*CREATE DATABASE testDB;*

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

*DROP TABLE testDB;*

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)  
);*

*update orders set first\_name='Rakesh' where id=1;*

INSERT INTO

*INSERT* *INTO* Customers (CustomerName, ContactName, Address, City, PostalCode, Country)  
*VALUES* (*'Cardinal'*, *'Tom B. Erichsen'*, *'Skagen21'*, *'Stavanger'*, *'4006'*, *'Norway'*);

Create Table Using Another Table

*CREATE TABLE TestTable AS  
SELECT customername, contactname FROM customers(*existing\_table\_name)*;*

The TRUNCATE TABLE delete the data inside a table, but not the table itself.

*TRUNCATE TABLE*table\_name*;*

**Delete a row:** *delete from test where age=10;*

MySQL ALTER TABLE Statement

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

Also used to add and drop various constraints on an existing table.

To add a column in a table.

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

To delete a column in a table.

*ALTER TABLE Customers DROP COLUMN Email;*

To change the data type of a column in a table.

***Alter table*** *person modify column current year;*

**MySQL Constraints**

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

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.

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/MySQL/mysql_notnull.asp) - *Ensures that a column cannot have a NULL value*

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

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

[FOREIGN KEY](https://www.w3schools.com/MySQL/mysql_foreignkey.asp) - *Prevents actions that would destroy links between tables*

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

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

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

**NOT NULL**

*By default, a column can hold NULL values.*

*The NOT NULL constraint enforces a column to NOT accept NULL values.*

[*ALTER*](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/alter-table.html)[*TABLE*](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/alter-table.html)*person add column country*[*varchar*](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/string-types.html)*(32)*[*not*](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/logical-operators.html#operator_not)*null;*

NOT NULL on CREATE TABLE

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

# **MySQL UNIQUE Constraint**

*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.*

*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 define a UNIQUE constraint on multiple columns.*

*CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    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.*

*ALTER TABLE Persons ADD UNIQUE (ID);*

*To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns.*

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

## DROP a UNIQUE Constraint

ALTER TABLE Persons DROP INDEX UC\_Person;

**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

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.*

*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.*

*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*

*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).*

*To drop a PRIMARY KEY constraint.*

*ALTER TABLE Persons DROP PRIMARY KEY;*

**MySQL 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/MySQL/mysql_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.*

Graphical user interface

Description automatically generated with low confidence

Table

Description automatically generated

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.

## FOREIGN KEY on CREATE TABLE

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

*To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns.*

*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)  
);*

## FOREIGN KEY on ALTER TABLE

*To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created.*

*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.*

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

*To drop a FOREIGN KEY constraint.*

*ALTER TABLE Orders DROP FOREIGN KEY FK\_PersonOrder;*

***MySQL CHECK Constraint***

*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 column it will allow 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.*

**CHECK on CREATE TABLE**

*The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:*

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

*To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns.*

*CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255),  
    CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes')  
);*

**CHECK on ALTER TABLE**

*To create a CHECK constraint on the "Age" column when the table is already created.*

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

To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns.

*ALTER TABLE Persons ADD CONSTRAINT CHK\_PersonAge CHECK (Age>=18 AND City='Sandnes');*

*To drop a CHECK constraint.*

*ALTER TABLE Persons DROP CHECK CHK\_PersonAge;*

**MySQL DEFAULT Constraint**

*The DEFAULT constraint is used to set a default value for a column.*

*The default value will be added to all new records, if no other value is specified.*

**DEFAULT on CREATE TABLE**

*The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:*

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

*The DEFAULT constraint can also be used to insert system values, by using functions like*[*CURRENT\_DATE()*](https://www.w3schools.com/MySQL/func_mysql_current_date.asp)*:*

*CREATE TABLE Orders (  
    ID int NOT NULL,  
    OrderNumber int NOT NULL,  
    OrderDate date DEFAULT CURRENT\_DATE()  
);*

## DEFAULT on ALTER TABLE

*To create a DEFAULT constraint on the "City" column when the table is already created.*

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

*To drop a DEFAULT constraint.*

*ALTER TABLE Persons ALTER City DROP DEFAULT;*

**MySQL CREATE INDEX Statement**

*The CREATE INDEX statement is used to create indexes in tables.*

*Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.*

**CREATE INDEX Syntax**

*Creates an index on a table. Duplicate values are allowed:*

*The SQL statement below creates an index named "idx\_lastname" on the "LastName" column in the "Persons" table:*

*CREATE INDEX idx\_lastname ON Persons (LastName);*

*If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:*

*CREATE* *INDEX* idx\_pname *ON* Persons (LastName, FirstName);

*The DROP INDEX statement is used to delete an index in a table.*

*ALTER TABLE*table\_name *DROP INDEX*index\_name*;*

**MySQL 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.*

*MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.*

*By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record.*

*The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:*

*CREATE TABLE Persons (  
    Personid int NOT NULL AUTO\_INCREMENT,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    PRIMARY KEY (Personid)  
);*

*To let the AUTO\_INCREMENT sequence start with another value.*

*ALTER TABLE Persons AUTO\_INCREMENT=100;*

*When we insert a new record into the "Persons" table, we do NOT have to specify a value for the "Personid" column (a unique value will be added automatically):*

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

**MySQL Working With Dates**

*As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.*

**MySQL Date Data Types**

*MySQL comes with the following data types for storing a date or a date/time value in the database:*

*DATE - format YYYY-MM-DD*

*DATETIME - format: YYYY-MM-DD HH:MI:SS*

*TIMESTAMP - format: YYYY-MM-DD HH:MI:SS*

*YEAR - format YYYY or YY*

*Note: The date data type are set for a column when you create a new table in your database!*

*Now we want to select the records with an OrderDate of "2008-11-11" from the table.*

*SELECT \* FROM Orders WHERE OrderDate='2008-11-11'*

MySQL CREATE VIEW Statement

*In SQL, a view is a virtual table based on the result-set of an SQL statement.*

*A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.*

*You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.*

A view is created with the CREATE VIEW statement.

*The following SQL creates a view that shows all customers from Brazil:*

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

*We can query the view above as follows:*

*SELECT \* FROM [Brazil Customers];*

*The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:*

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

*We can query the view above as follows:*

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

**MySQL Updating a View**

*A view can be updated with the CREATE OR REPLACE VIEW statement.*

*The following SQL adds the "City" column to the "Brazil Customers" view:*

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

*A view is deleted with the DROP VIEW statement.*

*DROP VIEW*view\_name*;*

**MySQL 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.*

*In MySQL there are three main data types: string, numeric, and date and time.*

### **String Data Types**

Graphical user interface, text, application, email

Description automatically generated

### **Numeric Data Types**

A picture containing text

Description automatically generated

### **Date and Time Data Types**

Text

Description automatically generated with low confidence

# **MySQL Functions**

**Joins:**

INNER JOIN: return rows from both tables that satisfy the given condition.

*SELECT employees.first\_name , employees.last\_name , orders.price*

*FROM employees ,orders WHERE orders.id = employees.id*

Cross JOIN: gives us combinations of each row of first table with all records in second table.

*SELECT \* FROM `employees` CROSS JOIN `orders`;*

Outer JOINs: return all records matching from both tables.

Left Join: *select orders.id, orders.items from Orders left JOIN employees on employees.first\_name = Orders.first\_name;*

Right Join: select orders.id, orders.price from Orders right JOIN employees on employees.id = *Orders.id*

ON clauses: to match the records between table.

USING:

*SELECT A.price , B.first\_name FROM `orders` AS A LEFT JOIN `employees` AS B USING ( first\_name )*

**SELECT DISTINCT:** return only distinct (different) values.

*SELECT DISTINCT price from orders;*

# [**Select distinct records on a join**](https://stackoverflow.com/questions/2068515/select-distinct-records-on-a-join)

*SELECT DISTINCT orders.price, orders.first\_name FROM orders*

*JOIN employees ON orders.id = employees.id*

*WHERE employees.last\_name = 'singh' ORDER BY employees.phone DESC;*

**SELECT with DISTINCT on multiple columns**

*SELECT distinct last\_name, status FROM employees WHERE last\_name='singh';*

## Sorted Data Using ‘Order By’

*SELECT price, items FROM orders WHERE id = 2 ORDER BY qty;*

*SELECT COUNT(price), price FROM orders*

*WHERE price < 70 GROUP BY price ORDER BY price;*

**Query for Creating a View**

*CREATE VIEW customers\_data AS*

*SELECT price, age FROM customers WHERE price > 40;*

Now, [SELECT](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/select.html) \* FROM customers\_data;

Query for Updating a View

*DROP VIEW customers\_data;*

Query to Display Primary Keys

*SHOW KEYS FROM customers WHERE Key\_name = 'PRIMARY'*

Searching for SQL Tables with Wildcards

[*SELECT*](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/select.html)*\* From Customers WHERE city*[*LIKE*](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/string-comparison-functions.html#operator_like)*'us%';*

**Between**

*SELECT Name FROM customers WHERE price BETWEEN 234 AND 2434;*

**Union**

*SELECT phone FROM Customers UNION SELECT items FROM Orders;*

[**Making Column Labels More Friendly**](https://bytescout.com/blog/deep-sql-queries-and-examples.html#28)

*SELECT city AS city\_description FROM customers;*

**Always and Everywhere!**

*SELECT city FROM customers WHERE id = ALL (SELECT price FROM customers WHERE phone > 123456790);*

**Query returns the** *age* **for each employee, along with total** *age* **of the employees by** *age***:**

[*SELECT*](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/select.html)*age, salary, city,*[*SUM*](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/8.0/en/group-by-functions.html#function_sum)*(age) OVER (PARTITION BY age) FROM customers;*