**How does it works on websites?**

All the dynamic websites existing on the internet all are database driven. Dynamic sites like online banking, social media sites, railway reservations, e-commerce are connected to the database. If there is a database, SQL will definitely come in handy.

However, SQL plays a crucial part but can't do the whole work alone. So, there are several elements that work together and for this, there is a need for some entities.

* RDBMS Database Program (for example MySQL, MS Access, Oracle, SQL Server etc)
* Server-side scripting (for example ASP, PHS etc)
* HTML, CSS
* SQL Commands

In your server MySQL like RDBMS software has to be installed and after that, through server-side scriptings like PHP and ASP you have to do programming first and then create dynamic web pages. Also, you have to do coding and tell which task perform you want. Now, for the user interface, you have to create pages for HTML and CSS so that you or the user can see the output of the task.

# 

# **https://www.edureka.co/blog/sql-commands**

# **CREATE DATABASE & CREATE TABLE Operations**

CREATE DATABASE Employee;

CREATE TABLE Employee\_Info

(

EmployeeID int,

EmployeeName varchar(255),

EmergencyContactName varchar(255),

PhoneNumber int,

Address varchar(255),

City varchar(255),

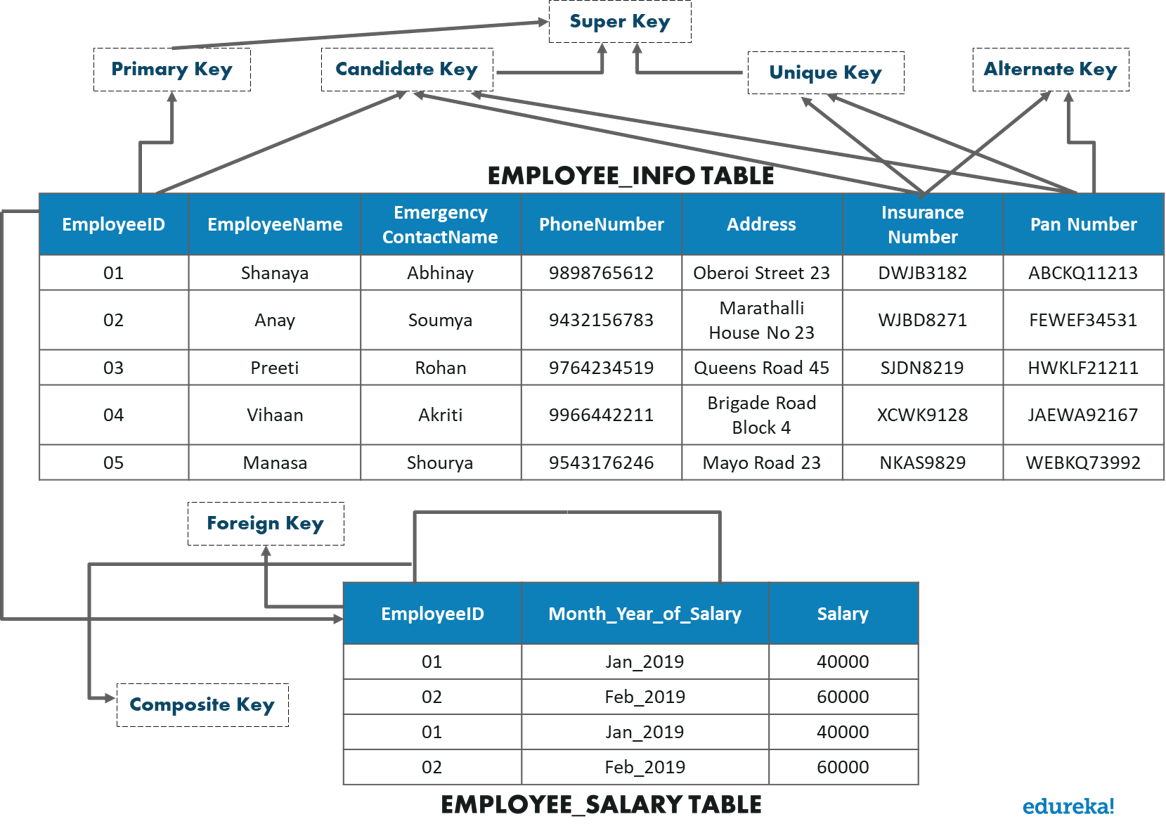
Country varchar(255)

);

## ****SQL Commands: Different Types Of Keys In Database****

There are mainly 7 types of Keys, that can be considered in a database. I am going to consider the below tables to explain to you the various keys.

* **Candidate Key –**A set of attributes which can uniquely identify a table can be termed as a Candidate Key. A table can have more than one candidate key, and out of the chosen candidate keys, one key can be chosen as a Primary Key. In the above example, since EmployeeID, InsuranceNumber and PanNumber can uniquely identify every tuple, they would be considered as a Candidate Key.
* **Super Key –**The set of attributes which can uniquely identify a tuple is known as Super Key. So, a candidate key, primary key, and a unique key is a superkey, but vice-versa isn’t true.
* **Primary Key –** A set of attributes which are used to uniquely identify every tuple is also a primary key. In the above example, since EmployeeID, InsuranceNumber and PanNumber are candidate keys, any one of them can be chosen as a Primary Key. Here EmployeeID is chosen as the primary key.
* **Alternate Key –**Alternate Keys are the candidate keys, which are not chosen as a Primary key. From the above example, the alternate keys are PanNumber and Insurance Number.
* **Unique Key –** The unique key is similar to the primary key, but allows one NULL value in the column. Here the Insurance Number and the Pan Number can be considered as unique keys.
* **Foreign Key –**An attribute that can only take the values present as the values of some other attribute, is the foreign key to the attribute to which it refers. in the above example, the Employee\_ID from the Employee\_Information Table is referred to the Employee\_ID from the Employee\_Salary Table.
* **Composite Key –**A composite key is a combination of two or more columns that identify each tuple uniquely. Here, the Employee\_ID and Month-Year\_Of\_Salary can be grouped together to uniquely identify every tuple in the table.



## 

## ****SQL Commands: Constraints Used In Database****

Constraints are used in a database to specify the rules for data in a table. The following are the different types of constraints:

* [NOT NULL](https://www.edureka.co/blog/sql-commands#NOT%20NULL)
* [UNIQUE](https://www.edureka.co/blog/sql-commands#UNIQUE)
* [CHECK](https://www.edureka.co/blog/sql-commands#CHECK)
* [DEFAULT](https://www.edureka.co/blog/sql-commands#DEFAULT)
* [INDEX](https://www.edureka.co/blog/sql-commands#INDEX)

### **NOT NULL**

This constraint ensures that a column cannot have a NULL value.

#### **Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | --NOT NULL on Create Table    CREATE TABLE Employee\_Info  (  EmployeeID int NOT NULL,  EmployeeName varchar(255) NOT NULL,  Emergency ContactName varchar(255),  PhoneNumber int NOT NULL,  Address varchar(255),  City varchar(255),  Country varchar(255)  );    --NOT NULL on ALTER TABLE    ALTER TABLE Employee\_Info  MODIFY PhoneNumber int NOT NULL; |

### **UNIQUE**

This constraint ensures that all the values in a column are unique.

#### **Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | --UNIQUE on Create Table    CREATE TABLE Employee\_Info  (  EmployeeID int NOT NULL UNIQUE,  EmployeeName varchar(255) NOT NULL,  Emergency ContactName varchar(255),  PhoneNumber int NOT NULL,  Address varchar(255),  City varchar(255),  Country varchar(255)  );    --UNIQUE on Multiple Columns    CREATE TABLE Employee\_Info  (  EmployeeID int NOT NULL,  EmployeeName varchar(255) NOT NULL,  Emergency ContactName varchar(255),  PhoneNumber int NOT NULL,  Address varchar(255),  City varchar(255),  Country varchar(255),  CONSTRAINT UC\_Employee\_Info UNIQUE(Employee\_ID, PhoneNumber)  );    --UNIQUE on ALTER TABLE    ALTER TABLE Employee\_Info  ADD UNIQUE (Employee\_ID);    --To drop a UNIQUE constraint    ALTER TABLE  Employee\_Info  DROP CONSTRAINT UC\_Employee\_Info; |

### **CHECK**

This constraint ensures that all the values in a column satisfy a specific condition.

#### **Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | --CHECK Constraint on CREATE TABLE    CREATE TABLE Employee\_Info  (  EmployeeID int NOT NULL,  EmployeeName varchar(255),  Emergency ContactName varchar(255),  PhoneNumber int,  Address varchar(255),  City varchar(255),  Country varchar(255) CHECK (Country=='India')  );    --CHECK Constraint on multiple columns    CREATE TABLE Employee\_Info  (  EmployeeID int NOT NULL,  EmployeeName varchar(255),  Emergency ContactName varchar(255),  PhoneNumber int,  Address varchar(255),  City varchar(255),  Country varchar(255) CHECK (Country = 'India' AND Cite = 'Hyderabad')  );    --CHECK Constraint on ALTER TABLE    ALTER TABLE Employee\_Info  ADD CHECK (Country=='India');    --To give a name to the CHECK Constraint    ALTER TABLE Employee\_Info  ADD CONSTRAINT CheckConstraintName CHECK (Country=='India');    --To drop a CHECK Constraint    ALTER TABLE Employee\_Info  DROP CONSTRAINT CheckConstraintName; |

### **DEFAULT**

This constraint consists of a set of default values for a column when no value is specified.

#### **Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | --DEFAULT Constraint on CREATE TABLE    CREATE TABLE Employee\_Info  (  EmployeeID int NOT NULL,  EmployeeName varchar(255),  Emergency ContactName varchar(255),  PhoneNumber int,  Address varchar(255),  City varchar(255),  Country varchar(255) DEFAULT 'India'  );    --DEFAULT Constraint on ALTER TABLE    ALTER TABLE Employee\_Info  ADD CONSTRAINT defau\_Country  DEFAULT 'India' FOR Country;    --To drop the Default Constraint    ALTER TABLE Employee\_Info  ALTER COLUMN Country DROP DEFAULT; |

### **INDEX**

This constraint is used to create indexes in the table, through which you can create and retrieve data from the database very quickly.

#### **Syntax**

--Create an Index where duplicate values are allowed

CREATE INDEX IndexName

ON TableName (Column1, Column2, ...ColumnN);

--Create an Index where duplicate values are not allowed

CREATE UNIQUE INDEX IndexName

ON TableName (Column1, Column2, ...ColumnN);

#### **Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | CREATE INDEX idex\_EmployeeName  ON Persons (EmployeeName);    --To delete an index in a table    DROP INDEX Employee\_Info.idex\_EmployeeName; |

Now, let us look into the next part of this article i.e. DML Commands.

## ****SQL Commands: Data Manipulation Language Commands (DML)****

This section of the article will give you an insight into the commands through which you can manipulate the database. The commands are as follows:

INSERT INTO Employee\_Info(EmployeeID, EmployeeName, EmergencyContactName, PhoneNumber, Address, City, Country)

VALUES ('06', 'Sanjana','Jagannath', '992', 'Camel Street House No 12', 'Chennai', 'India');

INSERT INTO Employee\_Info

VALUES ('07', 'Sayantini','Praveen', '9934', 'Nice Road 21', 'Pune', 'India');

INSERT INTO Employee\_Info

VALUES ('08', 'geetha','Pram', '9978', 'grr Road 21', 'Pune', 'India');

INSERT INTO Employee\_Info

VALUES ('09', 'san','dan', '9978', 'grr Road 21', 'Mumbai', 'India');

INSERT INTO Employee\_Info

VALUES ('10', 'satha','gree', '9934', 'mr Road 21', 'Pune', 'India', '400000');

INSERT INTO Employee\_Info

VALUES ('11', 'mammit','reem', '4567', 'sri Road 215', 'Mumbai', 'India', '400000');

INSERT INTO Employee\_Info

VALUES ('12', 'mohan','ram', '4324', 'srieaa Road 215', 'Mumbai', 'India', '600000');

### **UPDATE**

This statement is used to modify the records already present in the table.

UPDATE Employee\_Info

SET EmployeeName = 'Aahana', City= 'Ahmedabad'

WHERE EmployeeID = 1;

### **DELETE**

This statement is used to delete the existing records in a table.

DELETE FROM Employee\_Info

WHERE EmployeeName='Preeti';

### **SELECT**

This statement is used to select data from a database and the data returned is stored in a result table, called the **result-set**.

SELECT EmployeeID, EmployeeName

FROM Employee\_Info;

--(\*) is used to select all from the table

SELECT \* FROM Employee\_Info;

Apart from just using the SELECT keyword individually, you can use the following keywords with the SELECT statement:

* + [DISTINCT](https://www.edureka.co/blog/sql-commands#DISTINCT)
  + [ORDER BY](https://www.edureka.co/blog/sql-commands#ORDER%20BY)
  + [GROUP BY](https://www.edureka.co/blog/sql-commands#GROUP%20BY)
  + [HAVING Clause](https://www.edureka.co/blog/sql-commands#HAVING%20Clause)
  + [INTO](https://www.edureka.co/blog/sql-commands#INTO)

#### **The ‘SELECT DISTINCT’ Statement**

This statement is used to return only different values.

SELECT DISTINCT PhoneNumber FROM Employee\_Info;

#### **The ‘ORDER BY’ Statement**

The ‘ORDER BY’ statement is used to sort the required results in ascending or descending order. The results are sorted in ascending order by default. Yet, if you wish to get the required results in descending order, you have to use the **DESC** keyword.

-- Select all employees from the 'Employee\_Info' table sorted by EmergencyContactName:

SELECT \* FROM Employee\_Info

ORDER BY EmergencyContactName;

-- Select all employees from the 'Employee\_Info' table sorted by EmergencyContactName in Descending order:

SELECT \* FROM Employee\_Info

ORDER BY EmergencyContactName DESC;

-- Select all employees from the 'Employee\_Info' table sorted by EmergencyContactName and EmployeeName:

SELECT \* FROM Employee\_Info

ORDER BY EmergencyContactName, EmployeeName;

/\* Select all employees from the 'Employee\_Info' table sorted by EmergencyContactName in Descending order and EmployeeName in Ascending order: \*/

SELECT \* FROM Employee\_Info

ORDER BY EmergencyContactName ASC, EmployeeName DESC;

#### **The ‘GROUP BY’ Statement**

This ‘GROUP BY’ statement is used with the aggregate functions to group the result-set by one or more columns.

-- To list the number of employees from each city.

SELECT COUNT(EmployeeID), City

FROM Employee\_Info

GROUP BY City;

#### **The ‘HAVING’ Clause**

The ‘HAVING’ clause is used in SQL because the **WHERE keyword** cannot be used everywhere.

/\*  To list the number of employees in each city. The employees should be sorted high to low and only those cities must be included who have more than 5 employees:\*/

SELECT COUNT(EmployeeID), City

FROM Employee\_Info

GROUP BY City

HAVING COUNT(EmployeeID) > 1

ORDER BY COUNT(EmployeeID) DESC;

#### **The ‘SELECT INTO’ Statement**

The ‘SELECT INTO’ statement is used to copy data from one table to another.

-- To create a backup of database 'Employee'

SELECT \* INTO EmployeeBackup

FROM Employee;

--To select only few columns from Employee

SELECT EmployeeName, PhoneNumber INTO EmployeeContactDetails

FROM Employee;

SELECT \* INTO BlrEmployee

FROM Employee

WHERE City = 'Bangalore';

### **Operators in SQL**

The different set of operators available in SQL are as follows:



#### **Logical Operators**

The Logical operators present in SQL are as follows:

* + [AND](https://www.edureka.co/blog/sql-commands#AND)
  + [OR](https://www.edureka.co/blog/sql-commands#OR)
  + [NOT](https://www.edureka.co/blog/sql-commands#NOT)
  + [BETWEEN](https://www.edureka.co/blog/sql-commands#BETWEEN)
  + [LIKE](https://www.edureka.co/blog/sql-commands#LIKE)
  + [IN](https://www.edureka.co/blog/sql-commands#IN)
  + [EXISTS](https://www.edureka.co/blog/sql-commands#EXISTS)
  + [ALL](https://www.edureka.co/blog/sql-commands#ALL)
  + [ANY](https://www.edureka.co/blog/sql-commands#ANY)

##### **AND Operator**

This operator is used to filter records that rely on more than one condition. This operator displays the records, which satisfy all the conditions separated by AND, and give the output TRUE.

##### **AND Operator**

This operator is used to filter records that rely on more than one condition. This operator displays the records, which satisfy all the conditions separated by AND, and give the output TRUE.

SELECT \* FROM Employee\_Info

WHERE City='Mumbai' AND City='Hyderabad';

##### **OR Operator**

This operator displays all those records which satisfy any of the conditions separated by OR and give the output TRUE.

SELECT \* FROM Employee\_Info

WHERE City='Mumbai' OR City='Hyderabad';

##### **NOT Operator**

The NOT operator is used, when you want to display the records which do not satisfy a condition.

SELECT \* FROM Employee\_Info

WHERE NOT City='Mumbai';

##### **BETWEEN Operator**

The BETWEEN operator is used, when you want to select values within a given range. Since this is an inclusive operator, both the starting and ending values are considered.

SELECT \* FROM Employee\_Info

WHERE Employee\_Salary BETWEEN 30500 AND 49500;

SELECT SUM(Employee\_Salary)

FROM Employee\_Info;