Introduction to SQL

SQL is a standard language for accessing and manipulating databases.

What is SQL?

* SQL stands for Structured Query Language
* SQL lets you access and manipulate databases
* SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

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

SQL is a Standard - BUT....

Although SQL is an ANSI/ISO standard, there are different versions of the SQL language.

However, to be compliant with the ANSI standard, they all support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.

**Note:** Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

Using SQL in Your Web Site

To build a web site that shows data from a database, you will need:

* An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
* To use a server-side scripting language, like PHP or ASP
* To use SQL to get the data you want
* To use HTML / CSS to style the page

Now let's see what the MySQL's data types are. You can use any of them depending on your need. You should always try to not to underestimate or overestimate potential range of data when creating a database.

**DATA TYPES**

Data types define the nature of the data that can be stored in a particular column of a table

MySQL has **3** main categories of data types namely

1. Numeric,
2. Text
3. Date/time.

**Numeric Data types**

Numeric data types are used to store numeric values. It is very important to make sure range of your data is between lower and upper boundaries of numeric data types.

|  |  |
| --- | --- |
| TINYINT( ) | -128 to 127 normal0 to 255 UNSIGNED. |
| SMALLINT( ) | -32768 to 32767 normal0 to 65535 UNSIGNED. |
| MEDIUMINT( ) | -8388608 to 8388607 normal0 to 16777215 UNSIGNED. |
| INT( ) | -2147483648 to 2147483647 normal0 to 4294967295 UNSIGNED. |
| BIGINT( ) | -9223372036854775808 to 9223372036854775807 normal0 to 18446744073709551615 UNSIGNED. |
| FLOAT | A small approximate number with a floating decimal point. |
| DOUBLE( , ) | A large number with a floating decimal point. |
| DECIMAL( , ) | A DOUBLE stored as a string , allowing for a fixed decimal point. Choice for storing currency values. |

**Text Data Types**

As data type category name implies these are used to store text values. Always make sure you length of your textual data do not exceed maximum lengths.

|  |  |
| --- | --- |
| CHAR( ) | A fixed section from 0 to 255 characters long. |
| VARCHAR( ) | A variable section from 0 to 255 characters long. |
| TINYTEXT | A string with a maximum length of 255 characters. |
| TEXT | A string with a maximum length of 65535 characters. |
| BLOB | A string with a maximum length of 65535 characters. |
| MEDIUMTEXT | A string with a maximum length of 16777215 characters. |
| MEDIUMBLOB | A string with a maximum length of 16777215 characters. |
| LONGTEXT | A string with a maximum length of 4294967295 characters. |
| LONGBLOB | A string with a maximum length of 4294967295 characters. |

**Date / Time**

|  |  |
| --- | --- |
|  DATE | YYYY-MM-DD |
| DATETIME | YYYY-MM-DD HH:MM:SS |
| TIMESTAMP | YYYYMMDDHHMMSS |
| TIME | HH:MM:SS |

Apart from above there are some other data types in MySQL.

|  |  |
| --- | --- |
| ENUM | To store text value chosen from a list of predefined text values |
| SET | This is also used for storing text values chosen from a list of predefined text values. It can have multiple values. |
| BOOL | Synonym for TINYINT(1), used to store Boolean values |
| BINARY | Similar to CHAR, difference is texts are stored in binary format. |
| VARBINARY | Similar to VARCHAR, difference is texts are stored in binary format. |

**Creating Tables MySQL**



Tables can be created using **CREATE TABLE**statement and it actually has the following syntax.

CREATE TABLE [IF NOT EXISTS] `TableName` (`fieldname` dataType [optional parameters]) ENGINE = storage Engine;

**HERE**

* "CREATE TABLE" is the one responsible for the creation of the table in the database.
* "[IF NOT EXISTS]" is optional and only create the table if no matching table name is found.
* "`fieldName`" is the name of the field and "data Type" defines the nature of the data to be stored in the field.
* "[optional parameters]" additional information about a field such as "  AUTO\_INCREMENT" , NOT NULL etc

**Create Table Example:-**

CREATE TABLE IF NOT EXISTS `MyFlixDB`.`Members` (

 `membership\_number` INT AUTOINCREMENT ,

 `full\_names` VARCHAR(150) NOT NULL ,

 `gender` VARCHAR(6) ,

 `date\_of\_birth` DATE ,

 `physical\_address` VARCHAR(255) ,

 `postal\_address` VARCHAR(255) ,

 `contact\_number` VARCHAR(75) ,

 `email` VARCHAR(255) ,

 PRIMARY KEY (`membership\_number`) )

ENGINE = InnoDB;

# Using the WHERE SQL clause

**WHERE clause is used to specify/apply any condition while retrieving, updating or deleting data from a table. This clause is used mostly with SELECT, UPDATE and DELETEquery.**

**When we specify a condition using the WHERE clause then the query executes only for those records for which the condition specified by the WHERE clause is true.**

### Syntax for WHERE clause

**Here is how you can use the WHERE clause with a DELETE statement, or any other statement,**

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

**The WHERE clause is used at the end of any SQL query, to specify a condition for execution.**

### Time for an Example

**Consider a table student,**

|  |  |  |  |
| --- | --- | --- | --- |
| **s\_id** | **name** | **age** | **address** |
| **101** | **Adam** | **15** | **Chennai** |
| **102** | **Alex** | **18** | **Delhi** |
| **103** | **Abhi** | **17** | **Banglore** |
| **104** | **Ankit** | **22** | **Mumbai** |

**Now we will use the SELECT statement to display data of the table, based on a condition, which we will add to our SELECT query using WHERE clause.**

**Let's write a simple SQL query to display the record for student with s\_id as 101.**

**SELECT s\_id,**

 **name,**

 **age,**

 **address**

 **FROM student WHERE s\_id = 101;**

**Following will be the result of the above query.**

|  |  |  |  |
| --- | --- | --- | --- |
| **s\_id** | **name** | **age** | **address** |
| **101** | **Adam** | **15** | **Noida** |

#### Applying condition on Text Fields

**In the above example we have applied a condition to an integer value field, but what if we want to apply the condition on name field. In that case we must enclose the value in single quote ' '. Some databases even accept double quotes, but single quotes is accepted by all.**

**SELECT s\_id,**

 **name,**

 **age,**

 **address**

 **FROM student WHERE name = 'Adam';**

**Following will be the result of the above query.**

|  |  |  |  |
| --- | --- | --- | --- |
| **s\_id** | **name** | **age** | **address** |
| **101** | **Adam** | **15** | **Noida** |

## Operators for WHERE clause condition

**Following is a list of operators that can be used while specifying the WHERE clause condition.**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| **=** | **Equal to** |
| **!=** | **Not Equal to** |
| **<** | **Less than** |
| **>** | **Greater than** |
| **<=** | **Less than or Equal to** |
| **>=** | **Greate than or Equal to** |
| **BETWEEN** | **Between a specified range of values** |
| **LIKE** | **This is used to search for a pattern in value.** |
| **IN** | **In a given set of values** |

# DISTINCT keyword

The distinct keyword is used with SELECT statement to retrieve unique values from the table. Distinct removes all the duplicate records while retrieving records from any table in the database.

### Syntax for DISTINCT Keyword

SELECT DISTINCT column-name FROM table-name;

### Example using DISTINCT Keyword

Consider the following **Emp** table. As you can see in the table below, there is employee **name**, along with employee **salary** and **age**.

In the table below, multiple employees have the same salary, so we will be using DISTINCT keyword to list down distinct salary amount, that is currently being paid to the employees.

|  |  |  |  |
| --- | --- | --- | --- |
| **eid** | **name** | **age** | **salary** |
| 401 | Anu | 22 | 5000 |
| 402 | Shane | 29 | 8000 |
| 403 | Rohan | 34 | 10000 |
| 404 | Scott | 44 | 10000 |
| 405 | Tiger | 35 | 8000 |

SELECT DISTINCT salary FROM Emp;

The above query will return only the unique salary from **Emp** table.

|  |
| --- |
| **salary** |
| 5000 |
| 8000 |
| 10000 |

# SQL Constraints

**SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.**

**Constraints can be divided into the following two types,**

1. **Column level constraints: Limits only column data.**
2. **Table level constraints: Limits whole table data.**

**Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.**

* **NOT NULL**
* **UNIQUE**
* **PRIMARY KEY**
* **FOREIGN KEY**
* **CHECK**
* **DEFAULT**

## NOT NULL Constraint

**NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.**

**One important point to note about this constraint is that it cannot be defined at table level.**

### Example using NOT NULL constraint

**CREATE TABLE Student(s\_id int NOT NULL, Name varchar(60), Age int);**

**The above query will declare that the s\_id field of Student table will not take NULL value.**

## UNIQUE Constraint

**UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. This constraint can be applied at column level or table level.**

### Using UNIQUE constraint when creating a Table (Table Level)

**Here we have a simple CREATE query to create a table, which will have a column s\_id with unique values.**

**CREATE TABLE Student(s\_id int NOT NULL UNIQUE, Name varchar(60), Age int);**

**The above query will declare that the s\_id field of Student table will only have unique values and wont take NULL value.**

### Using UNIQUE constraint after Table is created (Column Level)

**ALTER TABLE Student ADD UNIQUE(s\_id);**

**The above query specifies that s\_id field of Student table will only have unique value.**

## Primary Key Constraint

**Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.**

### Using PRIMARY KEY constraint at Table Level

**CREATE table Student (s\_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);**

**The above command will creates a PRIMARY KEY on the s\_id.**

### Using PRIMARY KEY constraint at Column Level

**ALTER table Student ADD PRIMARY KEY (s\_id);**

**The above command will creates a PRIMARY KEY on the s\_id.**

## Foreign Key Constraint

**FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables:**

**Customer\_Detail Table**

|  |  |  |
| --- | --- | --- |
| **c\_id** | **Customer\_Name** | **address** |
| **101** | **Adam** | **Noida** |
| **102** | **Alex** | **Delhi** |
| **103** | **Stuart** | **Rohtak** |

**Order\_Detail Table**

|  |  |  |
| --- | --- | --- |
| **Order\_id** | **Order\_Name** | **c\_id** |
| **10** | **Order1** | **101** |
| **11** | **Order2** | **103** |
| **12** | **Order3** | **102** |

**In Customer\_Detail table, c\_id is the primary key which is set as foreign key in Order\_Detail table. The value that is entered in c\_id which is set as foreign key in Order\_Detail table must be present in Customer\_Detail table where it is set as primary key. This prevents invalid data to be inserted into c\_id column of Order\_Detail table.**

**If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.**

### Using FOREIGN KEY constraint at Table Level

**CREATE table Order\_Detail(**

 **order\_id int PRIMARY KEY,**

 **order\_name varchar(60) NOT NULL,**

 **c\_id int FOREIGN KEY REFERENCES Customer\_Detail(c\_id)**

**);**

**In this query, c\_id in table Order\_Detail is made as foriegn key, which is a reference of c\_id column in Customer\_Detail table.**

### Using FOREIGN KEY constraint at Column Level

**ALTER table Order\_Detail ADD FOREIGN KEY (c\_id) REFERENCES Customer\_Detail(c\_id);**

### Behaviour of Foriegn Key Column on Delete

**There are two ways to maintin the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which a record exits in the child table, then we must have some mechanism to save the integrity of data in the child table.**

****

1. **On Delete Cascade : This will remove the record from child table, if that value of foriegn key is deleted from the main table.**
2. **On Delete Null : This will set all the values in that record of child table as NULL, for which the value of foriegn key is deleted from the main table.**
3. **If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.**

**ERROR : Record in child table exist**

## CHECK Constraint

**CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.**

### Using CHECK constraint at Table Level

**CREATE table Student(**

 **s\_id int NOT NULL CHECK(s\_id > 0),**

 **Name varchar(60) NOT NULL,**

 **Age int**

**);**

**The above query will restrict the s\_id value to be greater than zero.**

### Using CHECK constraint at Column Level

**ALTER table Student ADD CHECK(s\_id > 0);**

SQL Alias - AS Keyword

**Alias** is used to give an alias name to a table or a column, which can be a resultset table too. This is quite useful in case of large or complex queries. Alias is mainly used for giving a short alias name for a column or a table with complex names.

Syntax of Alias for table names,

SELECT column-name FROM table-name AS alias-name

Following is an SQL query using **alias**,

SELECT \* FROM Employee\_detail AS ed;

**Syntax for defining alias for columns** will be like,

SELECT column-name AS alias-name FROM table-name;

Example using alias for columns,

SELECT customer\_id AS cid FROM Emp;

Example of Alias in SQL Query

Consider the following two tables,

The **class** table,

|  |  |
| --- | --- |
| **ID** | **Name** |
| 1 | abhi |
| 2 | adam |
| 3 | alex |
| 4 | anu |
| 5 | ashish |

and the **class\_info** table,

|  |  |
| --- | --- |
| **ID** | **Address** |
| 1 | DELHI |
| 2 | MUMBAI |
| 3 | CHENNAI |
| 7 | NOIDA |
| 8 | PANIPAT |

Below is the Query to fetch data from both the tables using SQL Alias,

SELECT C.id, C.Name, Ci.Address from Class AS C, Class\_info AS Ci where C.id = Ci.id;

and the resultset table will look like,

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Address** |
| 1 | abhi | DELHI |
| 2 | adam | MUMBAI |
| 3 | alex | CHENNAI |

SQL Alias seems to be quite a simple feature of SQL, but it is highly useful when you are working with more than 3 tables and have to use JOIN on them.