SQL

[**https://www.w3schools.com/sql/sql\_join.asp**](https://www.w3schools.com/sql/sql_join.asp)

<https://demo.phpmyadmin.net/master-config/index.php?route=/database/sql&db=sakila> - demo sakila database

SQL: SQL is a standard language for storing, manipulating and retrieving data in databases.

It was initially developed at IBM in 1970’s. SQL is standard language to communicate with relational database management systems like Oracle, MS Access, MS SQL server, MySQL, DB2, and Sybase Etc.

Purpose of SQL:

* SQL can execute queries against a database
* SQL can retrieve data from a database
* SQL can insert records in a database
* SQL can update records in a database
* SQL can delete records from a database
* SQL can create new databases
* SQL can create new tables in a database
* SQL can create stored procedures in a database
* SQL can create views in a database
* SQL can set permissions on tables, procedures, and views

## Some of the Most Important SQL Commands

* SELECT - extracts data from a database
* UPDATE - updates data in a database
* DELETE - deletes data from a database
* INSERT INTO - inserts new data into a database
* CREATE DATABASE - creates a new database
* ALTER DATABASE - modifies a database
* CREATE TABLE - creates a new table
* ALTER TABLE - modifies a table
* DROP TABLE - deletes a table
* CREATE INDEX - creates an index (search key)
* DROP INDEX - deletes an index

Database: A container to help organize data and a way to efficiently store and retrieve data

Relational: A way to describe data and the relationships between data entities

A relational database consist of tables and a table consist of rows (records) and columns (fields).

And intersection of rows and columns is called cell.

Primary Keys uniquely indentifies each row of table.

Foreign keys enforces referential integrity.

MySQL (version 8) is an open-source relational database management system. [Developed by](https://www.google.com/search?safe=active&rlz=1C1GCEB_enIN840IN840&sxsrf=ACYBGNQV-PMWSf5oAbIp2HtpYSAn0LXJ3w:1581139586803&q=mysql+developed+by&stick=H4sIAAAAAAAAAOPgE-LQz9U3MKk0ztZSySi30k_Oz8lJTS7JzM_TL85PKylPLEq1SkktS83JL0hNUUiqXMQqlFtZXJijgCwIALZ1lRdHAAAA&sa=X&ved=2ahUKEwjlk6X7m8HnAhUMCewKHeqWDiAQ6BMoADAcegQIEBAG): [Oracle Corporation](https://www.google.com/search?safe=active&rlz=1C1GCEB_enIN840IN840&sxsrf=ACYBGNQV-PMWSf5oAbIp2HtpYSAn0LXJ3w:1581139586803&q=Oracle&stick=H4sIAAAAAAAAAOPgE-LQz9U3MKk0zlYCs0zzssq1VDLKrfST83NyUpNLMvPz9Ivz00rKE4tSrVJSy1Jz8gtSUxSSKhexsvkXJSbnpO5gZQQA7gvd_0gAAAA&sa=X&ved=2ahUKEwjlk6X7m8HnAhUMCewKHeqWDiAQmxMoATAcegQIEBAH).

Commands for finding version:

Microsoft: select @@version;

Oracle – select \* from v$version;

MySQL & PostgreSQL – select version ();

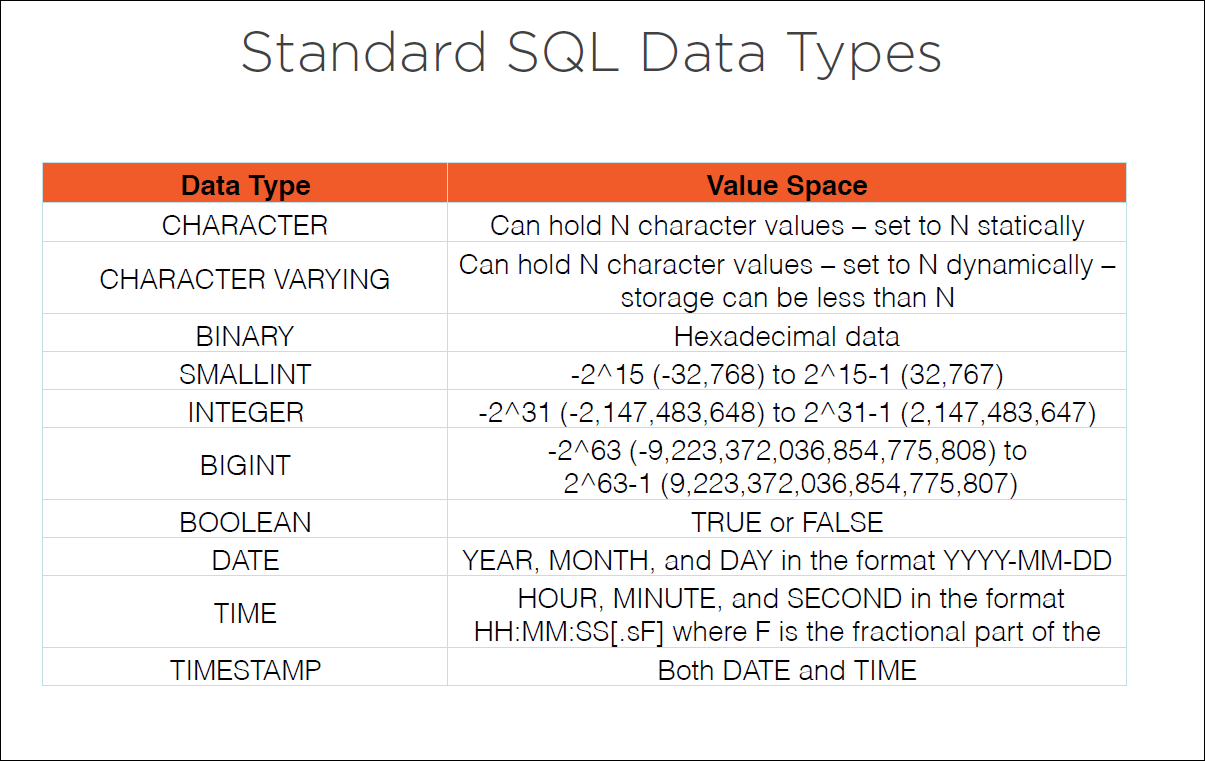
MySQL Workbench (version 6.3) is a visual database design tool that integrates SQL development, administration, database design, creation and maintenance into a single integrated development environment for the MySQL database system.

MySQL workbench divided into three major sections 1) development 2) Modeling 3) Administration

We can retrieve, view, edit or delete data from SQL editor or SQL Development section.

We can create database models as well as reverse engineer database from Data modeling section

Remember we can use MySQL command line editor if we have no access to MySQL workbench



A SQL statement is an expression that tells a database what you want it to do.

Basic SQL Statements:

|  |  |  |  |
| --- | --- | --- | --- |
| Data Definition Language | Data Manipulation Language | Data Control Language | Transaction Control Language |
| Deals with database schemas and descriptions, of how the data should reside in the databases | Deals with data manipulation, and includes most common SQL statements and it is used to store, modilfy, retrive, delete and update data in database | Includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system | Deals with transaction with in a database |
| Create Alter Drop  Truncate  Rename | Select insert update Delete | Grant Revoke | Commit Rollback savepoint |

Basic SQL Commands:

Use database name;



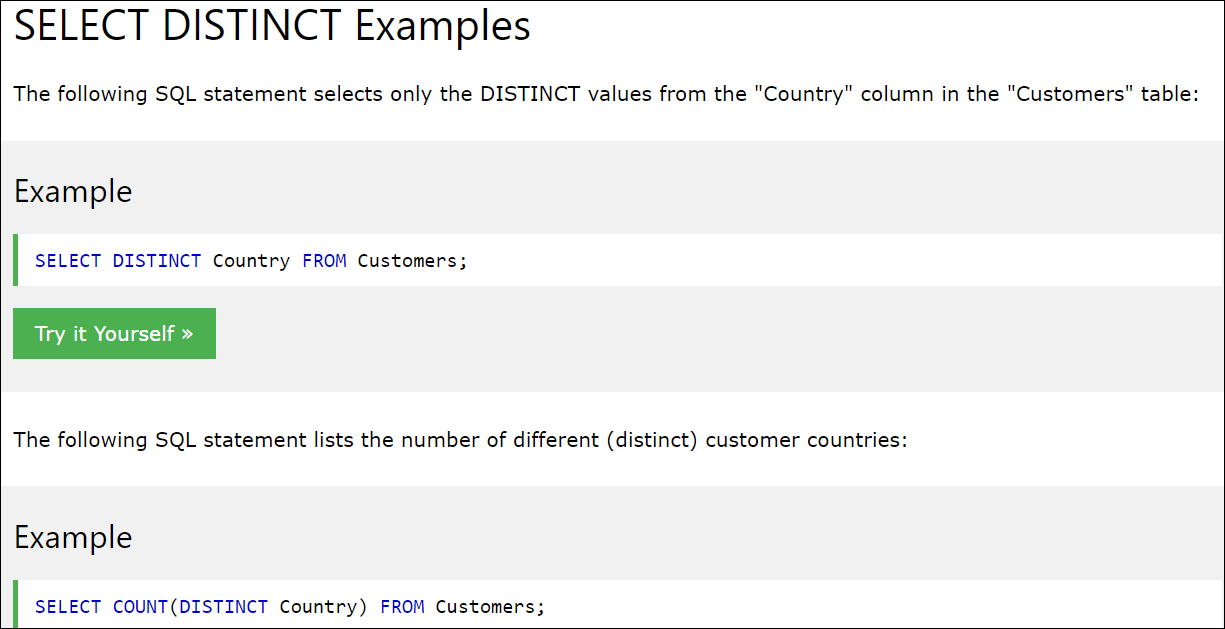
\* Pulls all the columns from a table



Not Distinct means first names of all the people.

Distinct means unique first names of the people.

Select DISTINCT first name from person;



SELECT “my first value’;

Select 1+1 AS two;

Select NOW ()

Select CURDATE ()

Select CURTIME ()

Select PI ();

Select MOD (45, 7);

Select SQRT (25);

Select first Name/5 AS FIRSTNAME

String operations:

SELECT CONCAT (first Name, ‘ ‘, Last Name) AS Full Name from Table Name;

Use Database;

Retrieve empty result set: SELECT \* FROM Table Name WHERE 1 = 2;

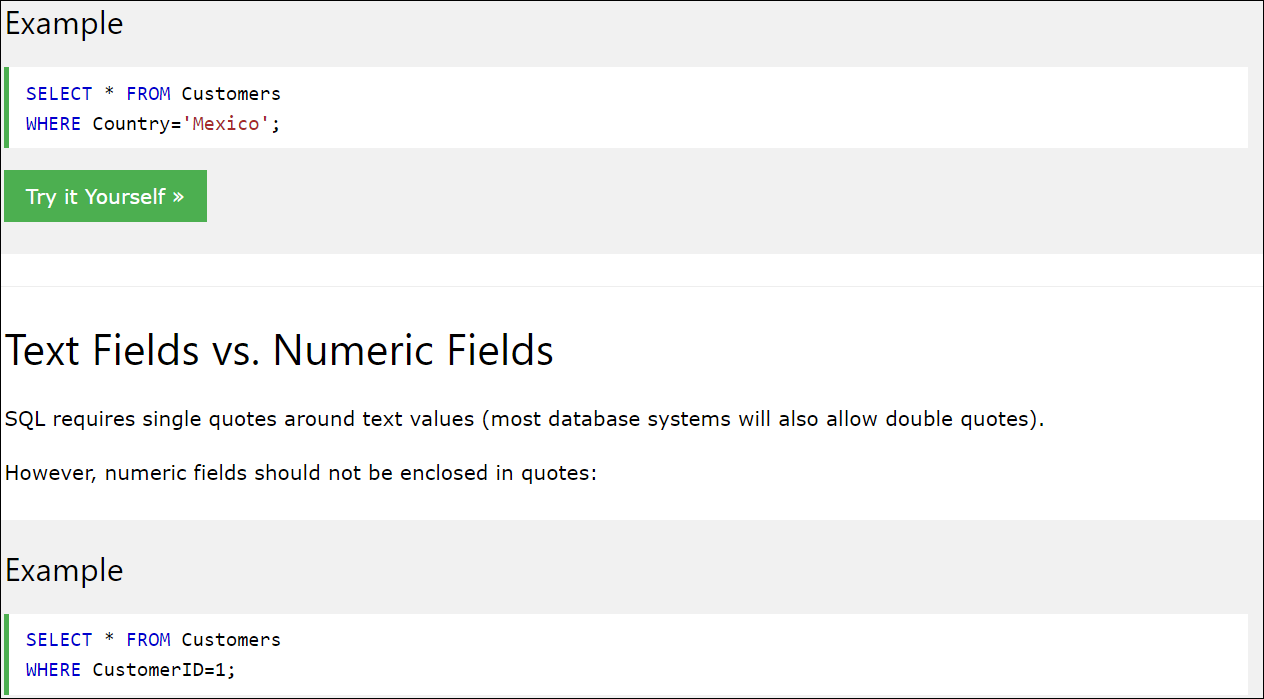
String operations:

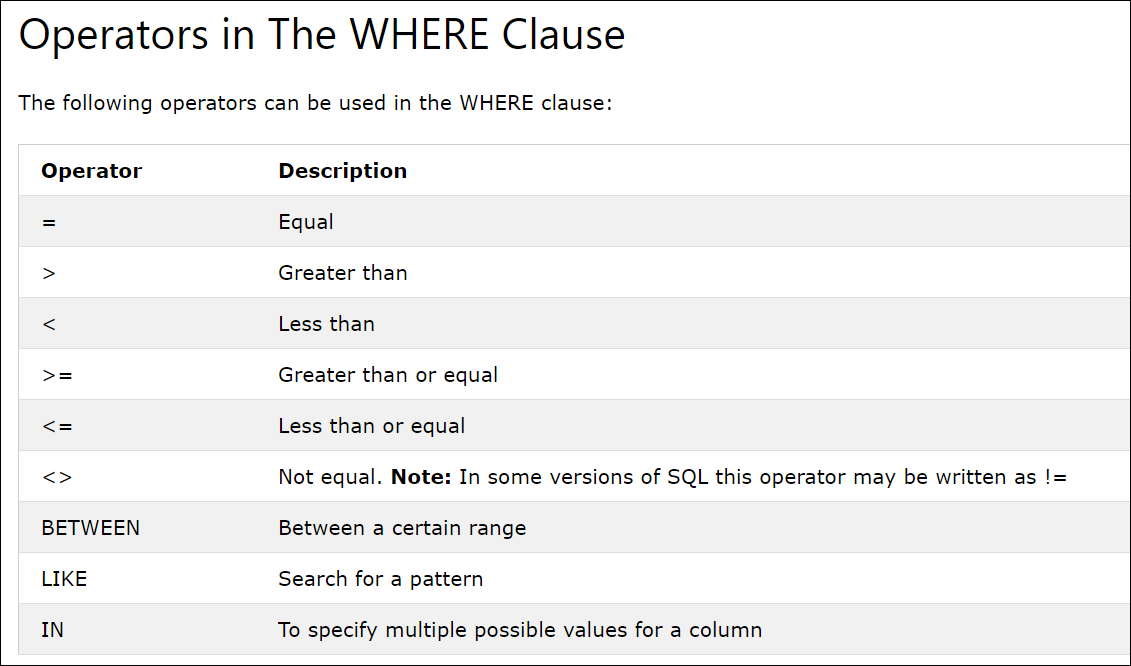
SELECT CONCAT (first Name, ‘ ‘, Last Name) AS Full Name from Table Name;

WHERE:

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.





Examples:

SELECT \* FROM Products

WHERE Price = 18;

WHERE Price > 30;

WHERE Price < 30;

WHERE Price >= 30;

WHERE Price <= 30;

WHERE Price <> 18;

WHERE Price BETWEEN 50 AND 60;

WHERE City LIKE 's%';

WHERE City LIKE 's\_r';

WHERE City LIKE 's%r';

WHERE City IN ('Paris','London');

WHERE City NOT IN ('Paris','London');

SQL AND, OR and NOT Operators:

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 separated by AND are TRUE.
* The OR operator displays a record if any of the conditions separated by OR is TRUE.

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

SELECT \* FROM Customers

WHERE Country='Germany' AND City='Berlin';

WHERE City='Berlin' OR City='München';

sWHERE NOT Country='Germany';

WHERE Country='Germany' AND (City='Berlin' OR City='München');

SQL ORDER BY Keyword

SELECT \* FROM Customers  
ORDER BY Country;

ORDER BY Country DESC;

ORDER BY Country ASC, CustomerName DESC;

String operations:

Computer new values from columns values

SELECT sum (contact number) from person

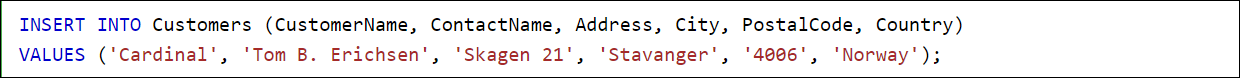
Length: SELECT first\_name, LENGTH(first\_name) AS Len FROM actor;

SELECT LEFT (first\_name,1) FROM actor # select first letter from column

SELECT CONCAT ( first\_name, ' ' , last\_name) FROM actor

SELECT SUBSTRING (first\_name, 1,3) FROM actor # 2 digit defines index values

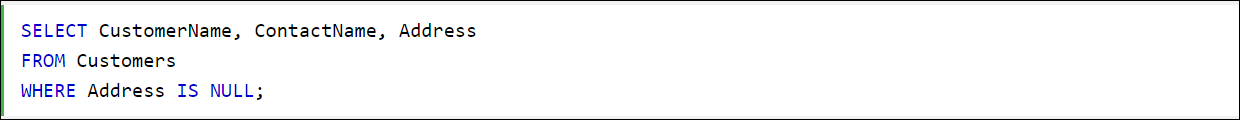
## **SQL INSERT INTO Statement**



Null Operator:

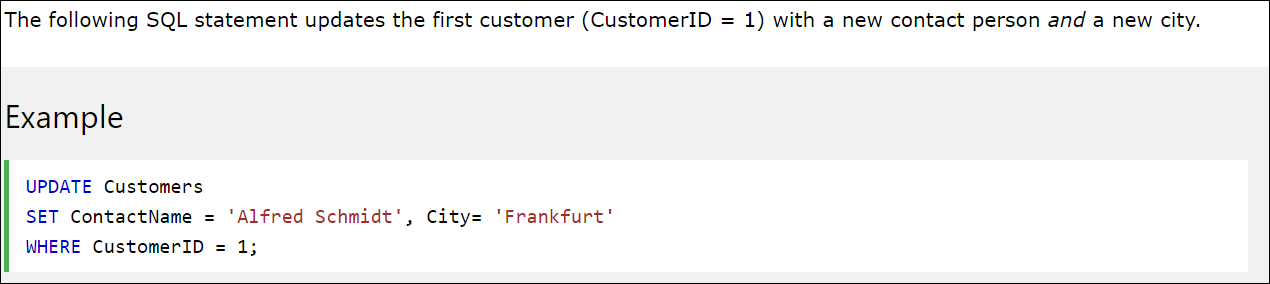
The IS NULL operator is used to test for empty vssalues (NULL values).

The following SQL lists all customers with a NULL value in the "Address" field:



## The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.



The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.



Filtering Results with the where Clause:

What is the last name of all the people I know whose first name is john?

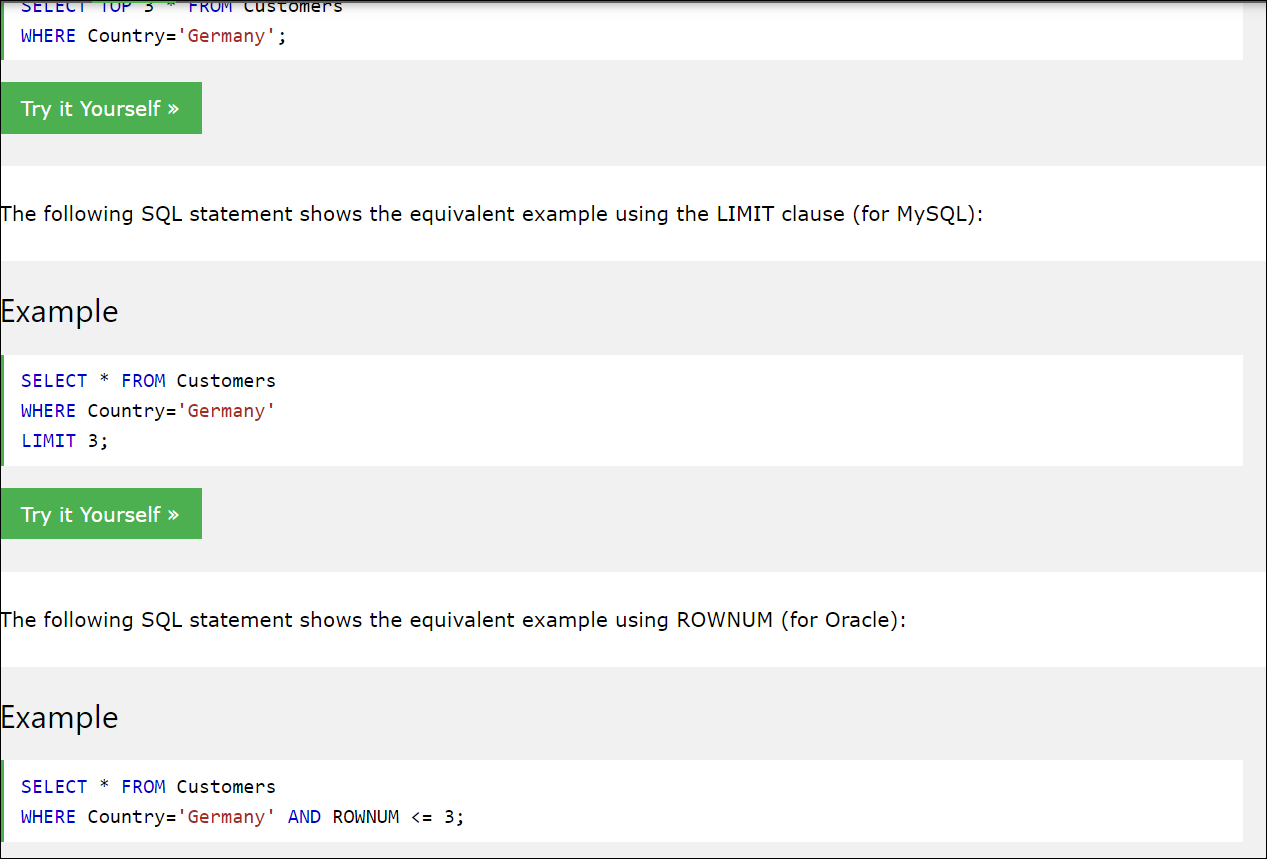
Select last name from person where first name = ‘john’;

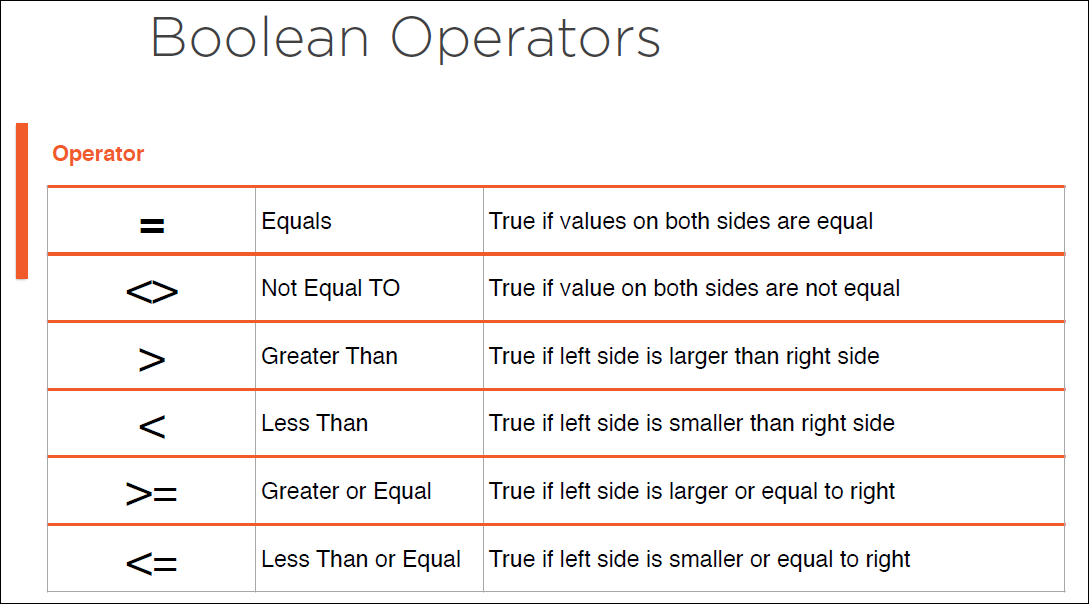
SQL TOP, LIMIT or ROWNUM Clause: The SELECT TOP clause is used to specify the number of records to return

SELECT TOP 3 \* FROM Customers;

SELECT \* FROM Customers LIMIT 3;

SELECT \* FROM Customers WHERE ROWNUM <= 3;

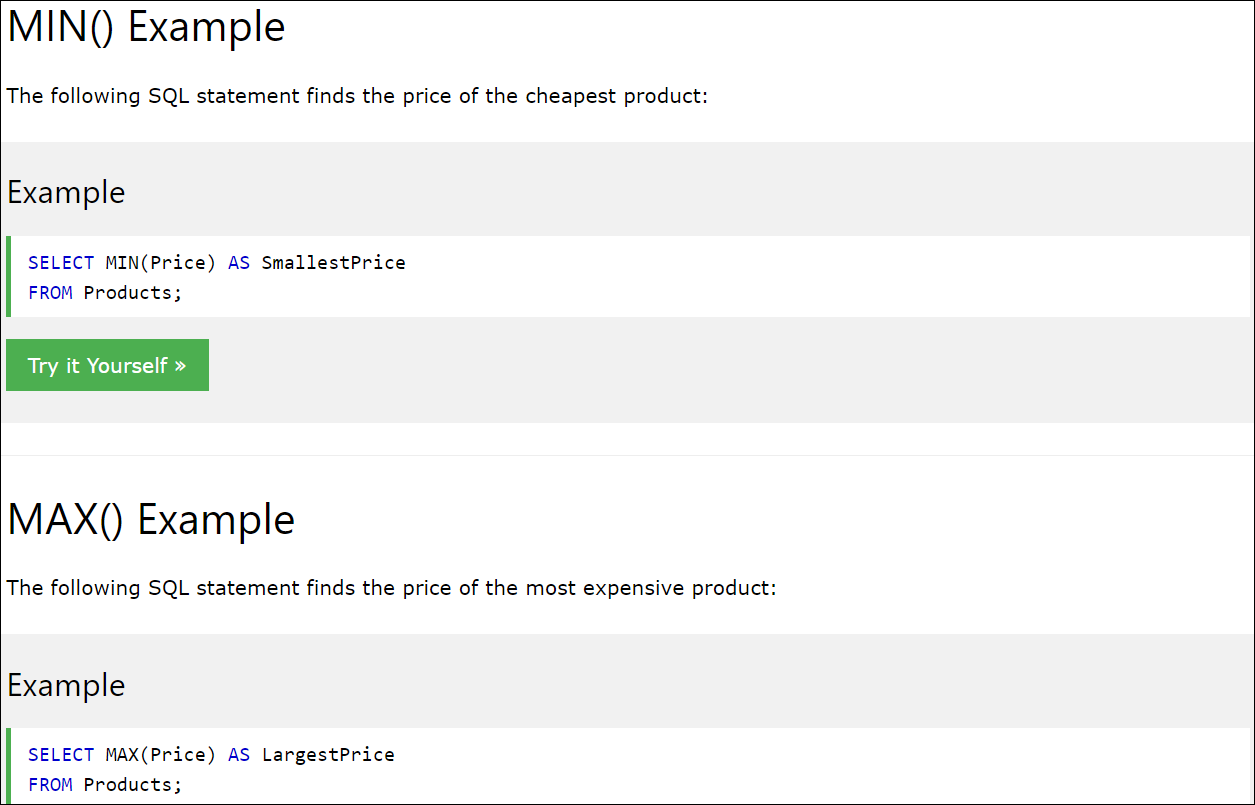




The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

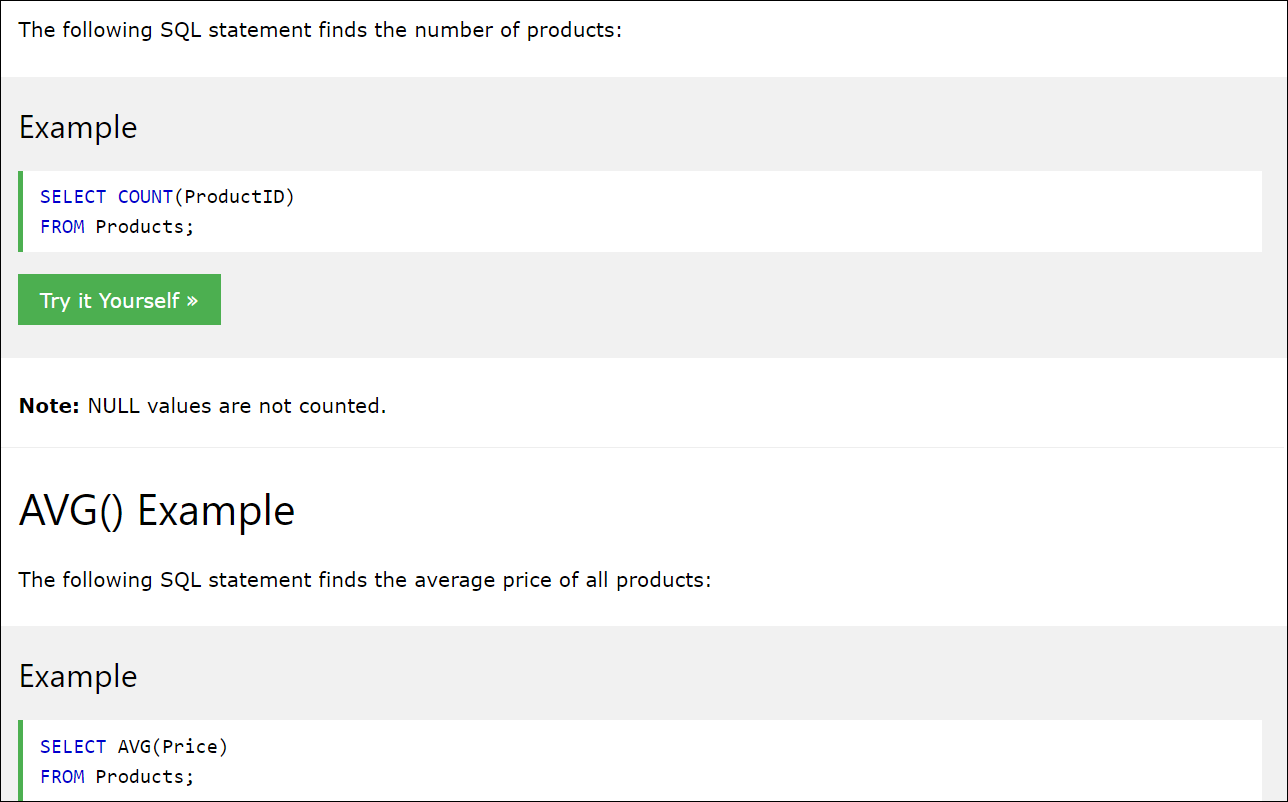


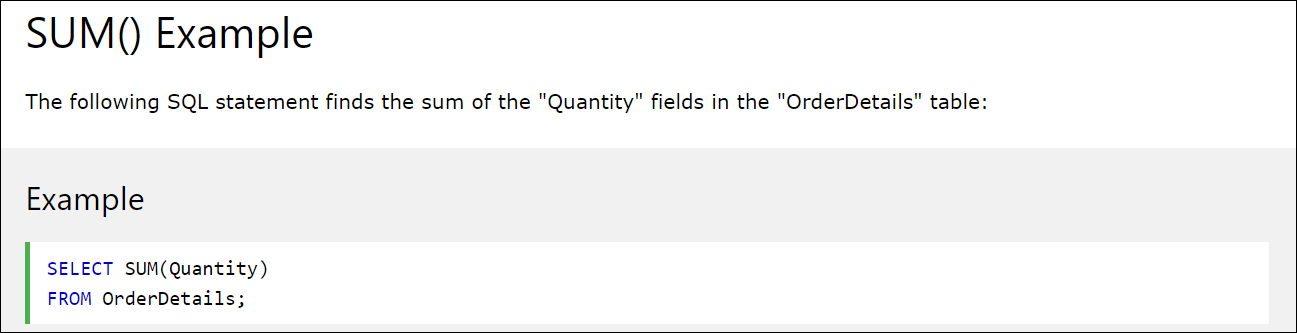
## The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criteria.

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

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





And: if both are true, row is included.

Select last name from person where first name = ‘john’ AND birthdate > ‘12/31/1965’;

OR: if either true, row is included.

Select last name from person where first name = ‘john’ OR birthdate > ‘12/31/1965’;

Between: True if column value between two values

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

Select first name from person where contacted BETWEEN 1 AND 20;

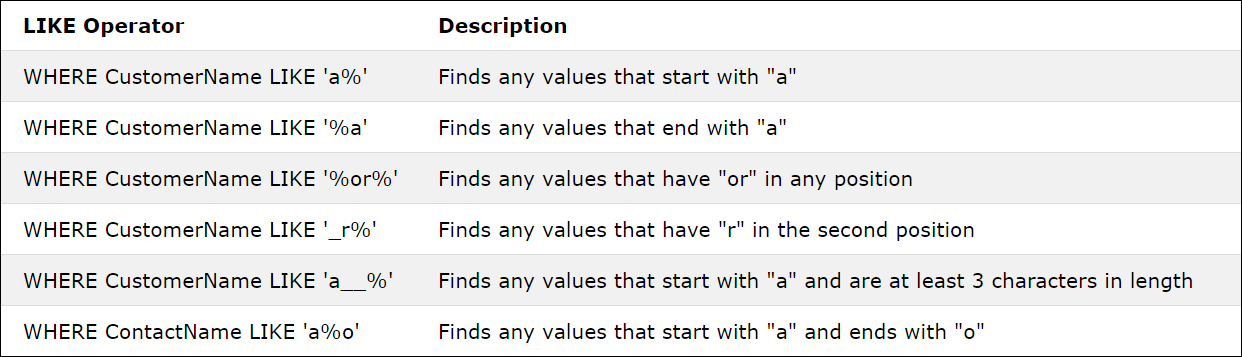
SELECT \* FROM Products WHERE Price BETWEEN 10 AND 20;

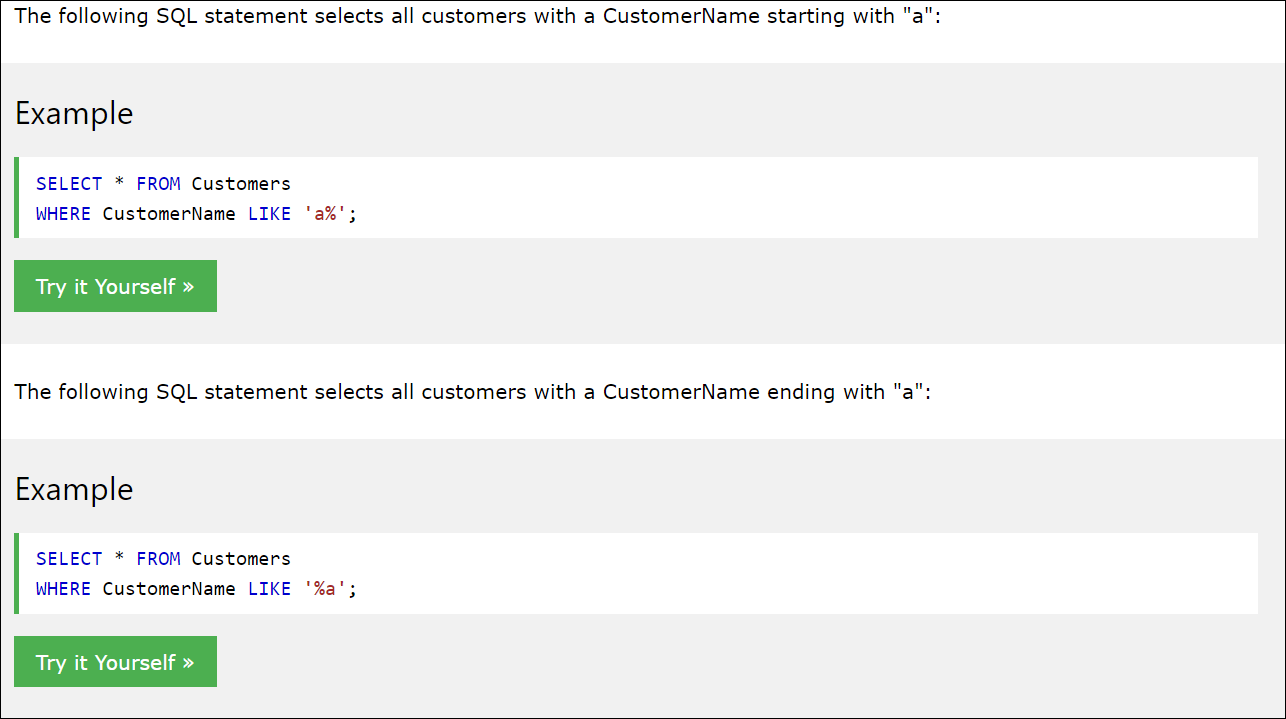
SELECT \* FROM Products WHERE Price NOT BETWEEN 10 AND 20;

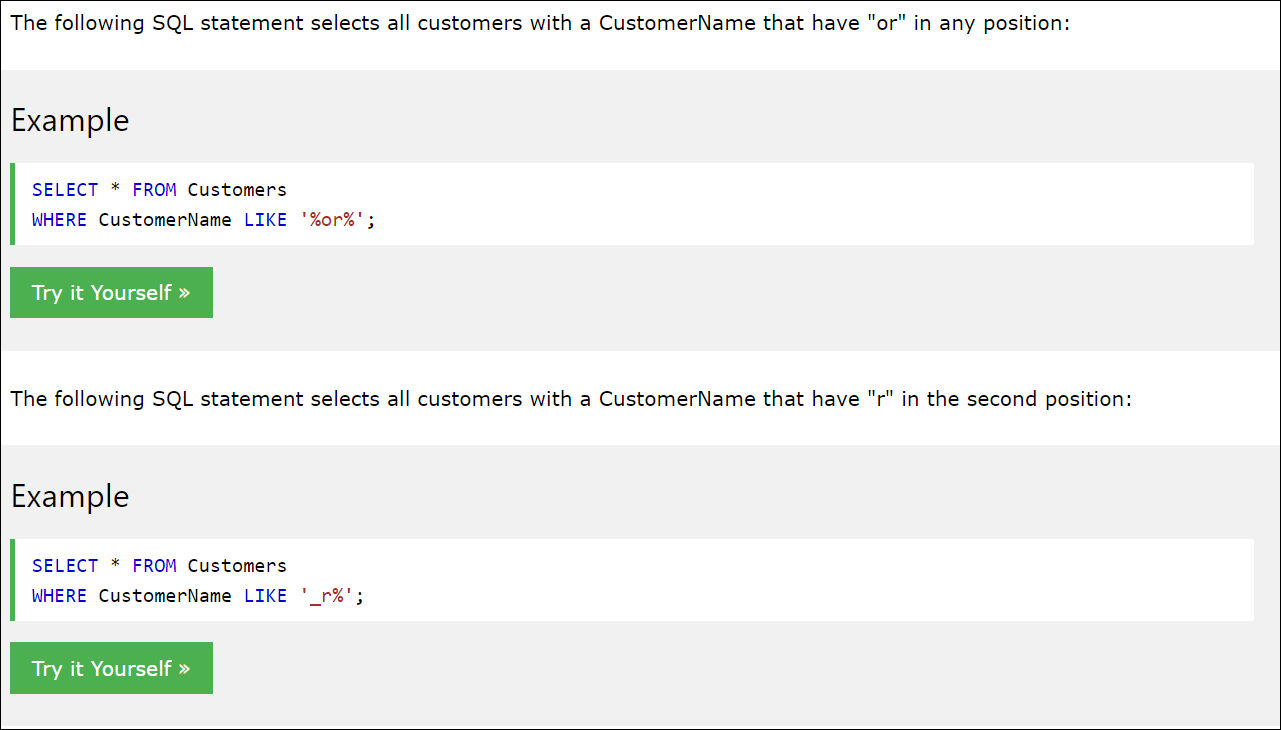
Like:

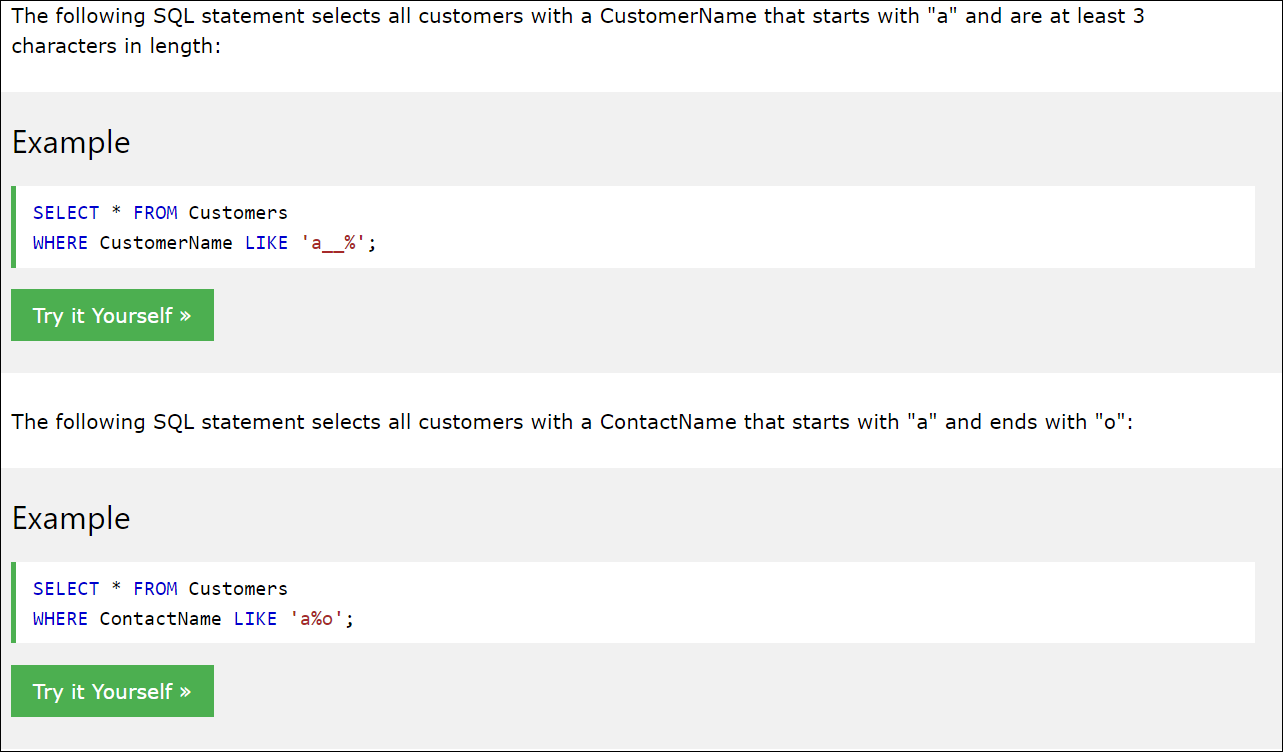
The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

* % - The percent sign represents zero, one, or multiple characters
* \_ - The underscore represents a single character









What are the all people in my contact list that have a first name? That begins with the letter J?

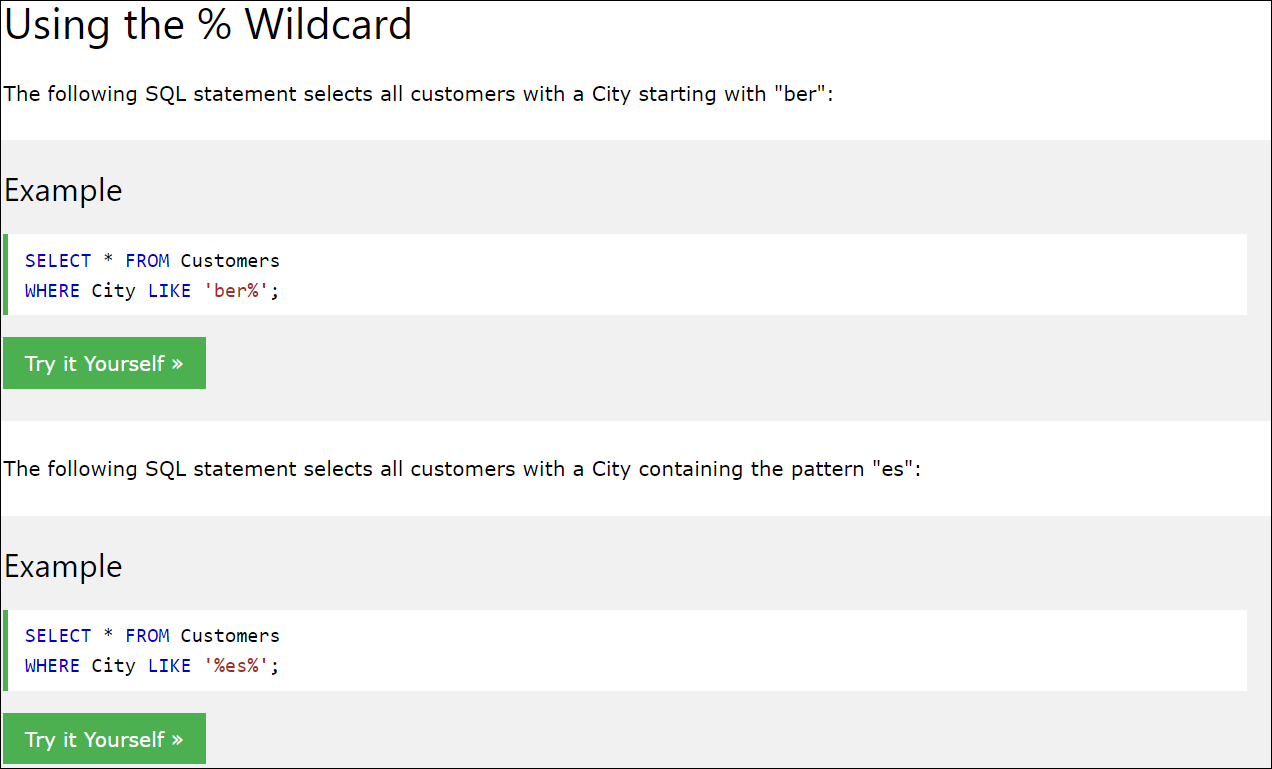
Select first name from person where p. First name like ‘j%’;

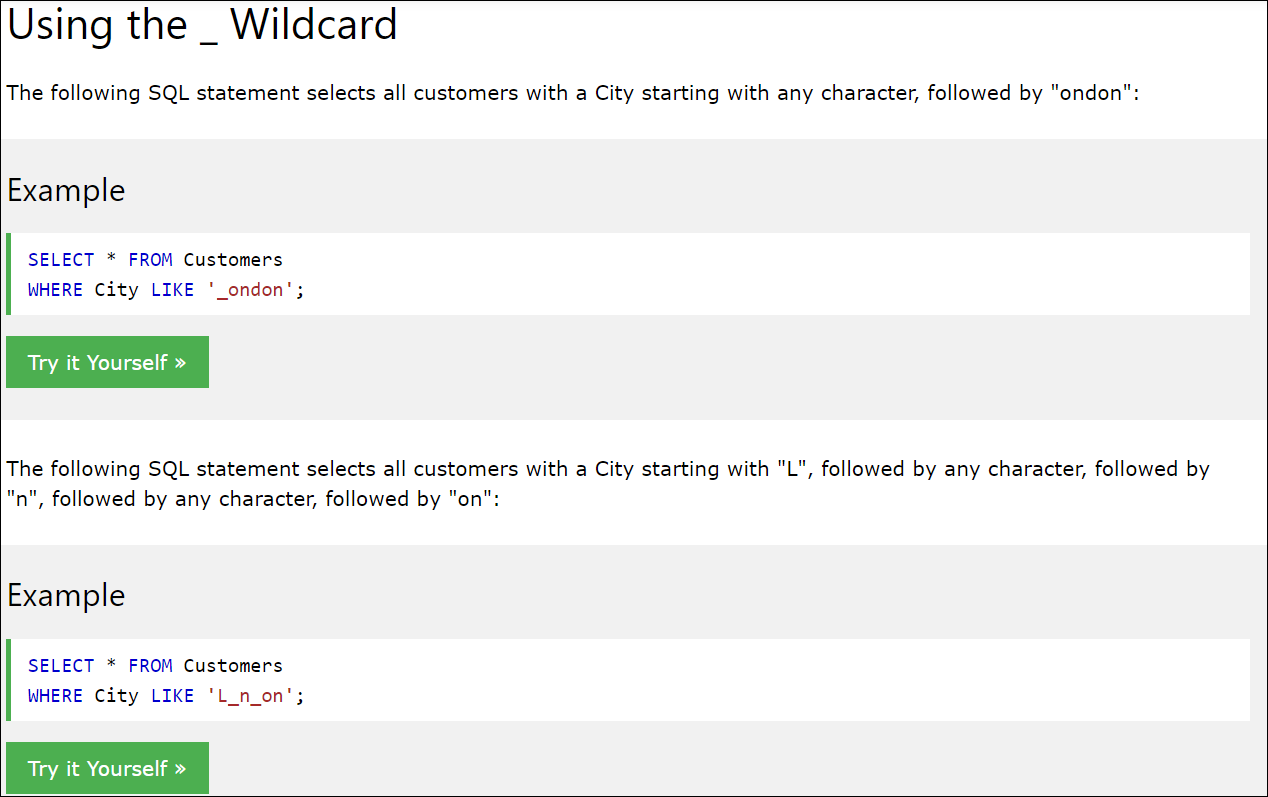
Select first name from person where p. First name like ‘A\_\_E’; 4th letter E

SQL Wildcards:

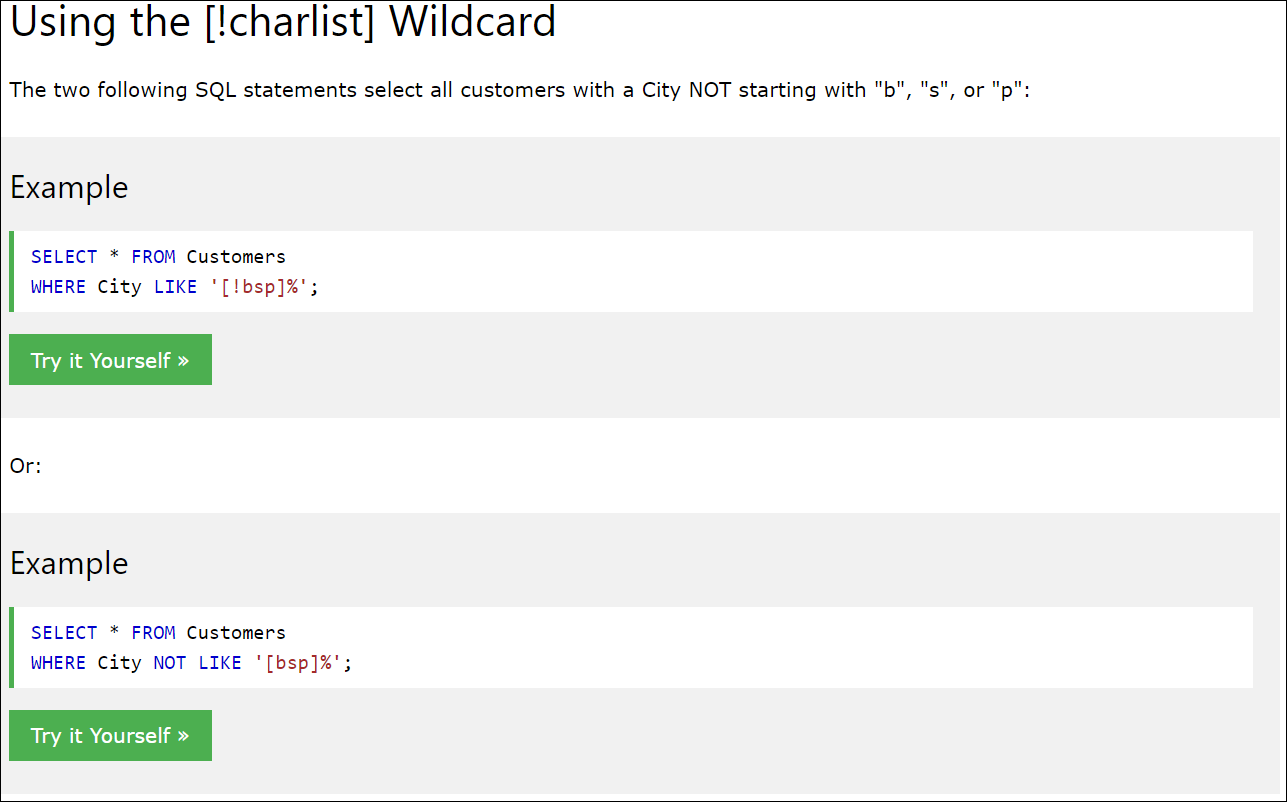
A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the [SQL LIKE](https://www.w3schools.com/sql/sql_like.asp) operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.









# SQL IN Operator: The IN operator allows you to specify multiple values in a WHERE clause.

IN: who are all the people in my contact list that are named john & Fritz)

Select first name from person where First name IN (‘john’,’Fritz’)



IS:

What are all the people in my contact list doesn’t have a last name?

Select first name from person where last name IS Null;

Is not:

What are all the people in my contact list have a last name?

Select first name from person where last name IS NOT Null;

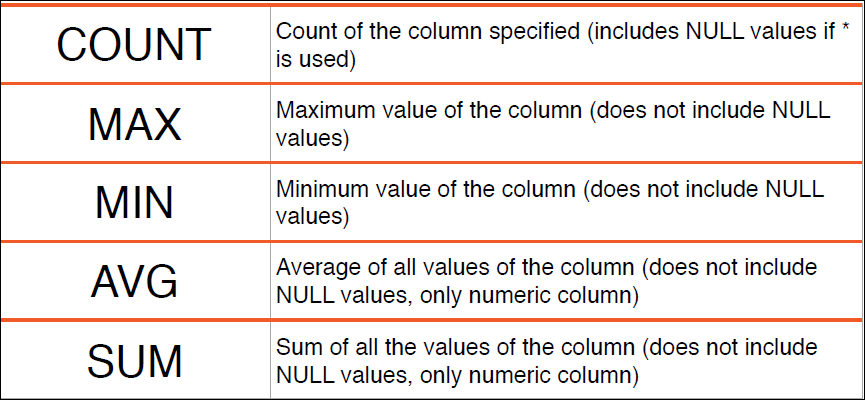
SQL Aliases: SQL aliases are used to give a table, or a column in a table, a temporary name.

SELECT column\_name AS alias\_name FROM table\_name;

SET functions:

Computer new values from columns values

SELECT sum (contact number) from person



# SQL MIN() and MAX() Functions :

# SELECT MIN(Price) AS SmallestPrice FROM Products;

# SELECT MAX(Price) AS LargestPrice FROM Products;

# SELECT COUNT(ProductID) FROM Products;

SELECT AVG(Price) FROM Products;

SELECT SUM(Quantity) FROM OrderDetails;

Set functions+ Qualifiers:

Select count (DISTINCT first Name) from person;

**SQL Joins:**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Let's look at a selection from the "Orders" table

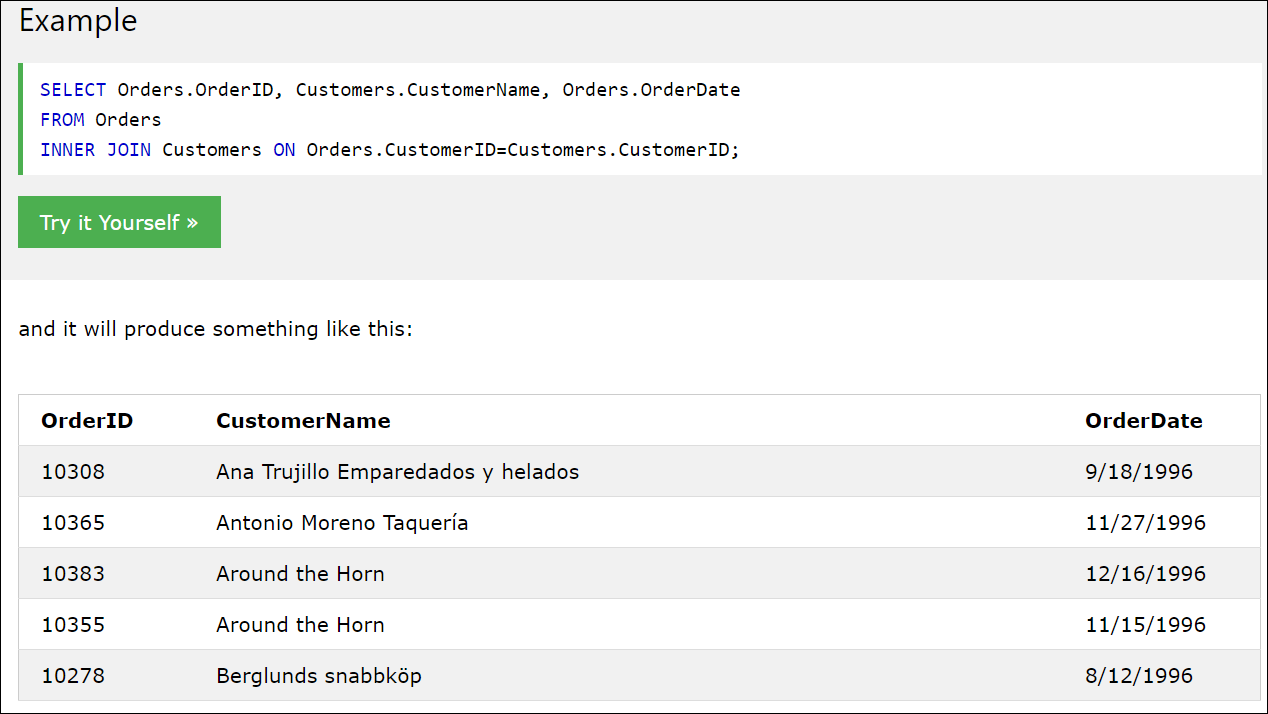
|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerID** | **OrderDate** |
| 10308 | 2 | 9/18/1996 |
| 10309 | 37 | 9/19/1996 |
| 10310 | 77 | 9/20/1996 |

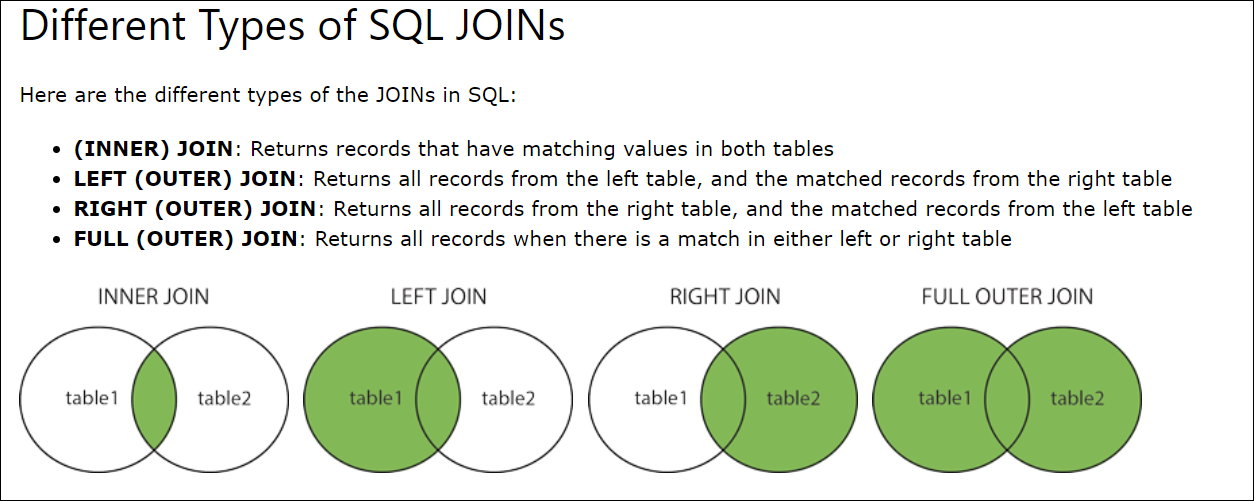
Then, look at a selection from the "Customers" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

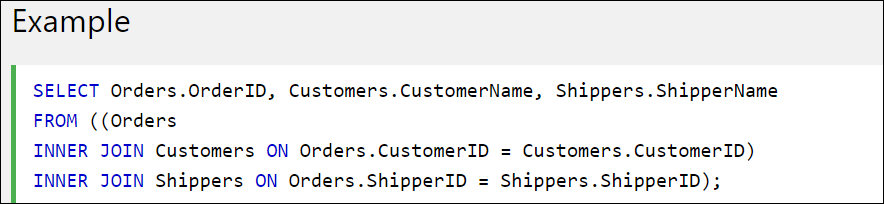
Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:





## JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:



## SQL LEFT JOIN Keyword:

## The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

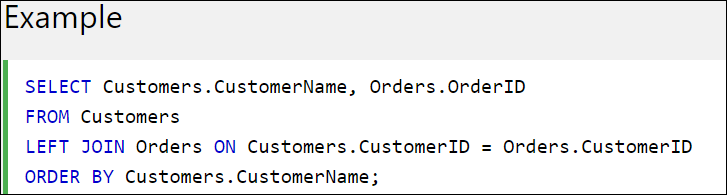
## Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.

## Below is a selection from the "Customers" table:

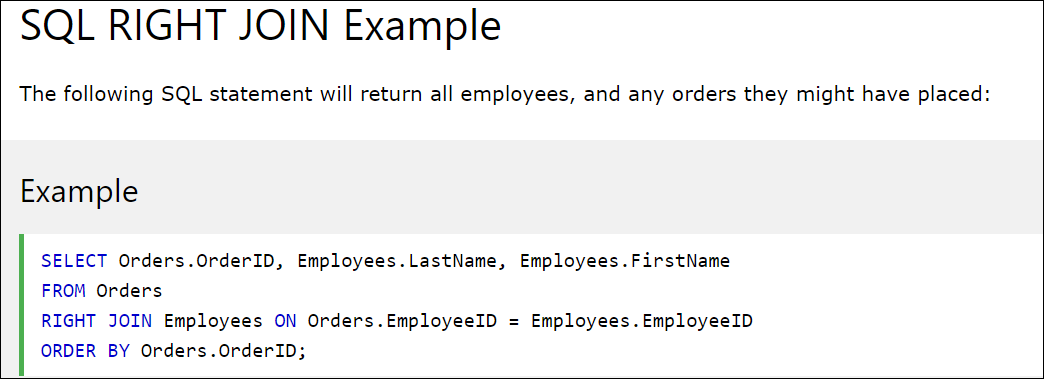
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Address** | **City** | **PostalCode** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Obere Str. 57 | Berlin | 12209 | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Avda. de la Constitución 2222 | México D.F. | 5021 | Mexico |

And a selection from the "Orders" table:

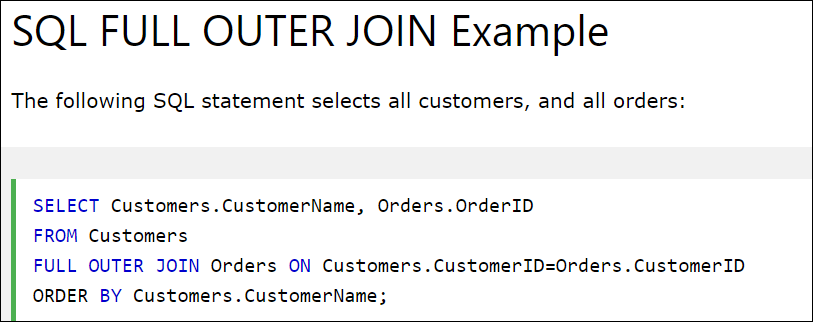
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **CustomerID** | **EmployeeID** | **OrderDate** | **ShipperID** |
| 10308 | 2 | 7 | 9/18/1996 | 3 |
| 10309 | 37 | 3 | 9/19/1996 | 1 |
| 10310 | 77 | 8 | 9/20/1996 | 2 |



**Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).



**Note:** The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).



**Note:** The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

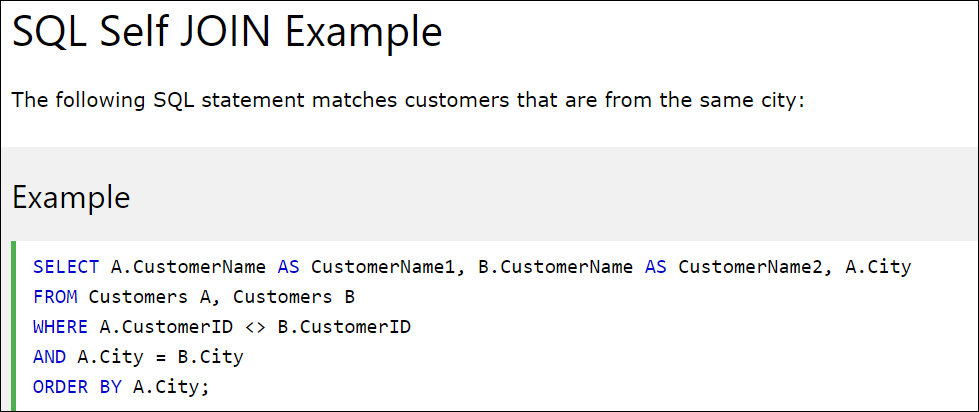
## SQL Self JOIN: A self-JOIN is a regular join, but the table is joined with itself.

## **Demo Database**

In this tutorial we will use the well-known Northwind sample database.

Below is a selection from 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. | 5021 | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mataderos 2312 | México D.F. | 5023 | Mexico |



Cross join: All rows from both tables

Group By:

Allows multiple columns with a set function

Brakes results into subsets

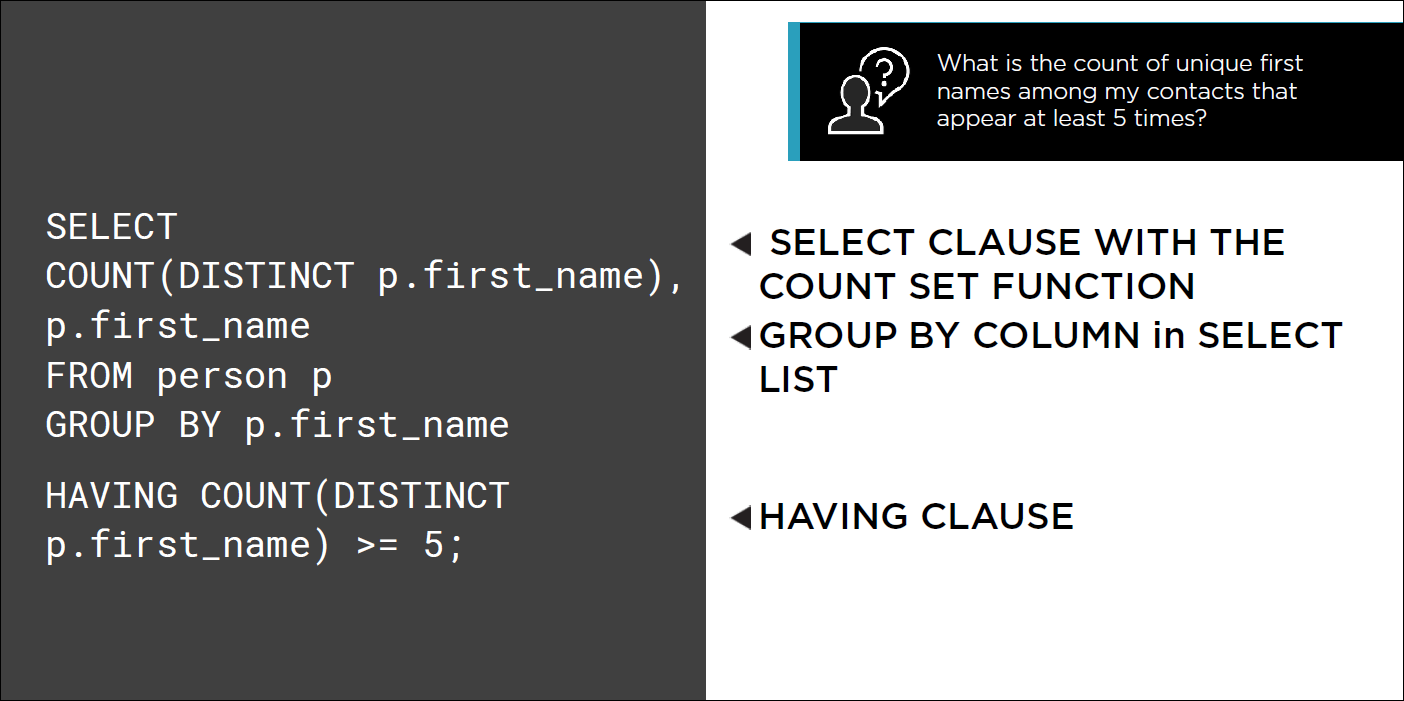
What are the count of every unique first name among my contacts?

Select count (first Name) from person group BY first name;

HAVING:

Works like WHERE work against SELECT

Restricts results set



**ssssData definition Language:**

# SQL CREATE DATABASE Statement

CREATE DATABASE testDB;

SHOW DATABASES;

DROP DATABASE testDB;

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

WITH DIFFERENTIAL;

## **The SQL CREATE TABLE Statement:** The CREATE TABLE statement is used to create a new table in a database

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

**SQL DROP TABLE Statement:** The following SQL statement drops the existing table "Shippers":

DROP TABLE Shippers;

## **SQL TRUNCATE TABLE:** The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

TRUNCATE TABLE Shippers;

## **SQL ALTER TABLE Statement:**

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

The following SQL adds an "Email" column to the "Customers" table:

ALTER TABLE Customers  
ADD Email varchar (255);

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE Customers  
DROP COLUMN Email;

To change the data type of a column in a table, use the following syntax:

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year;

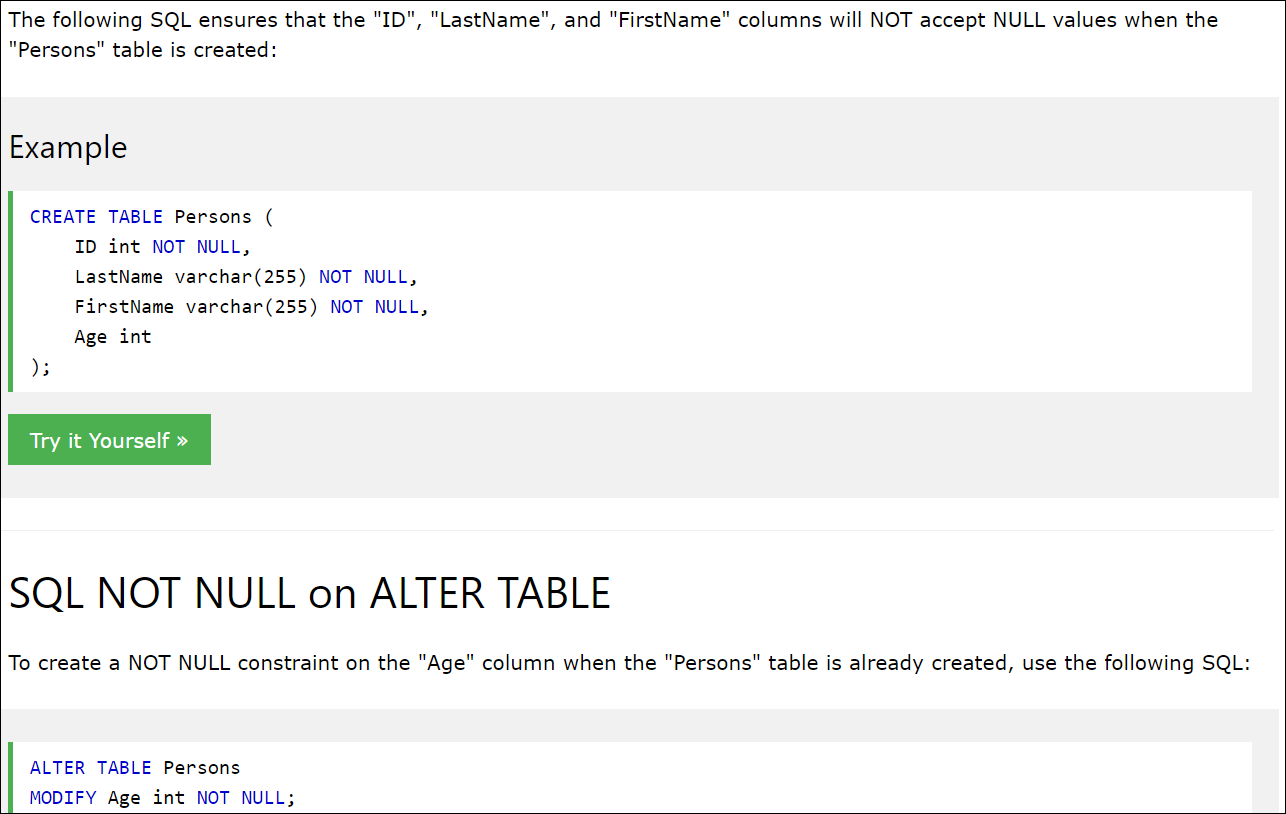
**SQL Constraints:** SQL constraints are used to specify rules for data in a table

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

* [NOT NULL](https://www.w3schools.com/sql/sql_notnull.asp) - Ensures that a column cannot have a NULL value

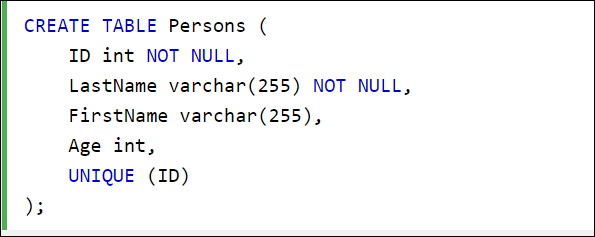


* [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different

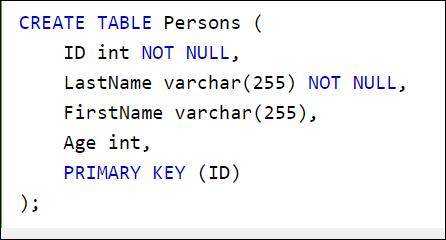
Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.



* [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table



* [FOREIGN KEY](https://www.w3schools.com/sql/sql_foreignkey.asp) - Uniquely identifies a row/record in another table

A FOREIGN KEY is a key used to link two tables together.

A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

Look at the following two tables:

"Persons" table:

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

"Orders" table:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **OrderNumber** | **PersonID** |
| 1 | 77895 | 3 |
| 2 | 44678 | 3 |
| 3 | 22456 | 2 |
| 4 | 24562 | 1 |

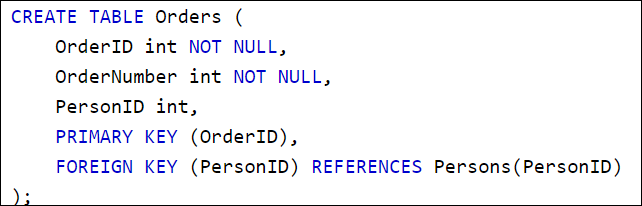
Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

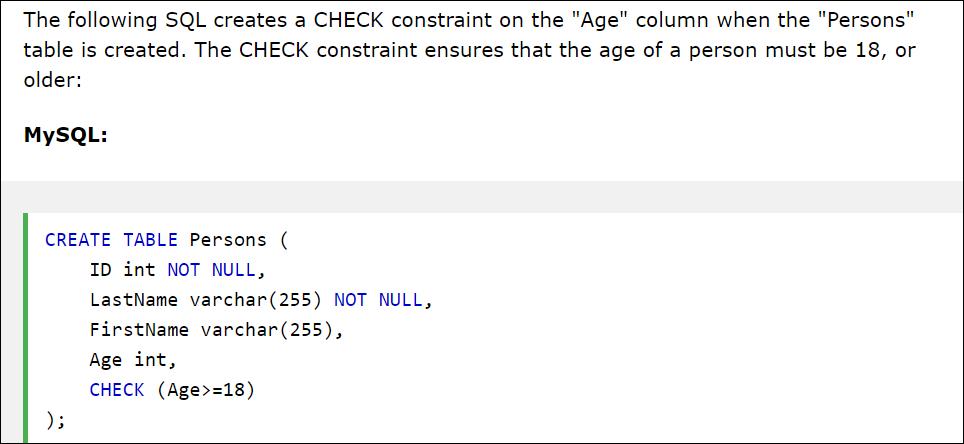


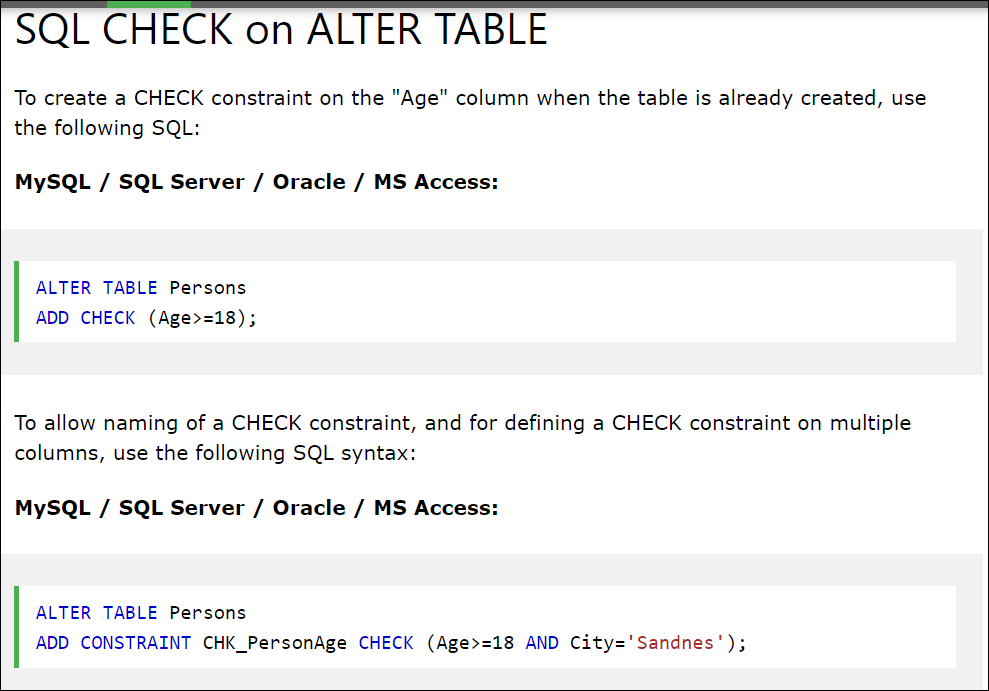
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that all values in a column satisfies a specific condition

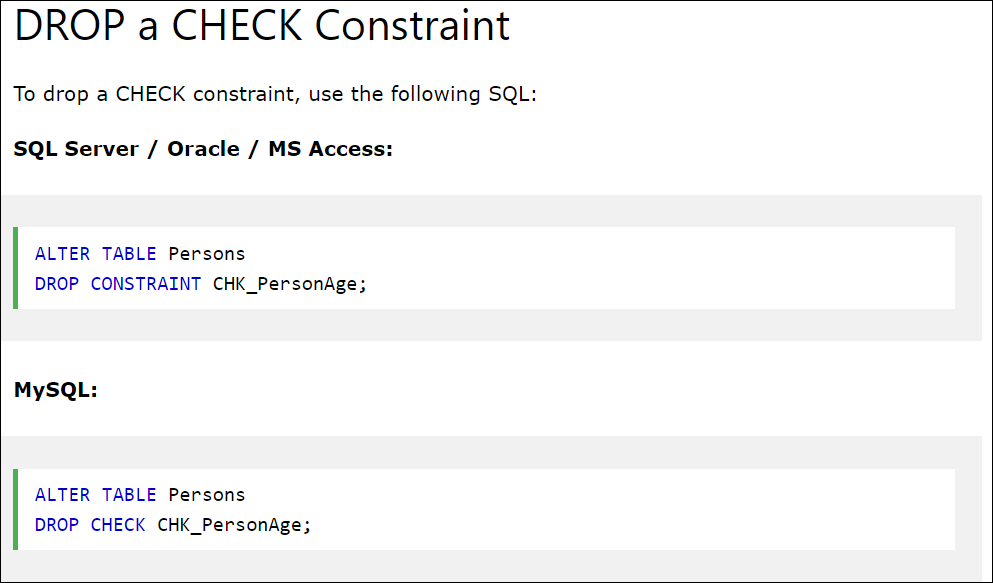
The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a single column it allows only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

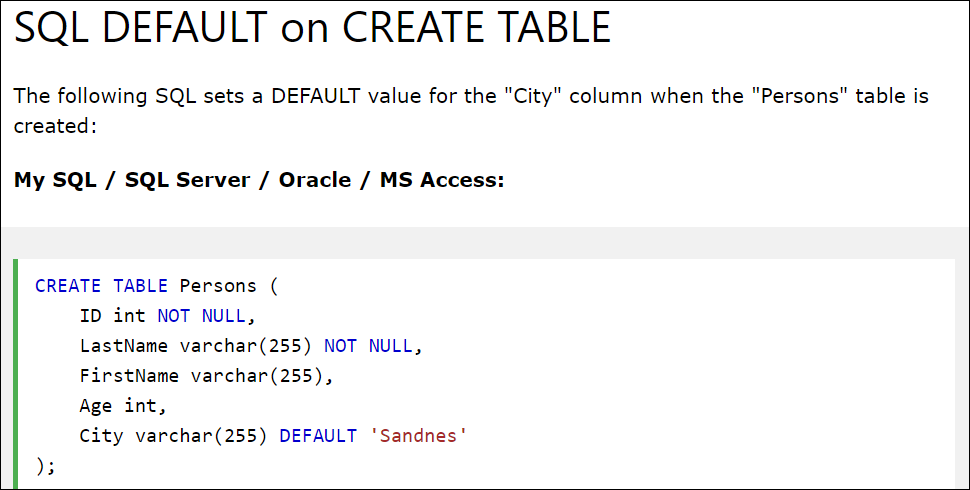


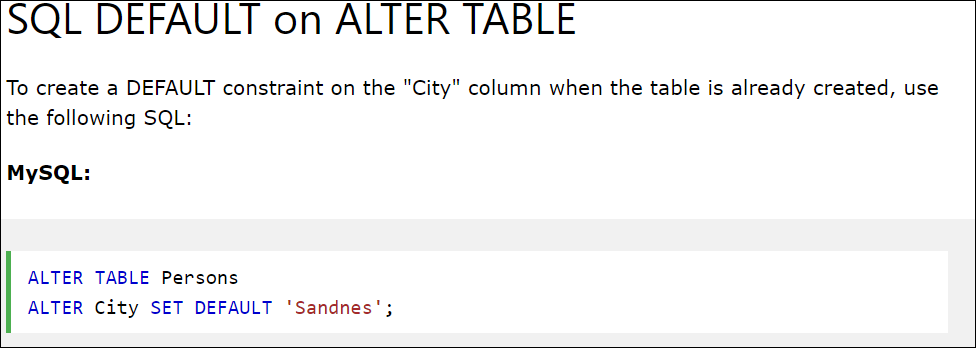


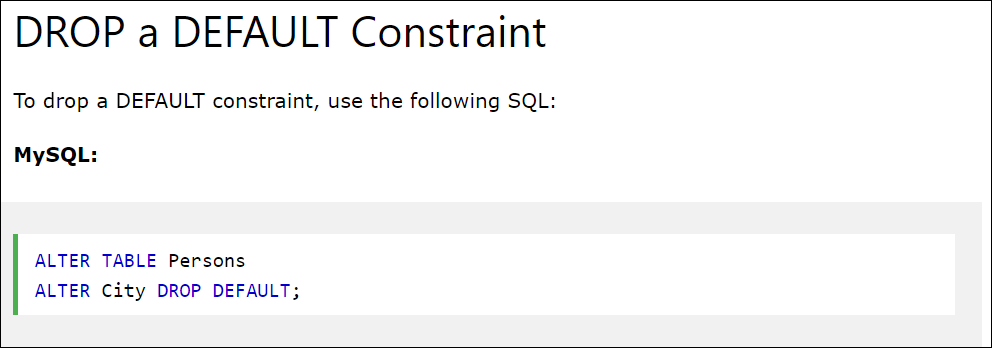


* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column when no value is specified

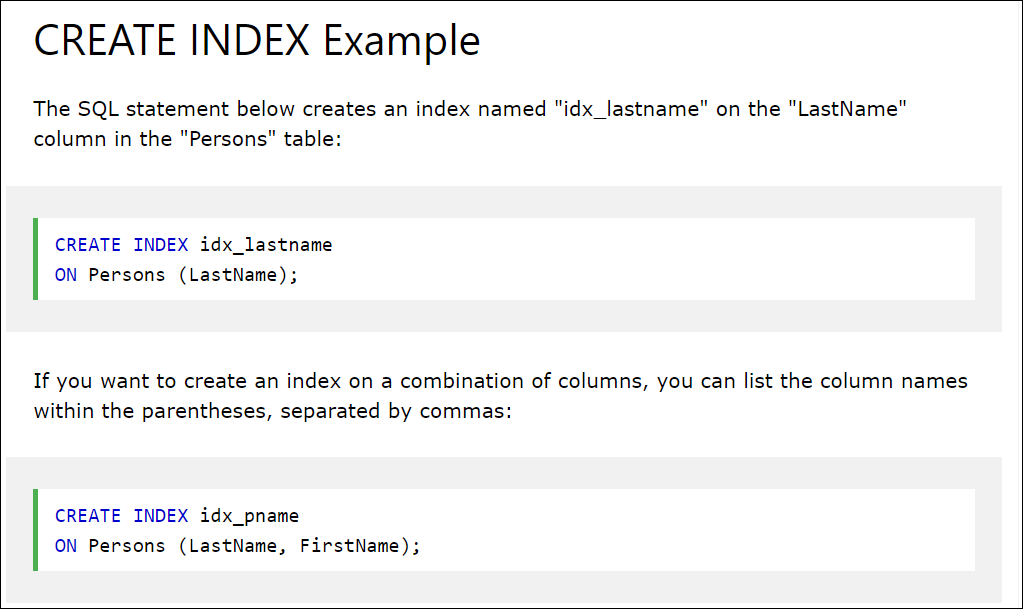
The default value will be added to all new records IF no other value is specified.







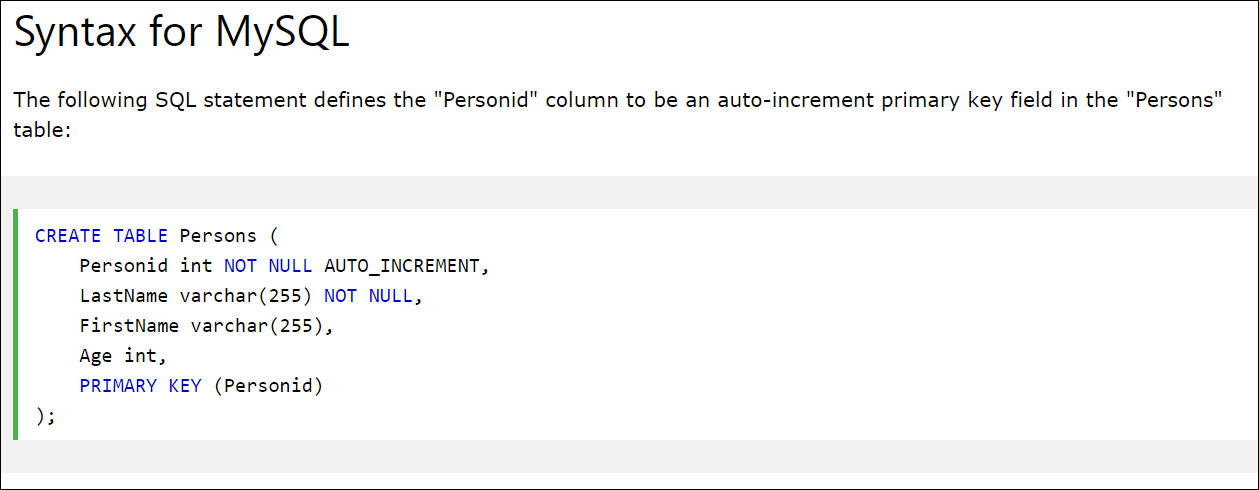
* [INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly

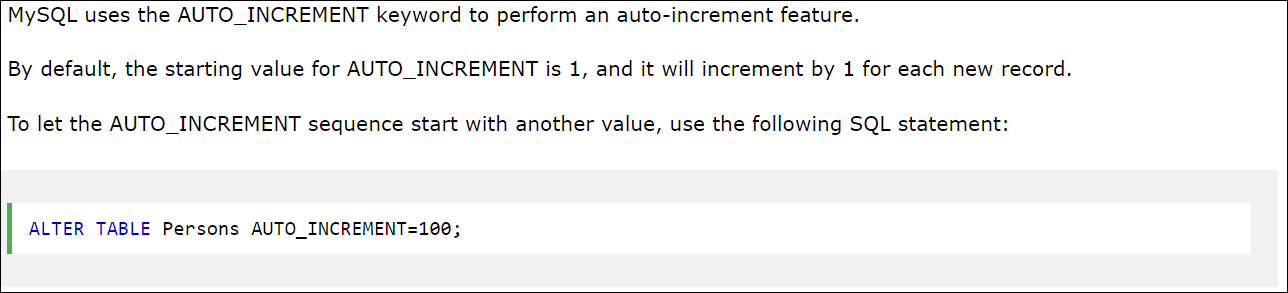


## **AUTO INCREMENT Field**

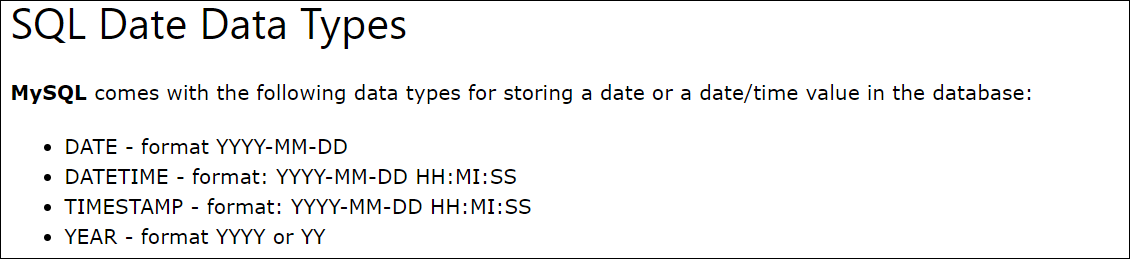
Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

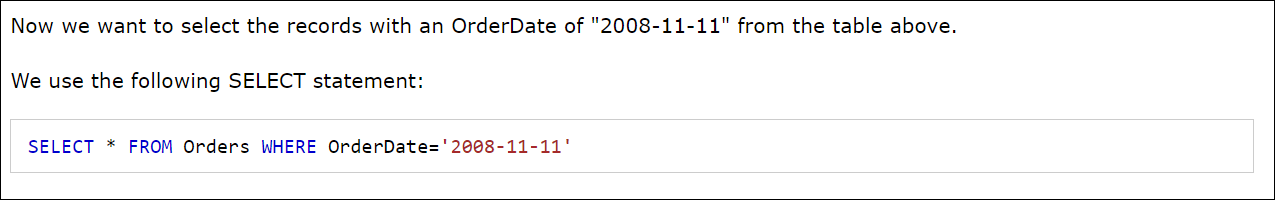
Often this is the primary key field that we would like to be created automatically every time a new record is inserted.





**SQL Dates**





SQL for Data science:

import pandas as pd

import sqlalchemy

import sqlite3

import PyMySQL

engine = sglalchemy.creat\_engine('mysql+pymysql://root:3306/application') # application - database Name

df = pd.read\_sql\_table(“Customers”, enginer, columns=[‘Name’])

query =’’’SELECT customers.name, customers.phonenumber, orders.name, orders. Amount

From Customer inner join orders

ON customers. ID = orders. Customer ID’’’

df= pr.read\_sql\_query(query, engine)

df

write back to SQL:from excel

df = pd.read\_csv(“Venkat.csv)

df

All the the names should match with table names

df.to\_sql(

name = “customers”, con = engine, index= False, if\_exists=’append’)

pd.read\_sql( it came used for both pd.read\_sql\_table, pd.read\_sql\_query)s

SQL for Data Science:

Order Table:

orderID, order Date, customer ID, product ID, sales, quantity

Customer table:

Customer ID, customer Name, city, state, postal\_code

Product Tabel:

Product ID, category, sub category,product

Libraries Install:

!pip install sqlalchemy

!pip install pymysql

From sqlalchemy import creat\_engine

Import pandas as pd

conn = creat\_engine('mysql+pymysql://username:password@df\_host/application') # application - database Name

conn.tabel\_names()

select data from orders table:

query=’SELECT\* from orders’

orders = pd.read\_sql(query,conn)

print(orders.shape)

orders

type(orders)

SQL Where condition:

query=’SELECT\* from products where category =” Furniture” and category =” Tech” ‘

products = pd.read\_sql(query,conn)

print(products.shape)

products

type(products)

SQL Joins:

(INNER join) : Return records that have matching value in both tables

Left(outer) join : returns all records from the left table and the matched records form the right table

Right(outer) join : Return all the records from the right table and the matched records from left table

SQL joins:

query=’SELECT\* from orders left join customers on oders.customer ID = customers.customer\_ID ,

data = pd.read\_sql(query,conn)

print(data.shape)

data

type(data)

R:

install.packages("RODBC")

library(RODBC)

connect <- odbcConnect("mysqlconnection")

my.data <- sqlQuery(connect, "SELECT \* FROM test.classroom")

my.data

SQL Project:

Extracting data from relational databases:

import package

# import sqlite3 package

import sqlite3

connect to database:

# create a connection to a database : create the database if not available

connection = sqlite3.connect("classroomDB.db")

connection.close()

create table:

# open connection

connection = sqlite3.connect("classroomDB.db")

# open cursor

cursor = connection.cursor()

# query for creating table

create\_table = """

CREATE TABLE classroom (

student\_id INTEGER PRIMARY KEY,

name VARCHAR(20),

gender CHAR(1),

physics\_marks INTEGER,

chemistry\_marks INTEGER,

mathematics\_marks INTEGER

);"""

# execute query

cursor.execute(create\_table)

# commit changes

connection.commit()

# close connection

connection.close()

insert data:

# sample data

classroom\_data = [( 1, "Raj","M", 70, 84, 92),

( 2, "Poonam","F", 87, 69, 93),

( 3, "Nik","M", 65, 83, 90),

( 4, "Rahul","F", 83, 76, 89)]

# open connection

connection = sqlite3.connect("classroomDB.db")

# open cursor

cursor = connection.cursor()

# insert each student record

for student in classroom\_data:

# formatted query string

insert\_statement = """INSERT INTO classroom

(student\_id, name, gender, physics\_marks, chemistry\_marks, mathematics\_marks)

VALUES

({0}, "{1}", "{2}", {3}, {4}, {5});""".format(student[0], student[1], student[2],

student[3],student[4], student[5])

# execute insert query

cursor.execute(insert\_statement)

# commit the changes

connection.commit()

# close the connection

connection.close()

insert data:

# sample data

classroom\_data = [( 1, "Raj","M", 70, 84, 92),

( 2, "Poonam","F", 87, 69, 93),

( 3, "Nik","M", 65, 83, 90),

( 4, "Rahul","F", 83, 76, 89)]

# open connection

connection = sqlite3.connect("classroomDB.db")

# open cursor

cursor = connection.cursor()

# insert each student record

for student in classroom\_data:

# formatted query string

insert\_statement = """INSERT INTO classroom

(student\_id, name, gender, physics\_marks, chemistry\_marks, mathematics\_marks)

VALUES

({0}, "{1}", "{2}", {3}, {4}, {5});""".format(student[0], student[1], student[2],

student[3],student[4], student[5])

# execute insert query

cursor.execute(insert\_statement)

# commit the changes

connection.commit()

# close the connection

connection.close()

extract data:

# open connection

connection = sqlite3.connect("classroomDB.db")

# open cursor

cursor = connection.cursor()

# query

query = "SELECT \* FROM classroom"

# execute query

cursor.execute(query)

# fetch results

result = cursor.fetchall()

# print results

for row in result:

print(row)

# close connection

connection.close()

MySQL database

install package

!conda install -y -q pymysql

# import package

import pymysql

connect to database

# Connection details

cnx= {'host': 'mysqldb.c12wj3xlqsae.us-west-2.rds.amazonaws.com',

'username': 'test',

'password': 'test123456',

'db': 'classroomDB'}

# Connect to the database

connection = pymysql.connect(cnx['host'],cnx['username'],cnx['password'],cnx['db'] )

# close database

connection.close()

create table:

# open connection

connection = pymysql.connect(cnx['host'],cnx['username'],cnx['password'],cnx['db'] )

# open cursor

cursor = connection.cursor()

# query for creating table

create\_table = """

CREATE TABLE classroom (

student\_id INTEGER PRIMARY KEY,

name VARCHAR(20),

gender CHAR(1),

physics\_marks INTEGER,

chemistry\_marks INTEGER,

mathematics\_marks INTEGER

);"""

# execute query

cursor.execute(create\_table)

# commit changes

connection.commit()

# close connection

connection.close()

insert data:

# sample data

classroom\_data = [( 1, "Raj","M", 70, 84, 92),

( 2, "Poonam","F", 87, 69, 93),

( 3, "Nik","M", 65, 83, 90),

( 4, "Rahul","F", 83, 76, 89)]

# open connection

connection = pymysql.connect(cnx['host'],cnx['username'],cnx['password'],cnx['db'] )

# open cursor

cursor = connection.cursor()

# insert each student record

for student in classroom\_data:

# formatted query string

insert\_statement = """INSERT INTO classroom

(student\_id, name, gender, physics\_marks, chemistry\_marks, mathematics\_marks)

VALUES

({0}, "{1}", "{2}", {3}, {4}, {5});""".format(student[0], student[1], student[2],

student[3],student[4], student[5])

# execute insert query

cursor.execute(insert\_statement)

# commit the changes

connection.commit()

# close the connection

connection.close()

extract data:

# open connection

connection = pymysql.connect(cnx['host'],cnx['username'],cnx['password'],cnx['db'] )

# open cursor

cursor = connection.cursor()

# query

query = "SELECT \* FROM classroom"

# execute query

cursor.execute(query)

# fetch results

result = cursor.fetchall()

# print results

for row in result:

print(row)

# close connection

connection.close()

Microsoft SQL Server database

install package

!conda install -y -q pymssql

import pymssql

connect to database

cnx= {

'host': 'mssqldb.c12wj3xlqsae.us-west-2.rds.amazonaws.com:1433',

'username': 'test',

'password': 'test123456',

'db': 'tempDB'}

conn = pymssql.connect(cnx['host'], cnx['username'], cnx['password'], cnx['db'])

conn.close()

create table

# open connection

connection = pymssql.connect(cnx['host'], cnx['username'], cnx['password'], cnx['db'])

# open cursor

cursor = connection.cursor()

# query for creating table

create\_table = """

CREATE TABLE classroom (

student\_id INTEGER PRIMARY KEY,

name VARCHAR(20),

gender CHAR(1),

physics\_marks INTEGER,

chemistry\_marks INTEGER,

mathematics\_marks INTEGER

);"""

# execute query

cursor.execute(create\_table)

# commit changes

connection.commit()

# close connection

connection.close()

insert data:

# sample data

classroom\_data = [( 1, "Raj","M", 70, 84, 92),

( 2, "Poonam","F", 87, 69, 93),

( 3, "Nik","M", 65, 83, 90),

( 4, "Rahul","F", 83, 76, 89)]

# open connection

connection = pymssql.connect(cnx['host'],cnx['username'],cnx['password'],cnx['db'] )

# open cursor

cursor = connection.cursor()

# insert each student record

for student in classroom\_data:

# formatted query string

insert\_statement = """INSERT INTO classroom

(student\_id, name, gender, physics\_marks, chemistry\_marks, mathematics\_marks)

VALUES

({0}, '{1}', '{2}', {3}, {4}, {5});""".format(student[0], student[1], student[2],

student[3],student[4], student[5])

# execute insert query

cursor.execute(insert\_statement)

# commit the changes

connection.commit()

# close the connection

connection.close()

extract data:

# open connection

connection = pymssql.connect(cnx['host'],cnx['username'],cnx['password'],cnx['db'] )

# open cursor

cursor = connection.cursor()

# query

query = "SELECT \* FROM classroom"

# execute query

cursor.execute(query)

# fetch results

result = cursor.fetchall()

# print results

for row in result:

print(row)

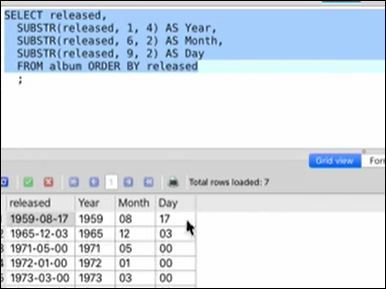
# close connection

connection.close()

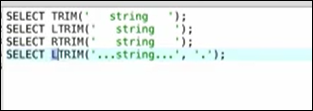
Strings:

SELECT SUBSTR (‘This string’, 6,3);

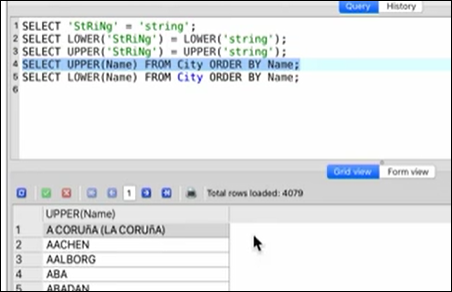
6 defines index string start and 3(end)

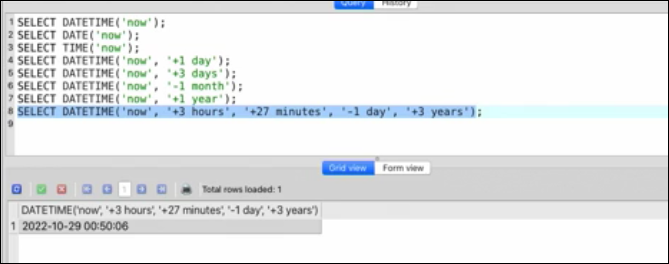


Remove spaces:



Folding case:





Python: