SQL Statements

Most of the actions you need to perform on a database are done with SQL statements.

SQL statements consists of keywords that are easy to understand.

The following SQL statement returns all records from a table named "Customers":

Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

Select all records from the Customers table:

SELECT \* FROM Customers;

In this tutorial we will teach you all about the different SQL statements.

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"), and contain records (rows) with data.

In this tutorial we will use the well-known Northwind sample database (included in MS Access and MS SQL Server).

Below is a selection from the [Customers](https://www.w3schools.com/sql/trysql.asp?filename=trysql_customers) table used in the 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 |

The table above contains five records (one for each customer) and seven columns (CustomerID, CustomerName, ContactName, Address, City, PostalCode, and Country).

Keep in Mind That...

* SQL keywords are NOT case sensitive: select is the same as SELECT

In this tutorial we will write all SQL keywords in upper-case.

Semicolon after SQL Statements?

Some database systems require a semicolon at the end of each SQL statement.

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

In this tutorial, we will use semicolon at the end of each SQL statement.

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

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table\_name* (*column1*, *column2*, *column3*, ...)  
VALUES (*value1*, *value2*, *value3*, ...);

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table\_name*  
VALUES (*value1*, *value2*, *value3*, ...);

Demo Database

Below is a selection from the [Customers](https://www.w3schools.com/sql/trysql.asp?filename=trysql_customers) table used in the examples:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |

INSERT INTO Example

The following SQL statement inserts a new record in the "Customers" table:

Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

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

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 92 | Cardinal | Tom B. Erichsen | Skagen 21 | Stavanger | 4006 | Norway |

Did you notice that we did not insert any number into the CustomerID field?  
The CustomerID column is an [auto-increment](https://www.w3schools.com/sql/sql_autoincrement.asp) field and will be generated automatically when a new record is inserted into the table.

Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

Example

INSERT INTO Customers (CustomerName, City, Country)  
VALUES ('Cardinal', 'Stavanger', 'Norway');

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 92 | Cardinal | null | null | Stavanger | null | Norway |

Insert Multiple Rows

It is also possible to insert multiple rows in one statement.

To insert multiple rows of data, we use the same INSERT INTO statement, but with multiple values:

Example

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)  
VALUES  
('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),  
('Greasy Burger', 'Per Olsen', 'Gateveien 15', 'Sandnes', '4306', 'Norway'),  
('Tasty Tee', 'Finn Egan', 'Streetroad 19B', 'Liverpool', 'L1 0AA', 'UK');

Make sure you separate each set of values with a comma ,.

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 92 | Cardinal | Tom B. Erichsen | Skagen 21 | Stavanger | 4006 | Norway |
| 93 | Greasy Burger | Per Olsen | Gateveien 15 | Sandnes | 4306 | Norway |
| 94 | Tasty Tee | Finn Egan | Streetroad 19B | Liverpool | L1 0AA | UK |

The SQL CREATE DATABASE Statement

The CREATE DATABASE statement is used to create a new SQL database.

Syntax

CREATE DATABASE *databasename*;

# SQL DROP DATABASE Statement

## The SQL DROP DATABASE Statement

The DROP DATABASE statement is used to drop an existing SQL database.

### Syntax

DROP DATABASE databasename;

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.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

ALTER TABLE *table\_name*  
ADD *column\_name datatype*;

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

Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

ALTER TABLE Customers  
ADD Email varchar(255);

ALTER TABLE - DROP COLUMN

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

ALTER TABLE *table\_name*  
DROP COLUMN *column\_name*;

The following SQL deletes the "Email" column from the "Customers" table:

Example

ALTER TABLE Customers  
DROP COLUMN Email;

ALTER TABLE - RENAME COLUMN

To rename a column in a table, use the following syntax:

ALTER TABLE *table\_name*  
RENAME COLUMN *old\_name* to *new\_name*;

ALTER TABLE - ALTER/MODIFY DATATYPE

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

SQL Server / MS Access:

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

My SQL / Oracle (prior version 10G):

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

Oracle 10G and later:

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

SQL ALTER TABLE Example

Look at the "Persons" table:

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

Now we want to add a column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

ALTER TABLE Persons  
ADD DateOfBirth date;

Notice that the new column, "DateOfBirth", is of type date and is going to hold a date. The data type specifies what type of data the column can hold. For a complete reference of all the data types available in MS Access, MySQL, and SQL Server, go to our complete [Data Types reference](https://www.w3schools.com/sql/sql_datatypes.asp).

The "Persons" table will now look like this:

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

Change Data Type Example

Now we want to change the data type of the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

ALTER TABLE Persons  
ALTER COLUMN DateOfBirth year;

Notice that the "DateOfBirth" column is now of type year and is going to hold a year in a two- or four-digit format.

DROP COLUMN Example

Next, we want to delete the column named "DateOfBirth" in the "Persons" table.

We use the following SQL statement:

ALTER TABLE Persons  
DROP COLUMN DateOfBirth;

The "Persons" table will now look like this:

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

The SQL DELETE Statement

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

DELETE Syntax

DELETE FROM *table\_name* WHERE *condition*;

Note: 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!

Demo Database

Below is a selection from the [Customers](https://www.w3schools.com/sql/trysql.asp?filename=trysql_customers) table used in the 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 |

SQL DELETE Example

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

The "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CustomerID | CustomerName | ContactName | Address | City | PostalCode | Country |
| 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 |

Delete All Records

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

DELETE FROM *table\_name*;

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example

DELETE FROM Customers;

Delete a Table

To delete the table completely, use the DROP TABLE statement:

Example

Remove the Customers table:

DROP TABLE Customers;

SQL Create Constraints

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

Syntax

CREATE TABLE *table\_name* (  
*column1 datatype* *constraint*,  
*column2 datatype* *constraint*,  
*column3 datatype* *constraint*,  
    ....  
);

SQL Constraints

SQL constraints are used to specify rules for the 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
* [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) - Prevents actions that would destroy links between tables
* [CHECK](https://www.w3schools.com/sql/sql_check.asp) - Ensures that the values in a column satisfies a specific condition
* [DEFAULT](https://www.w3schools.com/sql/sql_default.asp) - Sets a default value for a column if no value is specified
* [CREATE INDEX](https://www.w3schools.com/sql/sql_create_index.asp) - Used to create and retrieve data from the database very quickly
* SQL NOT NULL Constraint
* By default, a column can hold NULL values.
* The NOT NULL constraint enforces a column to NOT accept NULL values.
* This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.
* SQL NOT NULL on CREATE TABLE
* The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:
* Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)
* CREATE TABLE Persons (  
      ID int NOT NULL,  
      LastName varchar(255) NOT NULL,  
      FirstName varchar(255) NOT NULL,  
      Age int  
  );
* SQL NOT NULL on ALTER TABLE
* To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:
* SQL Server / MS Access:
* ALTER TABLE Persons  
  ALTER COLUMN Age int NOT NULL;
* My SQL / Oracle (prior version 10G):
* ALTER TABLE Persons  
  MODIFY COLUMN Age int NOT NULL;
* Oracle 10G and later:
* ALTER TABLE Persons  
  MODIFY Age int NOT NULL;

SQL UNIQUE Constraint

The UNIQUE constraint 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.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

SQL Server / Oracle / MS Access:

CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    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,  
    UNIQUE (ID)  
);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT UC\_Person UNIQUE (ID,LastName)  
);

SQL UNIQUE Constraint on ALTER TABLE

To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons  
ADD UNIQUE (ID);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons  
ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName);

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

MySQL:

ALTER TABLE Persons  
DROP INDEX UC\_Person;

SQL Server / Oracle / MS Access:

ALTER TABLE Persons  
DROP CONSTRAINT UC\_Person;

SQL PRIMARY KEY Constraint

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

MySQL:

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

SQL Server / Oracle / MS Access:

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

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

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: In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

SQL PRIMARY KEY on ALTER TABLE

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons  
ADD PRIMARY KEY (ID);

To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons  
ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

Note: If you use ALTER TABLE to add a primary key, the primary key column(s) must have been declared to not contain NULL values (when the table was first created).

DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

MySQL:

ALTER TABLE Persons  
DROP PRIMARY KEY;

SQL Server / Oracle / MS Access:

ALTER TABLE Persons  
DROP CONSTRAINT PK\_Person;

SQL FOREIGN KEY Constraint

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

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table.

The table with the foreign key is called the child table, and the table with the primary 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 prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

MySQL:

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

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)  
);

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID)  
    REFERENCES Persons(PersonID)  
);

SQL FOREIGN KEY on ALTER TABLE

To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Orders  
ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Orders  
ADD CONSTRAINT FK\_PersonOrder  
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL:

MySQL:

ALTER TABLE Orders  
DROP FOREIGN KEY FK\_PersonOrder;

SQL Server / Oracle / MS Access:

ALTER TABLE Orders  
DROP CONSTRAINT FK\_PersonOrder;

SQL Data Types

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

Note: Data types might have different names in different database. And even if the name is the same, the size and other details may be different! Always check the documentation!

MySQL Data Types (Version 8.0)

In MySQL there are three main data types: string, numeric, and date and 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 string 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 |

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) |

Note: All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

Date and 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. |

SQL Server Data Types

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 |  |

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 |

Date and 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 |  |

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 |

MS Access Data Types

|  |  |  |
| --- | --- | --- |
| 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 |

The SQL UPDATE Statement

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

UPDATE Syntax

UPDATE *table\_name*  
SET *column1* = *value1*, *column2* = *value2*, ...  
WHERE *condition*;

Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

Demo Database

Below is a selection from the [Customers](https://www.w3schools.com/sql/trysql.asp?filename=trysql_customers) table used in the 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 |

UPDATE Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

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

The selection from 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 |

UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

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

Example

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

The selection from 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 | Juan | Avda. de la Constitución 2222 | México D.F. | 05021 | Mexico |
| 3 | Antonio Moreno Taquería | Juan | 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 |

Update Warning!

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

Example

UPDATE Customers  
SET ContactName='Juan';

The selection from the "Customers" table will now look like this:

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

The SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO *table\_name* (*column1*, *column2*, *column3*, ...)  
VALUES (*value1*, *value2*, *value3*, ...);

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table\_name*  
VALUES (*value1*, *value2*, *value3*, ...);

Demo Database

Below is a selection from the [Customers](https://www.w3schools.com/sql/trysql.asp?filename=trysql_customers) table used in the examples:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |

INSERT INTO Example

The following SQL statement inserts a new record in the "Customers" table:

Example[Get your own SQL Server](https://www.w3schools.com/sql/sql_server.asp)

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

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 92 | Cardinal | Tom B. Erichsen | Skagen 21 | Stavanger | 4006 | Norway |

Did you notice that we did not insert any number into the CustomerID field?  
The CustomerID column is an [auto-increment](https://www.w3schools.com/sql/sql_autoincrement.asp) field and will be generated automatically when a new record is inserted into the table.

Insert Data Only in Specified Columns

It is also possible to only insert data in specific columns.

The following SQL statement will insert a new record, but only insert data in the "CustomerName", "City", and "Country" columns (CustomerID will be updated automatically):

Example

INSERT INTO Customers (CustomerName, City, Country)  
VALUES ('Cardinal', 'Stavanger', 'Norway');

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 92 | Cardinal | null | null | Stavanger | null | Norway |

Insert Multiple Rows

It is also possible to insert multiple rows in one statement.

To insert multiple rows of data, we use the same INSERT INTO statement, but with multiple values:

Example

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)  
VALUES  
('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),  
('Greasy Burger', 'Per Olsen', 'Gateveien 15', 'Sandnes', '4306', 'Norway'),  
('Tasty Tee', 'Finn Egan', 'Streetroad 19B', 'Liverpool', 'L1 0AA', 'UK');

Make sure you separate each set of values with a comma ,.

The selection from the "Customers" table will now look like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 92 | Cardinal | Tom B. Erichsen | Skagen 21 | Stavanger | 4006 | Norway |
| 93 | Greasy Burger | Per Olsen | Gateveien 15 | Sandnes | 4306 | Norway |
| 94 | Tasty Tee | Finn Egan | Streetroad 19B | Liverpool | L1 0AA | UK |

Certainly! In a relational database management system (RDBMS) like MySQL, the Data Definition Language (DDL) is used to define and manage the structure of the database, including tables, indexes, and other schema objects. Here are some common DDL statements along with examples for creating and managing a database in MySQL using phpMyAdmin:

### 1. \*\*Creating a Database:\*\*

CREATE DATABASE IF NOT EXISTS your\_database\_name;

### 2. \*\*Selecting a Database:\*\*

USE your\_database\_name;

### 3. \*\*Creating a Table:\*\*

CREATE TABLE your\_table\_name (

column1 datatype1,

column2 datatype2,

...

PRIMARY KEY (one\_or\_more\_columns)

);

Example:

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL,

email VARCHAR(100) NOT NULL,

birthdate DATE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

### 4. \*\*Altering a Table (Adding a Column):\*\*

ALTER TABLE your\_table\_name

ADD COLUMN new\_column\_name datatype;

Example:

ALTER TABLE users

ADD COLUMN phone\_number VARCHAR(20);

### 5. \*\*Dropping a Column:\*\*

ALTER TABLE your\_table\_name

DROP COLUMN column\_name;

Example:

ALTER TABLE users

DROP COLUMN phone\_number;

### 6. \*\*Creating an Index:\*\*

CREATE INDEX index\_name

ON your\_table\_name (column1, column2, ...);

Example:

CREATE INDEX idx\_username

ON users (username);

### 7. \*\*Dropping an Index:\*\*

DROP INDEX index\_name

ON your\_table\_name;

Example:

DROP INDEX idx\_username

ON users;

### 8. \*\*Renaming a Table:\*\*

RENAME TABLE old\_table\_name TO new\_table\_name;

Example:

RENAME TABLE users TO customers;

### 9. \*\*Dropping a Table:\*\*

DROP TABLE your\_table\_name;

Example:

DROP TABLE customers;

### Important Notes:

- Make sure to replace `your\_database\_name`, `your\_table\_name`, etc., with your actual database and table names.

- Adjust the column names and data types according to your specific requirements.

- Use `IF NOT EXISTS` to avoid errors if the database or table already exists.

- The examples assume a basic understanding of MySQL data types and constraints.

Remember to handle sensitive information, such as database credentials, securely in real-world applications.

### 1. \*\*SELECT Statement:\*\*

SELECT column1, column2, ...

FROM your\_table\_name

WHERE condition;

Example:

SELECT id, username, email

FROM users

WHERE id = 1;

### 2. \*\*INSERT Statement:\*\*

INSERT INTO your\_table\_name (column1, column2, ...)

VALUES (value1, value2, ...);

Example:

INSERT INTO users (username, email)

VALUES ('john\_doe', 'john.doe@example.com');

### 3. \*\*UPDATE Statement:\*\*

UPDATE your\_table\_name

SET column1 = value1, column2 = value2, ...

WHERE condition;

Example:

UPDATE users

SET email = 'new.email@example.com'

WHERE username = 'john\_doe';

### 4. \*\*DELETE Statement:\*\*

DELETE FROM your\_table\_name

WHERE condition;

Example:

DELETE FROM users

WHERE id = 1;

### 5. \*\*INSERT with Subquery:\*\*

INSERT INTO your\_table\_name (column1, column2, ...)

SELECT column1, column2, ...

FROM another\_table

WHERE condition;

Example:

INSERT INTO archived\_users (id, username, email)

SELECT id, username, email

FROM users

WHERE created\_at < '2022-01-01';

### 6. \*\*UPDATE with JOIN:\*\*

UPDATE table1

SET column1 = value

FROM table1

JOIN table2 ON table1.id = table2.id

WHERE condition;

Example:

UPDATE orders

SET status = 'Shipped'

FROM orders

JOIN customers ON orders.customer\_id = customers.id

WHERE customers.country = 'USA';

### 7. \*\*DELETE with JOIN:\*\*

DELETE table1

FROM table1

JOIN table2 ON table1.id = table2.id

WHERE condition;

Example:

DELETE users

FROM users

JOIN archived\_users ON users.id = archived\_users.id

WHERE archived\_users.created\_at < '2022-01-01';

These are basic examples of common DML statements. Make sure to replace `your\_table\_name`, `your\_column\_name`, etc., with your actual table and column names. Adjust conditions and values based on your specific requirements. Always be cautious when using `DELETE` statements to avoid unintended data loss.