

My sql

Data Types

- 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 <i>size</i> 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 <i>size</i> 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 <i>size</i> parameter specifies the column length in bytes. Default is 1
VARBINARY(size)	Equal to VARCHAR(), but stores binary byte strings. The <i>size</i> 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(<i>size</i>)	A bit-value type. The number of bits per value is specified in <i>size</i> . The <i>size</i> parameter can hold a value from 1 to 64. The default value for <i>size</i> is 1.
TINYINT(<i>size</i>)	A very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The <i>size</i> 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(<i>size</i>)	A small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The <i>size</i> parameter specifies the maximum display width (which is 255)
MEDIUMINT(<i>size</i>)	A medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The <i>size</i> parameter specifies the maximum display width (which is 255)

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(<i>fsp</i>)	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(<i>fsp</i>)	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(<i>fsp</i>)	A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'

The MySQL CREATE DATABASE Statement

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

```
CREATE DATABASE databasename;
```

Example

```
CREATE DATABASE testDB;
```


The MySQL DROP DATABASE Statement

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

```
DROP DATABASE databaseName;
```

Example

```
DROP DATABASE testDB;
```

The MySQL CREATE TABLE Statement

- The CREATE TABLE statement is used to create a new table in a database.

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

- The column parameters specify the names of the columns of the table.
- The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

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

- The PersonID column is of type int and will hold an integer.
- The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.
- The empty "Persons" table will now look like this:

PersonID	LastName	FirstName	Address	City

Tip: The empty "Persons" table can now be filled with data with the SQL [INSERT INTO](#) statement.

Create Table Using Another Table

- A copy of an existing table can also be created using CREATE TABLE.
- The new table gets the same column definitions. All columns or specific columns can be selected.
- If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax

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

Example

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

MySQL DROP TABLE Statement

- The DROP TABLE statement is used to drop an existing table in a database.

```
DROP TABLE table_name;
```

Example

```
DROP TABLE Shippers;
```

MySQL TRUNCATE TABLE

- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

```
TRUNCATE TABLE table_name;
```

MySQL 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

```
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 - MODIFY COLUMN

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

```
ALTER TABLE table_name  
MODIFY COLUMN column_name datatype;
```

MySQL 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:

Example

```
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.

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:

Example

```
ALTER TABLE Persons  
MODIFY 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:

Example

```
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

MySQL Constraints

SQL constraints are used to specify rules for data in a table.

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,  
    ....  
);
```

MySQL 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** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a **NOT NULL** and **UNIQUE**. Uniquely identifies each row in a table
- **FOREIGN KEY** - Prevents actions that would destroy links between tables
- **CHECK** - Ensures that the values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column if no value is specified
- **CREATE INDEX** - Used to create and retrieve data from the database very quickly

MySQL NOT NULL Constraint

MySQL 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.

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

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

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:

Example

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

MySQL UNIQUE Constraint

MySQL 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.

UNIQUE Constraint on CREATE TABLE

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

```
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:

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

UNIQUE Constraint on ALTER TABLE

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

```
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:

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

DROP a UNIQUE Constraint

To drop a `UNIQUE` constraint, use the following SQL:

```
ALTER TABLE Persons  
DROP INDEX UC_Person;
```


MySQL PRIMARY KEY Constraint

MySQL 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).

PRIMARY KEY on CREATE TABLE

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

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

```
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).

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:

```
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:

```
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:

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