

Another example creates two aliases again:

SELECT CustomerName AS Customer, ContactName AS [Contact Person]

FROM Customers;

Note that it is put in square brackets ([ ]) because the alias contains spaces. Quotation marks can be used as an alternative to square brackets.

You also have the option of creating an alias that contains one or more columns, let's look at the example below to see how it works:

SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS Address

FROM Customers;

The above statement changes a little bit in MySQL:

SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Address

FROM Customers;

Table Aliases

The following example selects all the orders from the customer table with CustomerID=4 (Around the Horn).

Here aliases are used to shorten the query:

SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;

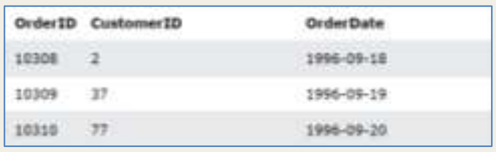
A query without aliases would look something like this:

SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.CustomerID;

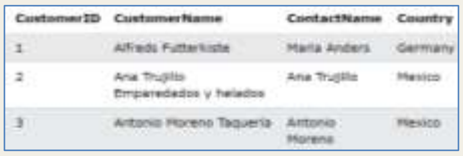
SQL Joins

A JOIN clause combines rows from two or more tables based on a related column found in both tables.

Let’s look at the Orders table and the Customers table:



OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20



CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Tables 22 & 23 *- Orders and Customers Tables in JOIN example (**Source:** https://www.w3schools.com/sql/sql_join.asp)*

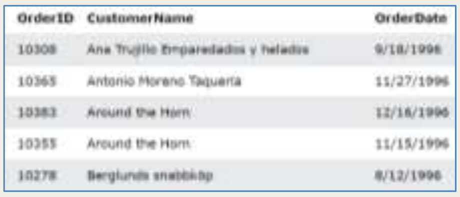
If you look at the two tables, you will notice a common column called the CustomerID. Based on the common column, we can create an SQL statement that uses an INNER JOIN, which selects records that have matching values in both tables.

Example:

SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders

INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;

This command will create something like the following table:



OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Table 24 *- JOIN example (**Source**: https://www.w3schools.com/sql/sql_join.asp)*

There are four different joins in SQL:

1. **(INNER) JOIN:** Returns records that have matching values in both tables;
2. **LEFT (OUTER) JOIN:** Returns all records from the left table and the corresponding matched records from the right table;
3. **RIGHT (OUTER) JOIN:** Returns all records from the right table and the corresponding matched records from the left table;
4. **FULL (OUTER) JOIN:** Returns all records when there is a match in either the left or the right table.

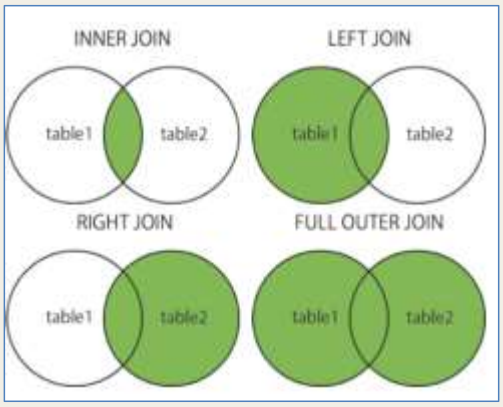


Figure 1 *- Different types of JOINS (**Source**: https://www.w3schools.com/sql/sql_join.asp)*

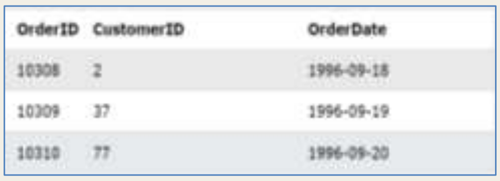
SQL Inner Join

The INNER JOIN keyword selects records that have matching values in both tables.

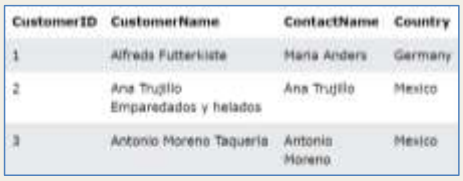
Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

The same tables as the example in the previous subsection are used to perform an inner join.



OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20



CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

***Tables 25 & 26** - Orders and Customers Tables in JOIN example (**Source**: https://www.w3schools.com/sql/sql_join_inner.asp)*

In this example, we want to retrieve the names of customers and their corresponding Order IDs:

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Note that the INNER JOIN keyword will select all rows from both tables that match. If records in the Orders table do not have matches in the Customers table, they will not be selected.

In the following example, we will see how to join three tables that contain customer and shipper information:

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

SQL Left Join

The LEFT JOIN keyword returns all records from the left table and the matching records from the right table. If no matches are found, zero records from the right table will be shown as a result.

Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Note that the LEFT JOIN is called LEFT OUTER JOIN in some databases.

As an example, let's select all customers and any orders that these customers might have:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note that all records from the left table Customers will be returned, even if there are no matches in the right table Orders.

SQL Right Join

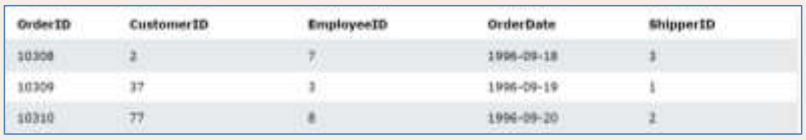
The RIGHT JOIN keyword essentially follows the same logic from the right side instead of the left one as described in the previous subsection.

The RIGHT JOIN will return all records from the right table and the matching records from the left table, if there are any.

Syntax:

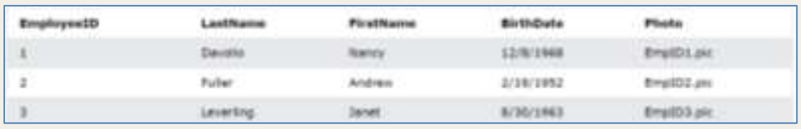
```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Consider the following two tables, Orders and Employees tables:



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

Table 28 *- Orders Table in RIGHT JOIN example (**Source**: https://www.w3schools.com/sql/sql_join_right.asp)*



EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

Table 29 *- Employees Table in RIGHT JOIN example (**Source:** https://www.w3schools.com/sql/sql_join_right.asp)*

The following example will return all employees and any orders that they might have placed:

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

Note that the RIGHT JOIN keyword will return all records from the right table, Employees, even if there are no matches found in the left table, Orders. The same logic applies as in the LEFT JOIN that we saw earlier.

SQL Full Join

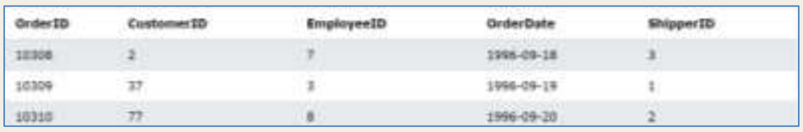
The FULL JOIN keyword returns all records when matching records are found in either the right or the left table.

Note that FULL OUTER JOIN and FULL JOIN are the same thing and a FULL JOIN can potentially return large result-sets.

Syntax:

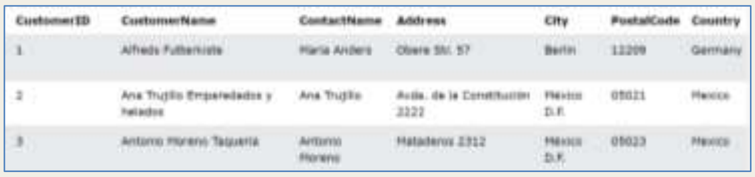
```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Consider the following two tables, the Orders table and the Customers table:



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

***Table 30** - Orders Table in FULL JOIN example (**Source:** https://www.w3schools.com/sql/sql_join_full.asp)*



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

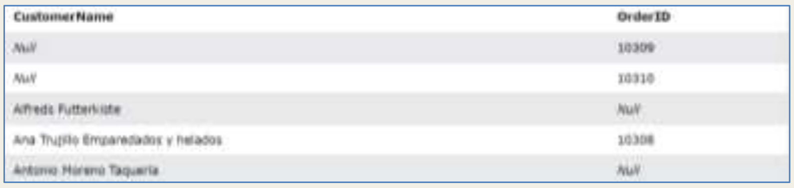
***Table 31** - Customers Table in FULL JOIN example (**Source**: https://www.w3schools.com/sql/sql_join_full.asp)*

An example that selects all customers and orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
```

ORDER BY Customers.CustomerName;

The result of this full join can look something like this:



CustomerName	OrderID
Null	10309
Null	10310
Alfreds Futterkiste	*Null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	*Null*

Table 32 *- FULL JOIN on Customers and Orders Tables Example (**Source**: https://www.w3schools.com/sql/sql_join_full.asp)*

Here we can see that it returns all matching records from both tables even if no common matches are found between the two tables. In the case of no common matches, a null value is assigned.

SQL Self Join

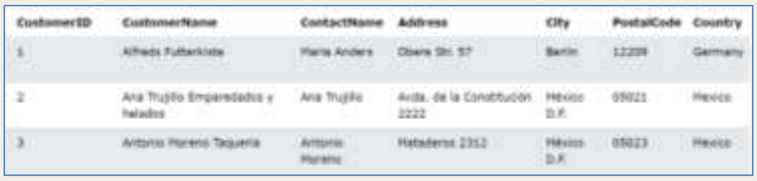
A self-join is considered a regular join, but the table is joined within.

Syntax:

SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;

T1 and T2 are aliases used for the same table.

Let's take the Customers table as an example:



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Table 33 *- Customers Table in SELF JOIN example (**Source**: https://www.w3schools.com/sql/sql_join_self.asp)*

Here, we want to select customers that are from the same city:

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS
CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

SQL Union

The UNION operator is used to combine the result-set of two or more SELECT statements.

There are a few requirements to enable a UNION:

1. Every SELECT statement within the UNION must have the same number of columns;
2. The columns must have similar data types;
3. The columns in every SELECT statement must be in the same order.

Syntax:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

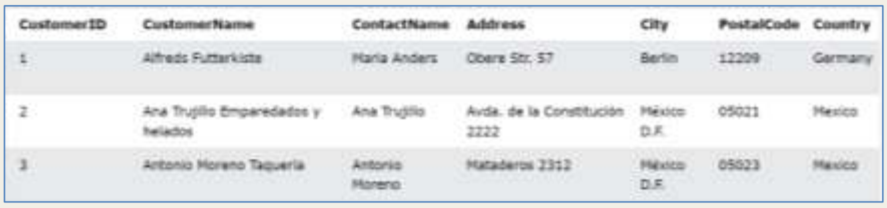
* Note that the UNION operator selects only distinct values by default.

To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

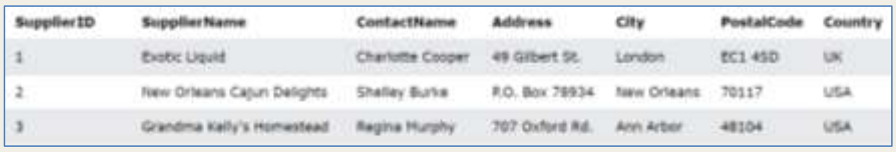
* Note that the column names in the two SELECT statements are usually equal.

Now let's see some examples of UNION, UNION ALL, and UNION with where statements to understand a little bit better how we can use it.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

*Table 34 - Customers Table in UNION example (**Source:** https://www.w3schools.com/sql/sql_union.asp)*



SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

*Table 35 - Suppliers Table in UNION example (**Source:** https://www.w3schools.com/sql/sql_union.asp)*

The first example is used to return distinct cities from both tables shown above:

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

Since we are using UNION, the suppliers from the same city will only be listed once.
If you want to see the duplicated values, use UNION ALL.
The following example will do precisely that and return any duplicate values from both tables:

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

The following example will return the distinct German cities from both the “Customers” and “Suppliers” tables with the use of WHERE:

```
SELECT City, Country FROM Customers
WHERE Country = ‘Germany’
UNION
SELECT City, Country FROM Customers
WHERE Country = ‘Germany’
ORDER BY City;
```

This example is similar to the previous one, but we will be returning possible duplicate values:

```
SELECT City, Country FROM Customers
WHERE Country = ‘Germany’
UNION ALL
SELECT City, Country FROM Customers
WHERE Country = ‘Germany’
ORDER BY City;
```

Another example will list all customers and suppliers:

```
SELECT 'Customer' AS Type, ContactName, City, Country
FROM Customers
UNION
SELECT 'Supplier', ContactName, City, Country
FROM Suppliers;
```

See that we used AS here to create an alias for the given query that will disappear after completing it.

SQL Group By

The GROUP BY statement groups rows with the same values into summary rows. As an example, consider that you want to find the number of customers in each country.

Also, the GROUP BY statement is often used with aggregate functions such as COUNT(), MAX(), MIN(), SUM(), AVG() to group the result by one or more columns.

Syntax:

SELECT column_name(s)

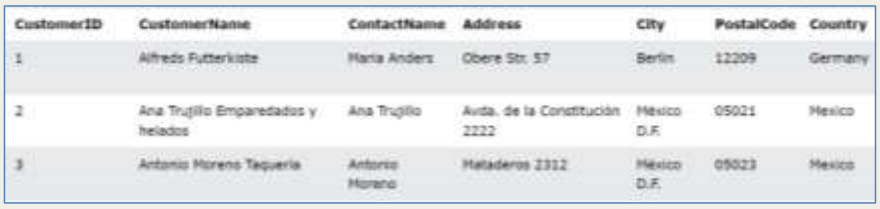
FROM table_name

WHERE condition

GROUP BY column_name(s)

ORDER BY column_name(s);

We will be using the Customers table, shown below, in our examples.



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Table 36** - Customers Table in GROUP BY example (**Source:** https://www.w3schools.com/sql/sql_groupby.asp)*

As a first example, let's list the number of customers found in each county:

SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country;

As a second example, we will again list the number of customers in each country, but in descending order:

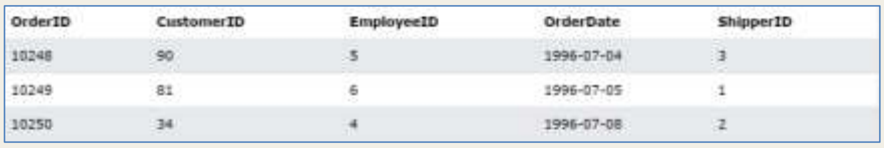
SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country;

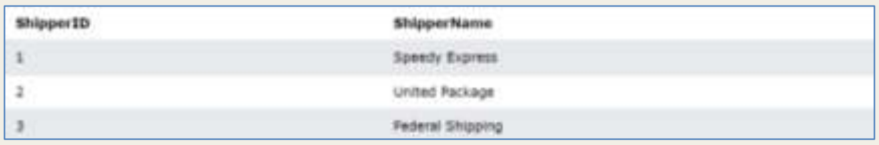
ORDER BY COUNT(CustomerID) DESC;

In the following example, we will use GROUP BY with JOIN by using the Orders and Shippers tables.



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

Table 37 - Orders Table in GROUP BY example (**Source**: https://www.w3schools.com/sql/sql_groupby.asp)



ShipperID	ShipperName
1	Speedy Express
2	United Package
3	Federal Shipping

Table 38 - Shippers Table in GROUP BY example (**Source:** https://www.w3schools.com/sql/sql_groupby.asp)

In this example, we will list the number of orders sent by each shipper:

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

See here we started by selecting the Shippers' names in the Shippers table and counted the orders based on their OrderID saved as an alias.

Then we performed a LEFT JOIN to join the Shippers table (table 2) on the Orders table (table 1) and group them by the Shipper's name.

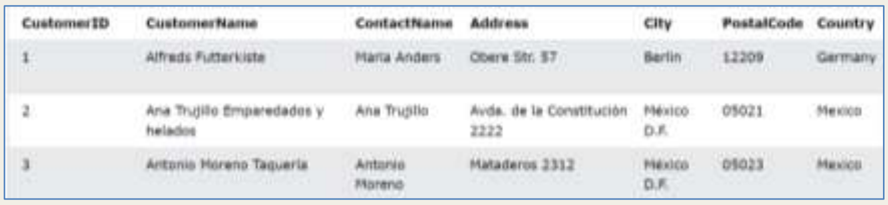
SQL Having

The HAVING clause was added to SQL because WHERE cannot be used with aggregate functions.

Syntax:

```
SELECT column_name(s)
FROM table_name
```

```
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

***Table 39** - Customers Table in HAVING example (**Source:** https://www.w3schools.com/sql/sql_having.asp)*

In this example, we will be using the Customers table once again. Here, we want to list the number of customers found in each country, but we also want to include countries that have more than five customers.

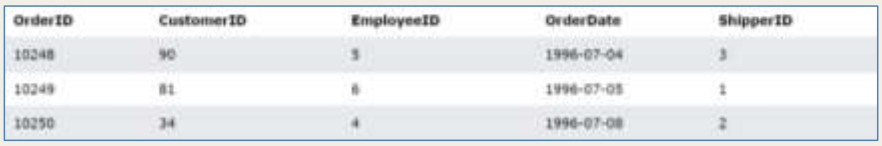
```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

In this example, we want to list the number of customers per country again and include countries with more than five customers in descending order.

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

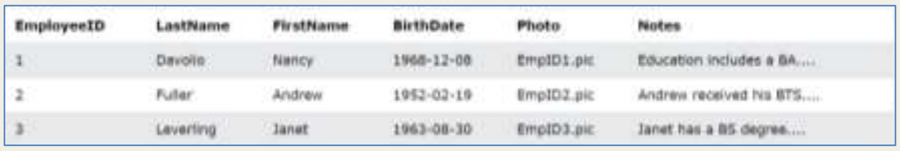
Let’s try a few other examples by combining what we have learnt so far.

We will use the Orders and Employees tables in the following two examples.



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

Table 40 *- Orders Table in HAVING example (**Source:** https://www.w3schools.com/sql/sql_having.asp)*



EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA....
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his BTS....
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree....

Table 41 *- Orders Table in HAVING example (**Source:** https://www.w3schools.com/sql/sql_having.asp)*

The following example will list the employees that have registered more than ten orders:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;
```

In this example, we will list the employees “Davolio” or “Fuller” if they have registered orders more than 25 times:

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
WHERE LastName = ‘Davolio’ OR LastName = ‘Fuller’
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

SQL Select Into

The SELECT INTO statement copies data from one table into a new table.

Syntax to copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Syntax to copy only some columns into a new table:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

The new table will keep the column names and types the same as the old table. You can create new columns with the AS clause.

Example of creating a backup copy of Customers:

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

Example of using IN clause to copy the table into a new table in another database:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

Example to copy only a few columns into a new table:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

Example to copy only the German customers into a new table:

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

Example to copy data from multiple tables into a new table:

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

SELECT INTO can also be used to create a new, empty table using the schema of another.

To do that, add a WHERE clause that returns no data:

```
SELECT * INTO newtable
FROM oldtable
WHERE 1 = 0;
```

SQL Insert Into Select

The INSERT INTO SELECT statement copies data from one table and inserts it into another. It requires the data types in the source and target table to match.

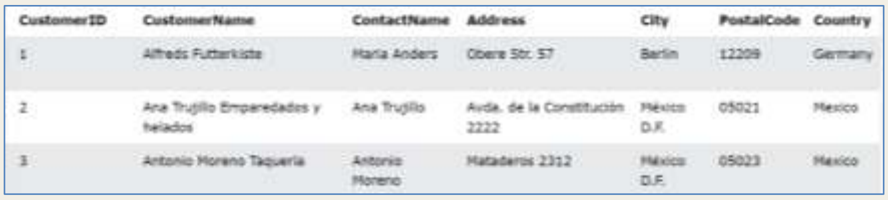
Syntax to copy all columns from one table to another:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Syntax to copy only some columns from one table to another:

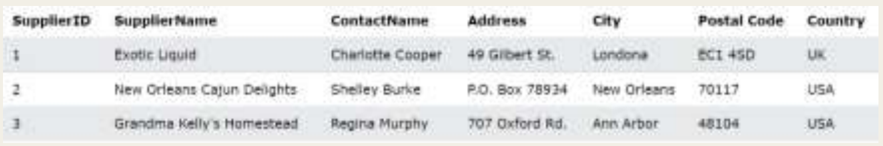
```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
```

WHERE condition;



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Table 42 *- Customers Table in INSERT INTO SELECT example (**Source:** https://www.w3schools.com/sql/sql_insert_into_select.asp)*



SupplierID	SupplierName	ContactName	Address	City	Postal Code	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

Table 43 *- Suppliers Table in INSERT INTO SELECT example (**Source:** https://www.w3schools.com/sql/sql_insert_into_select.asp)*

We will be using the Customers and Suppliers tables, shown above, in the following examples.

The first example copies Suppliers into Customers (note that the columns that are not filled with data will contain null values):
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;

This example copies Suppliers into Customers to fill all columns:
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;

The third example copies only the German suppliers into Customers:
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers
WHERE Country='Germany';

SQL Case

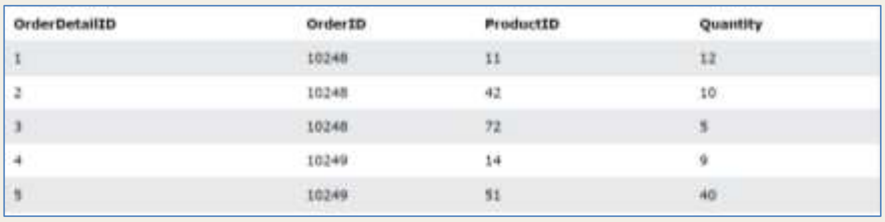
The CASE statement goes through a series of conditions and returns a value when the first condition is met. Think of it as an if, then, else statement.

When one condition is found true, it will stop going through the loop. If no conditions are found true, it will return the value in the ELSE clause.

Note that if there isn't an ELSE clause and no conditions are found true, it will return NULL.

Syntax:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```



OrderDetailID	OrderID	ProductID	Quantity
1	10248	11	12
2	10248	42	10
3	10248	72	5
4	10249	14	9
5	10249	51	40

***Table 44** - Orders Table in CASE example (**Source:** https://www.w3schools.com/sql/sql_case.asp)*

In the following examples, we will use the Orders table.

The first example will go through a series of conditions and return a value when the first condition is met:

```
SELECT OrderID, Quantity,
CASE
        WHEN Quantity > 30 THEN 'The quantity is greater than 30'
        WHEN Quantity = 30 THEN 'The quantity is 30’
        ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

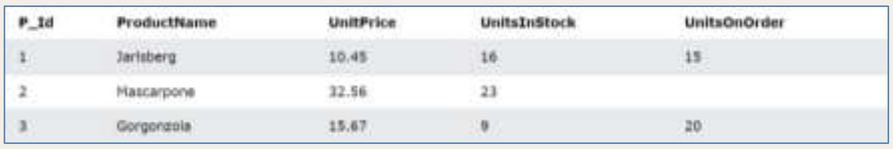
In this second example, we will order the customers by City. Note that if City is NULL, it will be ordered by Country.

```
SELECT CustomerName, City, Country
FROM Customers
ORDER BY
     (CASE
           WHEN City IS NULL THEN Country
           ELSE City
END);
```

SQL Null Functions

The NULL functions include the following: IFNULL(), ISNULL(), COALESCE(), and NVL().

Here, we will use the “Products” table:



P_Id	ProductName	UnitPrice	UnitsInStock	UnitsOnOrder
1	Jarlsberg	10.45	16	15
2	Mascarpone	32.56	23	
3	Gorgonzola	15.67	9	20

Table 45 - *Products Table in NULL functions example (**Source:** https://www.w3schools.com/sql/sql_isnull.asp)*

Let’s say that the “UnitsOnOrder” column is optional and may contain NULL values.

Example:

```
SELECT ProductName, UnitPrice * (UnitsInStock + UnitsOnOrder)
```

FROM Products;

Here we can see that if any of the UnitsOnOrder values are null, the result will also be null.

Let’s see how we can overcome this issue.

In MySQL, you can use the ISNULL() function that lets you return an alternative value if an expression is null:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

Or we can use the COALESCE() function:

```
SELECT ProductName, UnitPrice * (UnitsInStock + COALESCE(UnitsOnOrder, 0))
FROM Products;
```

In SQL Server, the ISNULL() function does the same thing as in MySQL:

```
SELECT ProductName, UnitPrice * (UnitsInStock + IFNULL(UnitsOnOrder, 0))
FROM Products;
```

In MS Access IsNull() function returns TRUE(-1) if the expression is a null value, otherwise FALSE (0):

```
SELECT ProductName, UnitPrice * (UnitsInStock + IIF(IsNull(UnitsOnOrder), 0,
UnitsOnOrder))
FROM Products;
```

In Oracle, the NVL() function does the same thing:

```
SELECT ProductName, UnitPrice * (UnitsInStock + NVL(UnitsOnOrder, 0))
FROM Products;
```

SQL Comments

SQL Comments explain SQL statement sections or prevent their execution.

Note that the examples in this section are not supported in Firefox and Microsoft Edge, which are Microsoft Access databases. Comments are generally not supported in Microsoft Access databases.

Single line comments in SQL start with - - (two dashes):

```
--Select all:
SELECT * FROM Customers;
```

Or it can be used like this to ignore the end of the line:

```
SELECT * FROM Customers -- WHERE City='Berlin';
```

Or to ignore a statement:

```
--SELECT * FROM Customers;
SELECT * FROM Products;
```

Multiple-line comments start with /* and end with */. Any text written between these two will be ignored.

Example:

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
```

To ignore part of a statement, you can also use /**/.

Example 1:

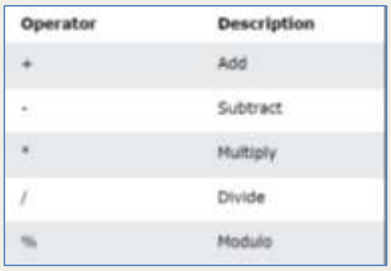
```
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

Example 2:

```
SELECT * FROM Customers WHERE (CustomerName LIKE 'L%'
OR CustomerName LIKE 'R%' /*OR CustomerName LIKE 'S%'
OR CustomerName LIKE 'T%'*/ OR CustomerName LIKE 'W%')
AND Country='USA'
ORDER BY CustomerName;
```

SQL Operators

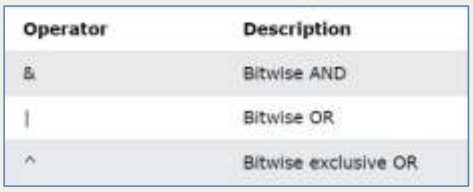
Arithmetic Operators used in SQL:



Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Table 46 *- Arithmetic Operators (**Source:** https://www.w3schools.com/sql/sql_operators.asp)*

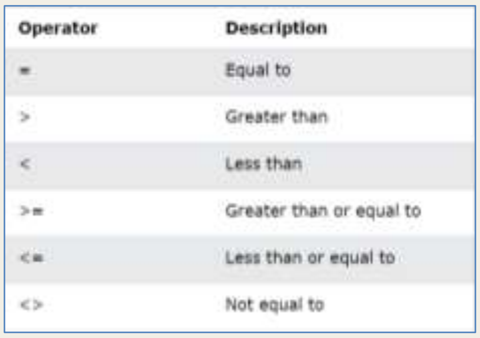
Bitwise operators used in SQL:



Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

Table 47 *– Bitwise operators (**Source**: https://www.w3schools.com/sql/sql_operators.asp)*

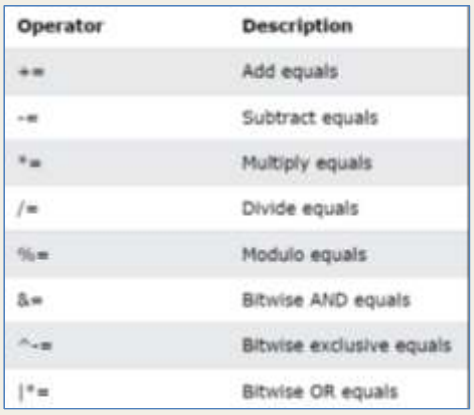
Comparison operators used in SQL:



Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Table 48 *– Comparison operators (**Source**: https://www.w3schools.com/sql/sql_operators.asp)*

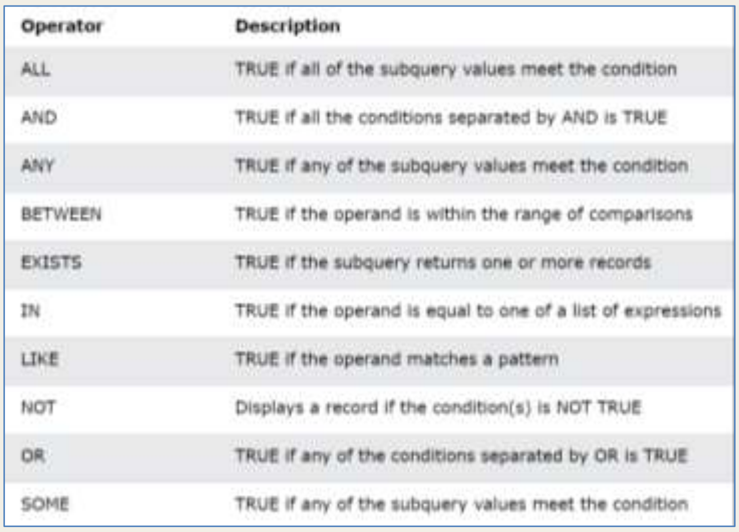
Compound operators used in SQL:



Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

Table 49 *- Compound operators (**Source:** https://www.w3schools.com/sql/sql_operators.asp)*

Logical operators used in SQL:



Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Table 50 *- Logical operators (**Source**: https://www.w3schools.com/sql/sql_operators.asp)*

SQL Database

As we have mentioned in the previous section that was dedicated to the basic statements used in SQL, this programming language is mainly used for relational databases. Therefore, in this section, we will learn how to create a database, modify it, and manipulate it with SQL.

Let's start simple, and we will build into slightly more complicated statements.

SQL Create DB

The CREATE DATABASE statement creates a new SQL database.

Syntax:

CREATE DATABASE DatabaseName;

Note: Always remember that the name of the database should be unique within the Relational Database Management System (RDMS) that you are using, and make sure that you have admin privileges before creating any database.

Let's say you want to create a test database. You would use the following statement:

```
CREATE DATABASE testDB;
```

SQL Drop DB

The DROP DATABASE statement deletes an existing SQL database.

```
DROP DATABASE DatabaseName;
```

Before you delete the database, make sure that you don't need any of the information that it contains because it completely deletes it.

Remember the database that we just created called "testDB"? Now we are going to delete it.

Example:

```
DROP DATABASE testDB;
```

SQL Backup DB

The BACKUP DATABASE statement does a complete backup on an existing SQL database.

To use this statement, you need to provide two things: the name of the database and the file path.

```
BACKUP DATABASE DatabaseName
TO DISK = 'filepath'
```

Example:

```
BACKUP DATABASE testDB
```

```
TO DISK = 'D:\backups\testDB.bak';
```

Note: To avoid technical problems, it is better to back up the database to a different drive than the one the existing database is on.

There is also another option where you perform a differential backup based on changes that have been made since the last complete database backup. This type of backup also reduces the backup time.

To do this, you follow this syntax:

```
BACKUP DATABASE DatabaseName
TO DISK = 'filepath'
WITH DIFFERENTIAL;
```

Example:

```
BACKUP DATABASE testDB
TO DISK = 'D:\backups\testDB.bak'
WITH DIFFERENTIAL;
```

SQL Create Table

The CREATE TABLE statement creates a new table in a database.

Syntax:

```
CREATE TABLE table_name (
                    column1 datatype,
                    column2 datatype,
                    column3 datatype,
                    …….
);
```

In this statement, you need to specify the **names of the columns** and the **type of data** that the column will contain.

Co-funded by the Erasmus+ Programme of the European Union

There are many data types such as integer, date or varchar. Depending on the type of data that you want to store, you choose the most suitable option. For instance, if you have a column named “Date of Birth”, then you would probably choose the Date as the data type.

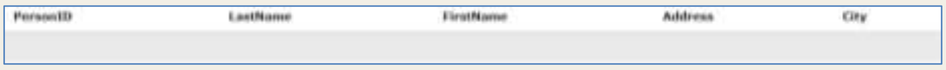
Example:

```
CREATE TABLE Persons (
                PersonID int,
                LastName varchar(255),
                FirstName varchar(255),
                Address varchar(255),
                City varchar(255)
);
```

This example will create a table with the name Persons and will contain 5 columns.

The PersonID will contain an integer (int); the columns LastName, FirstName, Address, and City will contain characters with a maximum length of 255.

The table will look something like this without any data:



PersonID	LastName	FirstName	Address	City

Table 51 *- Empty table in CREATE TABLE Example (**Source:** https://www.w3schools.com/sql/sql_create_table.asp)*

You can also create a table by using another table and choosing which columns you want in the new table. Keep in mind that the data of the existing table will fill the entries of the new table.

The syntax is as follows:

```
CREATE TABLE new_table_name AS
SELECT column1, column2,...
FROM existing_table_name
```

WHERE ....;

As you have learnt in the previous section:

- SELECT specifies the columns from the existing table,
- FROM specifies the name of the existing table, and
- WHERE can be used if you want a set of records that fulfil a specified condition.

Example:

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

SQL Drop Table

Similar to the DROP DATABASE statement that we saw earlier, the DROP TABLE statement deletes an existing table in a database.

Remember that you need to be sure that you do not need any of the information contained in a table before deleting it.

Syntax:

```
DROP TABLE TableName;
```

Example:

```
DROP TABLE Persons;
```

You can also choose to delete the data contained in a table, but not the table itself.

Maybe you created a new table from an existing table that has the structure that you want, but you want to add completely new entries. That is where TRUNCATE TABLE is useful.

Syntax:

```
TRUNCATE TABLE TableName;
```

Example:

```
TRUNCATE TABLE Persons;
```

SQL Alter Table

The ALTER TABLE statement can add, delete or modify columns in an existing table. Also, it can be used to add and drop constraints on an existing table.

Let's see the syntax of adding a column first:

```
ALTER TABLE TableName
ADD column_name datatype;
```

This is familiar to how we created a table by specifying the name of the column and the type of data to be contained in that column.

Example:

```
ALTER TABLE Customers
ADD Email varchar(255);
```

To delete a column in a table, as we have seen before, you use the DROP statement.

Keep in mind that some database systems do not allow for users to delete a column.

Syntax:

```
ALTER TABLE TableName
DROP COLUMN ColumnName;
```

As an example, let's delete the column that we created:

```
ALTER TABLE Customers
DROP COLUMN Email;
```

To change the data type of a column, you can use the following statements depending on the RDBMS that you are using:

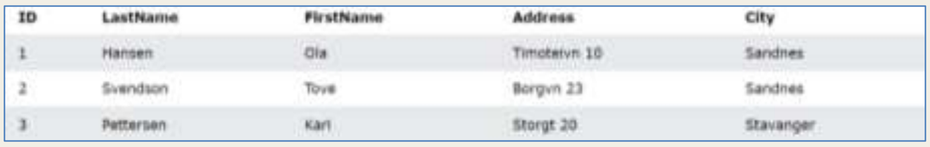
- ALTER COLUMN (for SQL Server/MS Access);
- MODIFY COLUMN (for My SQL/ Oracle prior to version 10G);
- MODIFY (for Oracle version 10G and later).

Syntax:

```
ALTER TABLE TableName
ALTER COLUMN ColumnName datatype;
```

Note that the second statement is the one that changes depending on the RDBMS that you are using from ALTER COLUMN to MODIFY COLUMN or MODIFY. The rest stays the same.

Let's see an example to understand this statement a bit better. The table underneath is the "Persons" table and contains information about different people.



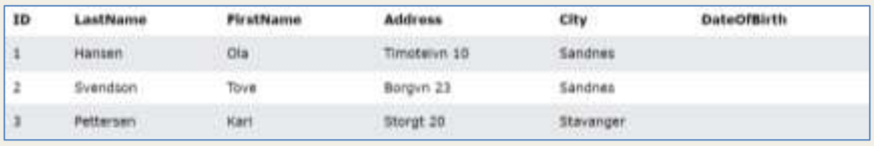
ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Table 52 *- ALTER TABLE Example (**Source:** https://www.w3schools.com/sql/sql_alter.asp)*

As an example, let's say that we wanted to add a column named "DateofBirth" in this table. We will use the following statement:

```
ALTER TABLE Persons
ADD DateofBirth date;
```

The new column that we added to the table has the data type of date, which means that it stores data in a date format. Underneath, you can see the table with the new column added.



ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

Table 53 *- ALTER TABLE Example (**Source**: https://www.w3schools.com/sql/sql_alter.asp)*

However, what if you changed your mind and wanted to change the data type of the new column, then you can use the ALTER COLUMN statement. For example, we can change the newly added column’s type from date to year. To do this, use the following statement:

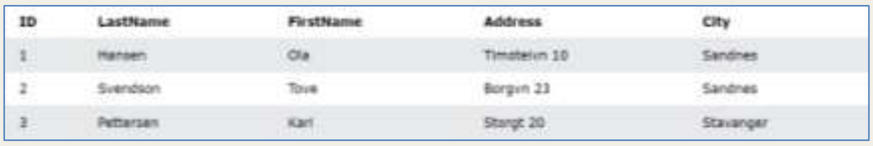
```
ALTER TABLE Persons
ALTER COLUMN DateofBirth year;
```

The year data type holds a year in two- or four-digits format.

To delete the column that we just altered, we use the DROP COLUMN statement.

```
ALTER TABLE Persons
DROP COLUMN DateofBirth;
```

Our table will go back looking the way it did in the beginning.



ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

Table 54 *- ALTER TABLE Example (**Source**: https://www.w3schools.com/sql/sql_alter.asp)*

SQL Constraints

SQL Constraints are used when the table is created with the statement CREATE TABLE or after the table is created with the statement ALTER TABLE.

Syntax:

```
CREATE TABLE table_name (
        column1 datatype constraint,
        column2 datatype constraint,
        column3 datatype constraint,
        ....
);
```

Constraints are used to specify a set of rules and restrictions that apply to a column or a table. They are used to ensure the integrity, accuracy, and reliability of the data. If the constraints are applied to a table, then all columns need to adhere to these constraints.

The following constraints are the ones that are most commonly used:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT
- CREATE INDEX

We will go through each of these constraints to explain their usage and syntax with examples.

SQL Not Null

In SQL, columns can hold null values by default. The NOT NULL constraint is used to avoid null values in columns. This is particularly important to ensure that when a new entry is added to a table all the necessary fields are filled.

As an example, let's say that we want to create a table named "Persons" and we want to ensure that the columns "ID", "LastName", and "FirstName" do not hold any null values:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255) NOT NULL,
        Age int
);
```

If, for some reason, you want to alter an already existing table to add constraints, you can use the following statement:

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

SQL Unique

The UNIQUE constraint is used to ensure that all values stored in a column are unique among the rows in a table. To make this clearer, think of the variable ID. You wouldn't want two people to have the same ID, therefore you would use the constraint UNIQUE on this occasion.

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL UNIQUE,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

My SQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        UNIQUE (ID)
);
```

As you can see, depending on the RDBMS that you are using, there are a few adjustments on where the UNIQUE constraint is put in the code.

If you want to name or define a UNIQUE constraint on multiple columns, use the following:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
       CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

You can also add a UNIQUE constraint after the table has been created by using the ALTER TABLE statement that we learnt earlier.

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (ID);
```

If you also want to name and define a UNIQUE constraint on multiple already existing columns, you use the following statement:

```
ALTER TABLE Persons
ADD CONSTRAINT UC_Persons UNIQUE (ID, LastName);
```

To delete the UNIQUE constraint, you can use the following statement:

My SQL:

```
ALTER TABLE Persons
DROP INDEX UC_Persons;
```

SQL Server/Oracle/ MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT UC_Persons;
```

SQL Primary Key

The PRIMARY KEY constraint is used to uniquely identify each row or record in a table. Note that primary keys must contain unique values, but cannot contain null values.

A table can only have **ONE** primary key and that primary key can consist of one or multiple columns.

SQL Server/Oracle/MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

MySQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        PRIMARY KEY (ID)
);
```

The following example allows you to name and define a PRIMARY KEY constraint on multiple columns:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Note that the PRIMARY KEY is still one, but the value of the primary key encompasses two columns.

You can also create a PRIMARY KEY constraint on an existing table by using the following statement:

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);
```

To add and define a PRIMARY KEY constraint on an existing table, use the following statement:

```
ALTER TABLE Persons
ADD CONSTRAINT PK_Persons PRIMARY KEY (ID, LastName);
```

To drop a PRIMARY KEY constraint, use the following statements according to your RDBMS.

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL Server / Oracle / MS Access:

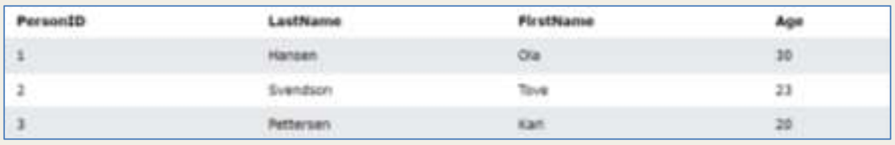
```
ALTER TABLE Persons
DROP PRIMARY KEY;
```

SQL Foreign Key

The FOREING KEY represents the columns of a table that are linked to a primary key in another table. The table that has a foreign key is called the child table, whereas the table that has the primary key is called the referenced or parent table.

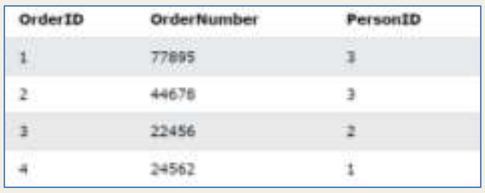
This type of constraint is used to prevent any actions that would destroy links between parent and child tables.

Let’s consider the following two tables:



PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Table 55 - *Persons table in FOREING KEY Example (**Source**: https://www.w3schools.com/sql/sql_foreignkey.asp)*



OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Table 56 - *Orders table in FOREING KEY Example (**Source**: https://www.w3schools.com/sql/sql_foreignkey.asp)*

These two tables are linked by the column “PersonID” that is found in both tables. Now, the primary key is located in the Persons table and the foreign key is the “PersonID” in the Orders table.

The FOREIGN KEY constraint works by preventing the input of invalid data in the foreign key column, because it is linked with the parent table and its values need to be identical.

To use the FOREIGN KEY constraint when creating a table, you can use the following statement according to your RDBMS.

SQL Server / Oracle / MS Access:

```
CREATE TABLE Orders (
        OrderID int NOT NULL PRIMARY KEY,
        OrderNumber int NOT NULL,
        PersonID int FOREIGN KEY REFERENCES Persons(PersonID)
);
```

My SQL:

```
CREATE TABLE Orders (
        OrderID int NOT NULL,
        OrderNumber int NOT NULL,
        PersonID int,
        PRIMARY KEY (OrderID),
        FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

This statement linked the Orders table to the Persons table with the FOREIGN KEY constraint based on PersonID column.

SQL Check

The CHECK constraint is used to specify the values allowed in a column or in certain columns of a table based on values found in other columns of the same row.

Example of CHECK constraint on CREATE TABLE

The following example is used to ensure that a person is not under the age of 18, so the CHECK constraint is added to the “Age” column.

MySQL:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
          CHECK (Age>=18)
);
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int CHECK (Age>=18)
);
```

If you want to name a CHECK constraint and use the constraint on multiple columns, you can use the following statement.

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
       City varchar(255),
       CONSTRAINT CHK_Person CHECK (Age>=18 AND City= ‘Sandnes’)
);
```

Example of CHECK constraint on ALTER TABLE

To create a constraint on an already existing table, use the following statement.

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD CHECK (Age>=18);
```

To name a constraint and create it on multiple columns, you can use:

```
ALTER TABLE Persons
ADD CONSTRAINT CHK_Person CHECK (Age>=18 AND City= ‘Sandnes’);
```

Example of DROP a CHECK constraint

To eliminate a CHECK constraint, you can use the following according to the RDMBS.

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
DROP CONSTRAINT CHK_PersonAge;
```

MySQL:

```
ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

SQL Default

The DEFAULT constraint is used to specify a default value for a column. If there are no other values specified, the default value will be added to all new records.

Example of DEFAULT constraint on CREATE TABLE

The following example adds a default value to the City column when the Persons table is created:

```
CREATE TABLE Persons (
        ID int NOT NULL,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int,
        City varchar(255) DEFAULT 'Sandnes'
);
```

This constraint can also be used to insert system values with functions such as GETDATE():

```
CREATE TABLE Orders (
        ID int NOT NULL,
        OrderNumber int NOT NULL,
        OrderDate date DEFAULT GETDATE()
);
```

Example of DEFAULT constraint on ALTER TABLE

In this example, the column “City” is used to create a DEFAULT constraint when we are altering an already existing table.

MySQL:

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

SQL Server:

```
ALTER TABLE Persons
ADD CONSTRAINT df_City
DEFAULT 'Sandnes' FOR City;
```

MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City SET DEFAULT ‘Sandnes’;
```

Oracle:

```
ALTER TABLE Persons
```

```
MODIFY City DEFAULT 'Sandnes';
```

Example of DROP a DEFAULT constraint

MySQL:

```
ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ALTER COLUMN City DROP DEFAULT;
```

SQL Index

The CREATE INDEX statement creates an index on a table. Indexes are useful when you want to retrieve data more quickly.

Please note that tables with indexes take more time to update in comparison to tables without. Therefore, it is suggested to only create indexes on columns that are frequently searched.
To CREATE INDEX on a table where duplicate values are allowed, use the following syntax:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

To CREATE UNIQUE INDEX on a table where duplicate values are not allowed, use the following syntax:

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

Keep in mind that creating indexes varies from database to database, so always check the syntax to create one in your database.

Examples of CREATE INDEX

In this example, we are creating an index on the LastName column by specifying the name idx_lastname:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

To create an index on a combination of columns, use the following statement:

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

If you want, you can add more columns in the parenthesis.

Examples of DROP INDEX

If you want to delete an index, use the following statement according to your RDBMS.

MS Access:

```
DROP INDEX index_name ON table_name;
```

SQL Server:

```
DROP INDEX table_name.index_name;
```

DB2/Oracle:

```
DROP INDEX index_name;
```

MySQL:

```
ALTER TABLE table_name
DROP INDEX index_name;
```

SQL Auto Increment

Auto-increment is used to generate unique numbers automatically when a new record is entered into a table. This is usually used on the primary key field in order to ensure that no one person has the same ID.

This feature uses different syntax in MySQL, SQL Server, Access and Oracle. Therefore, we will be going through each of these to explain how to use Auto-Increment.

MySQL:

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

In MySQL, AUTO_INCREMENT adds the auto-increment feature and by default, the value set is 1 and it goes up by 1 each time.

If you would like the sequence to start from a different value, use the following statement:

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

If you enter a new record into the Persons table, you will not have to specify a value for the "PersonID" column since it will be generated automatically:

```
INSERT INTO Persons (FirstName,LastName)
VALUES ('Lars','Monsen');
```

SQL Server

We are following the same example as above, where we use the “Personsid” column as the primary key in the Persons table:

```
CREATE TABLE Persons (
        Personid int IDENTITY(1,1) PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

In SQL Server, the auto-increment feature uses the keyword IDENTIFY to be activated. The two values in the parenthesis indicate (starting value, adding value for each new record). It will start at 1 and go up by 1 each time a new record is entered.

If you wanted to change the starting value to 10 and to add 5 each time a new record is added, you would write it like this IDENTIFY (10,5).

When entering new records, you do not need to specify the Personsid. It will be automatically generated as in the example above.

MS Access

```
CREATE TABLE Persons (
        Personid AUTOINCREMENT PRIMARY KEY,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Age int
);
```

MS Access uses AUTOINCREMENT keyword to activate the auto-increment feature. Similar to the other two, the starting value is one and it adds up by one each time a record is added.

You can specify different values such as 10 for starting value and 5 for each addition with AUTOINCREMENT(10,5).

Again, note that each time we add a new record, we do not need to specify the Personid value. It is generated automatically.

Oracle

In Oracle, the code is a bit trickier. To create an auto-increment field, you need to create a sequence of numbers:

```
CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10;
```

This sequence creates a sequence object named “seq_person”, sets the minimum value to start from (which is 1 in this instance), then specifies the increment by 1. The cache specifies how many sequence values should be stored in memory for faster access.

Unlike the previous examples, to enter a new record into the Persons table, you need to use the nextval function. This function is used to retrieve the next value from the sequence object that we created.

```
INSERT INTO Persons (Personid,FirstName,LastName)
VALUES (seq_person.nextval,'Lars','Monsen');
```

Here, we can see that the Personid column is selected to be assigned the next number from the sequence object that we created called “seq_person”.

SQL Dates

One of the most challenging parts when working with dates is to ensure that the format of the date you are trying to enter is the same with the format of the date column in the database.

It is important to note that data that contains only date portions will work as expected in queries. However, if there is a time portion, things get a bit more complicated.

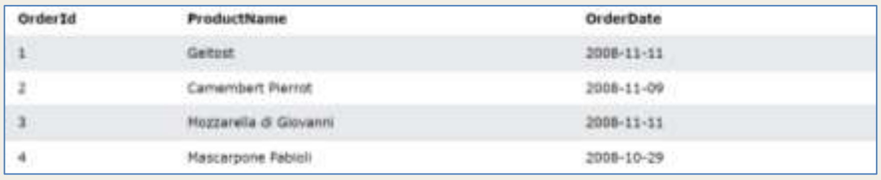
Date Data types found in MySQL:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
- YEAR - format YYYY or YY

Data types found in SQL Server:

- DATE - format YYYY-MM-DD
- DATETIME - format: YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

Keep in mind that the data types are chosen when you are creating a new table in your database.



OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
2	Camembert Pierrot	2008-11-09
3	Mozzarella di Giovanni	2008-11-11
4	Mascarpone Fabioli	2008-10-29

Table 57 *- Orders table in Dates Example (**Source**: https://www.w3schools.com/sql/sql_dates.asp)*

We will use the Orders table in our example to select the records with an OrderDate of “2008-11-11”.

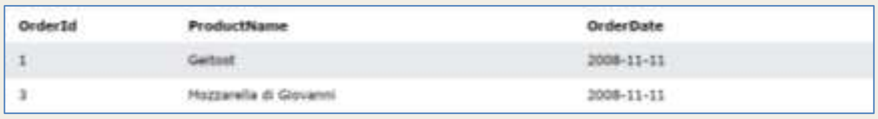
Example:

SELECT *

FROM Orders

WHERE OrderDate='2008-11-11';

The expected result will look something like this:

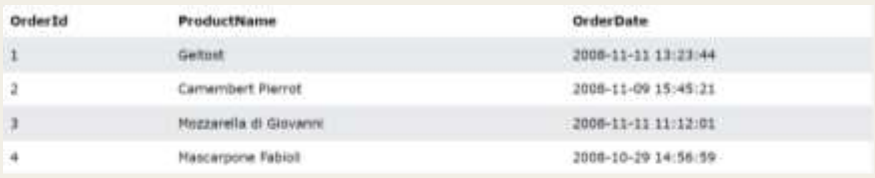


OrderId	ProductName	OrderDate
1	Geitost	2008-11-11
3	Mozzarella di Giovanni	2008-11-11

Table 58 *- Result of OrderDate query in Dates Example (**Source**: https://www.w3schools.com/sql/sql_dates.asp)*

Note that two dates can be easily compared when there is no time stamp involved.

Suppose that you have the Orders table, but with a timestamp in the OrderDate column.



OrderId	ProductName	OrderDate
1	Geitost	2008-11-11 13:23:44
2	Camembert Pierrot	2008-11-09 15:45:21
3	Mozzarella di Giovanni	2008-11-11 11:12:01
4	Mascarpone Fabioli	2008-10-29 14:56:59

Table 59 *- Orders table with timestamp in Dates Example (**Source**: https://www.w3schools.com/sql/sql_dates.asp)*

Here, if you attempted to use the same query as we used above:

SELECT *

FROM Orders

WHERE OrderDate='2008-11-11';

You would get no result, because the query is not taking into account the time stamp. It is recommended to not use time stamps unless you absolutely have to.

SQL Views

In SQL, a view is a virtual table of a result-set created from a specific query. A view is useful when you want to view and present data through a combination of tables.

Syntax:

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note that a view always shows up-to-date data since the database recreates the virtual table, every time users query it.

Example to query all customers from Brazil:

```
CREATE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName
FROM Customers
WHERE Country = 'Brazil';
```

To query the view:

```
SELECT * FROM [Brazil Customers];
```

Another example is to create a view that selects every product in the Products table with a price that is higher than the average price:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName, Price
FROM Products
WHERE Price > (SELECT AVG(Price) FROM Products);
```

To query the view above, use the following statement:

```
SELECT * FROM [Products Above Average Price];
```

To update a view, use the CREATE OR REPLACE VIEW statement:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

The following example adds the “City” column to the Brazil Customer view that we created earlier:

```
CREATE OR REPLACE VIEW [Brazil Customers] AS
SELECT CustomerName, ContactName, City
FROM Customers
WHERE Country = 'Brazil';
```

To delete a view, use the DROP VIEW statement:

```
DROP VIEW view_name;
```

For example, suppose we want to delete the “Brazil customers” view:

```
DROP VIEW [Brazil Customers];
```

SQL Data Types

Generally, each column in a table requires a name and a data type.

An SQL developer will need to decide the type of data that will be stored inside each column when creating a table. The data type is used for SQL to understand the data that will be contained in each column and also how it will interact with the data.

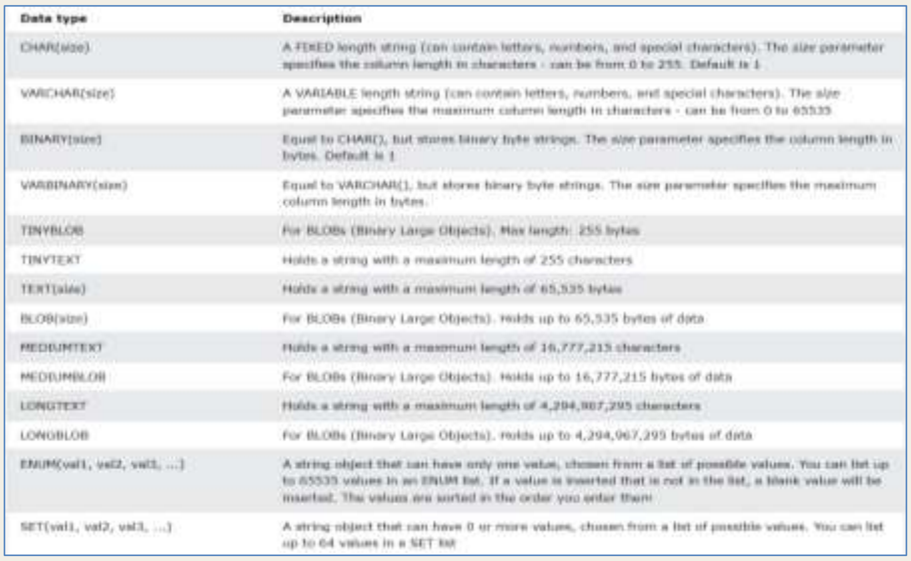
Please keep in mind that data types might have different names in different databases.

Always check the documentation even if the name is the same because other details might be different like the size.

Data types in MySQL (Version 8.0)

MySQL has three main data types: string, numeric, and date/time.

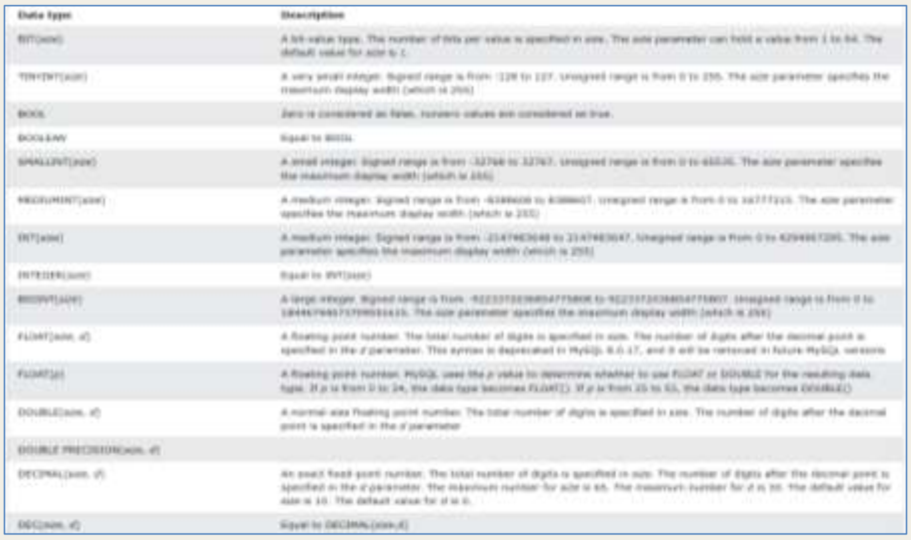
String Data Types



Data type	Description
CHAR(size)	A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1
VARCHAR(size)	A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum column length in characters - can be from 0 to 65535
BINARY(size)	Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.
TINYBLOB	For BLOBs (Binary Large Objects). Max length: 255 bytes
TINYTEXT	Holds a string with a maximum length of 255 characters
TEXT(size)	Holds a string with a maximum length of 65,535 bytes
BLOB(size)	For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters
MEDIUMBLOB	For BLOBs (Binary Large Objects). Holds up to 16,777,215 bytes of data
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters
LONGBLOB	For BLOBs (Binary Large Objects). Holds up to 4,294,967,295 bytes of data
ENUM(val1, val2, val3, ...)	A string object that can have only one value, chosen from a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted. The values are sorted in the order you enter them
SET(val1, val2, val3, ...)	A string object that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values in a SET list

Table 60 *– String Data types (MySQL) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

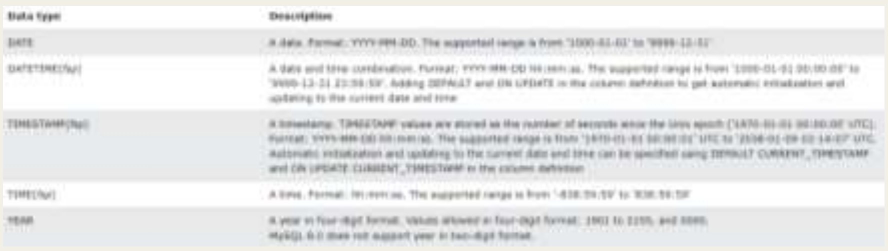
Numeric Data Types



Data type	Description
BIT(size)	A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.
TINYINT(size)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255)
BOOL	Zero is considered as false, nonzero values are considered as true.
BOOLEAN	Equal to BOOL
SMALLINT(size)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255)
MEDIUMINT(size)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255)
INT(size)	A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)
INTEGER(size)	Equal to INT(size)
BIGINT(size)	A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255)
FLOAT(size, d)	A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions
FLOAT(p)	A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()
DOUBLE(size, d)	A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter
DOUBLE PRECISION(size, d)	
DECIMAL(size, d)	An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0.
DEC(size, d)	Equal to DECIMAL(size,d)

Table 61 *– Numeric Data types (MySQL) (****Source****:* *https://www.w3schools.com/sql/sql_datatypes.asp)*

Date/Time Data Types

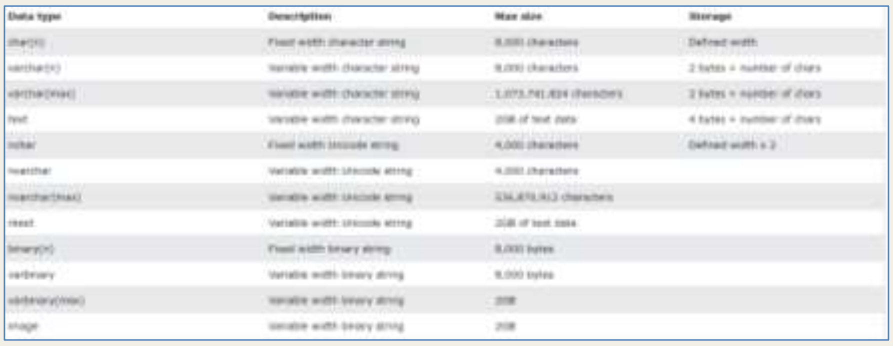


Data type	Description
DATE	A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'
DATETIME(fsp)	A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time
TIMESTAMP(fsp)	A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP in the column definition
TIME(fsp)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'
YEAR	A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000. MySQL 8.0 does not support year in two-digit format.

Table 62 *– Date/Time Data types (MySQL) (****Source****:* *https://www.w3schools.com/sql/sql_datatypes.asp****)***

Data Types in SQL Server

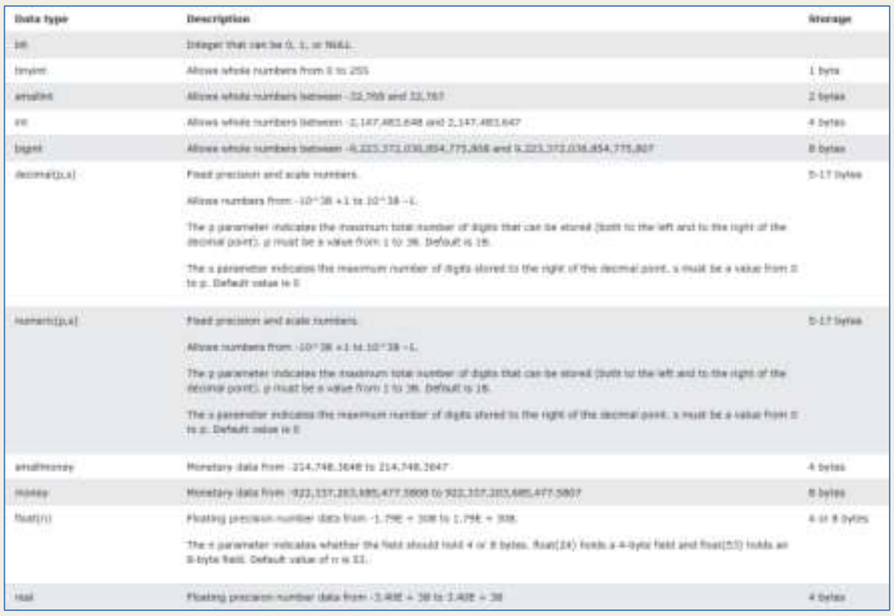
String Data Types



Data type	Description	Max size	Storage
char(n)	Fixed width character string	8,000 characters	Defined width
varchar(n)	Variable width character string	8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string	1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string	2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string	4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string	4,000 characters	
nvarchar(max)	Variable width Unicode string	536,870,912 characters	
ntext	Variable width Unicode string	2GB of text data	
binary(n)	Fixed width binary string	8,000 bytes	
varbinary	Variable width binary string	8,000 bytes	
varbinary(max)	Variable width binary string	2GB	
image	Variable width binary string	2GB	

Table 63 – *String Data types (SQL Server) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

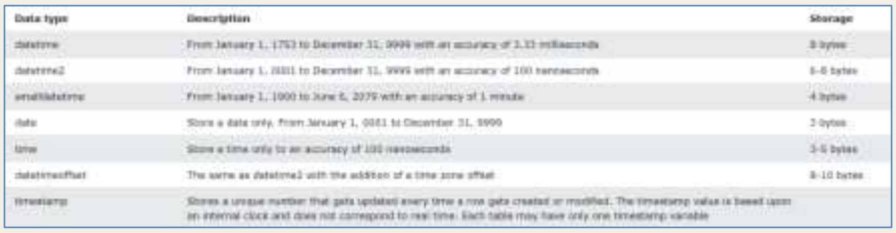
Numeric Data Types



Data type	Description	Storage
bit	Integer that can be 0, 1, or NULL	
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from -1.79E + 308 to 1.79E + 308. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from -3.40E + 38 to 3.40E + 38	4 bytes

Table 64 – *Numeric Data types (SQL Server) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

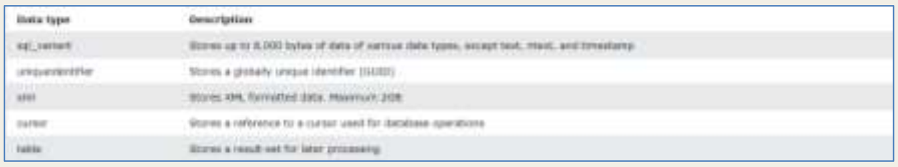
Date/Time Data Types



Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

Table 65 *– Date/Time Data types (SQL Server) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

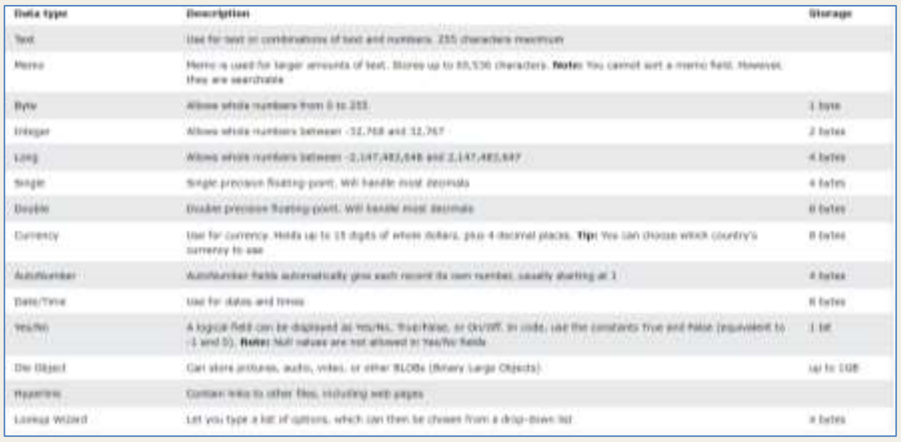
Other Data Types



Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

Table 66 *– Other Data types (SQL Server) (**Source**: https://www.w3schools.com/sql/sql_datatypes.asp)*

Data Types in MS Access



Data type	Description	Storage
Text	Use for text or combinations of text and numbers. 255 characters maximum	
Memo	Memo is used for larger amounts of text. Stores up to 65,536 characters. **Note:** You cannot sort a memo field. However, they are searchable	
Byte	Allows whole numbers from 0 to 255	1 byte
Integer	Allows whole numbers between -32,768 and 32,767	2 bytes
Long	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
Single	Single precision floating-point. Will handle most decimals	4 bytes
Double	Double precision floating-point. Will handle most decimals	8 bytes
Currency	Use for currency. Holds up to 15 digits of whole dollars, plus 4 decimal places. **Tip:** You can choose which country's currency to use	8 bytes
AutoNumber	AutoNumber fields automatically give each record its own number, usually starting at 1	4 bytes
Date/Time	Use for dates and times	8 bytes
Yes/No	A logical field can be displayed as Yes/No, True/False, or On/Off. In code, use the constants True and False (equivalent to -1 and 0). **Note:** Null values are not allowed in Yes/No fields	1 bit
Ole Object	Can store pictures, audio, video, or other BLOBs (Binary Large Objects)	up to 1GB
Hyperlink	Contain links to other files, including web pages	
Lookup Wizard	Let you type a list of options, which can then be chosen from a drop-down list	4 bytes

Table 67 *– Data types in Access (**Source:** https://www.w3schools.com/sql/sql_datatypes.asp)*

SQL References

SQL Keywords

Keyword	Description
ADD	Adds a column in an existing table
ADD CONSTRAINT	Adds a constraint after a table is already created
ALL	Returns true if all of the subquery values meet the condition
ALTER	Adds, deletes, or modifies columns in a table, or changes the data type of a column in a table
ALTER COLUMN	Changes the data type of a column in a table
ALTER TABLE	Adds, deletes, or modifies columns in a table
AND	Only includes rows where both conditions is true
ANY	Returns true if any of the subquery values meet the condition
AS	Renames a column or table with an alias

ASC	Sorts the result set in ascending order
BACKUP DATABASE	Creates a backup of an existing database
BETWEEN	Selects values within a given range
CASE	Creates different outputs based on conditions
CHECK	A constraint that limits the value that can be placed in a column
COLUMN	Changes the data type of a column or deletes a column in a table
CONSTRAINT	Adds or deletes a constraint
CREATE	Creates a database, index, view, table, or procedure
CREATE DATABASE	Creates a new SQL database
CREATE INDEX	Creates an index on a table (allows duplicate values)
CREATE OR REPLACE VIEW	Updates a view

CREATE TABLE	Creates a new table in the database
CREATE PROCEDURE	Creates a stored procedure
CREATE UNIQUE INDEX	Creates a unique index on a table (no duplicate values)
CREATE VIEW	Creates a view based on the result set of a SELECT statement
DATABASE	Creates or deletes an SQL database
DEFAULT	A constraint that provides a default value for a column
DELETE	Deletes rows from a table
DESC	Sorts the result set in descending order
DISTINCT	Selects only distinct (different) values
DROP	Deletes a column, constraint, database, index, table, or view
DROP COLUMN	Deletes a column in a table

DROP CONSTRAINT	Deletes a UNIQUE, PRIMARY KEY, FOREIGN KEY, or CHECK constraint
DROP DATABASE	Deletes an existing SQL database
DROP DEFAULT	Deletes a DEFAULT constraint
DROP INDEX	Deletes an index in a table
DROP TABLE	Deletes an existing table in the database
DROP VIEW	Deletes a view
EXEC	Executes a stored procedure
EXISTS	Tests for the existence of any record in a subquery
FOREIGN KEY	A constraint that is a key used to link two tables together
FROM	Specifies which table to select or delete data from
FULL OUTER JOIN	Returns all rows when there is a match in either left table or right table

GROUP BY	Groups the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG)
HAVING	Used instead of WHERE with aggregate functions
IN	Allows you to specify multiple values in a WHERE clause
INDEX	Creates or deletes an index in a table
INNER JOIN	Returns rows that have matching values in both tables
INSERT INTO	Inserts new rows in a table
INSERT INTO SELECT	Copies data from one table into another table
IS NULL	Tests for empty values
IS NOT NULL	Tests for non-empty values
JOIN	Joins tables
LEFT JOIN	Returns all rows from the left table, and the matching rows from the right table

LIKE	Searches for a specified pattern in a column
LIMIT	Specifies the number of records to return in the result set
NOT	Only includes rows where a condition is not true
NOT NULL	A constraint that enforces a column to not accept NULL values
OR	Includes rows where either condition is true
ORDER BY	Sorts the result set in ascending or descending order
OUTER JOIN	Returns all rows when there is a match in either left table or right table
PRIMARY KEY	A constraint that uniquely identifies each record in a database table
PROCEDURE	A stored procedure
RIGHT JOIN	Returns all rows from the right table, and the matching rows from the left table
ROWNUM	Specifies the number of records to return in the result set

SELECT	Selects data from a database
SELECT DISTINCT	Selects only distinct (different) values
SELECT INTO	Copies data from one table into a new table
SELECT TOP	Specifies the number of records to return in the result set
SET	Specifies which columns and values that should be updated in a table
TABLE	Creates a table, or adds, deletes, or modifies columns in a table, or deletes a table or data inside a table
TOP	Specifies the number of records to return in the result set
TRUNCATE TABLE	Deletes the data inside a table, but not the table itself
UNION	Combines the result set of two or more SELECT statements (only distinct values)
UNION ALL	Combines the result set of two or more SELECT statements (allows duplicate values)

UNIQUE	A constraint that ensures that all values in a column are unique
UPDATE	Updates existing rows in a table
VALUES	Specifies the values of an INSERT INTO statement
VIEW	Creates, updates, or deletes a view

***Table 68** – SQL Keywords and Descriptions (**Source:** https://www.w3schools.com/sql/sql_ref_keywords.asp)*

MySQL Functions

For more details and a comprehensive list on specific functions used in MySQL, learners can refer to this link.

SQL Server Functions

For more details and a comprehensive list on specific functions used in SQL Server, learners can refer to this link.

MS Access Functions

For more details and a comprehensive list on specific functions used in SQL Server, learners can refer to this link.

SQL Quick Ref

For a comprehensive list of SQL statements and their corresponding syntax, learners can refer to this link.

4.4. SQL Examples

SQL Examples

There is comprehensive list of examples in the W3Schools website that learners can use to self-study and practice their SQL skills further.

SQL Quiz

For learners that want to assess their knowledge and skills on SQL, please refer to one of the following websites:

- W3Schools
- Tutorialspoint

SQL Exercises

For a comprehensive list of exercises, learners can use the W3Schools website to practice their SQL skills.