SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)
Re-accredited by NAAC with A+ grade, Accredited by NBA(CSE, IT, ECE, EEE & Mechanical)
Approvedy by AICTE, New Delhi, Recognized by UGC, Affiliated to Anna University, Chennai

Department of MCA

DBMS SQL Constraints

Course Name : 23CAT603 - DATA BASE MANAGEMENT SYSTEM

Class : I Year / I Semester

Unit II – SQL Constraints

SQL Constraints

SQL Create 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.

Syntax

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

SQL 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

SQL Constraints

SQL Constraints are the rules applied to a data columns or the complete table to limit the type of data that can go into a table. When you try to perform any INSERT, UPDATE, or DELETE operation on the table, RDBMS will check whether that data violates any existing constraints and if there is any violation between the defined constraint and the data action, it aborts the operation and returns an error.
We can define a column level or a table level constraints. The column level constraints are applied only to one column, whereas the table level constraints are applied to the whole table.
SQL Create Constraints

We can create constraints on a table at the time of a table creation using the CREATE TABLE statement, or after the table is created, we can use the ALTER TABLE statement to create or delete table constraints.

Different RDBMS allows to define different constraints. This tutorial will discuss about 7 most important constraints available in MySQL.

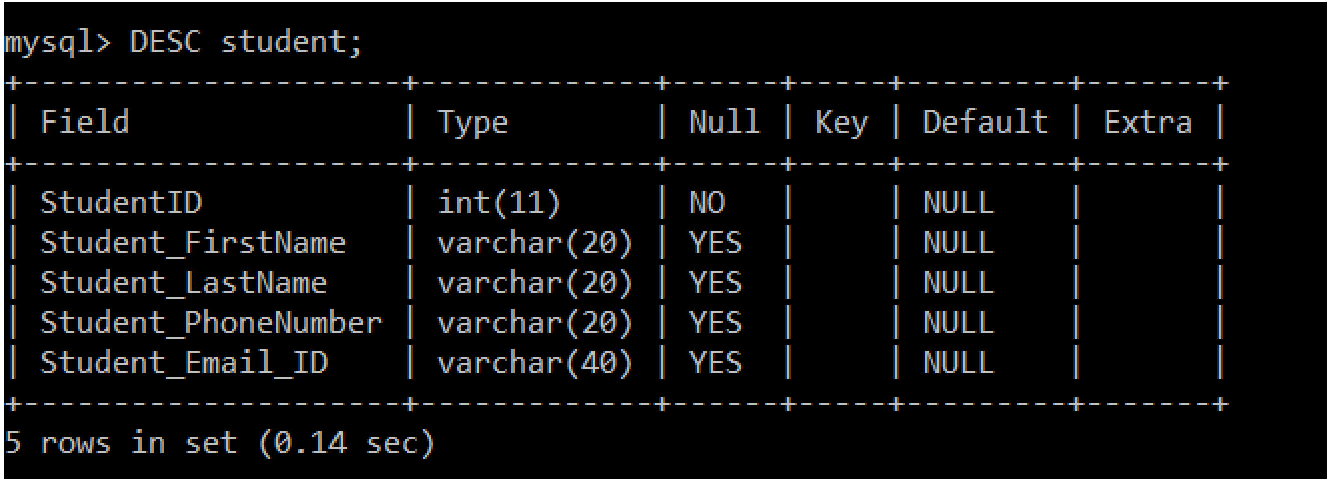
```
CREATE TABLE table_name ( column1
datatype constraint, column2
datatype constraint, .... columnN
datatype constraint );
```

1. NOT NULL

NULL means empty, i.e., the value is not available.
Whenever a table's column is declared as NOT NULL, then the value for that column cannot be empty for any of the table's records.
There must exist a value in the column to which the NOT NULL constraint is applied.
NOTE: NULL does not mean zero. NULL means empty column, not even zero.
Syntax to apply the NOT NULL constraint during table creation:
CREATE TABLE TableName (ColumnName1 datatype NOT NULL, ColumnName2 datatype,…., ColumnNameN datatype);
Example:
Create a student table and apply a NOT NULL constraint on one of the table's column while creating a table.
CREATE TABLE student(StudentID **INT** NOT NULL, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40));
To verify that the not null constraint is applied to the table's column and the student table is created successfully, we will execute the following query:
mysql> **DESC** student;

1. NOT NULL

mysql> **DESC** student;



```
mysql> DESC student;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| StudentID            | int(11)     | NO   |     | NULL    |       |
| Student_FirstName    | varchar(20) | YES  |     | NULL    |       |
| Student_LastName     | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber  | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID     | varchar(40) | YES  |     | NULL    |       |
+----------------------+-------------+------+-----+---------+-------+
5 rows in set (0.14 sec)
```

2. UNIQUE

Duplicate values are not allowed in the columns to which the UNIQUE constraint is applied.
The column with the unique constraint will always contain a unique value.
This constraint can be applied to one or more than one column of a table, which means more than one unique constraint can exist on a single table.
Using the UNIQUE constraint, you can also modify the already created tables.

Syntax to apply the UNIQUE constraint on a single column:

CREATE TABLE TableName (ColumnName1 datatype **UNIQUE**, ColumnName2 datatype,…., Column NameN datatype);

Example:

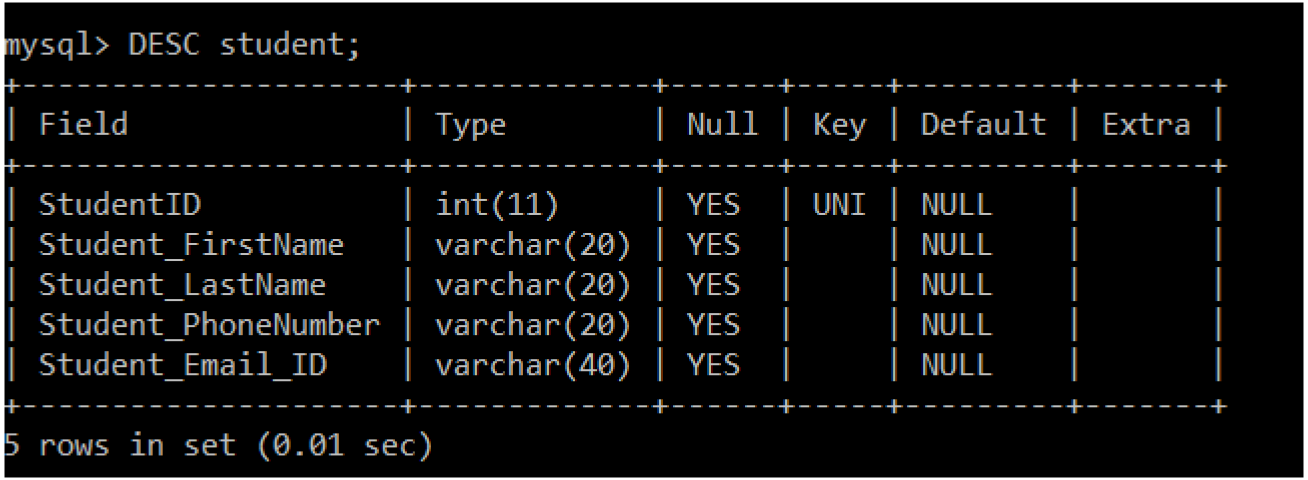
Create a student table and apply a UNIQUE constraint on one of the table's column while creating a table.

mysql> **CREATE TABLE** student(StudentID **INT UNIQUE**, Student_FirstName **VARCHAR**(20), Student_ LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40));

To verify that the unique constraint is applied to the table's column and the student table is created successfully, we will execute the following query:

2. UNIQUE

mysql> **DESC** student;



```
mysql> DESC student;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| StudentID            | int(11)     | YES  | UNI | NULL    |       |
| Student_FirstName    | varchar(20) | YES  |     | NULL    |       |
| Student_LastName     | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber  | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID     | varchar(40) | YES  |     | NULL    |       |
+----------------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

2. UNIQUE

Syntax to apply the UNIQUE constraint on more than one column:
CREATE TABLE TableName (ColumnName1 datatype, ColumnName2 datatype,…., ColumnNameN d atatype, **UNIQUE** (ColumnName1, ColumnName 2));

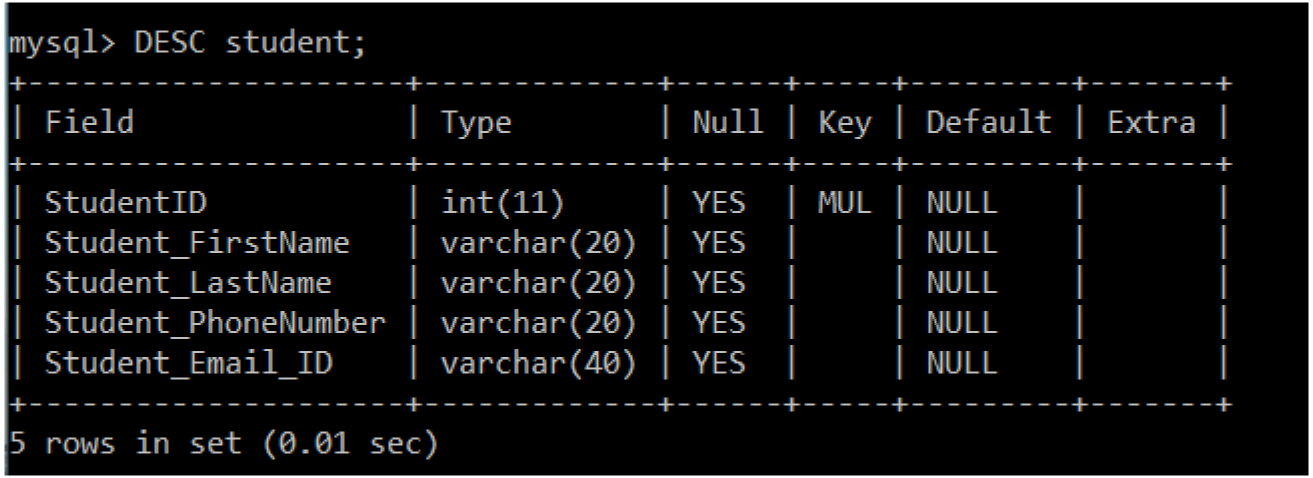
Example:
Create a student table and apply a UNIQUE constraint on more than one table's column while creating a table.

mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), Student_LastNam e **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40), **UNIQUE**(StudentID, Student_PhoneNumber));

To verify that the unique constraint is applied to more than one table's column and the student table is created successfully, we will execute the following query:
mysql> **DESC** student;

2. UNIQUE

mysql> **DESC** student;



```
mysql> DESC student;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| StudentID            | int(11)     | YES  | MUL | NULL    |       |
| Student_FirstName    | varchar(20) | YES  |     | NULL    |       |
| Student_LastName     | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber  | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID     | varchar(40) | YES  |     | NULL    |       |
+----------------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

2. UNIQUE

ALTER **TABLE** TableName **ADD UNIQUE** (ColumnName);

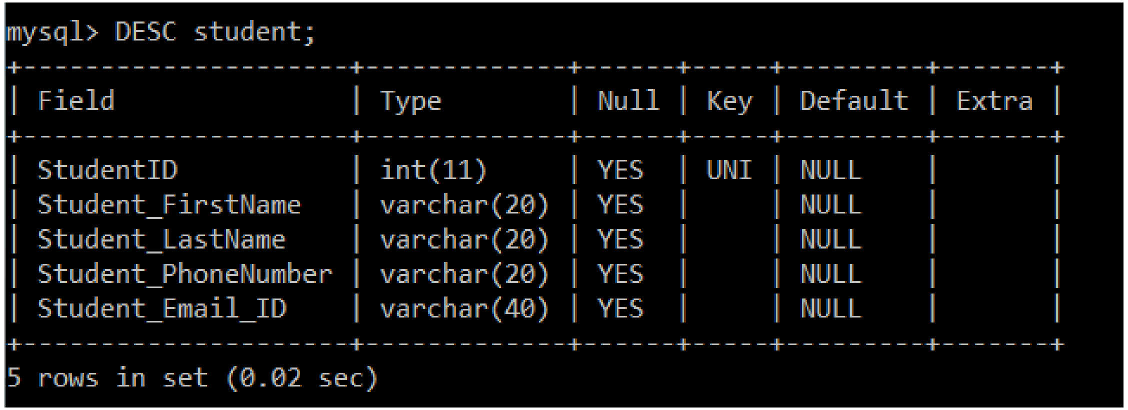
Example:

Consider we have an existing table student, without any constraints applied to it. Later, we decided to apply a UNIQUE constraint to one of the table's column. Then we will execute the following query:

mysql> **ALTER TABLE** student **ADD UNIQUE** (StudentID);

To verify that the unique constraint is applied to the table's column and the student table is created successfully, we will execute the following query:

mysql> **DESC** student;



```
mysql> DESC student;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| StudentID            | int(11)     | YES  | UNI | NULL    |       |
| Student_FirstName    | varchar(20) | YES  |     | NULL    |       |
| Student_LastName     | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber  | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID     | varchar(40) | YES  |     | NULL    |       |
+----------------------+-------------+------+-----+---------+-------+
5 rows in set (0.02 sec)
```

3. PRIMARY KEY

PRIMARY KEY Constraint is a combination of NOT NULL and Unique constraints.
NOT NULL constraint and a UNIQUE constraint together forms a PRIMARY constraint.
The column to which we have applied the primary constraint will always contain a unique value and will not allow null values.

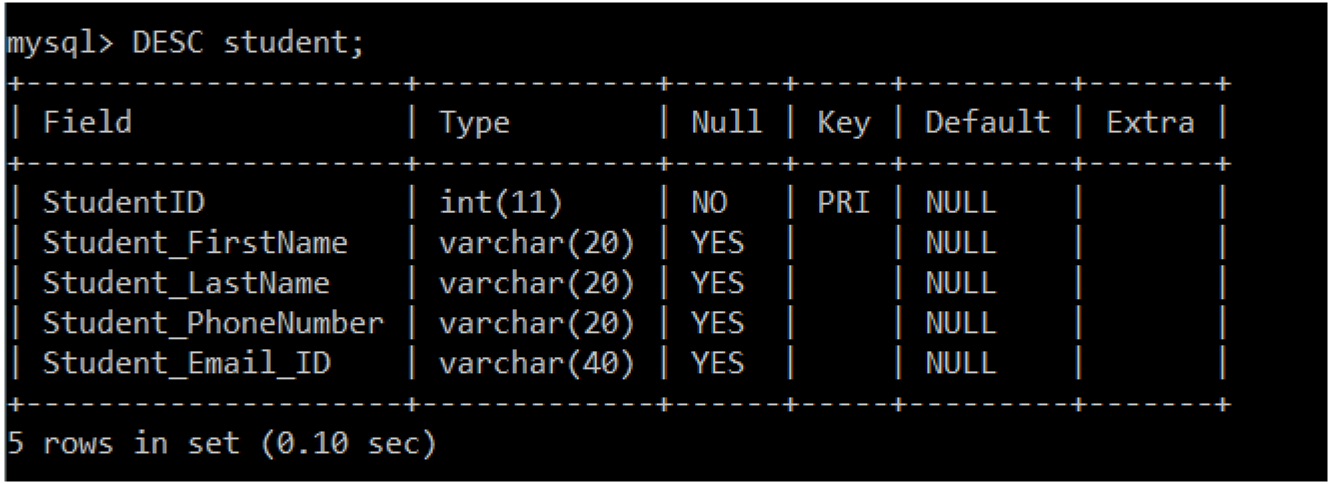
Syntax of primary key constraint during table creation:
CREATE TABLE TableName (ColumnName1 datatype **PRIMARY KEY**, ColumnName2 datatype,…., ColumnNameN datatype);

Example:
Create a student table and apply the PRIMARY KEY constraint while creating a table.
mysql> **CREATE TABLE** student(StudentID **INT PRIMARY KEY**, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40));

To verify that the primary key constraint is applied to the table's column and the student table is created successfully, we will execute the following query:

3. PRIMARY KEY

mysql> **DESC** student;



```
mysql> DESC student;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| StudentID            | int(11)     | NO   | PRI | NULL    |       |
| Student_FirstName    | varchar(20) | YES  |     | NULL    |       |
| Student_LastName     | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber  | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID     | varchar(40) | YES  |     | NULL    |       |
+----------------------+-------------+------+-----+---------+-------+
5 rows in set (0.10 sec)
```

3. PRIMARY KEY

Syntax to apply the primary key constraint on an existing table's column:

ALTER TABLE TableName **ADD PRIMARY KEY** (ColumnName);

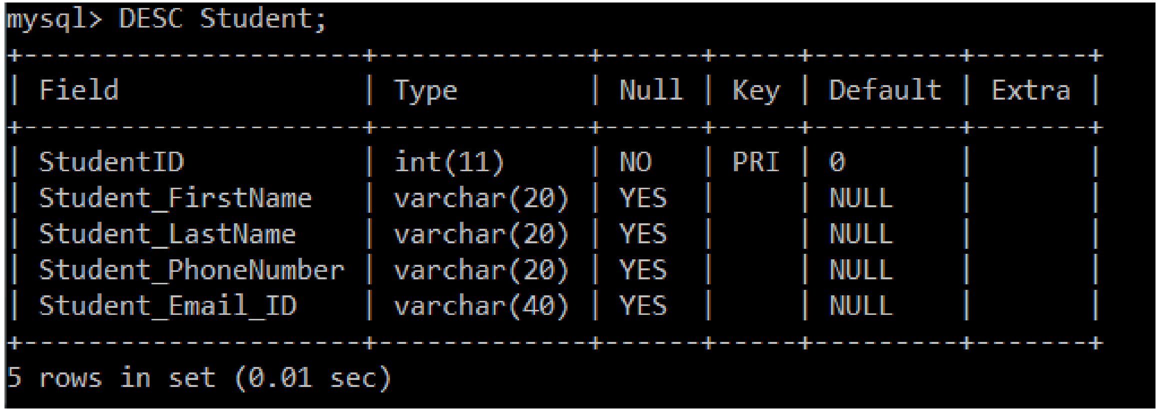
Example:

Consider we have an existing table student, without any constraints applied to it. Later, we decided to apply the PRIMARY KEY constraint to the table's column. Then we will execute the following query:

mysql> **ALTER TABLE** student **ADD PRIMARY KEY** (StudentID);

To verify that the primary key constraint is applied to the student table's column, we will execute the following query:

mysql> **DESC** student;



```
mysql> DESC Student;
+---------------------+-------------+------+-----+---------+-------+
| Field               | Type        | Null | Key | Default | Extra |
+---------------------+-------------+------+-----+---------+-------+
| StudentID           | int(11)     | NO   | PRI | 0       |       |
| Student_FirstName   | varchar(20) | YES  |     | NULL    |       |
| Student_LastName    | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID    | varchar(40) | YES  |     | NULL    |       |
+---------------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

4. FOREIGN KEY

A foreign key is used for referential integrity.

When we have two tables, and one table takes reference from another table, i.e., the same column is present in both the tables and that column acts as a primary key in one table. That particular column will act as a foreign key in another table.

Syntax to apply a foreign key constraint during table creation:

CREATE TABLE tablename(ColumnName1 Datatype(**SIZE**) **PRIMARY KEY**, ColumnNameN Datatype(**SIZE**), **FOREIGN KEY**(ColumnName) **REFERENCES** PARENT_TABLE_NAME(Primary_Key_ColumnName));

Example:

Create an employee table and apply the FOREIGN KEY constraint while creating a table.

To create a foreign key on any table, first, we need to create a primary key on a table.

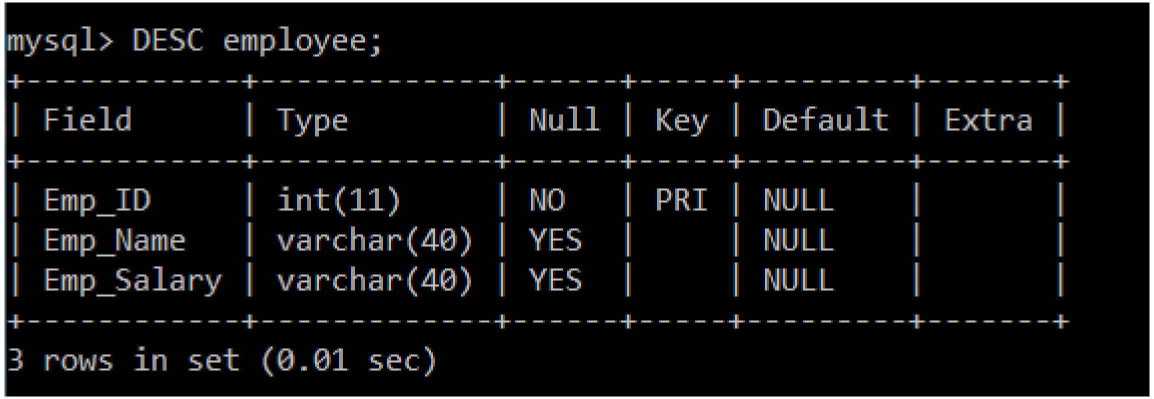
mysql> **CREATE TABLE** employee (Emp_ID **INT** NOT NULL **PRIMARY KEY**, Emp_Name **VARCHAR** (40), Emp_Salary **VARCHAR** (40));

To verify that the primary key constraint is applied to the employee table's column, we will execute the following query:

mysql> **DESC** employee;

4. FOREIGN KEY

mysql> **DESC** employee;



```
mysql> DESC employee;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Emp_ID     | int(11)     | NO   | PRI | NULL    |       |
| Emp_Name   | varchar(40) | YES  |     | NULL    |       |
| Emp_Salary | varchar(40) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

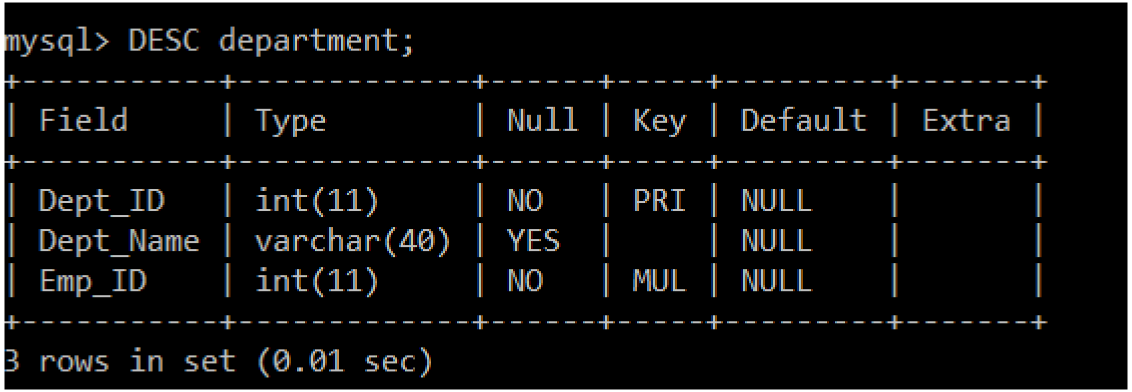
Now, we will write a query to apply a foreign key on the department table referring to the primary key of the employee table, i.e., Emp_ID.

mysql> **CREATE TABLE** department(Dept_ID **INT** NOT NULL **PRIMARY KEY**, Dept_Name **VARCHAR**(40), Emp_ID **INT** NOT NULL, **FOREIGN KEY**(Emp_ID) **REFERENCES** employee(Emp_ID));

To verify that the foreign key constraint is applied to the department table's column, we will execute the following query:

4. FOREIGN KEY

mysql> **DESC** employee;



```
mysql> DESC department;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Dept_ID   | int(11)     | NO   | PRI | NULL    |       |
| Dept_Name | varchar(40) | YES  |     | NULL    |       |
| Emp_ID    | int(11)     | NO   | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

4. FOREIGN KEY

Syntax to apply the foreign key constraint with constraint name:

CREATE TABLE tablename(ColumnName1 Datatype **PRIMARY KEY**, ColumnNameN Datatype(**SIZE**), **CONSTRAINT** ConstraintName **FOREIGN KEY**(ColumnName) **REFERENCES** PARENT_TABLE_NAME(Primary_Key_ColumnName));

Example:

Create an employee table and apply the FOREIGN KEY constraint with a constraint name while creating a table.

To create a foreign key on any table, first, we need to create a primary key on a table.

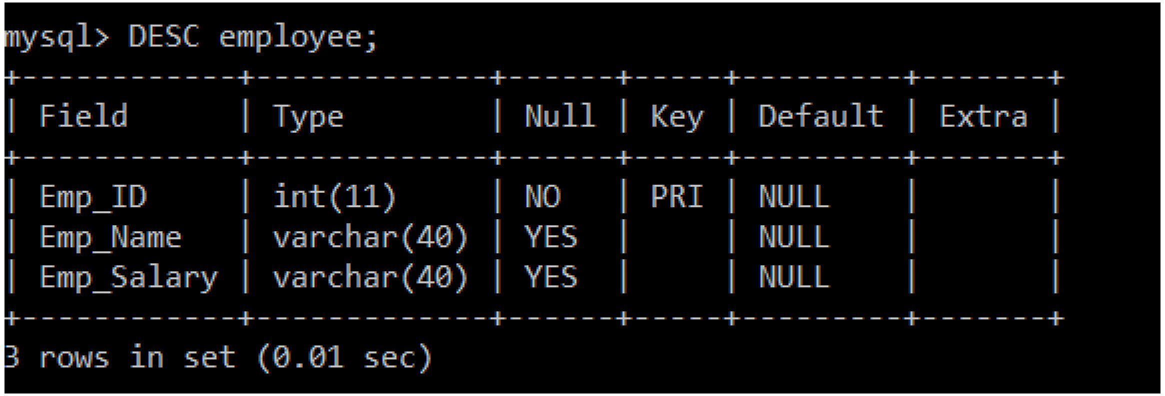
mysql> **CREATE TABLE** employee (Emp_ID **INT** NOT NULL **PRIMARY KEY**, Emp_Name **VARCHAR** (40), Emp_Salary **VARCHAR** (40));

To verify that the primary key constraint is applied to the student table's column, we will execute the following query:

mysql> **DESC** employee;

4. FOREIGN KEY

mysql> **DESC** employee;



```
mysql> DESC employee;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Emp_ID     | int(11)     | NO   | PRI | NULL    |       |
| Emp_Name   | varchar(40) | YES  |     | NULL    |       |
| Emp_Salary | varchar(40) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

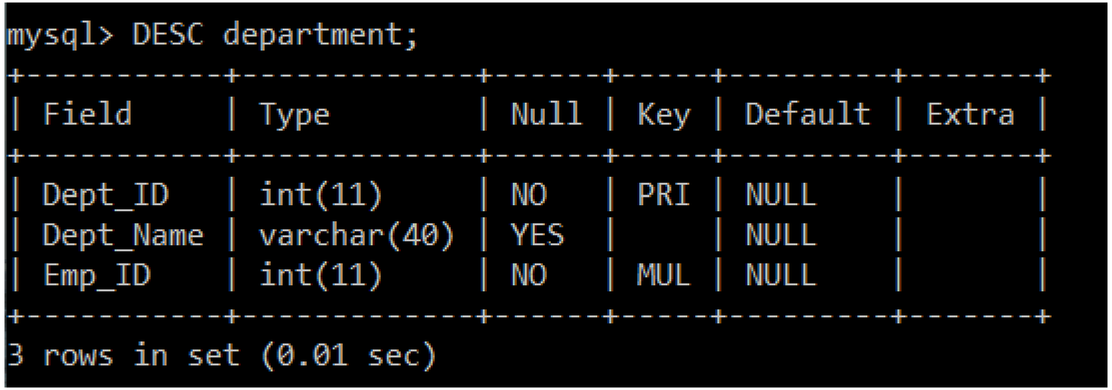
Now, we will write a query to apply a foreign key with a constraint name on the department table referring to the primary key of the employee table, i.e., Emp_ID.

mysql> **CREATE TABLE** department(Dept_ID **INT** NOT NULL **PRIMARY KEY**, Dept_Name **VARCHAR**(40), Emp_ID **INT** NOT NULL, **CONST RAINT** emp_id_fk **FOREIGN KEY**(Emp_ID) **REFERENCES** employee(Emp_ID));

To verify that the foreign key constraint is applied to the department table's column, we will execute the following query:

4. FOREIGN KEY

mysql> **DESC** employee;



```
mysql> DESC department;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Dept_ID   | int(11)     | NO   | PRI | NULL    |       |
| Dept_Name | varchar(40) | YES  |     | NULL    |       |
| Emp_ID    | int(11)     | NO   | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

4. FOREIGN KEY

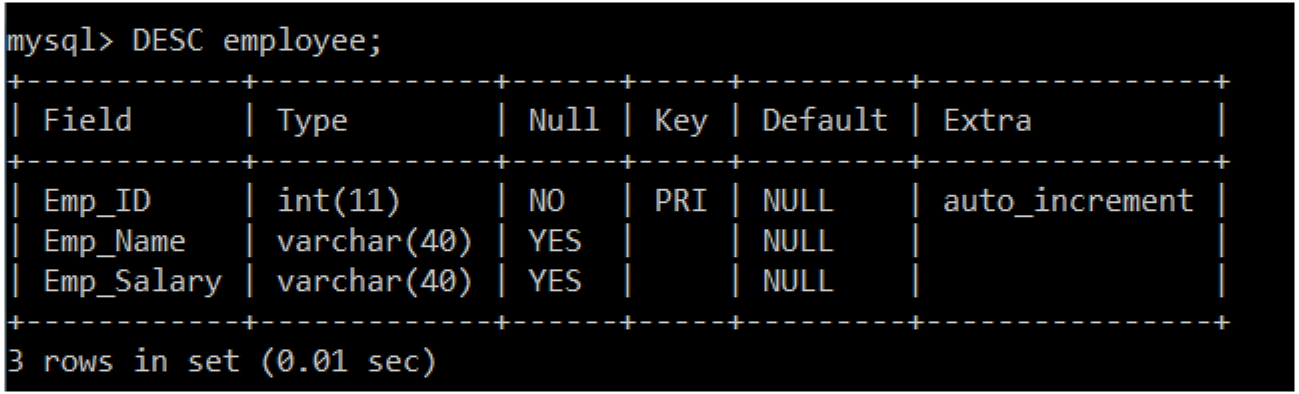
Syntax to apply the foreign key constraint on an existing table's column:

ALTER TABLE Parent_TableName **ADD FOREIGN KEY** (ColumnName) **REFERENCES** Child_TableName (ColumnName);

Example:

Consider we have an existing table employee and department. Later, we decided to apply a FOREIGN KEY constraint to the department table's column. Then we will execute the following query:

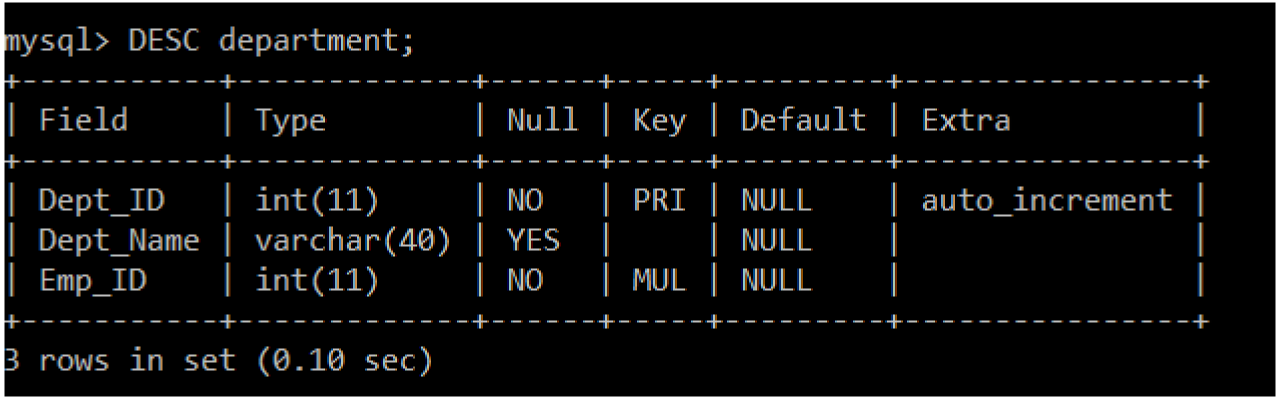
mysql> **DESC** employee;



```
mysql> DESC employee;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| Emp_ID     | int(11)     | NO   | PRI | NULL    | auto_increment |
| Emp_Name   | varchar(40) | YES  |     | NULL    |                |
| Emp_Salary | varchar(40) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

4. FOREIGN KEY

mysql> **ALTER TABLE** department **ADD FOREIGN KEY** (Emp_ID) **REFERENCES** employee (Emp_ID);
To verify that the foreign key constraint is applied to the department table's column, we will execute the following query:
mysql> **DESC** department;



```
mysql> DESC department;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| Dept_ID   | int(11)     | NO   | PRI | NULL    | auto_increment |
| Dept_Name | varchar(40) | YES  |     | NULL    |                |
| Emp_ID    | int(11)     | NO   | MUL | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.10 sec)
```

5. CHECK

Whenever a check constraint is applied to the table's column, and the user wants to insert the value in it, then the value will first be checked for certain conditions before inserting the value into that column.

For example: if we have an age column in a table, then the user will insert any value of his choice. The user will also enter even a negative value or any other invalid value. But, if the user has applied check constraint on the age column with the condition age greater than 18. Then in such cases, even if a user tries to insert an invalid value such as zero or any other value less than 18, then the age column will not accept that value and will not allow the user to insert it due to the application of check constraint on the age column.

Syntax to apply check constraint on a single column:

CREATE TABLE TableName (ColumnName1 datatype **CHECK** (ColumnName1 Condition), ColumnName2 datatype,…., ColumnNameN datatype);

Example:

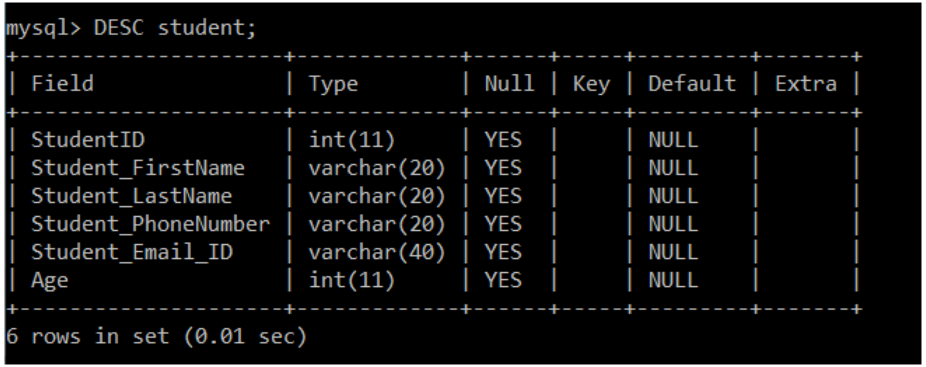
Create a student table and apply CHECK constraint to check for the age less than or equal to 15 while creating a table.

5. CHECK

mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40), Age **INT CHECK**(Age <= 15));

To verify that the check constraint is applied to the student table's column, we will execute the following query:

mysql> **DESC** student;



```
mysql> DESC student;
+---------------------+-------------+------+-----+---------+-------+
| Field               | Type        | Null | Key | Default | Extra |
+---------------------+-------------+------+-----+---------+-------+
| StudentID           | int(11)     | YES  |     | NULL    |       |
| Student_FirstName   | varchar(20) | YES  |     | NULL    |       |
| Student_LastName    | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID    | varchar(40) | YES  |     | NULL    |       |
| Age                 | int(11)     | YES  |     | NULL    |       |
+---------------------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

5. CHECK

Syntax to apply check constraint on multiple columns:

1.**CREATE TABLE** TableName (ColumnName1 datatype, ColumnName2 datatype **CHECK** (ColumnName1 Condition AND ColumnName2 Condition),…., ColumnNameN datatype);

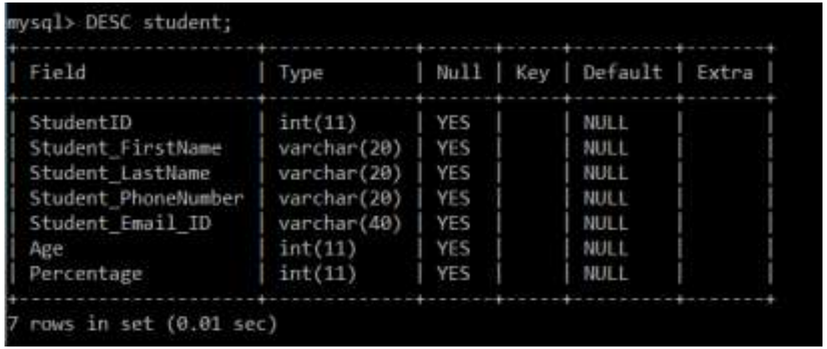
Example:

Create a student table and apply CHECK constraint to check for the age less than or equal to 15 and a percentage greater than 85 while creating a table.

1.mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40), Age **INT**, Percentage **INT**, **CHECK**(Age <= 15 AND Percentage > 85));

To verify that the check constraint is applied to the age and percentage column, we will execute the following query:

mysql> **DESC** student;



```
mysql> DESC student;
+---------------------+-------------+------+-----+---------+-------+
| Field               | Type        | Null | Key | Default | Extra |
+---------------------+-------------+------+-----+---------+-------+
| StudentID           | int(11)     | YES  |     | NULL    |       |
| Student_FirstName   | varchar(20) | YES  |     | NULL    |       |
| Student_LastName    | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID    | varchar(40) | YES  |     | NULL    |       |
| Age                 | int(11)     | YES  |     | NULL    |       |
| Percentage          | int(11)     | YES  |     | NULL    |       |
+---------------------+-------------+------+-----+---------+-------+
7 rows in set (0.01 sec)
```

5. CHECK

Syntax to apply check constraint on an existing table's column:

ALTER TABLE TableName **ADD CHECK** (ColumnName Condition);

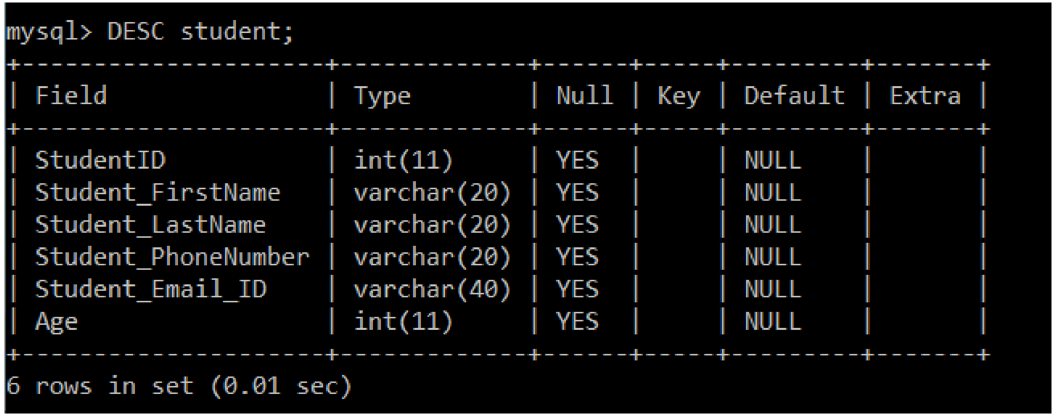
Example:

Consider we have an existing table student. Later, we decided to apply the CHECK constraint on the student table's column. Then we will execute the following query:

mysql> **ALTER TABLE** student **ADD CHECK** (Age <=15);

To verify that the check constraint is applied to the student table's column, we will execute the following query:

mysql> **DESC** student;



```
mysql> DESC student;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| StudentID            | int(11)     | YES  |     | NULL    |       |
| Student_FirstName    | varchar(20) | YES  |     | NULL    |       |
| Student_LastName     | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber  | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID     | varchar(40) | YES  |     | NULL    |       |
| Age                  | int(11)     | YES  |     | NULL    |       |
+----------------------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

6. DEFAULT

Whenever a default constraint is applied to the table's column, and the user has not specified the value to be inserted in it, then the default value which was specified while applying the default constraint will be inserted into that particular column.

Syntax to apply default constraint during table creation:

CREATE TABLE TableName (ColumnName1 datatype **DEFAULT** Value, ColumnName2 datatype,…., ColumnNameN datatype);

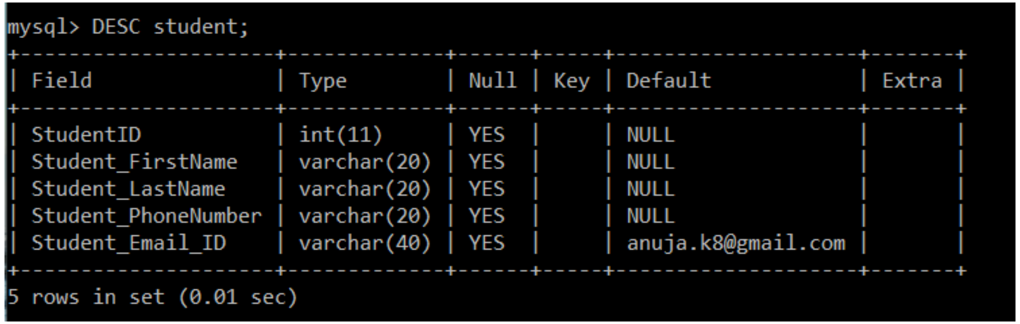
Example:

Create a student table and apply the default constraint while creating a table.

mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40) **DEFAULT** "anuja.k8@gmail.com");

6. DEFAULT

mysql> **DESC** student;



```
mysql> DESC student;
+---------------------+-------------+------+-----+--------------------+-------+
| Field               | Type        | Null | Key | Default            | Extra |
+---------------------+-------------+------+-----+--------------------+-------+
| StudentID           | int(11)     | YES  |     | NULL               |       |
| Student_FirstName   | varchar(20) | YES  |     | NULL               |       |
| Student_LastName    | varchar(20) | YES  |     | NULL               |       |
| Student_PhoneNumber | varchar(20) | YES  |     | NULL               |       |
| Student_Email_ID    | varchar(40) | YES  |     | anuja.k8@gmail.com |       |
+---------------------+-------------+------+-----+--------------------+-------+
5 rows in set (0.01 sec)
```

Syntax to apply default constraint on an existing table's column:

ALTER TABLE TableName **ALTER** ColumnName **SET DEFAULT** Value;

Example:

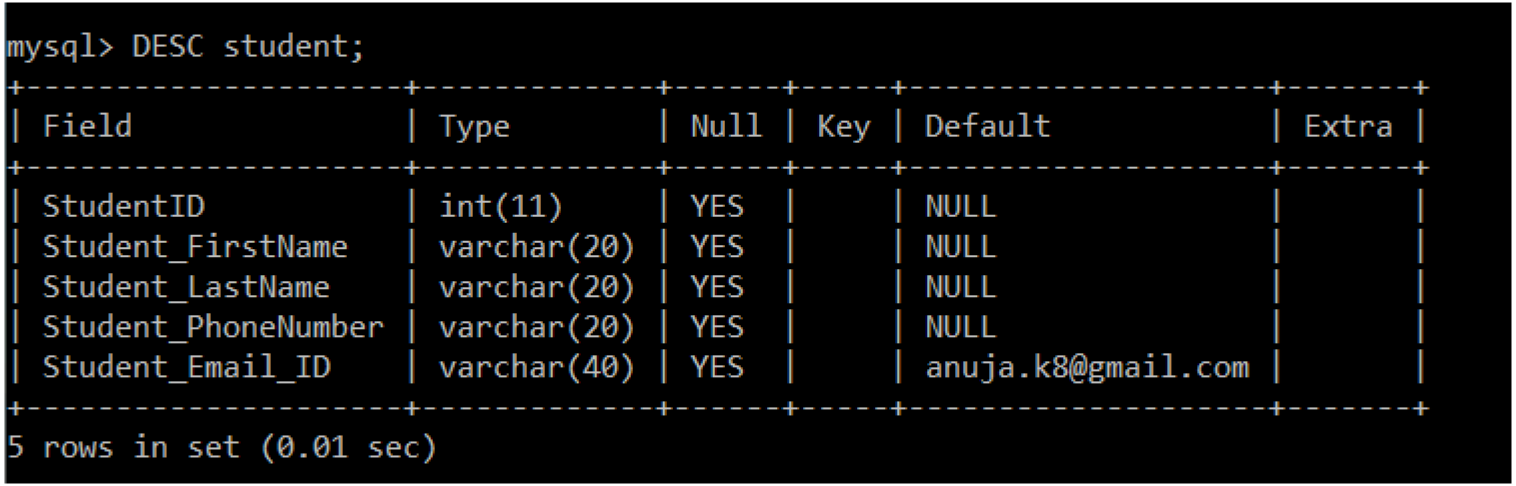
Consider we have an existing table student. Later, we decided to apply the DEFAULT constraint on the student table's column. Then we will execute the following query:

mysql> **ALTER TABLE** student **ALTER** Student_Email_ID **SET DEFAULT** "anuja.k8@gmail.com";

To verify that the default constraint is applied to the student table's column, we will execute the following query:

6. DEFAULT

mysql> **DESC** student;



```
mysql> DESC student;
+----------------------+-------------+------+-----+---------------------+-------+
| Field                | Type        | Null | Key | Default             | Extra |
+----------------------+-------------+------+-----+---------------------+-------+
| StudentID            | int(11)     | YES  |     | NULL                |       |
| Student_FirstName    | varchar(20) | YES  |     | NULL                |       |
| Student_LastName     | varchar(20) | YES  |     | NULL                |       |
| Student_PhoneNumber  | varchar(20) | YES  |     | NULL                |       |
| Student_Email_ID     | varchar(40) | YES  |     | anuja.k8@gmail.com  |       |
+----------------------+-------------+------+-----+---------------------+-------+
5 rows in set (0.01 sec)
```

7. CREATE INDEX

CREATE INDEX constraint is used to create an index on the table. Indexes are not visible to the user, but they help the user to speed up the searching speed or retrieval of data from the database.

Syntax to create an index on single column:

CREATE INDEX IndexName **ON** TableName (ColumnName 1);

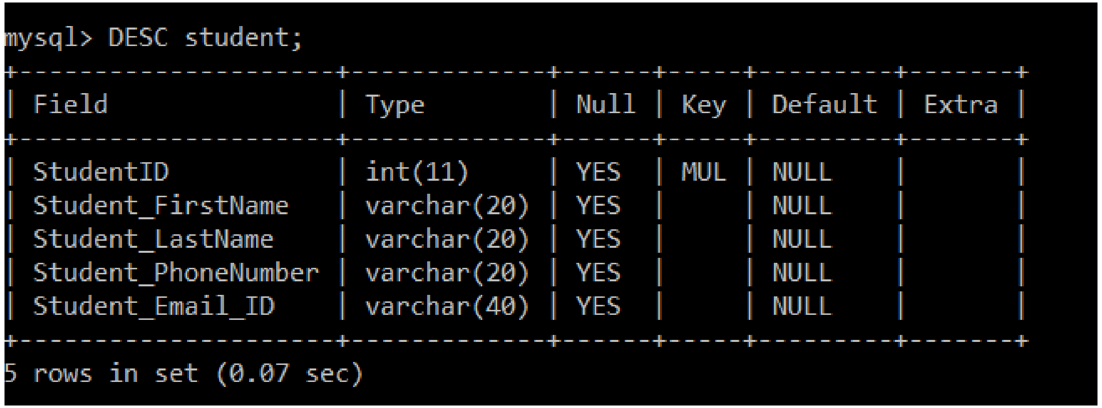
Example:

Create an index on the student table and apply the default constraint while creating a table.

mysql> **CREATE INDEX** idx_StudentID **ON** student (StudentID);

To verify that the create index constraint is applied to the student table's column, we will execute the following query:

mysql> **DESC** student;



```
mysql> DESC student;
+---------------------+--------------+------+-----+---------+-------+
| Field               | Type         | Null | Key | Default | Extra |
+---------------------+--------------+------+-----+---------+-------+
| StudentID           | int(11)      | YES  | MUL | NULL    |       |
| Student_FirstName   | varchar(20)  | YES  |     | NULL    |       |
| Student_LastName    | varchar(20)  | YES  |     | NULL    |       |
| Student_PhoneNumber | varchar(20)  | YES  |     | NULL    |       |
| Student_Email_ID    | varchar(40)  | YES  |     | NULL    |       |
+---------------------+--------------+------+-----+---------+-------+
5 rows in set (0.07 sec)
```

7. CREATE INDEX

Syntax to create an index on multiple columns:

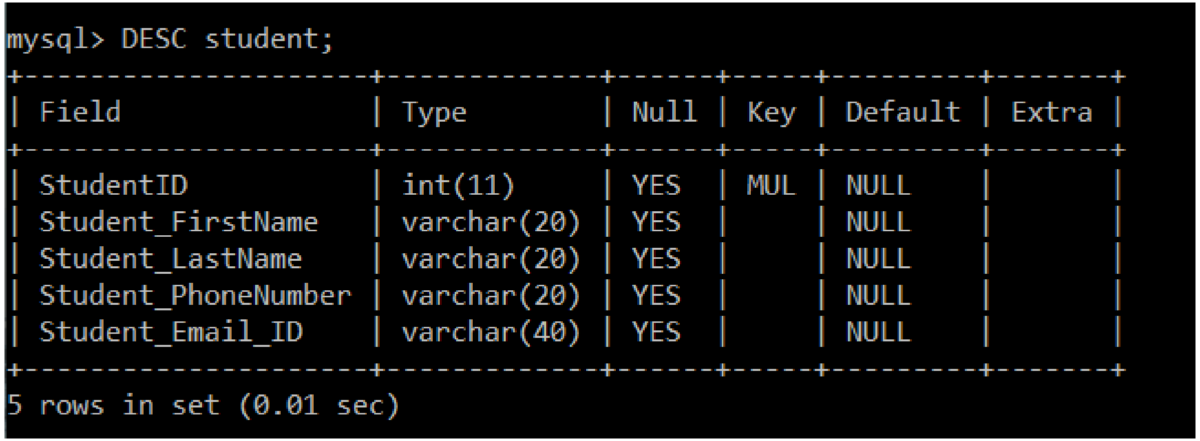
CREATE INDEX IndexName **ON** TableName (ColumnName 1, ColumnName 2, ColumnName N);

Example:

mysql> **CREATE INDEX** idx_Student **ON** student (StudentID, Student_PhoneNumber);

To verify that the create index constraint is applied to the student table's column, we will execute the following query:

mysql> **DESC** student;



```
mysql> DESC student;
+---------------------+-------------+------+-----+---------+-------+
| Field               | Type        | Null | Key | Default | Extra |
+---------------------+-------------+------+-----+---------+-------+
| StudentID           | int(11)     | YES  | MUL | NULL    |       |
| Student_FirstName   | varchar(20) | YES  |     | NULL    |       |
| Student_LastName    | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID    | varchar(40) | YES  |     | NULL    |       |
+---------------------+-------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

7. CREATE INDEX

Syntax to create an index on an existing table:

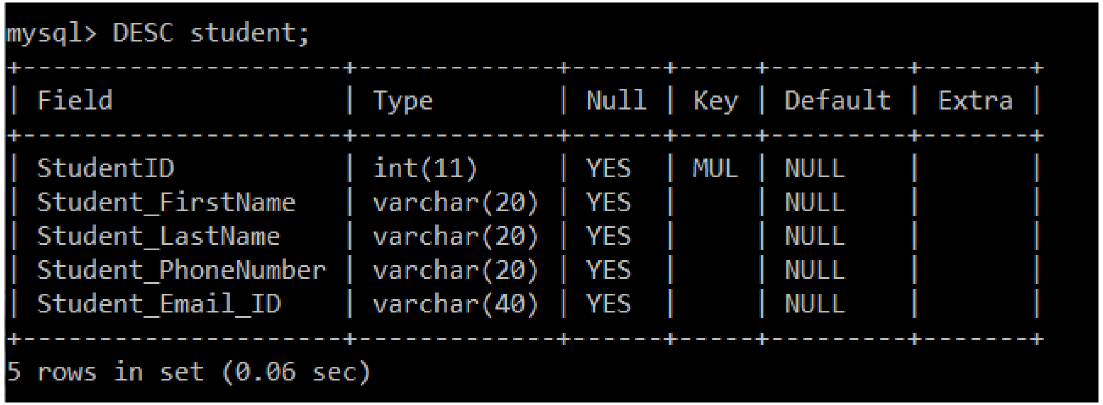
ALTER TABLE TableName **ADD INDEX** (ColumnName);

Consider we have an existing table student. Later, we decided to apply the DEFAULT constraint on the student table's column. Then we will execute the following query:

mysql> **ALTER TABLE** student **ADD INDEX** (StudentID);

To verify that the create index constraint is applied to the student table's column, we will execute the following query:

mysql> **DESC** student;



```
mysql> DESC student;
+----------------------+-------------+------+-----+---------+-------+
| Field                | Type        | Null | Key | Default | Extra |
+----------------------+-------------+------+-----+---------+-------+
| StudentID            | int(11)     | YES  | MUL | NULL    |       |
| Student_FirstName    | varchar(20) | YES  |     | NULL    |       |
| Student_LastName     | varchar(20) | YES  |     | NULL    |       |
| Student_PhoneNumber  | varchar(20) | YES  |     | NULL    |       |
| Student_Email_ID     | varchar(40) | YES  |     | NULL    |       |
+----------------------+-------------+------+-----+---------+-------+
5 rows in set (0.06 sec)
```

Constraints

In **Database Management Systems (DBMS)**, **constraints** are rules applied to table columns to ensure the accuracy and reliability of the data stored in the database. These rules help to enforce data integrity, prevent invalid data entry, and maintain relationships between tables.

Here's an overview of the most common types of constraints in DBMS:

Types of Constraints in DBMS

1. **NOT NULL Constraint**
2. **UNIQUE Constraint**
3. **PRIMARY KEY Constraint**
4. **FOREIGN KEY Constraint**
5. **CHECK Constraint**
6. **DEFAULT Constraint**
7. **INDEX Constraint**

(though not always considered a strict "constraint," it's used for performance optimization)

8. ASSERTION

(a general constraint on multiple tables or the whole database, typically supported in advanced DBMS)

Constraints

NOT NULL Constraint

•**Definition**: Ensures that a column cannot store NULL values. In other words, every row must have a valid value for this column.

•**Purpose**: Used when certain fields in a table are mandatory and must always have a value.

•**Example**:

```
CREATE TABLE Employees (emp_id INT NOT NULL,       -- emp_id cannot be NULL
first_name VARCHAR(50) NOT NULL,  -- first_name cannot be NULL
 last_name VARCHAR(50));
```

In the above table, emp_id and first_name cannot have NULL values.

Constraints

2. UNIQUE Constraint

Definition: Ensures that all values in a column (or a combination of columns) are unique across all rows in the table. It allows NULL values (unless the column is also defined as NOT NULL), but each non-NULL value must be unique.

Purpose: Used to enforce uniqueness, for example, for email addresses or usernames.

Example:

```
CREATE TABLE Customers (
customer_id INT NOT NULL PRIMARY KEY,
email VARCHAR(100) UNIQUE -- email must be unique but can be NULL );
```

Here, the **email column** must contain unique values, but it can also be **NULL.**

Constraints

3. PRIMARY KEY Constraint

Definition: A primary key is a combination of columns that uniquely identifies each row in a table. A primary key is automatically NOT NULL and must contain unique values.

Purpose: It ensures that each row can be uniquely identified, and it enforces both uniqueness and non-nullability.

Example:

```
CREATE TABLE Employees (
emp_id INT PRIMARY KEY, -- emp_id is the primary key
 first_name VARCHAR(50),
 last_name VARCHAR(50) );
```

In this table, the emp_id column is the primary key. It automatically ensures that no two employees can have the same emp_id, and the emp_id cannot be NULL.

Constraints

4. FOREIGN KEY Constraint

Definition: A foreign key is a column (or a combination of columns) in one table that refers to the primary key or a unique key in another table. It establishes and enforces a link between the data in the two tables.

Purpose: It ensures referential integrity by ensuring that every value in the foreign key column corresponds to an existing value in the referenced table.

Example:

```
CREATE TABLE Orders (
order_id INT PRIMARY KEY, customer_id INT,
order_date DATE,
FOREIGN KEY (customer_id) REFERENCES Customers(customer_id) );
```

In this table, the emp_id column is the primary key. It automatically ensures that no two employees can have the same emp_id, and the emp_id cannot be NULL.

In this example, customer_id in the Orders table is a **foreign key** referencing the customer_id column in the Customers table. This ensures that an order can only be placed by a customer who exists in the Customers table.

Constraints

5. CHECK Constraint

Definition: The CHECK constraint is used to limit the values that can be inserted into a column. It ensures that the data meets a specific condition.

Purpose: Used for enforcing domain integrity by restricting the values in a column based on certain criteria.

Example:

```
CREATE TABLE Employees (
emp_id INT PRIMARY KEY,
salary DECIMAL(10, 2),
CHECK (salary >= 0) -- salary must be greater than or equal to 0 );
```

In this table, the salary column must always have a value greater than or equal to 0.
If you try to insert a negative salary value, it will be rejected.

Constraints

6. DEFAULT Constraint

Definition: The DEFAULT constraint is used to provide a default value for a column when no value is specified during record insertion.

Purpose: Used to insert a default value into a column if no value is provided during the INSERT operation.

Example:

```
CREATE TABLE Employees (
emp_id INT PRIMARY KEY,
first_name VARCHAR(50),
last_name VARCHAR(50),
status VARCHAR(20) DEFAULT 'Active' -- Default value is 'Active' );
```

In this case, if the status column is not provided during insertion, it will default to 'Active'.

Constraints

7. INDEX Constraint (Optional)

- **Definition**: An index is a database object that improves the speed of data retrieval operations on a table. While it’s not always classified as a "constraint" in traditional terms, it can be used to enforce uniqueness in some cases.
- **Purpose**: Used for optimizing query performance, especially for searching, sorting, or joining tables.
- **Example**:

```
CREATE INDEX idx_last_name ON Employees (last_name);
```

This creates an index on the last_name column of the Employees table to speed up queries that filter or sort by last_name.

Constraints

8. ASSERTION (Advanced Constraint)

•**Definition**: An assertion is a more complex type of constraint that is used to define a condition that must hold true for the entire database or multiple tables. Assertions are not widely supported in many DBMS products but can be used for global checks.

•**Example**: This is a conceptual example of how assertions might look, but it’s not widely supported in databases like MySQL or PostgreSQL.

```
CREATE ASSERTION employee_salary_check CHECK (NOT EXISTS
(SELECT * FROM Employees WHERE salary < 0));
```

•This example ensures that there are no employees with a salary less than 0 across the entire database.

Constraints

Summary of Constraints and Their Purposes

Constraint	Purpose
NOT NULL	Ensures that a column cannot have NULL values.
UNIQUE	Ensures that all values in a column (or combination of columns) are unique across all rows.
PRIMARY KEY	Uniquely identifies each row in a table and automatically enforces NOT NULL and UNIQUE constraints.
FOREIGN KEY	Enforces a link between two tables by referencing a primary or unique key in another table.
CHECK	Ensures that values in a column meet a specific condition (e.g., salary >= 0).
DEFAULT	Provides a default value for a column when no value is specified.
INDEX	Optimizes query performance by creating a fast lookup mechanism.
ASSERTION	Enforces a condition that must be true across multiple tables or the whole database (less common).

Constraints

- **Data Integrity**: Constraints are primarily used to enforce data integrity,
- ensuring that the data stored in the database is valid, accurate, and consistent.
- **Relationships**: Constraints like

PRIMARY KEY and FOREIGN KEY help establish and maintain relationships between tables, ensuring referential integrity.

- **Validation**: Constraints such as

CHECK and NOT NULL are used to validate data before it's entered into the table, preventing invalid data from being stored.

- **Default Values**: Constraints like

DEFAULT make it easier to manage optional data by automatically filling in values when none are provided.

By applying constraints, you can maintain data consistency and improve the reliability and performance of your database.

References

1. https://www.javatpoint.com/constraints-in-sql
2. https://www.geeksforgeeks.org/sql-constraints/
3. https://www.tutorialspoint.com/sql/sql-constraints.htm