EXPERIMENT-2

Lab 2. To understand and apply the concept of Constraints.

Objective: To understand the concept of data constraints that are enforced on data being stored in the table. Focus on Primary Key and the Foreign Key.

Constraints in SQL

Constraints in SQL means we are applying certain conditions or restrictions on the database. This further means that before inserting data into the database, we are checking for some conditions. If the condition we have applied to the database holds true for the data which is to be inserted, then only the data will be inserted into the database tables.

Constraints in SQL can be categorized into two types:

1. **Column Level Constraint:** Column Level Constraint is used to apply a constraint on a single column.
2. **Table Level Constraint:** Table Level Constraint is used to apply a constraint on multiple columns.

Some of the real-life examples of constraints are as follows:

1. Every person has a unique email id. This is because while creating an email account for any user, the email providing services such as Gmail, Yahoo or any other email providing service will always check for the availability of the email id that the user wants for himself. If some other user already takes the email id that the user wants, then that id cannot be assigned to another user. This simply means that no two users can have the same email ids on the same email providing service. So, here the email id is the constraint on the database of email providing services.

2. Whenever we set a password for any system, there are certain constraints that are to be followed. These constraints may include the following:
    - There must be one uppercase character in the password.
    - Password must be of at least eight characters in length.
    - Password must contain at least one special symbol.

Constraints available in SQL are:

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

Now let us try to understand the different constraints available in SQL in more detail with the help of examples. We will use MySQL database for writing all the queries.

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:

1. **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.

1. **CREATE TABLE** student(StudentID **INT** NOT NULL, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40));

```
mysql> CREATE TABLE student(StudentID INT NOT NULL, Student_FirstName VARCHAR(20), Student_LastName VARCHAR(20), Student_PhoneNumber VARCHAR(20), Student_
Email_ID VARCHAR(40));
Query OK, 0 rows affected (0.28 sec)
```

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:

1. 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)
```

Syntax to apply the NOT NULL constraint on an existing table's column:

1. **ALTER TABLE** TableName CHANGE Old_ColumnName New_ColumnName Datatype NOT NULL;

Example:

Consider we have an existing table student, without any constraints applied to it. Later, we decided to apply a NOT NULL constraint to one of the table's column. Then we will execute the following query:

1. mysql> **ALTER TABLE** student CHANGE StudentID StudentID **INT** NOT NULL;

```
mysql> ALTER TABLE student CHANGE StudentID StudentID INT NOT NULL;
Query OK, 0 rows affected (0.26 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

To verify that the not null constraint is applied to the student table's column, we will execute the following query:

1. 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 (0.01 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:

1. **CREATE TABLE** TableName (ColumnName1 datatype **UNIQUE**, ColumnName2 datatype ,…., ColumnNameN datatype);

Example:

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

1. mysql> **CREATE TABLE** student(StudentID **INT UNIQUE**, Student_FirstName **VARCH AR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Stud ent_Email_ID **VARCHAR**(40));

```
mysql> CREATE TABLE student(StudentID INT UNIQUE, Student_FirstName VARCHAR(20), Student_LastName VARCHAR(20), Student_PhoneNumber VARCHAR(20),
 Student_Email_ID VARCHAR(40));
Query OK, 0 rows affected (0.12 sec)
```

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:

1. 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)
```

Syntax to apply the UNIQUE constraint on more than one column:

1. **CREATE TABLE** TableName (ColumnName1 datatype, ColumnName2 datatype,…., ColumnNameN datatype, **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.

1. mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ID **VARCHAR**(40), **UNIQUE**(StudentID, Student_PhoneNumber));

```
mysql> CREATE TABLE student(StudentID INT, Student_FirstName VARCHAR(20), Student_LastName VARCHAR(20), Student_PhoneNumber VARCHAR(20), Student_Email_ID VARCHAR(40), UNIQUE(StudentID, Student_PhoneNumber));
Query OK, 0 rows affected (0.15 sec)
```

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:

1. 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)
```

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

1. **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:

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

```
mysql> ALTER TABLE student ADD UNIQUE(StudentID);
Query OK, 0 rows affected (0.17 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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:

1. 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:

1. **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.

1. mysql> **CREATE TABLE** student(StudentID **INT PRIMARY KEY**, Student_FirstName **V ARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(2 0), Student_Email_ID **VARCHAR**(40));

```
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));
Query OK, 0 rows affected (0.12 sec)
```

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:

1. 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)
```

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

1. **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:

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

```
mysql> ALTER TABLE student ADD PRIMARY KEY(StudentID);
Query OK, 0 rows affected (0.21 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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

1. 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:

1. **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.

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

```
mysql> CREATE TABLE employee(Emp_ID INT NOT NULL PRIMARY KEY, Emp_Name VARCHAR(40), Emp_Salary VARCHAR(40));
Query OK, 0 rows affected (0.13 sec)
```

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

1. 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.

1. 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));

```
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));
Query OK, 0 rows affected (0.10 sec)
```

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

1. mysql> **DESC** department;

```
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)
```

Syntax to apply the foreign key constraint with constraint name:

1. **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.

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

```
mysql> CREATE TABLE employee(Emp_ID INT NOT NULL PRIMARY KEY, Emp_Name VARCHAR(40), Emp_Salary VARCHAR(40));
Query OK, 0 rows affected (0.11 sec)
```

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

1. 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.

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

```
mysql> CREATE TABLE department(Dept_ID INT NOT NULL PRIMARY KEY, Dept_Name VARCHAR(40), Emp_ID INT NOT NULL, CONSTRAINT emp_id_fk FOREIGN KEY(Emp_ID
) REFERENCES employee(Emp_ID));
Query OK, 0 rows affected (0.12 sec)
```

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

1. mysql> **DESC** department;

```
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)
```

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

1. **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:

1. 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)
```

1. mysql> **ALTER TABLE** department **ADD FOREIGN KEY** (Emp_ID) **REFERENCES** employee (Emp_ID);

```
mysql> ALTER TABLE department ADD FOREIGN KEY(Emp_ID) REFERENCES employee(Emp_ID);
Query OK, 0 rows affected (0.21 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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

1. 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:

1. **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.

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 CHECK**(Age <= 15));

```
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));
Query OK, 0 rows affected (0.22 sec)
```

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

1. 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)
```

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));

```
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));
Query OK, 0 rows affected (0.13 sec)
```

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

1. 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)
```

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

1. **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:

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

```
mysql> ALTER TABLE student ADD CHECK( Age <=15 );
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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

1. 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:

1. **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.

1. mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), St udent_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ ID **VARCHAR**(40) **DEFAULT** "anuja.k8@gmail.com");

```
mysql> CREATE TABLE student(StudentID INT, Student_FirstName VARCHAR(20), Student_LastName VARCHAR(20), Student_PhoneNumber VARCHAR(20), Student_Ema
il_ID VARCHAR(40) DEFAULT "anuja.k8@gmail.com");
Query OK, 0 rows affected (0.13 sec)
```

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

1. 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:

1. **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:

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

```
mysql> ALTER TABLE student ALTER Student_Email_ID SET DEFAULT "anuja.k8@gmail.com";
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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

1. 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:

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

Example:

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

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

```
mysql> CREATE INDEX idx_StudentID ON student(StudentID);
Query OK, 0 rows affected (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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

1. 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)
```

Syntax to create an index on multiple columns:

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

Example:

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

```
mysql> CREATE INDEX idx_Student ON student(StudentID, Student_PhoneNumber);
Query OK, 0 rows affected (0.16 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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

1. 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)
```

Syntax to create an index on an existing table:

1. **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:

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

```
mysql> ALTER TABLE student ADD INDEX(StudentID);
Query OK, 0 rows affected (0.14 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

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

1. 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)
```

LAB Performance Questions:

You are a junior database administrator at a company called TechCo. The company manages information about its products, customers, and orders through a relational database. Your task is to perform various SQL queries to retrieve and manipulate data within this database.

1. Write an SQL script that includes DDL statements to create the following tables:
   - `Products` with columns: `ProductID` (integer), `ProductName` (varchar), `UnitPrice` (decimal), `Manufacturer` (varchar).
   - `Customers` with columns: `CustomerID` (integer), `CustomerName` (varchar), `Email` (varchar).
   Ensure appropriate constraints for data integrity.

2. Insert at least three sample records into the `Products` table created in Question 1. Ensure each record has a unique product ID.
   Insert at least three sample records into the `Customers` table created in Question 1. Each record should have a unique customer ID.

3. Add a column named `PhoneNumber` to the `Customers` table. Apply the NOT NULL constraint to ensure that this information is always provided.

4. Implement a UNIQUE constraint on the `Email` column in the `Customers` table. Provide an example of how this constraint prevents data duplication.

5. Set the `ProductID` column as the primary key for the `Products` table. Also, define `CustomerID` as the primary key for the `Customers` table.

6. Establish a relationship between the `Orders` table (containing columns `OrderID` and `CustomerID`) and the `Customers` table. Ensure referential integrity by using a FOREIGN KEY constraint.

7. Add a column named `OrderQuantity` to the `Orders` table. Apply a CHECK constraint to limit the order quantity to a range between 1 and 100.

8. Include a column named `OrderStatus` in the `Orders` table. Set a DEFAULT constraint so that if the status is not specified during insertion, it defaults to 'Pending.'

9. Create an index on the `UnitPrice` column in the `Products` table. Explain how this index can improve query performance.

10. Write an SQL query to retrieve the names and email addresses of all customers who placed orders. Join the `Customers` and `Orders` tables to achieve this.

11. Modify the `OrderStatus` of all orders with a quantity greater than 50 to 'Shipped.' Provide the SQL statement for this update.

12. Delete all records from the `Products` table where the `UnitPrice` is less than $10. Include the SQL statement for this deletion.

13. The `UnitPrice` column in the `Products` table needs to be changed to a new datatype, `float`. Write an SQL statement using the ALTER TABLE statement to make this change.

14. Write an SQL query to calculate the average `UnitPrice` of all products in the `Products` table.