MySQL Constraints

The MySQL **constraints** can be used to set certain rules to the column(s) in a table. These constraints can restrict the type of data that can be inserted or updated in a particular column. This helps you to maintain the data accuracy and reliability in a table.

There are two types of MySQL constraints.

- **Column level constraints:** These type of constraints will only apply to a column in a table.
- **Table level constraints:** These constraints will apply to the complete table.

The commonly used constraints in MySQL are NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT, CREATE INDEX, AUTO_INCREMENT, etc.

Syntax

Following is the basic syntax to add a constraint for the column(s) in a table −

```
CREATE TABLE table_name (

   Column_name1 datatype constraint,

   Column_name2 datatype constraint,

   Column_name3 datatype constraint,

   .........

);
```

MySQL NOT NULL Constraint

By default, a column in a MySQL table can contain NULL values. In some scenarios, we may want a particular column to not accept or contain NULL values. To do so, we can use the MySQL NOT NULL constraint.

This constraint enforces a specific field to always contain a value, which means that we cannot insert or update a record without adding a value to this field.

Example

In the following query, we are adding the NOT NULL constraint on the **ID** and **NAME** columns of the **CUSTOMERS** table. As a result, the **ID** and **NAME** columns will not accept NULL values at the time of record insertion.

```
CREATE TABLE CUSTOMERS (
  ID int NOT NULL,
  NAME varchar(20) NOT NULL,
  AGE int
);
```

Let's try inserting records into this table. The following statement will insert a record into the CUSTOMERS table −

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(1, 'Nikhil', 18);
```

But, if we try to insert records with NULL values as ID as −

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(Null, 'Varun', 26);
```

An error will be generated saying "Column 'ID' cannot be null".

```
ERROR 1048 (23000): Column 'ID' cannot be null
```

In the same way if we try to pass NULLs as values to the NAME column, similar error will be generated.

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(3, Null, 19);
```

This will generate the following error −

```
ERROR 1048 (23000): Column 'NAME' cannot be null
```

As we can see in the above queries, the first record is successfully inserted because it does not have null values in the ID and Name columns. Whereas, the second and third records are not inserted because we are trying to insert NULL values in the columns which shouldn't be NULL.

MySQL UNIQUE Constraint

The UNIQUE constraint in MySQL ensures that every value in a column must be distinct. This means the column with the UNIQUE constraint cannot have the same value repeated; each value must be unique.

Note: We can have one or more UNIQUE constraints on a single table.

Example

The following query creates a UNIQUE constraint on the ID column of the CUSTOMERS table −

```
CREATE TABLE CUSTOMERS (
  ID int NOT NULL,
  NAME varchar(20) NOT NULL,
  AGE int,
  UNIQUE (ID)
);
```

Now, let us insert the following records into the above-created table −

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(1, 'Nikhil', 18);
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(1, 'Varun', 26);
```

In the above code block, the second insert statement returned an error saying "Duplicate entry '1' for key 'customers.ID'' because the ID value we are inserting already exists in the table. Therefore, it is a duplicate and the query generates the following error −

```
ERROR 1062 (23000): Duplicate entry '1' for key 'customers.ID'
```

MySQL PRIMARY KEY Constraint

The PRIMARY KEY constraint in MySQL is used to uniquely identify each record in a table. This means that, if we define primary key on a particular column in a table, it must contain UNIQUE values, and cannot contain NULL values.

Note: We can have only a single primary key on a table.

Example

The following query creates a PRIMARY KEY on the ID column of the CUSTOMERS table −

```
CREATE TABLE CUSTOMERS (
  ID int NOT NULL,
  NAME varchar(20) NOT NULL,
  AGE int,
  PRIMARY KEY (ID)
);
```

Once the table is created, insert the following record into the above-created table –

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES (1, 'Nikhil', 18);
```

```
Query OK, 1 row affected (0.01 sec)
```

Since we added the PRIMARY KEY constraint on the ID column, if you try to insert a record with duplicate ID value or NULL value, it will generate an error.

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES (1, 'Varun', 26);
```

```
ERROR 1062 (23000): Duplicate entry '1' for key 'customers.PRIMARY'
```

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES (NULL, 'Datta', 19);
```

```
ERROR 1048 (23000): Column 'ID' cannot be null
```

As we can see in the above queries, the first insert statement is successfully inserted into the table. Whereas the second and third statements returned an error because they contain a duplicate and a NULL value in the primary key column i.e. (ID).

MySQL FOREIGN KEY Constraint

The FOREIGN KEY constraint in MySQL is used to link a field or collection of fields in one table to the primary key of another table.

A table with the foreign key is called a ***child table*** and the table with the primary key is called the ***parent table*** or referenced table.

Example

The following query creates a FOREIGN KEY on the CUST_ID column when the ORDERS table is created –

Table: Customers

```
CREATE TABLE CUSTOMERS (
  CUST_ID int NOT NULL,
  NAME varchar(20) NOT NULL,
  AGE int,
  PRIMARY KEY (CUST_ID)
);
```

Table: Orders

```
CREATE TABLE ORDERS (
  ORDER_ID int NOT NULL,
  ORDER_NUMBER int NOT NULL,
  CUST_ID int,
  FOREIGN KEY (CUST_ID) REFERENCES CUSTOMERS (CUST_ID)
);
```

MySQL CHECK Constraint

The CHECK constraint in MySQL restricts the range of values that can be inserted into a column. This constraint ensures that the inserted value in a column must be satisfied with the provided condition.

Example

The following query creates a CHECK constraint on the AGE column of the CUSTOMERS table, where it ensures that the age of the student must be 18 or older −

```
CREATE TABLE CUSTOMERS (
  ID int NOT NULL,
  NAME varchar(20) NOT NULL,
  AGE int,
  CHECK (AGE >= 18)
 );
```

Once the table is created, we can insert the records into the above created table as shown below –

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(1, 'Nikhil', 18);

INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(3, 'Datta', 19);
```

Since we added the CHECK constraint on the AGE column such that the age of the student should be equal or greater than 18. If you try to insert a record with age value less than 18, an error will be generated.

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(2, 'Varun', 16);
```

```
ERROR 3819 (HY000): Check constraint 'customers_chk_1' is violated.
```

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(4, 'Karthik', 15);
```

```
ERROR 3819 (HY000): Check constraint 'customers_chk_1' is violated.
```

Example

Here, the following query creates a CHECK constraint on **multiple columns** (AGE and ADDRESS) −

```
CREATE TABLE CUSTOMERS (
  ID int NOT NULL,
  NAME varchar(20) NOT NULL,
  AGE int,
  ADDRESS varchar(40),
  CONSTRAINT CHECK_AGE CHECK (AGE >= 18 AND ADDRESS = "Mumbai")
);
```

Now, let us insert the following records into the above-created table −

```
INSERT INTO CUSTOMERS(ID, NAME, AGE, ADDRESS) VALUES(1, 'Nikhil', 18, 'Mumbai');
```

```
Query OK, 1 row affected (0.01 sec)
```

```
INSERT INTO CUSTOMERS(ID, NAME, AGE, ADDRESS) VALUES(3, 'Datta', 19, 'Delhi');
```

```
ERROR 3819 (HY000): Check constraint 'CHECK_AGE_AND_ADDRESS' is violated.
```

The second insert statement returned an error because it is violating the condition of the check constraint i.e. (AGE >= 18 AND ADDRESS = "Mumbai").

MySQL DEFAULT Constraint

The DEFAULT constraint in MySQL is used to assign a default value to a specific column in a table. This default value gets applied to any new records in the DEFAULT specified column when no other value is provided during insertion.

Example

In the following query, we are defining the DEFAULT constraint on the ADDRESS column of the CUSTOMERS table. We assigned "Mumbai" as default value when no value is inserted. −

```
CREATE TABLE CUSTOMERS (
  ID int NOT NULL,
  NAME varchar(20) NOT NULL,
  AGE int,
  ADDRESS varchar(40) DEFAULT "Mumbai"
);
```

Here, we are inserting the first two records without any value in the ADDRESS column. In the third record, we are inserting the ADDRESS value as 'Delhi'.

```
INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(1, 'Nikhil', 18);


INSERT INTO CUSTOMERS(ID, NAME, AGE) VALUES(2, 'Varun', 16);


INSERT INTO CUSTOMERS(ID, NAME, AGE, ADDRESS) VALUES(3, 'Datta', 19, 'Delhi');
```

Exeucte the following query to display the records inserted in the above-created table −

```
Select * from CUSTOMERS;
```

In the following output, we can see that the value in the ADDRESS column for the first two rows is by default "Mumbai".

ID	NAME	AGE	ADDRESS
1	Nikhil	18	Mumbai

2	Varun	16	Mumbai
3	Datta	19	Delhi

MySQL CREATE INDEX Constraint

The CREATE INDEX constraint in MySQL is used to create indexes for one more columns in a table.

The indexes are used to fetch the data from the database much quicker. However, the users cannot see the indexes in action, instead, they are just used to speed up the searches and queries.

Example

Here, we are creating a table named CUSTOMERS using the query below −

```
CREATE TABLE CUSTOMERS (
  ID int NOT NULL,
  NAME varchar(20) NOT NULL,
  AGE int,
  ADDRESS varchar(40),
  PRIMARY KEY (ID)
);
```

The following query creates an index named "index_address" on the ADDRESS column of the CUSTOMERS table −

```
CREATE INDEX index_address ON CUSTOMERS (ADDRESS);
```

MySQL AUTO_INCREMENT Constraint

When a **AUTO_INCREMENT** constraint is defined on a particular column of a table, it will automatically generate a unique number when a new record is inserted into that column.

By default, the starting value is 1, and it will automatically increment its value by 1 for each new record.

Example

The following query adds an AUTO_INCREMENT constraint on the ID column of the CUSTOMERS table −

```
CREATE TABLE CUSTOMERS (
  ID int NOT NULL AUTO_INCREMENT,
  NAME varchar(20) NOT NULL,
  AGE int,
  PRIMARY KEY (ID)
);
```

In the insert statements below, we are not inserting ID values.

```
INSERT INTO STUDENTS(NAME, AGE) VALUES('Nikhil', 18);
INSERT INTO STUDENTS(NAME, AGE) VALUES('Varun', 16);
INSERT INTO STUDENTS(NAME, AGE) VALUES('Datta', 19);
```

Now, execute the following query to display the records of the above-created table −

```
Select * from CUSTOMERS;
```

As we can see in the STUDENTS table below, the values in the ID column are automatically incremented because of the AUTO_INCREMENT constraint on the ID column.