NAME: Divyansh Kumar Singh
SAP ID: 590012175
BATCH: B1
DBMS LAB 2

AIM: *To Implement and Understand the Concept Of Constraints*

Lab 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.

Name: Divyansh Kumar Singh
SAP ID: 590012175
Batch: B1

classmate
Date 19/8/25
Page

Lab-2

Aim: To implement and understand the concept of constraints.

1. CREATE ⇒ For creating a database and tables namely "Products" and "Customers".

   INT, VARCHAR(N), DECIMAL ⇒ Data Types.

   NOT NULL ⇒ Constraint such that the field shouldn't be empty.

   DESC ⇒ For displaying the structure of tables.

2. INSERT ⇒ For inserting values into table

   VALUES ⇒ The rowwise values for the table is sequence to schema.

   SELECT * FROM ⇒ For displaying the entire table.

3. ALTER ⇒ To modify Customers table.
   ADD ⇒ To add a column PhoneNumber in table.
   NOTNULL, DEFAULT ⇒ Constraints to make the field to not be empty and default number as "0000000000".

   CHECK ⇒ Checking the length of phone number.

4. Altering table by adding a columns "unique email" having constraint UNIQUE.
Two fields with same email will show error.

5. ADD PRIMRY KEY (column) ⇒ To make the column as Primary Key.

6. Creating a new table "Orders"
FOREIGN KEY (customerID) References Customers (CustomerID);
- FOREIGN KEY → To create a foreign key
- (customerID) → column
- References Customers → To which table & column it is referencing
- (CustomerID) → column name
- Customers → Table

7. BETWEEN 1 AND 100 ⇒ The value should be within the range (1 to 100)

8. DEFAULT ⇒ Making "Pending" as default value for order status if no value is given.

9. The index will speed up queries that filter or sort by UnitPrice.

10. Joining Customers object c CustomerId and Orders object O. [JOIN]

11. Verify OrderQuantity > 50 for getting shipped. using WHERE clause.

12. Delete fields from Products where unitprice is less than 10.

13. MODIFY the data type to FLOAT.

14. AVG(UnitPrice) will give average of values in UnitPrice and AS will store it in column AveragePrice. FROM table products.

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 an empty column, not even zero.

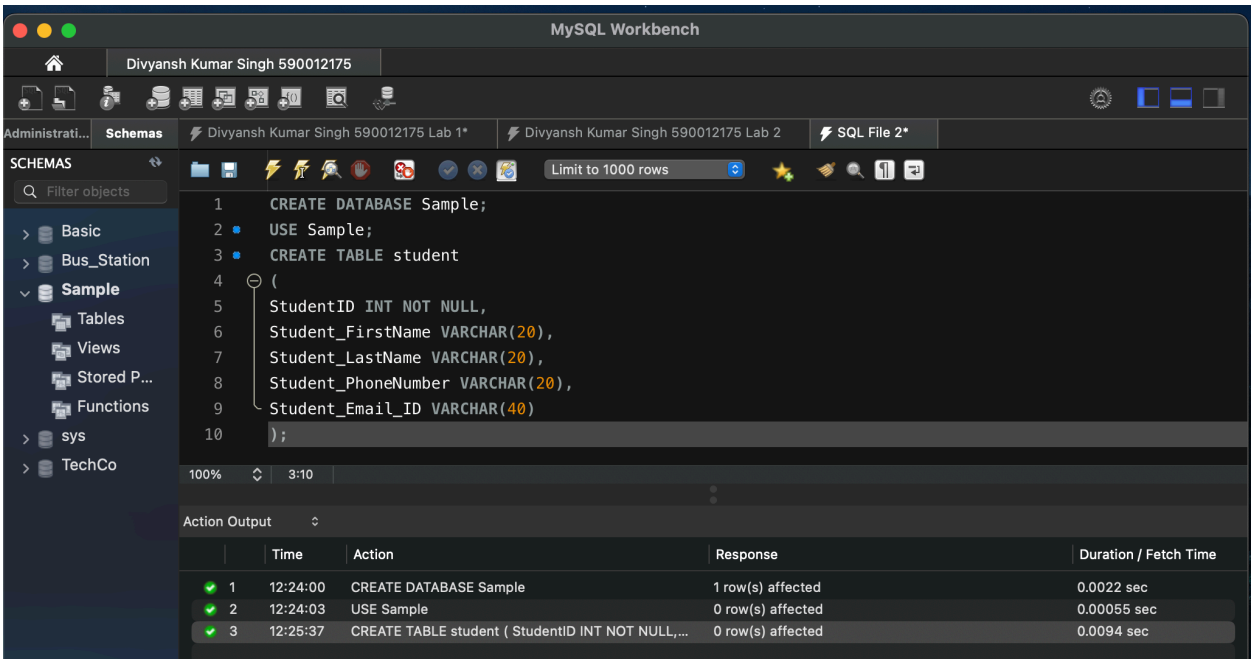
Syntax to apply the NOT NULL constraint during table creation:

1. **CREATE TABLE** TableName (ColumnName1 datatype NOT NULL, ColumnName2 dataty pe,…., ColumnNameN datatype);

Example:

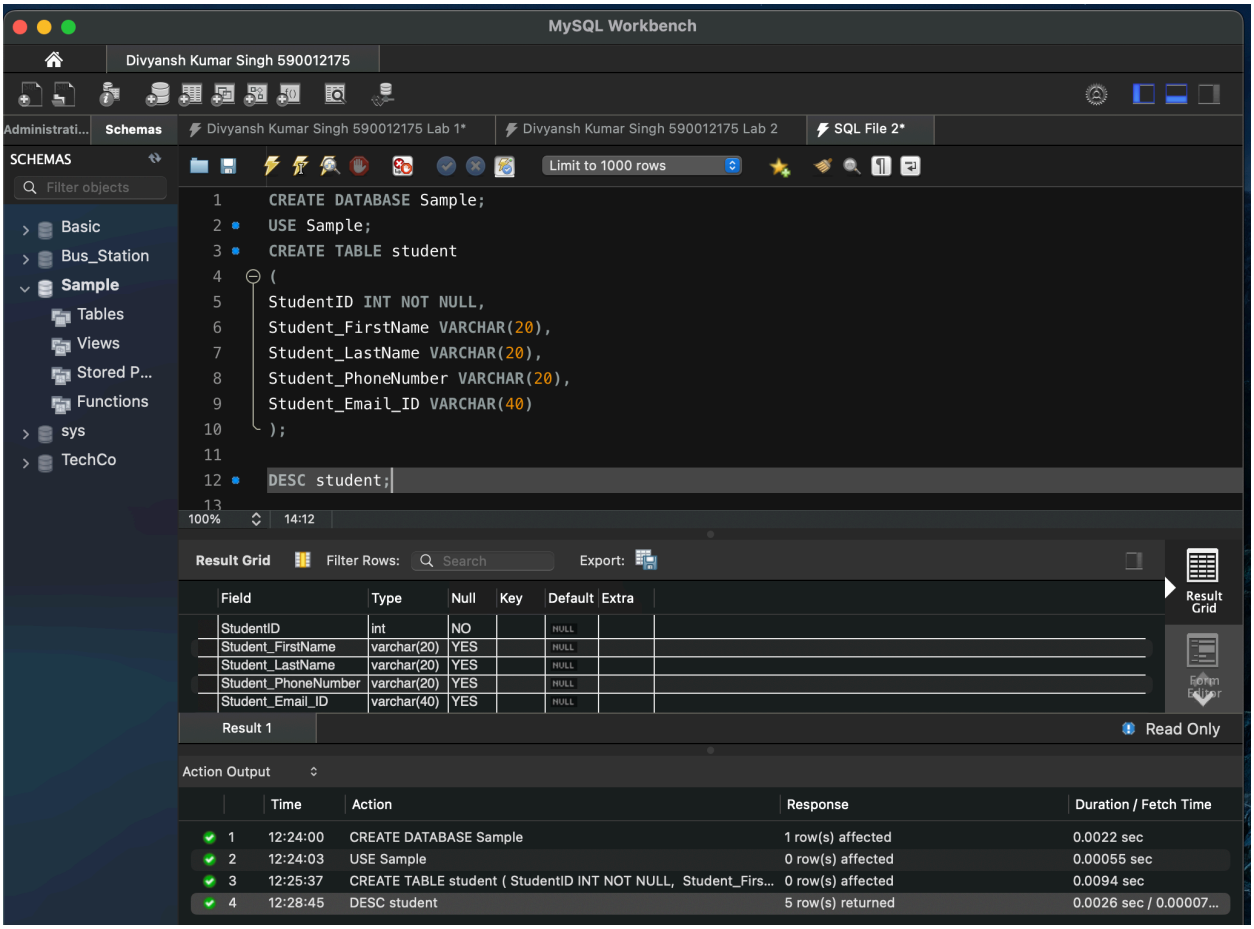
Create a student table and apply a NOT NULL constraint on one of the table's column while creating a table.

1. **CREATE TABLE** student(StudentID **INT** NOT NULL, Student_FirstName **VARCHAR**(20), Student_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_E mail_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:

1. mysql> **DESC** student;



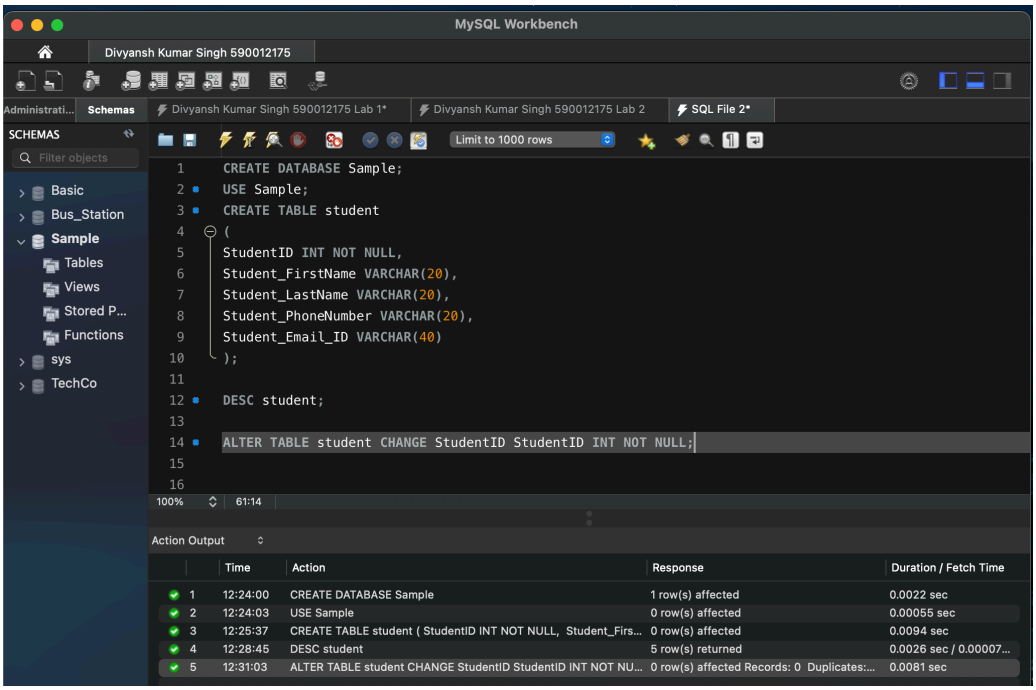
Syntax to apply the NOT NULL constraint on an existing table's column:

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

Example:

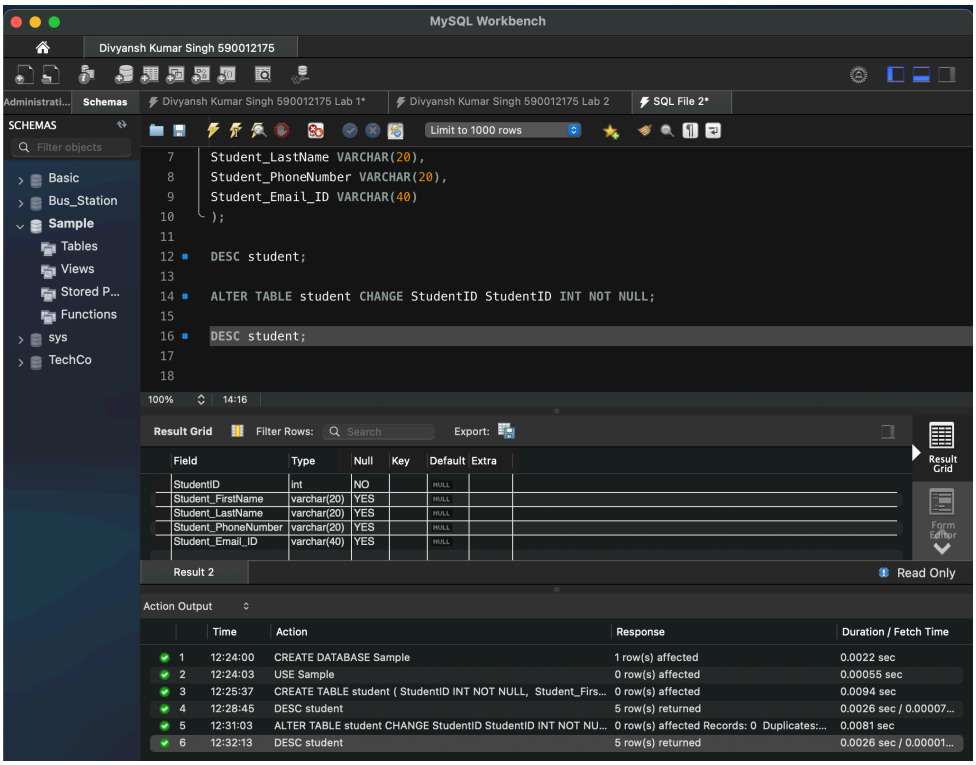
Consider we have an existing table student, without any constraints applied to it. Later, we decided to apply a NOT NULL constraint to one of the table's column. Then we will execute the following query:

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



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

2. mysql> **DESC** student;



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:

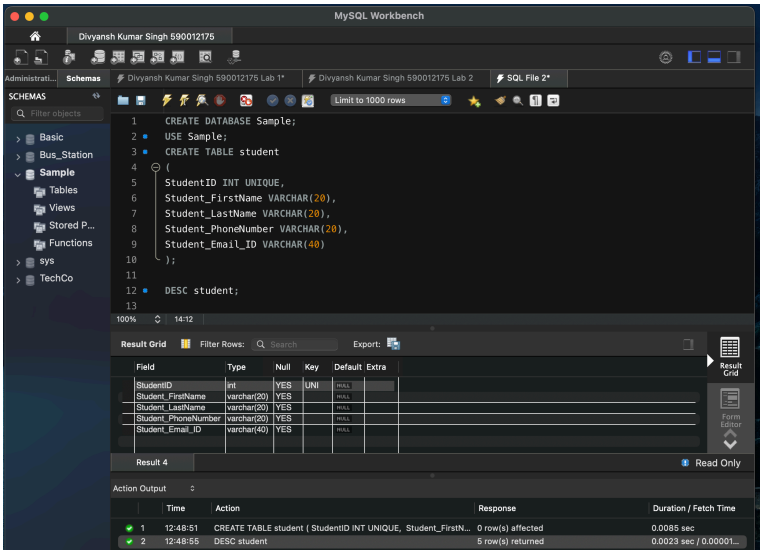
Create a student table and apply a UNIQUE constraint on one of the table's column while creating a table.

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

```
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. mysql> **DESC** student;



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));

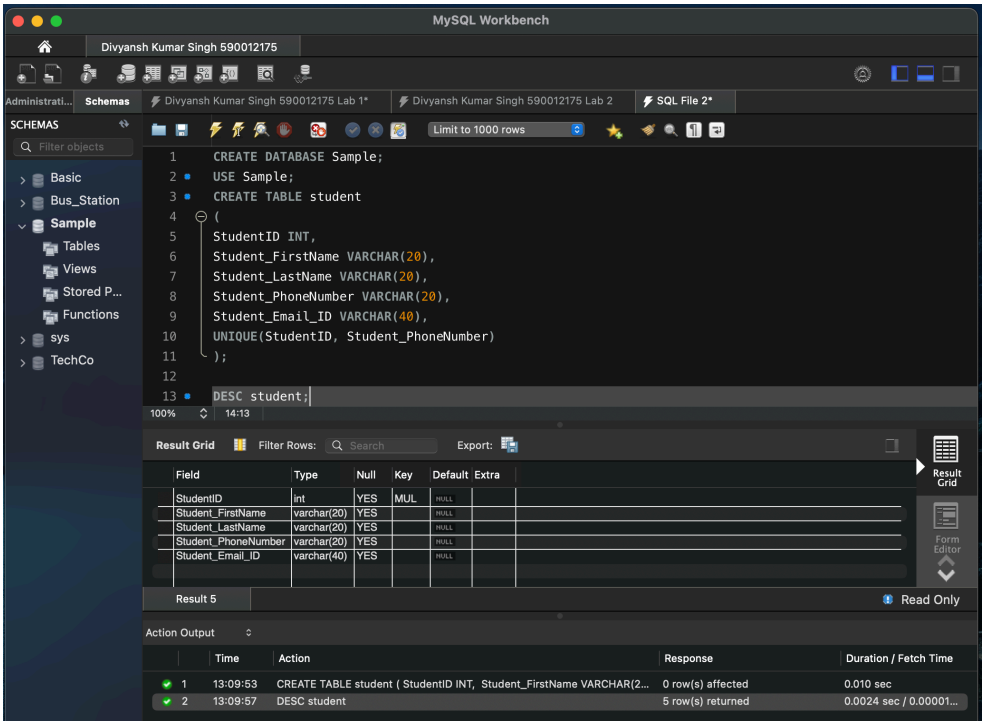
Example:
Create a student table and apply a UNIQUE constraint on more than one table's column while creating a table.

1. mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), St udent_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ ID **VARCHAR**(40), **UNIQUE**(StudentID, Student_PhoneNumber));

```
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)
);
```

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:

2. mysql> **DESC** student;



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:

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

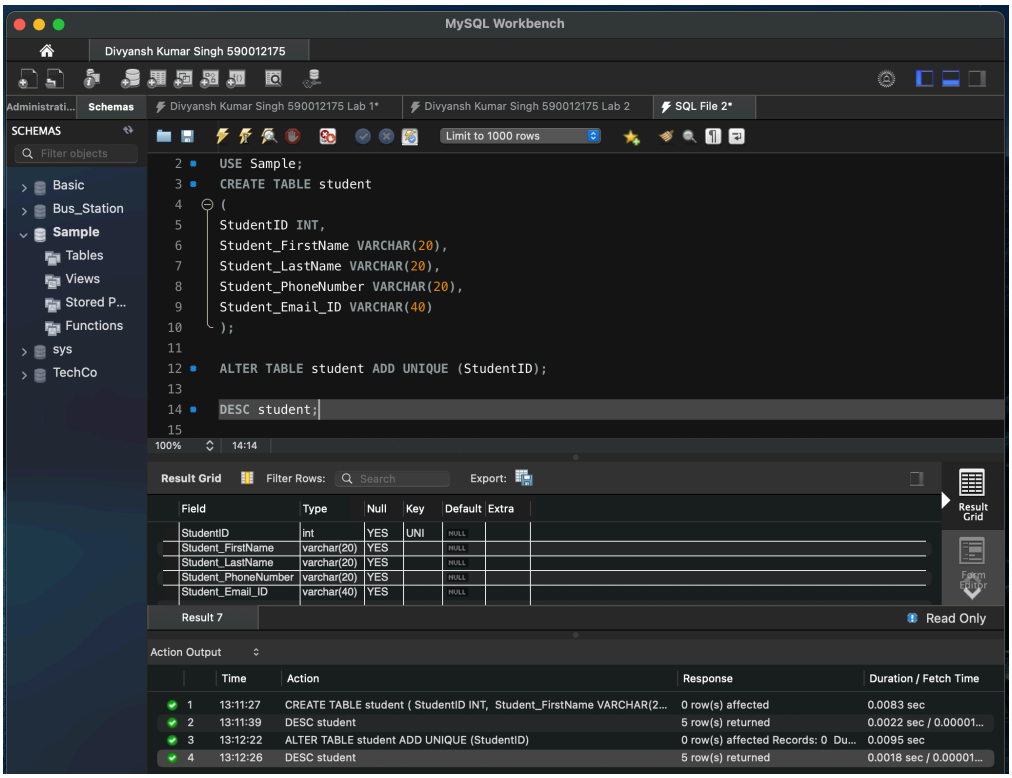
```
CREATE TABLE student
(
StudentID INT,
Student_FirstName VARCHAR(20),
Student_LastName VARCHAR(20),
Student_PhoneNumber VARCHAR(20),
Student_Email_ID VARCHAR(40)
);

ALTER TABLE student ADD UNIQUE (StudentID);

DESC student;
```

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:

3. mysql> **DESC** student;



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:

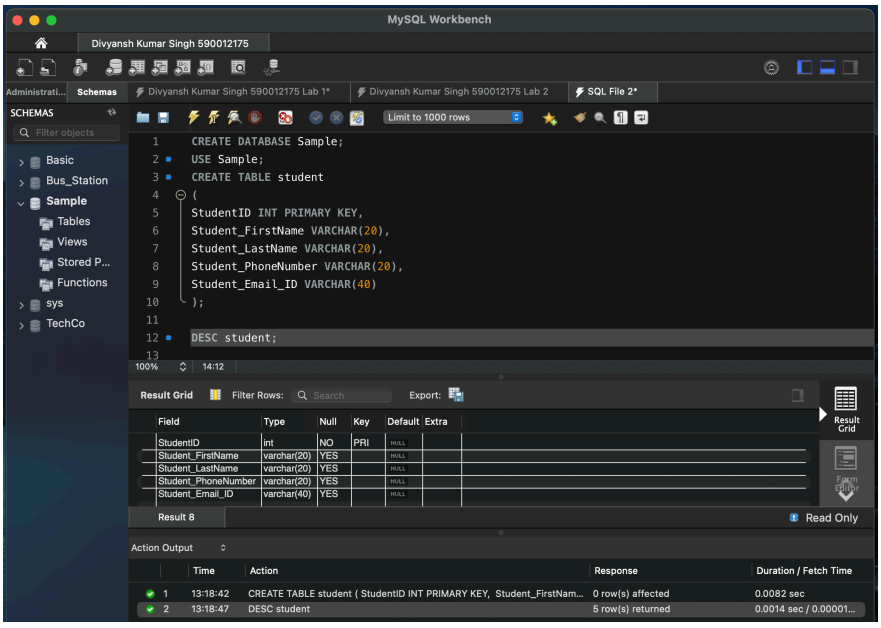
Create a student table and apply the PRIMARY KEY constraint while creating a table.

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));

```
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:

2. mysql> **DESC** student;



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);

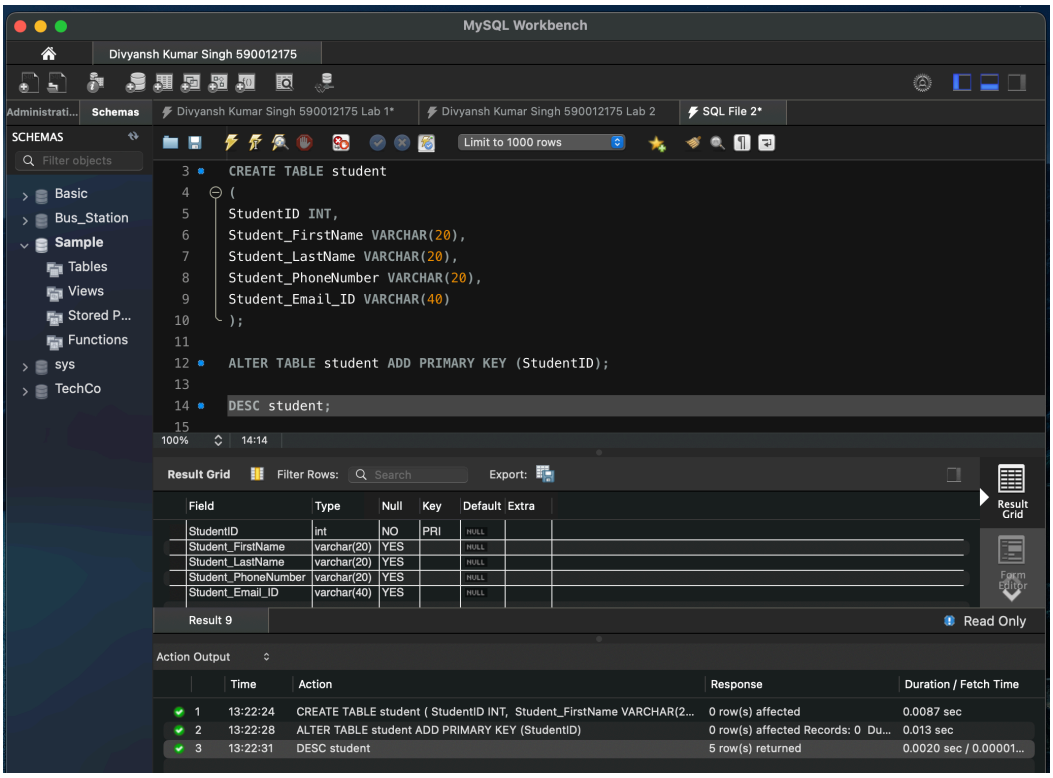
```
CREATE TABLE student
(
StudentID INT,
Student_FirstName VARCHAR(20),
Student_LastName VARCHAR(20),
Student_PhoneNumber VARCHAR(20),
Student_Email_ID VARCHAR(40)
);

ALTER TABLE student ADD PRIMARY KEY (StudentID);

DESC student;
```

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

2. mysql> **DESC** student;



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**, ColumnN ameN Datatype(**SIZE**), **FOREIGN KEY**(ColumnName) **REFERENCES** PARENT_TAB LE_NAME(Primary_Key_ColumnName));

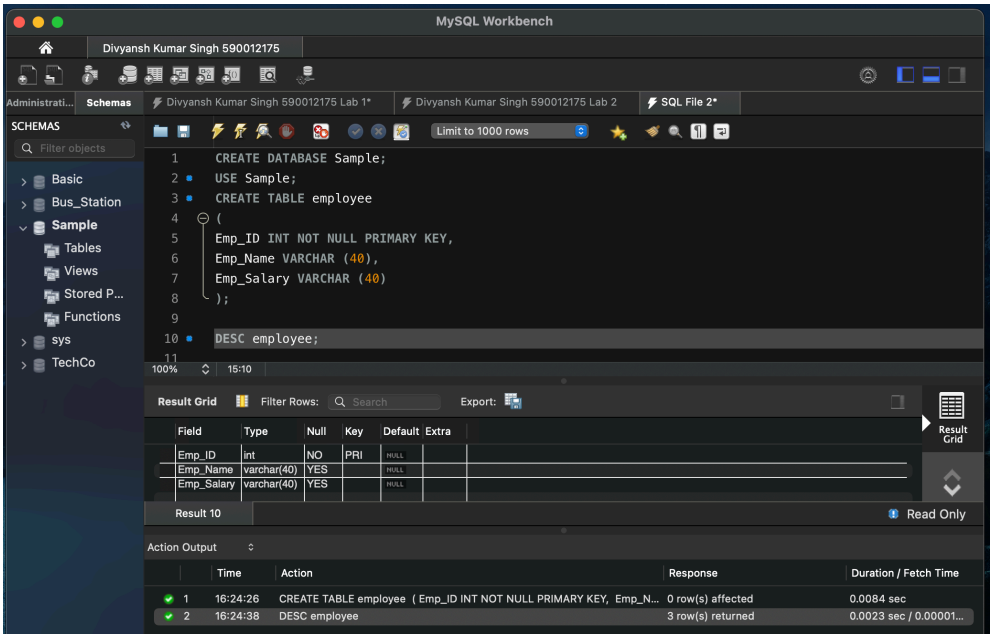
Example:
Create an employee table and apply the FOREIGN KEY constraint while creating a table.
To create a foreign key on any table, first, we need to create a primary key on a table.

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

```
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:

2. mysql> **DESC** employee;



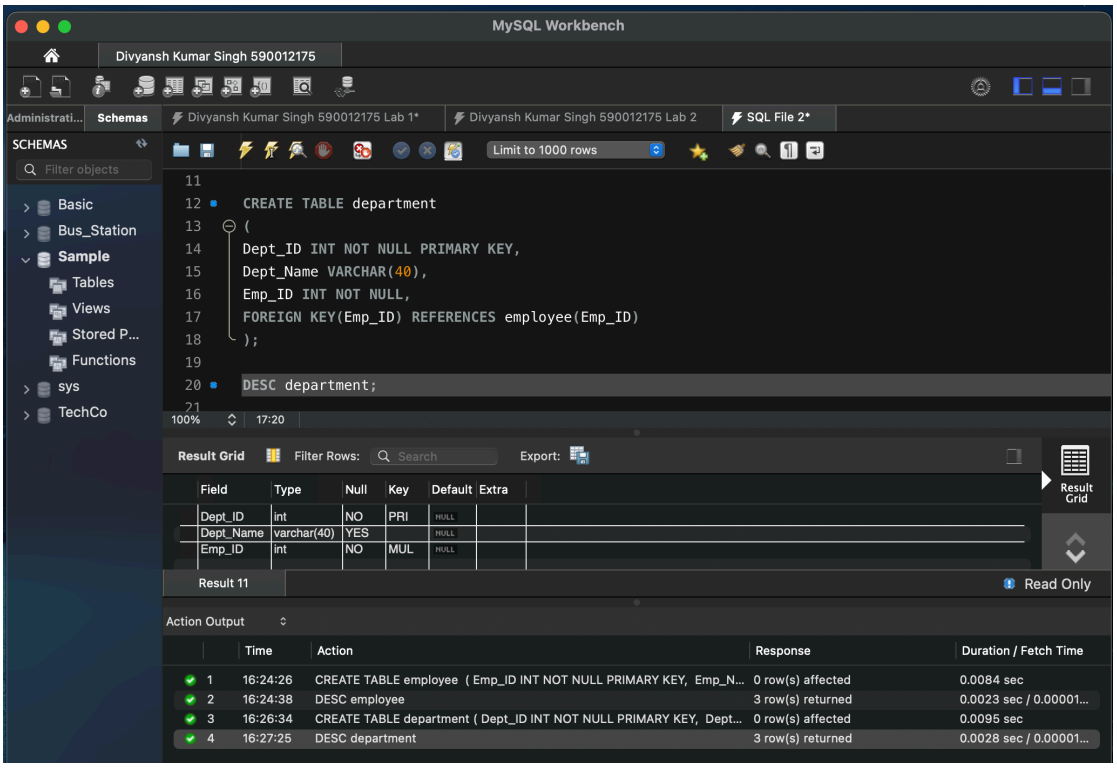
Now, we will write a query to apply a foreign key on the department table referring to the primary key of the employee table, i.e., Emp_ID.

3. 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));

```
12    CREATE TABLE department
13    (
14    Dept_ID INT NOT NULL PRIMARY KEY,
15    Dept_Name VARCHAR(40),
16    Emp_ID INT NOT NULL,
17    FOREIGN KEY(Emp_ID) REFERENCES employee(Emp_ID)
18    );
19
```

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

4. mysql> **DESC** department;



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:

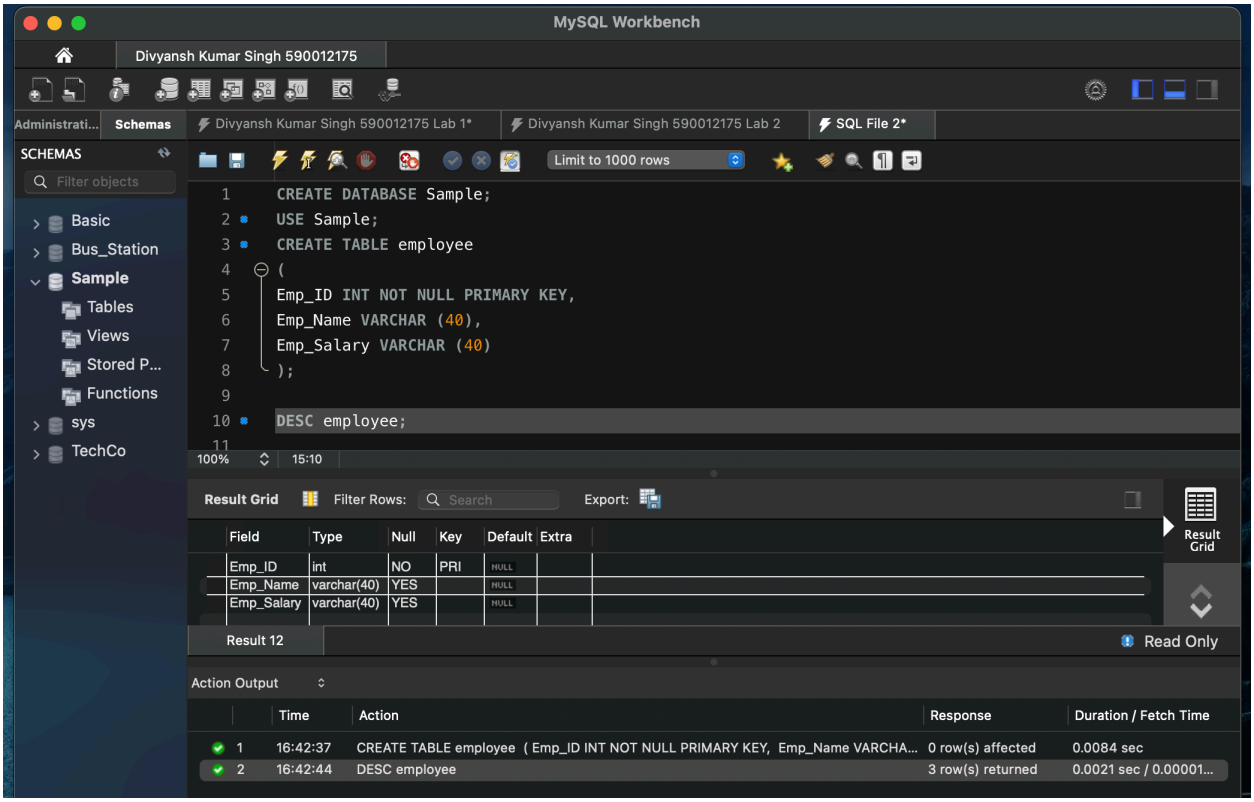
Create an employee table and apply the FOREIGN KEY constraint with a constraint name while creating a table.

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));

```
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:

2. mysql> **DESC** employee;



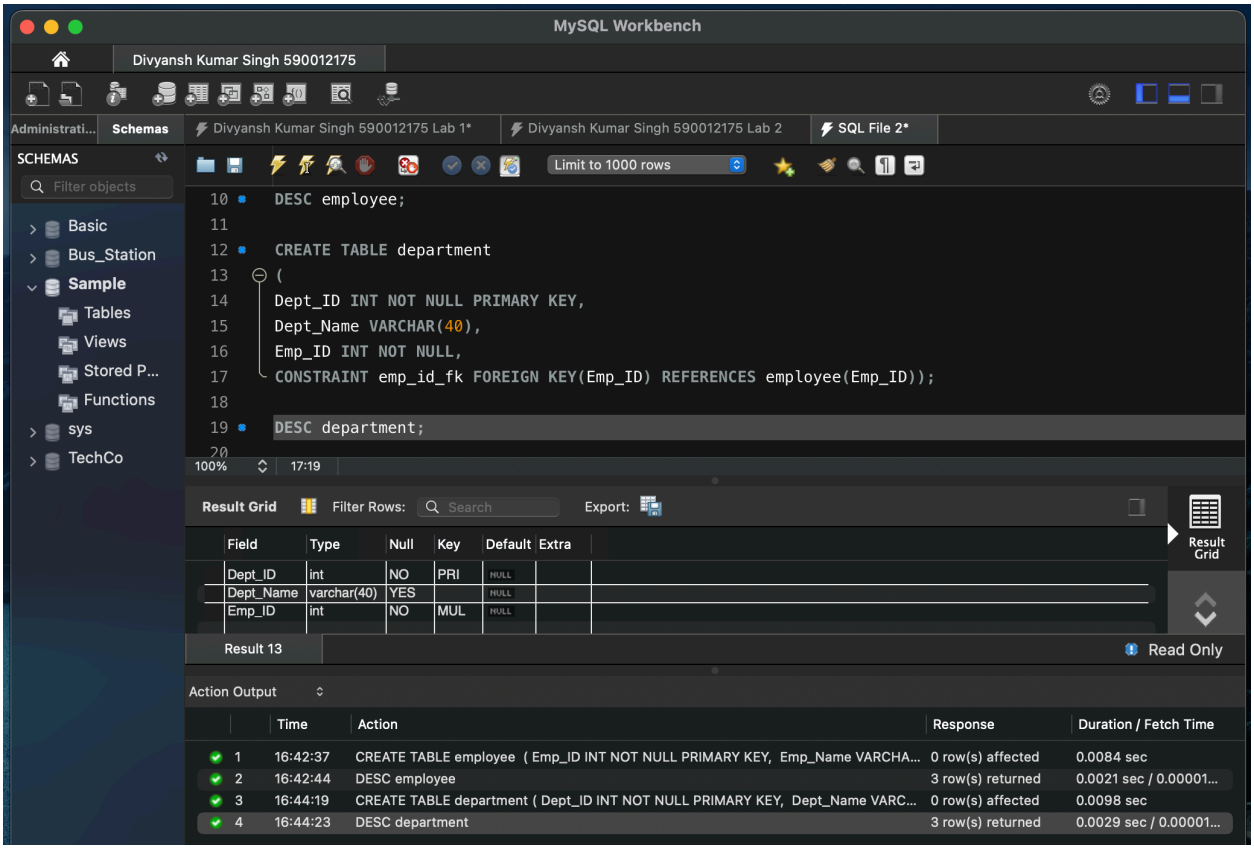
Now, we will write a query to apply a foreign key with a constraint name on the department table referring to the primary key of the employee table, i.e., Emp_ID.

3. 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));

```
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));
```

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

4. mysql> **DESC** department;

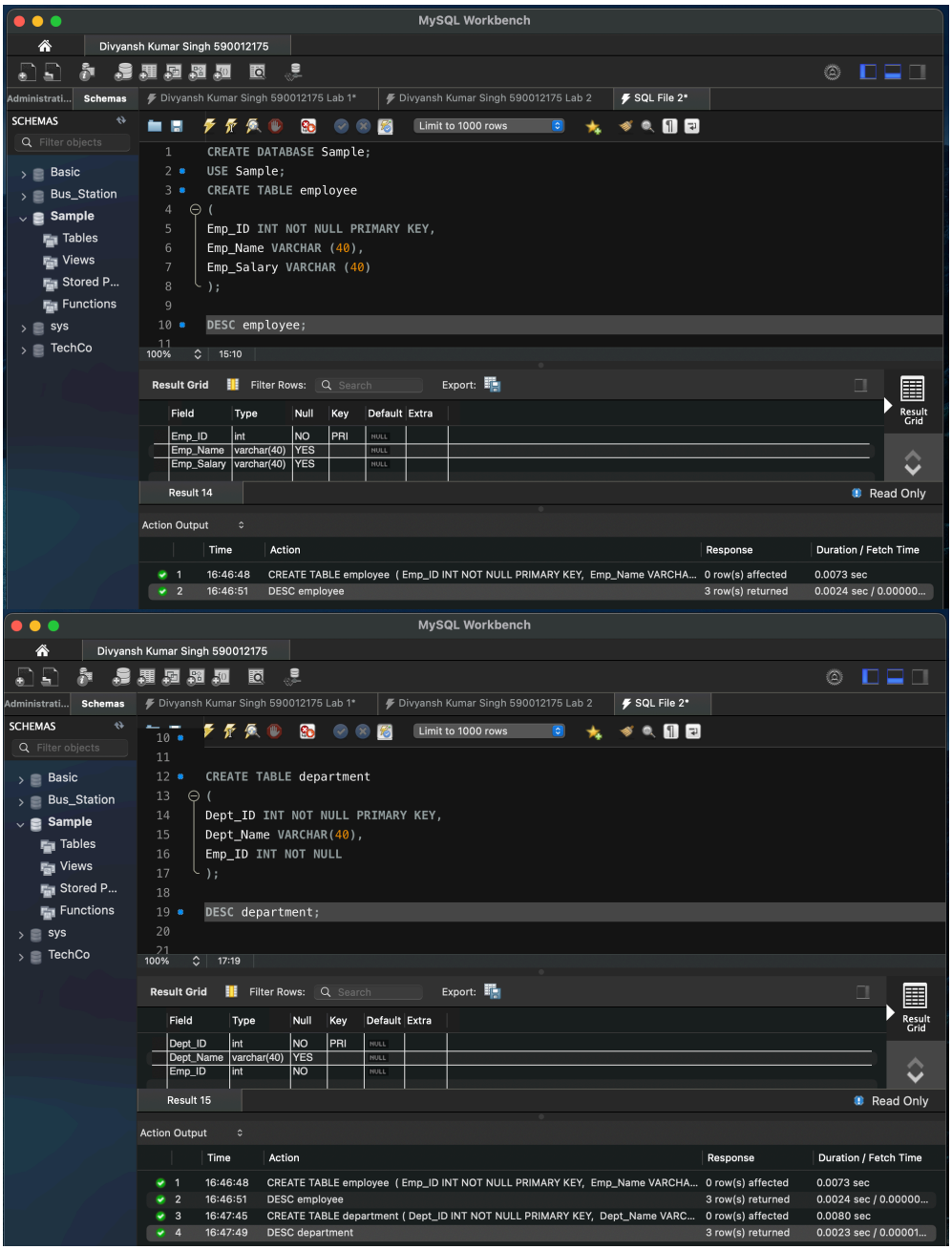


Syntax to apply the foreign key constraint on an existing table's column:

1. **ALTER TABLE** Parent_TableName **ADD FOREIGN KEY** (ColumnName) **REFERENCE S** 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;

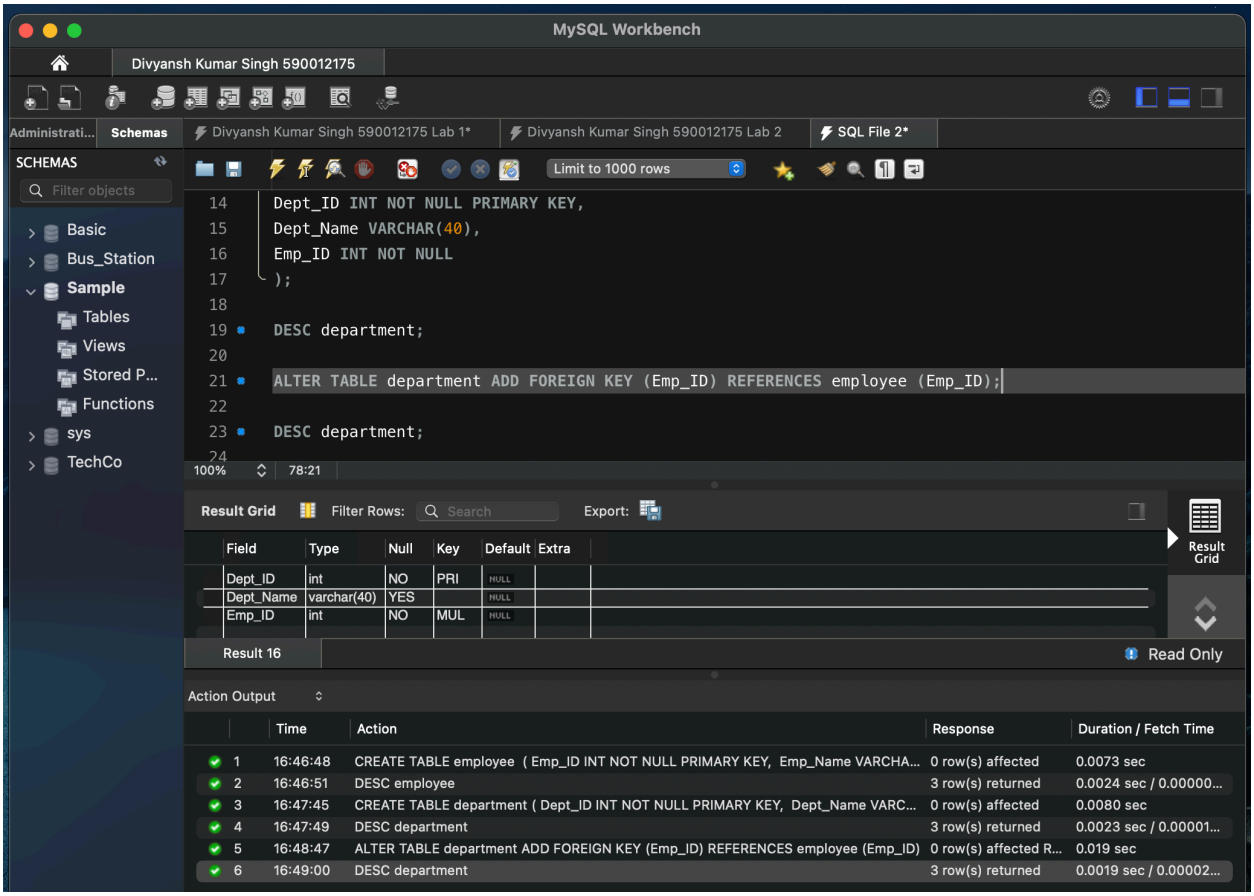


2. mysql> **ALTER TABLE** department **ADD FOREIGN KEY** (Emp_ID) **REFERENCES** em
ployee (Emp_ID);

```
21 •  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:

3. mysql> **DESC** department;



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 Conditio n), ColumnName2 datatype,…., ColumnNameN datatype);

Example:

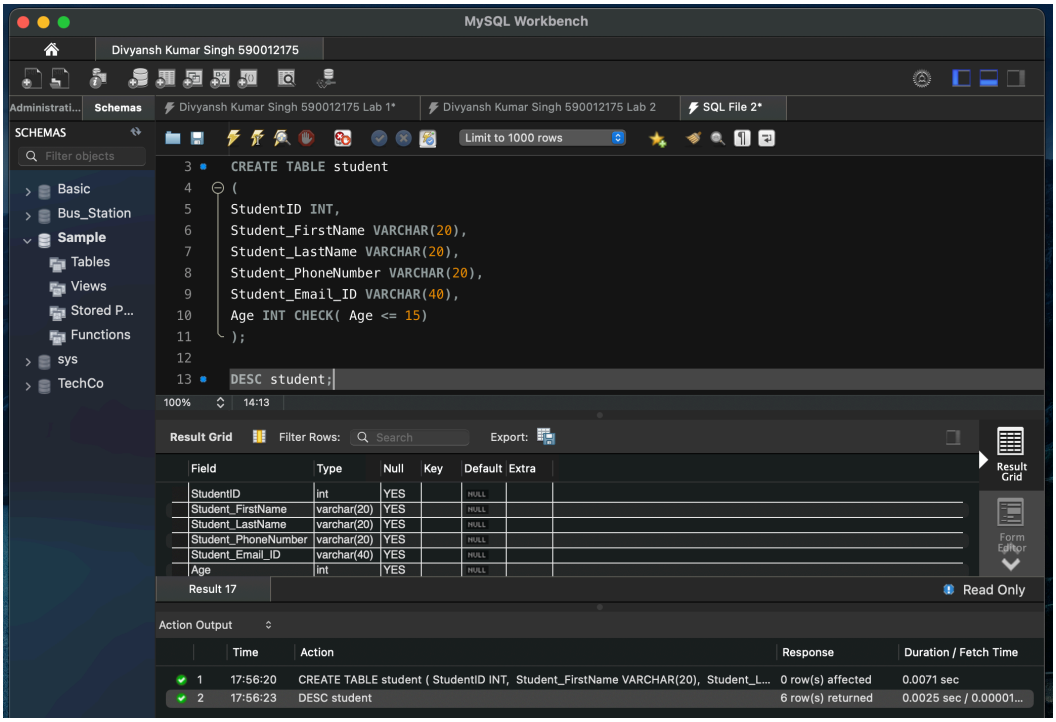
Create a student table and apply CHECK constraint to check for the age less than or equal to 15 while creating a table.

1. mysql> **CREATE TABLE** student(StudentID **INT**, Student_FirstName **VARCHAR**(20), St udent_LastName **VARCHAR**(20), Student_PhoneNumber **VARCHAR**(20), Student_Email_ ID **VARCHAR**(40), Age **INT CHECK**(Age <= 15));

```
3   CREATE TABLE student
4   (
5   StudentID INT,
6   Student_FirstName VARCHAR(20),
7   Student_LastName VARCHAR(20),
8   Student_PhoneNumber VARCHAR(20),
9   Student_Email_ID VARCHAR(40),
10  Age INT CHECK( Age <= 15)
11  );
```

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

2. mysql> **DESC** student;



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:

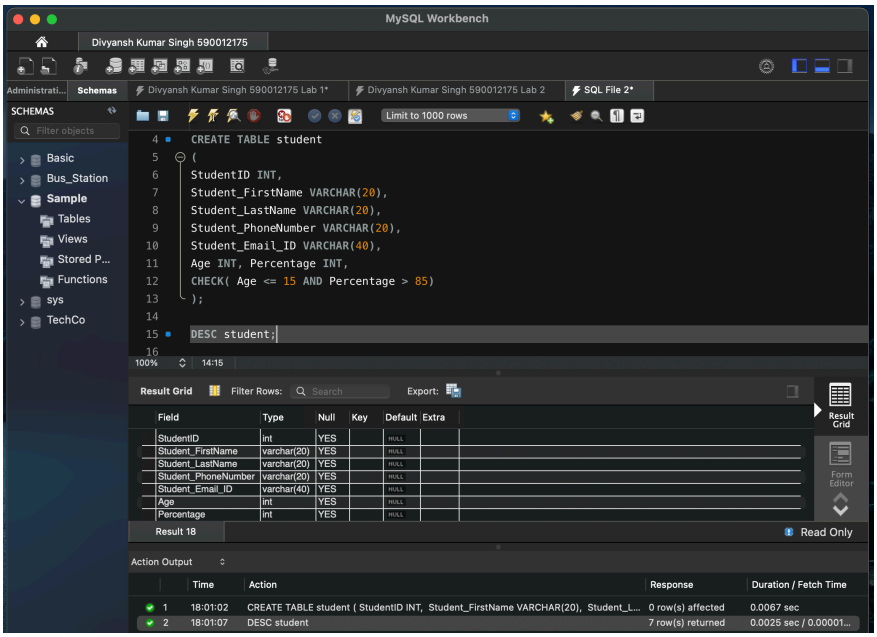
Create a student table and apply CHECK constraint to check for the age less than or equal to 15 and a percentage greater than 85 while creating a table.

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

```
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:

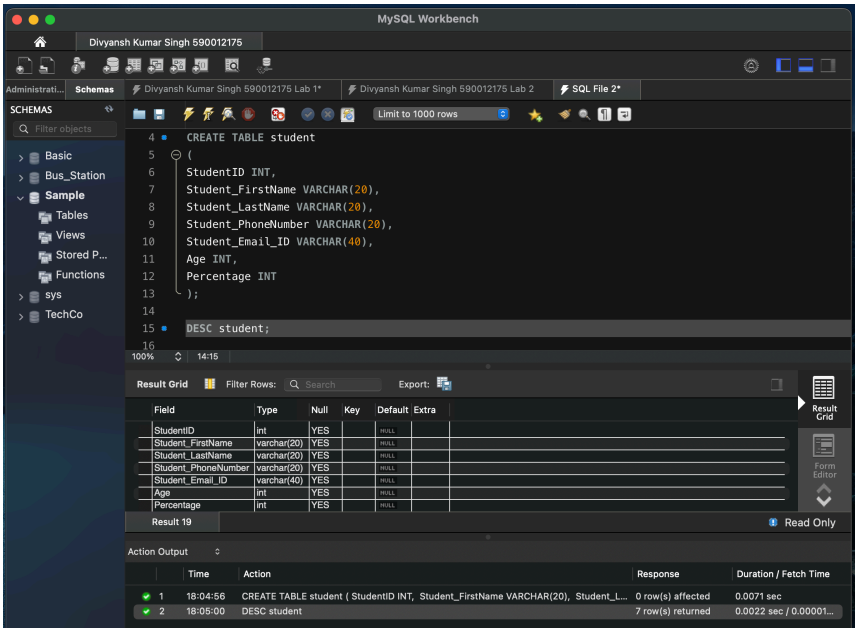
2. mysql> **DESC** student;



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:

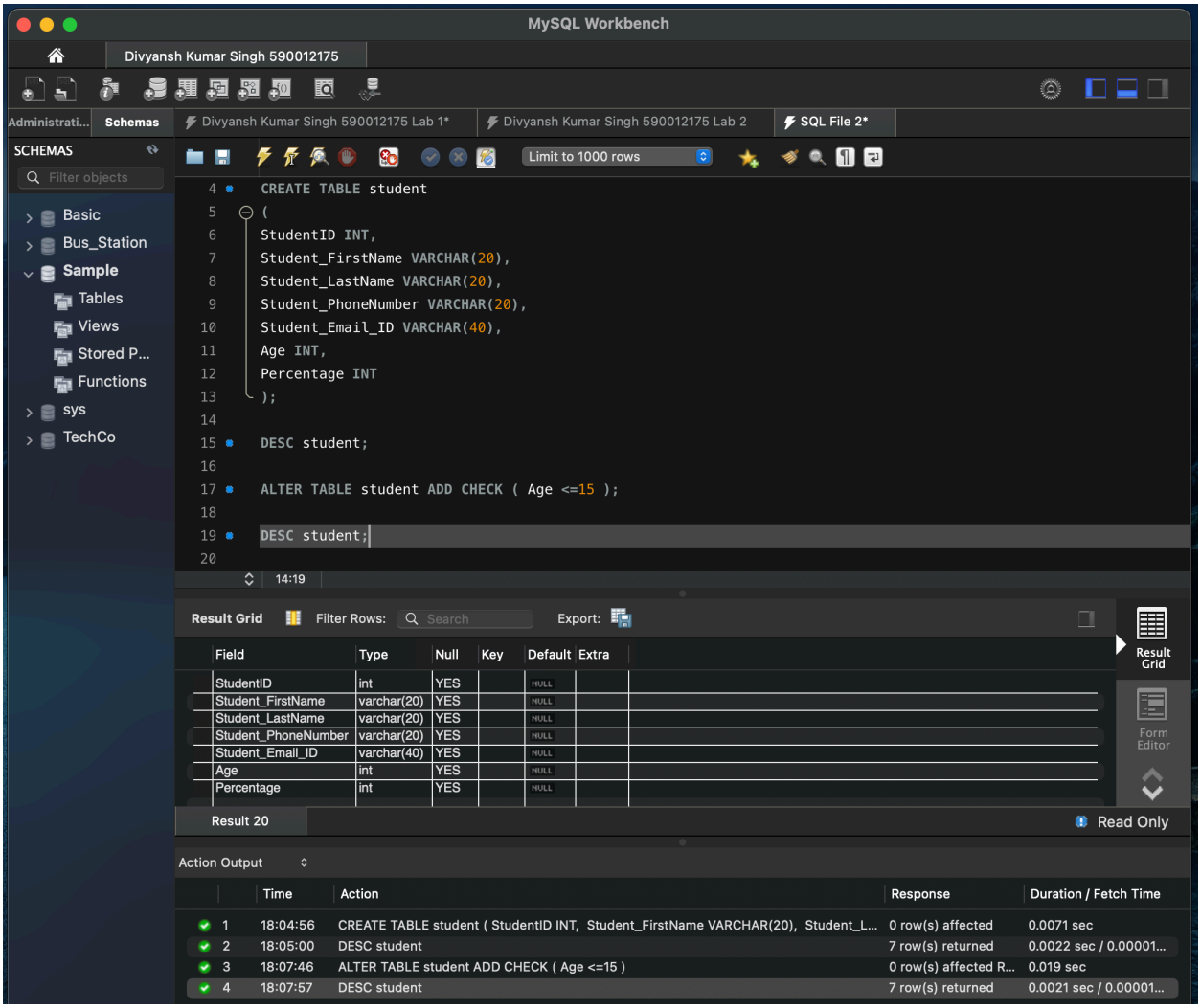


1. mysql> **ALTER TABLE** student **ADD CHECK** (Age <=15);

```
17 •    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:

2. mysql> **DESC** student;



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:

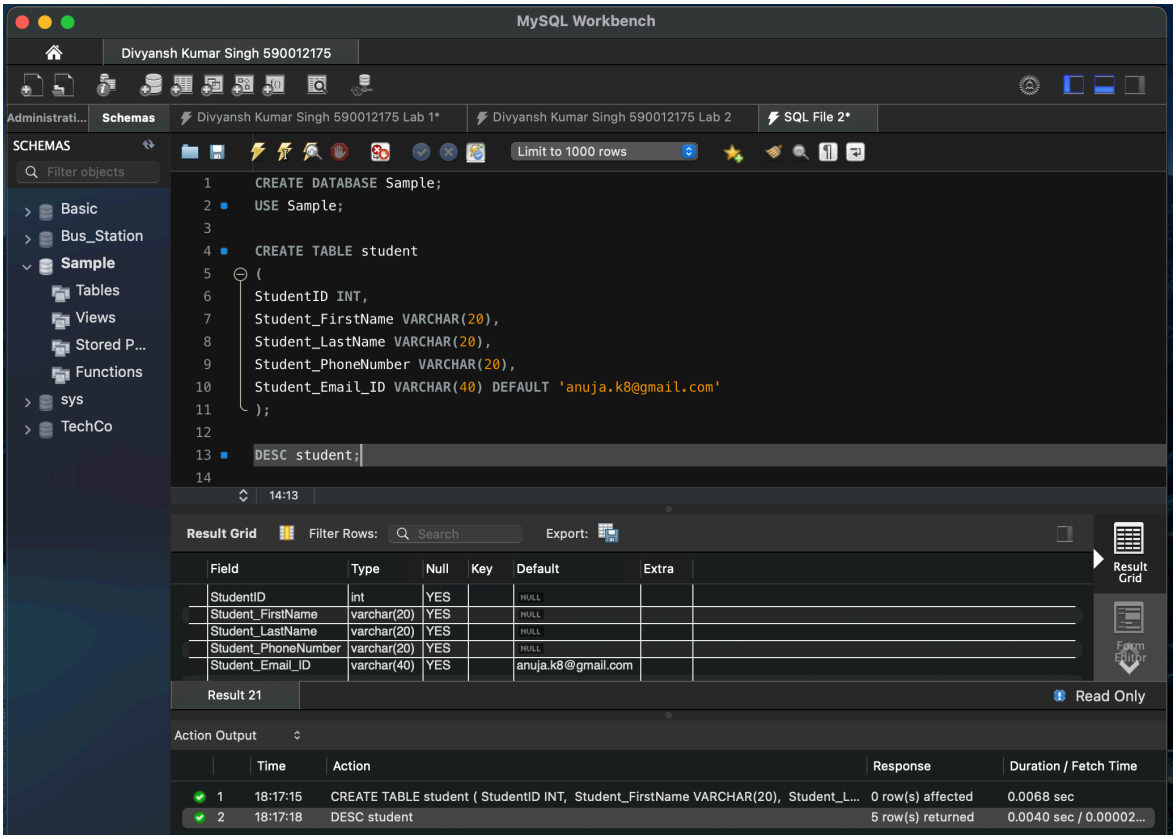
Create a student table and apply the default constraint while creating a table.

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");

```
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'
);
```

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

2. mysql> **DESC** student;



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:

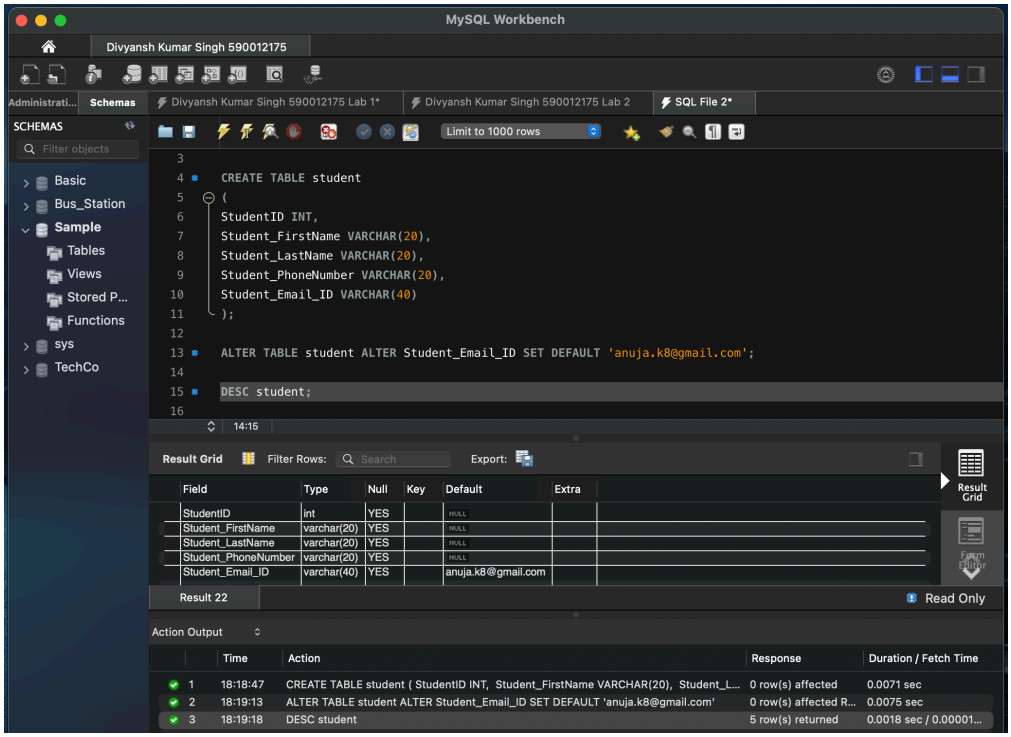
1. mysql> **ALTER TABLE** student **ALTER** Student_Email_ID **SET DEFAULT** "anuja.k8@gmail.com";

```
CREATE TABLE student
(
StudentID INT,
Student_FirstName VARCHAR(20),
Student_LastName VARCHAR(20),
Student_PhoneNumber VARCHAR(20),
Student_Email_ID VARCHAR(40)
);

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:

1. mysql> **DESC** student;



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:

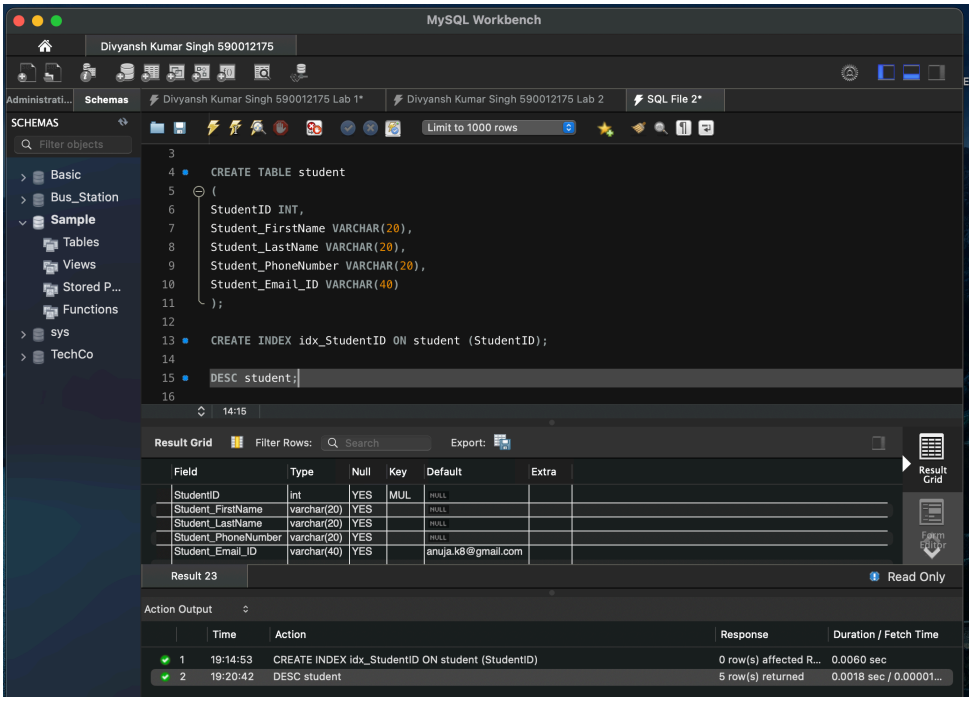
Create an index on the student table and apply the default constraint while creating a table.

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

```
4   CREATE TABLE student
5   (
6   StudentID INT,
7   Student_FirstName VARCHAR(20),
8   Student_LastName VARCHAR(20),
9   Student_PhoneNumber VARCHAR(20),
10  Student_Email_ID VARCHAR(40)
11  );
12
13  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:

2. mysql> DESC student;

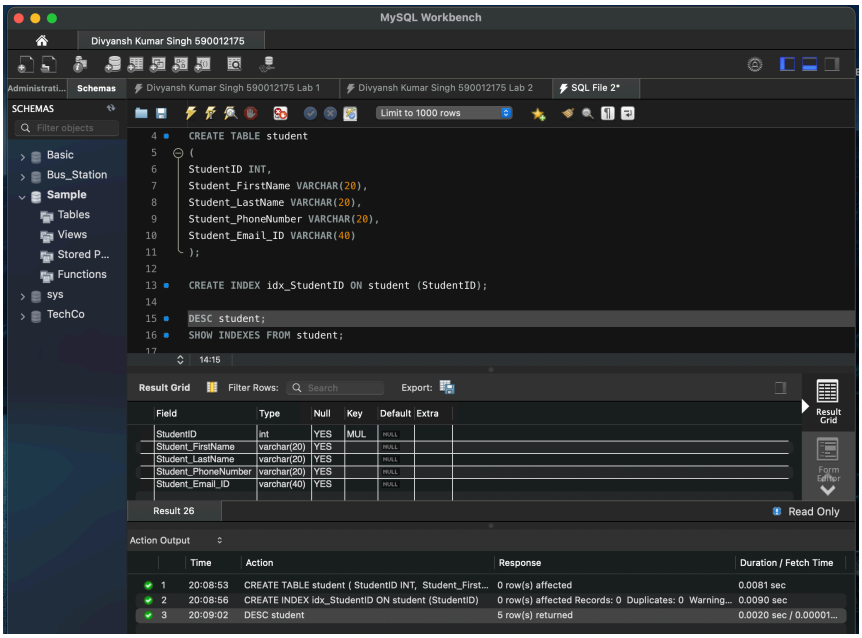


Syntax to create an index on multiple columns:

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

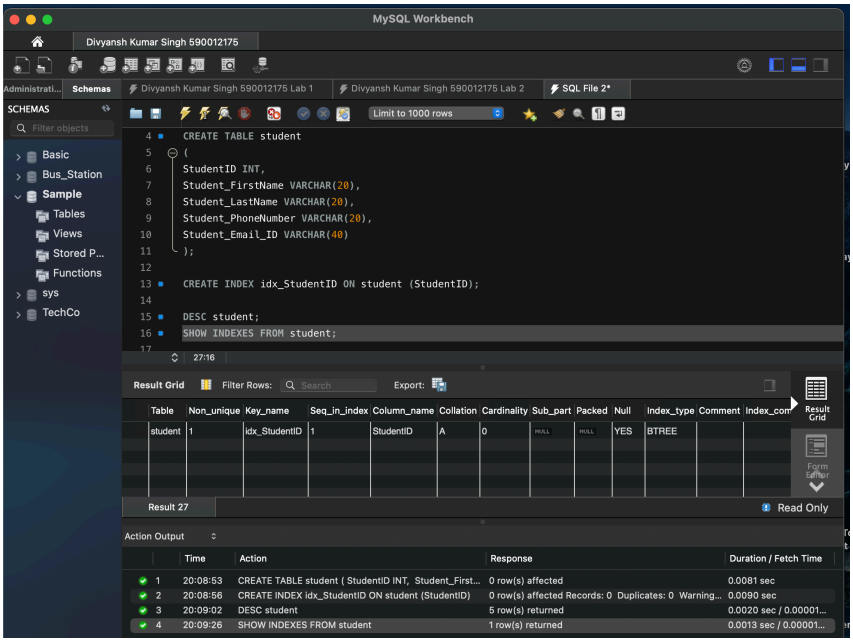
Example:

1. 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:

2. mysql> **DESC** student;

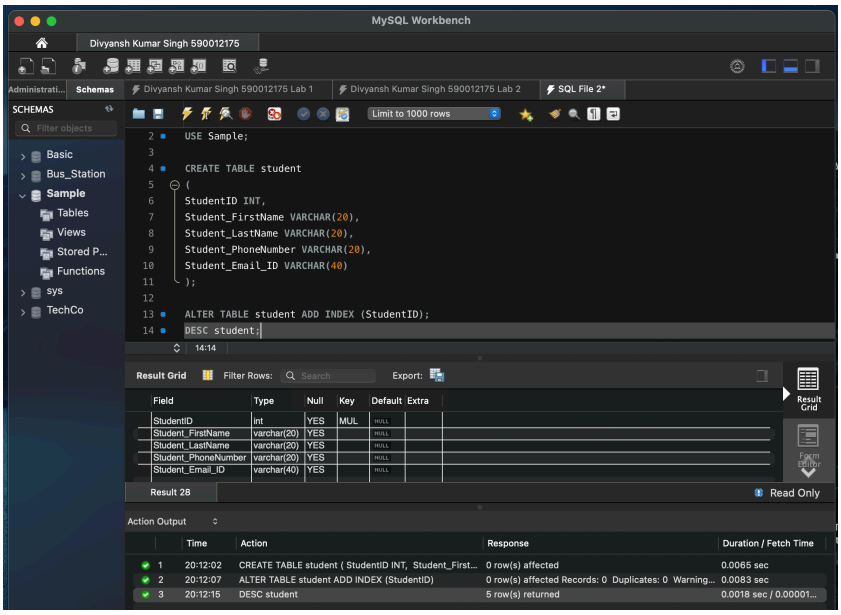


Syntax to create an index on an existing table:

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

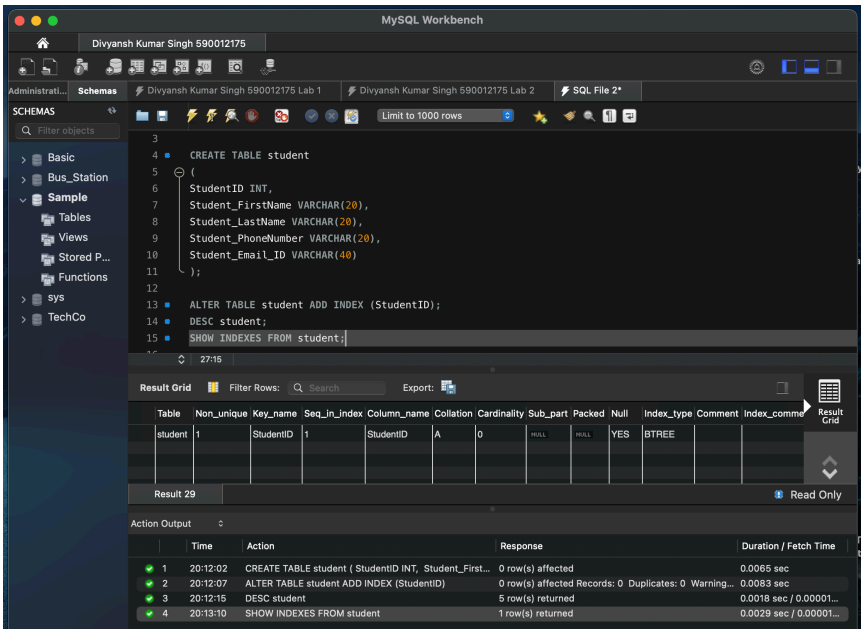
Consider we have an existing table student. Later, we decided to apply the DEFAULT constraint on the student table's column. Then we will execute the following query:

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



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

2. mysql> **DESC** student;



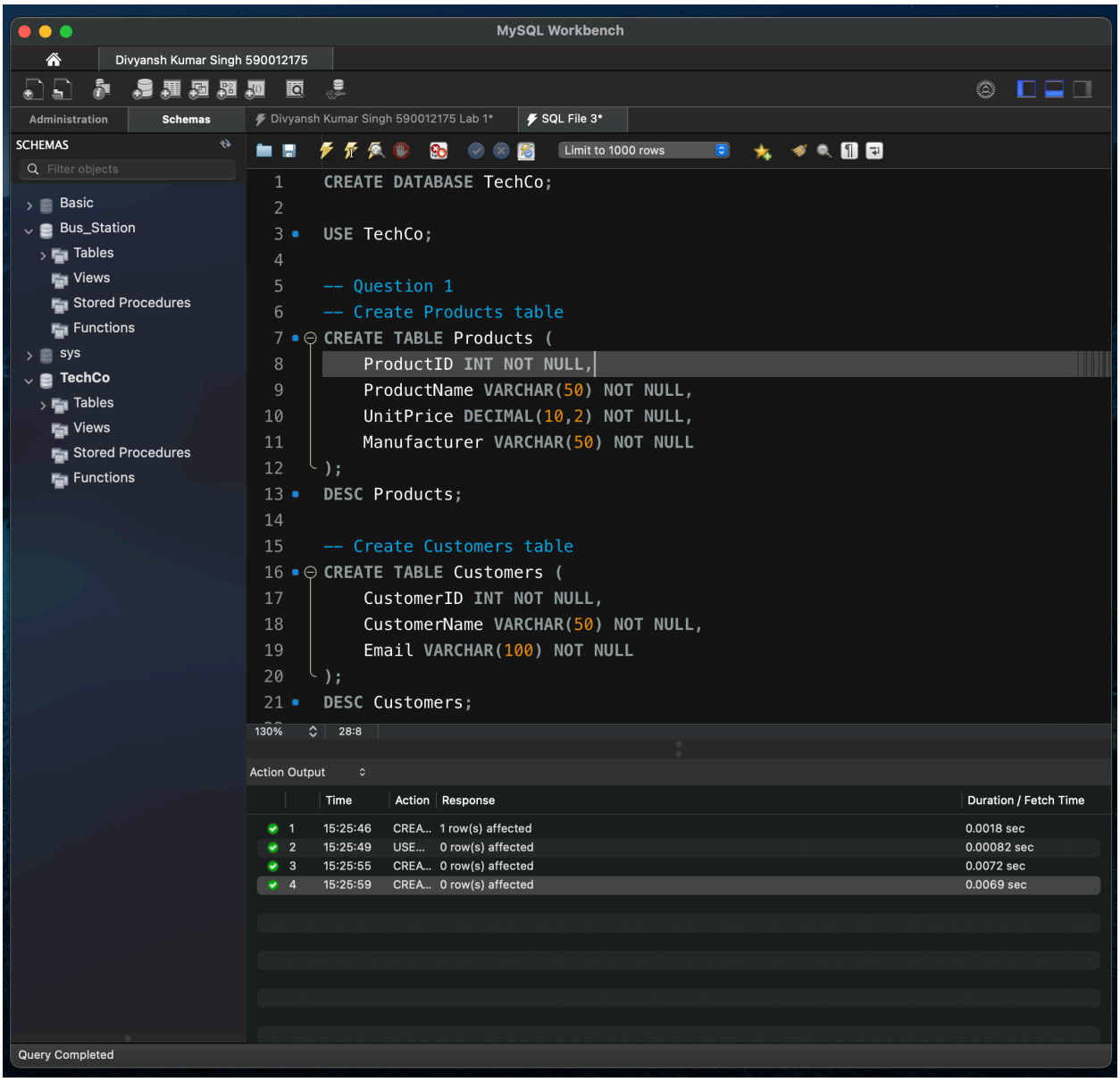
LAB Performance Questions

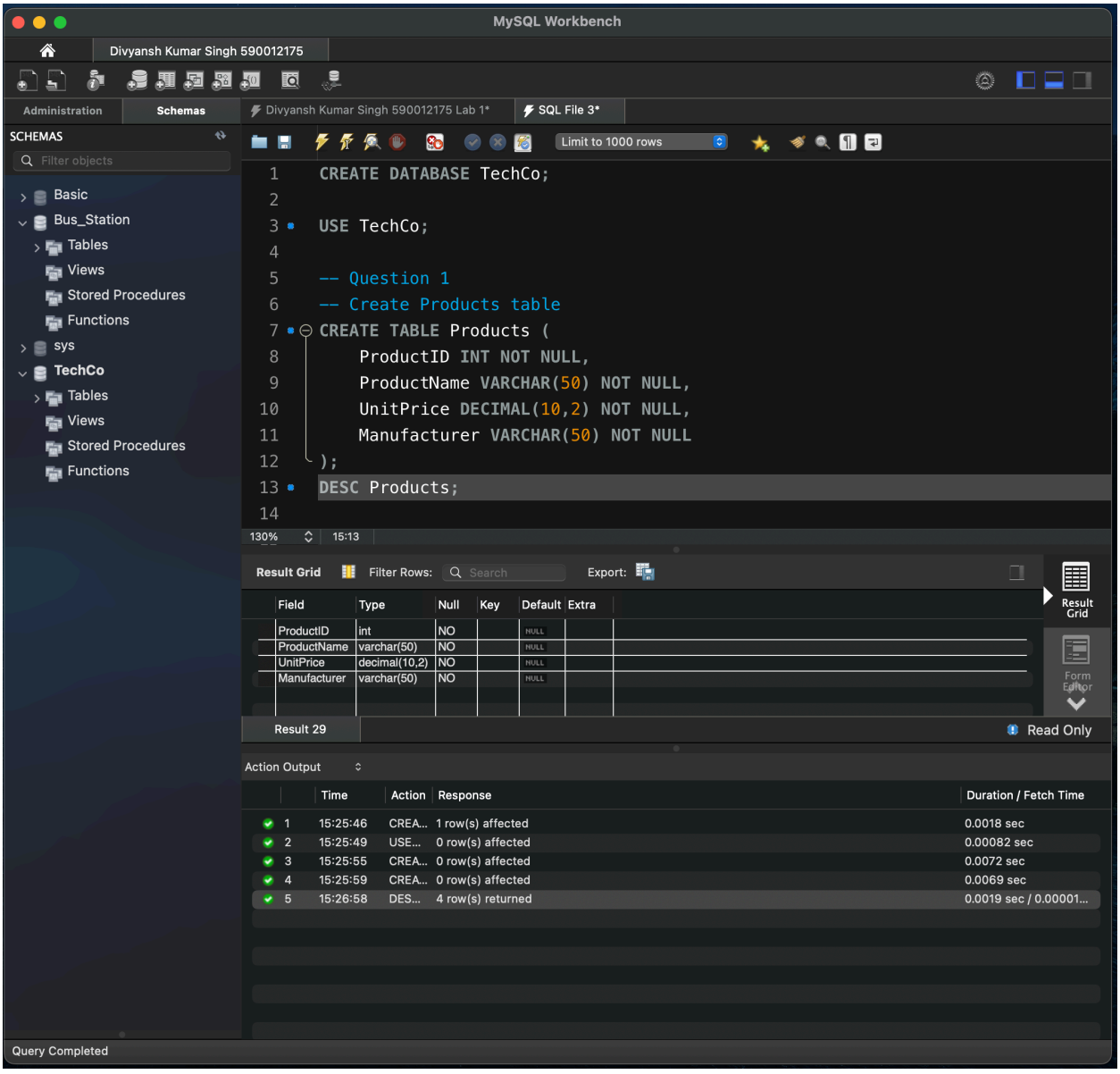
You are a junior database administrator at a company called TechCo. The company manages information about its products, customers, and orders through a relational database. Your task is toperform various SQL queries to retrieve and manipulate data within this database.

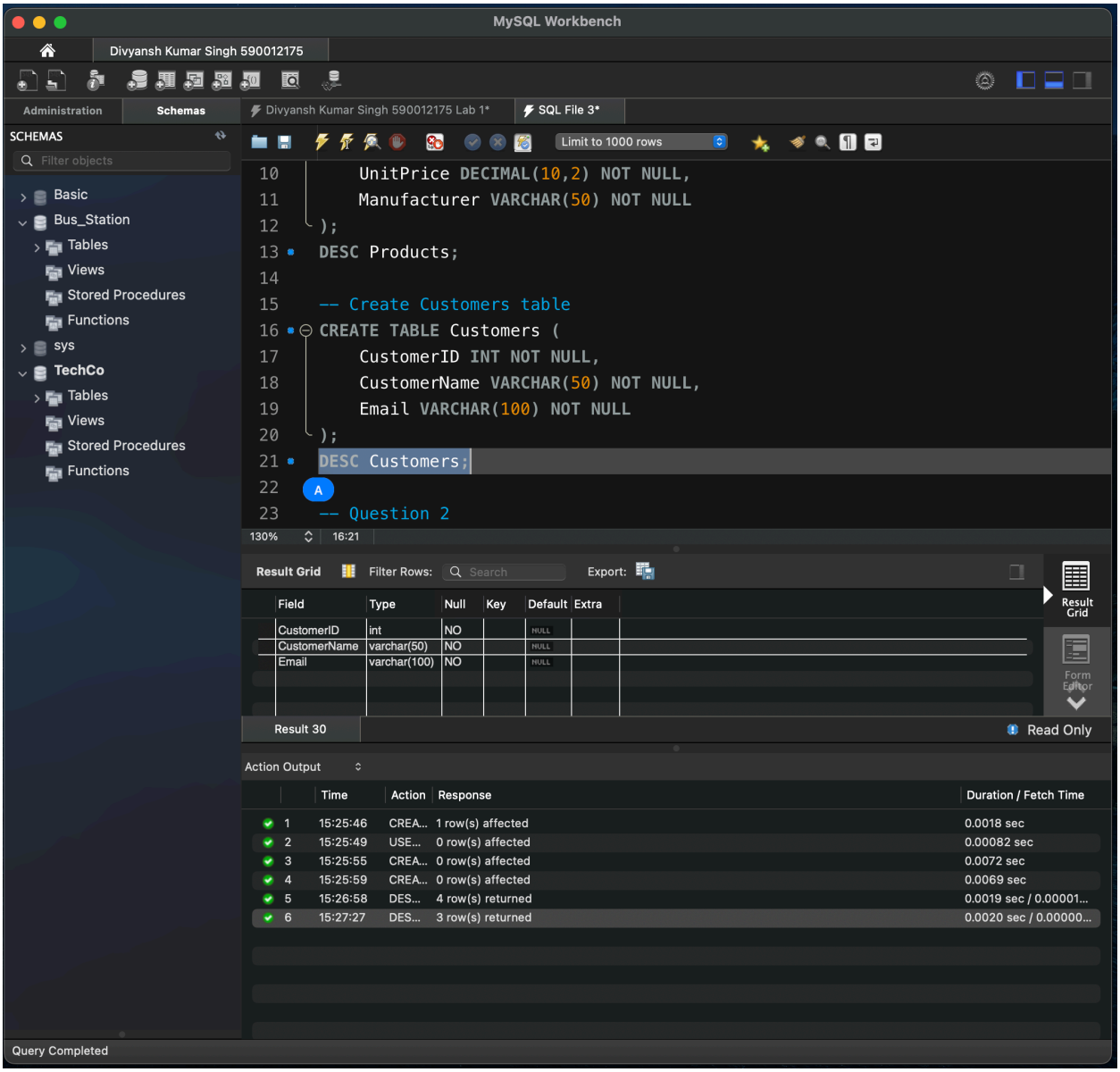
1. Write an SQL script that includes DDL statements to create the following tables:

- `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.

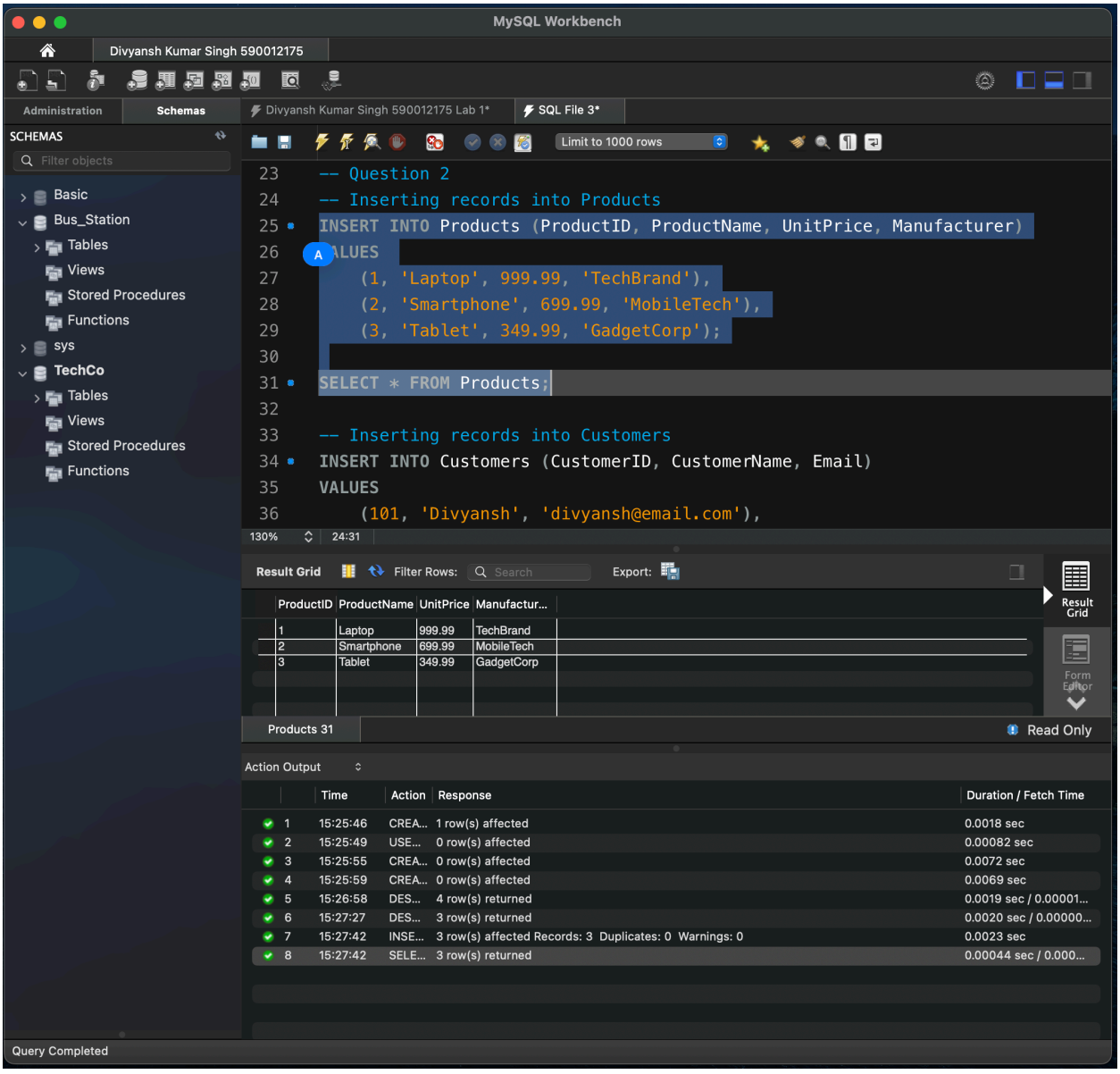


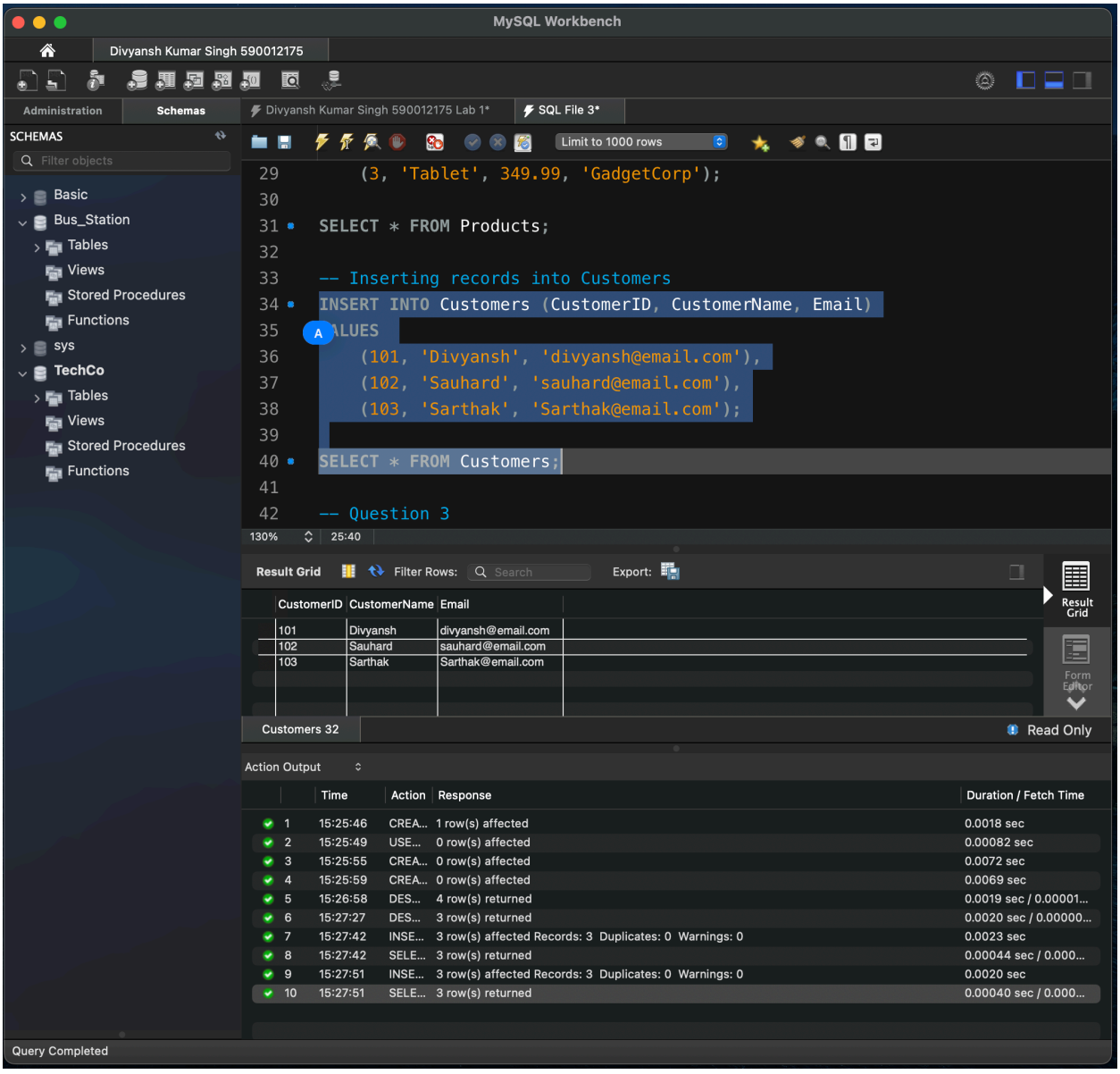

MySQL Workbench
Divyansh Kumar Singh 590012175
Administration
Schemas
Divyansh Kumar Singh 590012175 Lab 1*
SQL File 3*
SCHEMAS
Filter objects
Basic
Bus_Station
Tables
Views
Stored Procedures
Functions
sys
TechCo
Tables
Views
Stored Procedures
Functions
Limit to 1000 rows
1 CREATE DATABASE TechCo;
2
3 USE TechCo;
4
5 -- Question 1
6 -- Create Products table
7 CREATE TABLE Products (
8 ProductID INT NOT NULL,
9 ProductName VARCHAR(50) NOT NULL,
10 UnitPrice DECIMAL(10,2) NOT NULL,
11 Manufacturer VARCHAR(50) NOT NULL
12);
13 DESC Products;
14
130%
15:13
Result Grid
Filter Rows:
Search
Export:
Field Type Null Key Default Extra
ProductID int NO NULL
ProductName varchar(50) NO NULL
UnitPrice decimal(10,2) NO NULL
Manufacturer varchar(50) NO NULL
Result Grid
Form Editor
Result 29
Read Only
Action Output
Time Action Response Duration / Fetch Time
1 15:25:46 CREA... 1 row(s) affected 0.0018 sec
2 15:25:49 USE... 0 row(s) affected 0.00082 sec
3 15:25:55 CREA... 0 row(s) affected 0.0072 sec
4 15:25:59 CREA... 0 row(s) affected 0.0069 sec
5 15:26:58 DES... 4 row(s) returned 0.0019 sec / 0.00001...
Query Completed



- **CREATE TABLE** → Creates a new table.
- **INTEGER** → Stores whole numbers.
- **VARCHAR(n)** → Stores variable-length text up to *n* characters.
- **DECIMAL** → Stores fixed-precision numbers (like prices).
- **PRIMARY KEY** → Uniquely identifies each row, cannot be NULL.

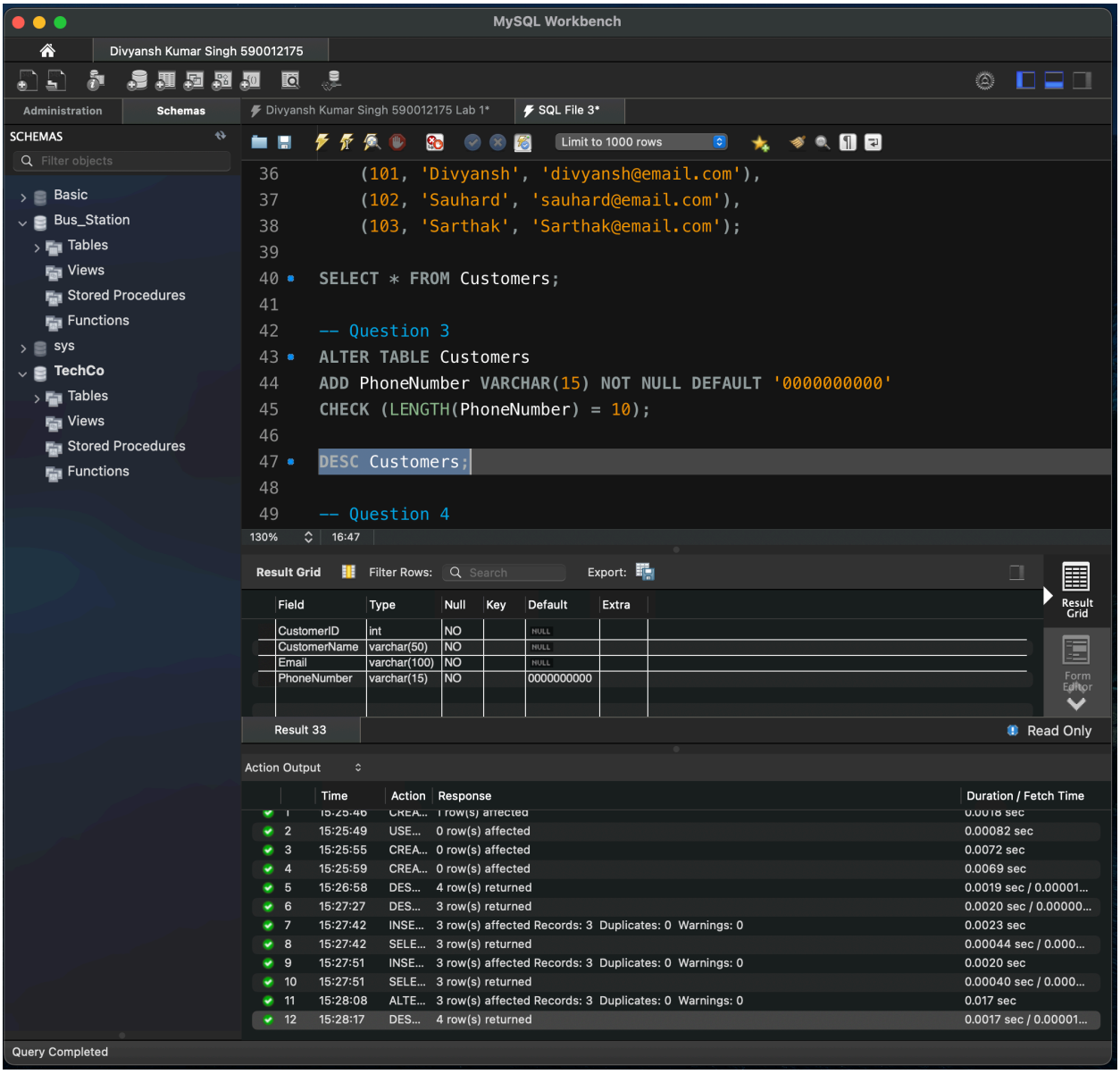
2. Insert at least three sample records into the `Products` table created in Question 1. Ensure each record has a unique product ID.
Insert at least three sample records into the `Customers` table created in Question 1. Each record should have a unique customer ID.





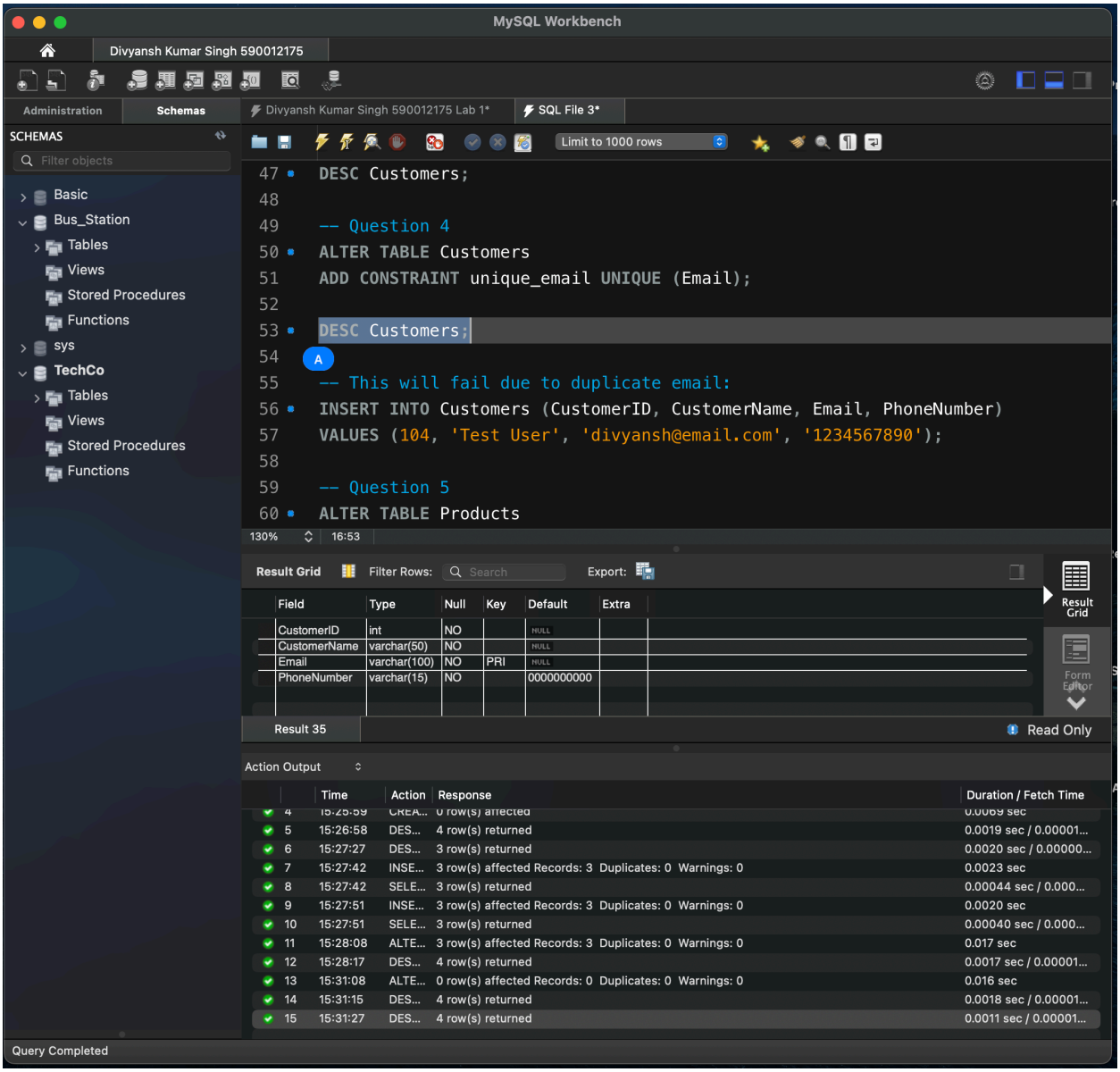
- **INSERT INTO** → Adds new rows to a table.
- **VALUES** → Specifies the data to insert into the columns.

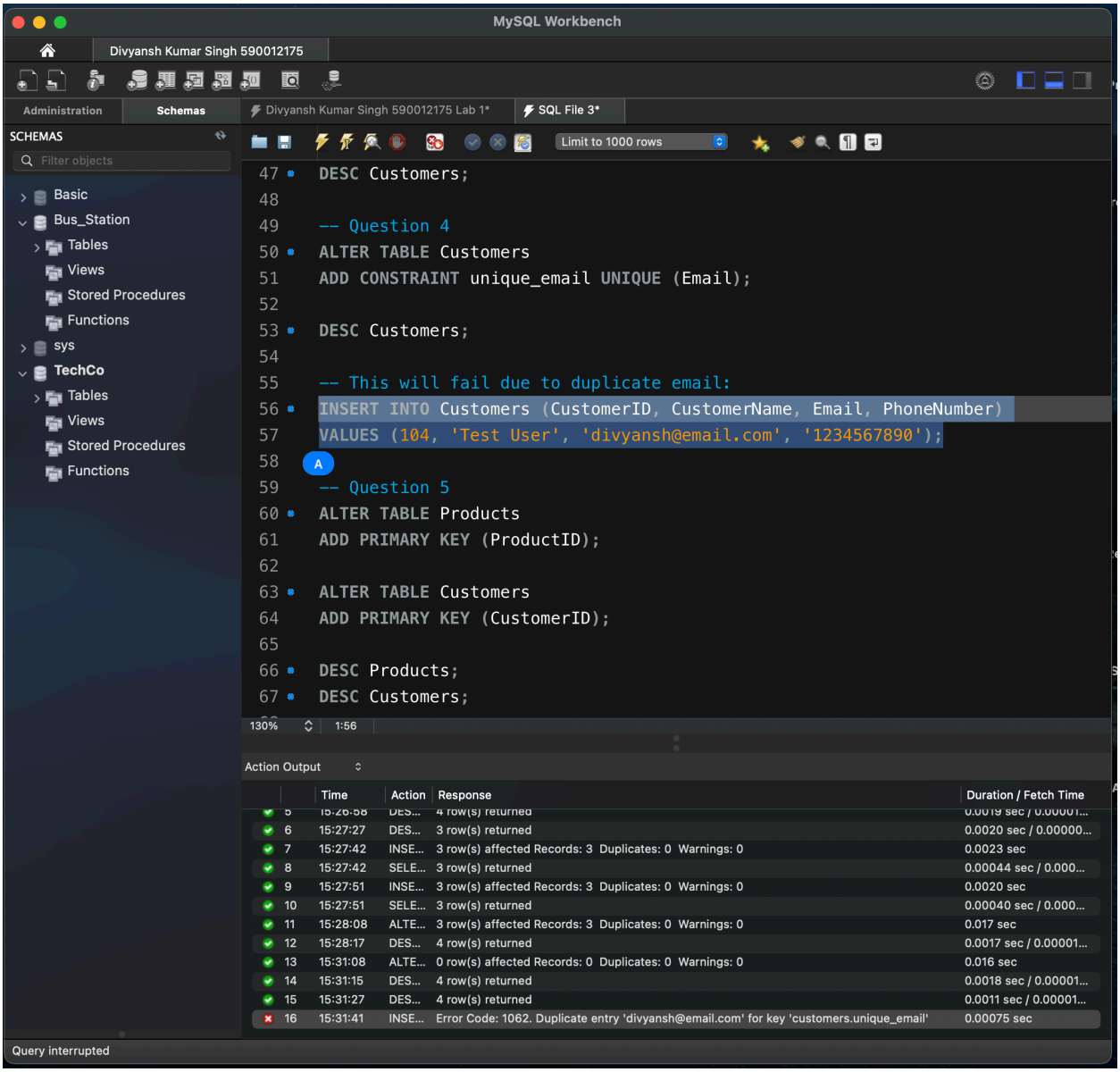
3. Add a column named `PhoneNumber` to the `Customers` table. Apply the NOT NULL constraint to ensure that this information is always provided.



- **ALTER TABLE** → Modifies an existing table.
- **ADD COLUMN** → Introduces a new column.
- **NOT NULL** → Ensures the column must always have a value.

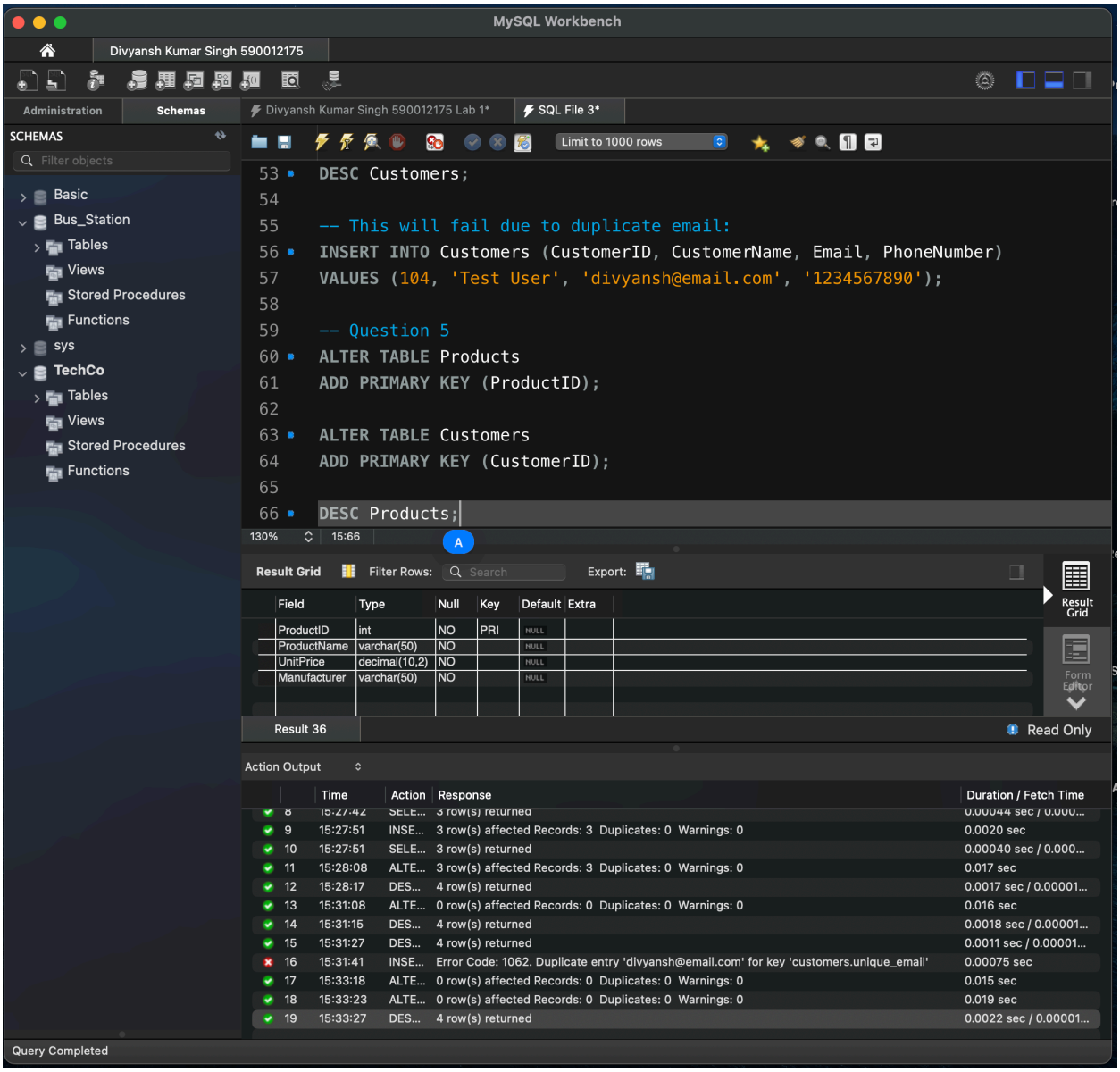
4. Implement a UNIQUE constraint on the `Email` column in the `Customers` table. Provide an example of how this constraint prevents data duplication.

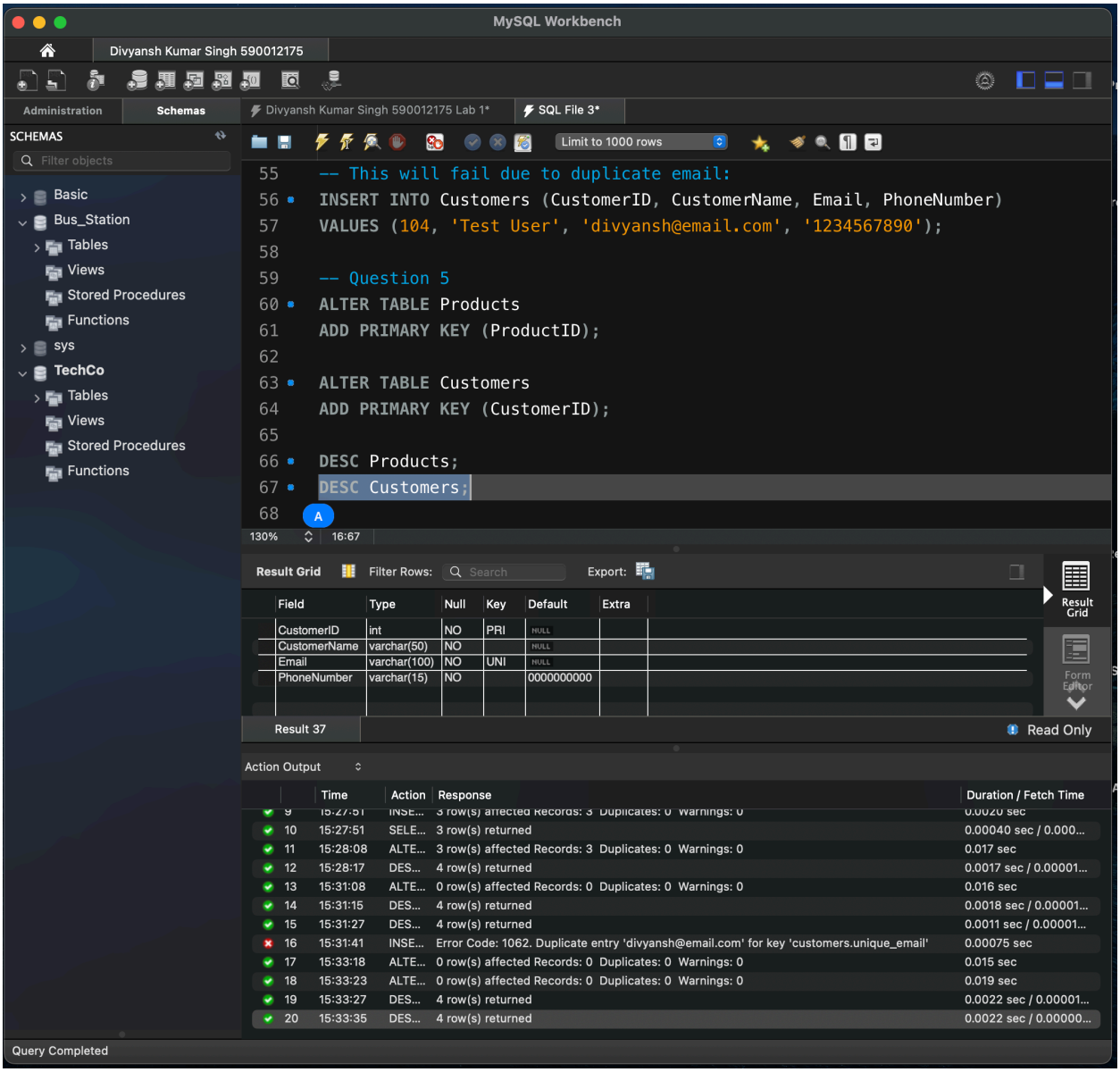




- **UNIQUE** → Ensures no duplicate values exist in the column.

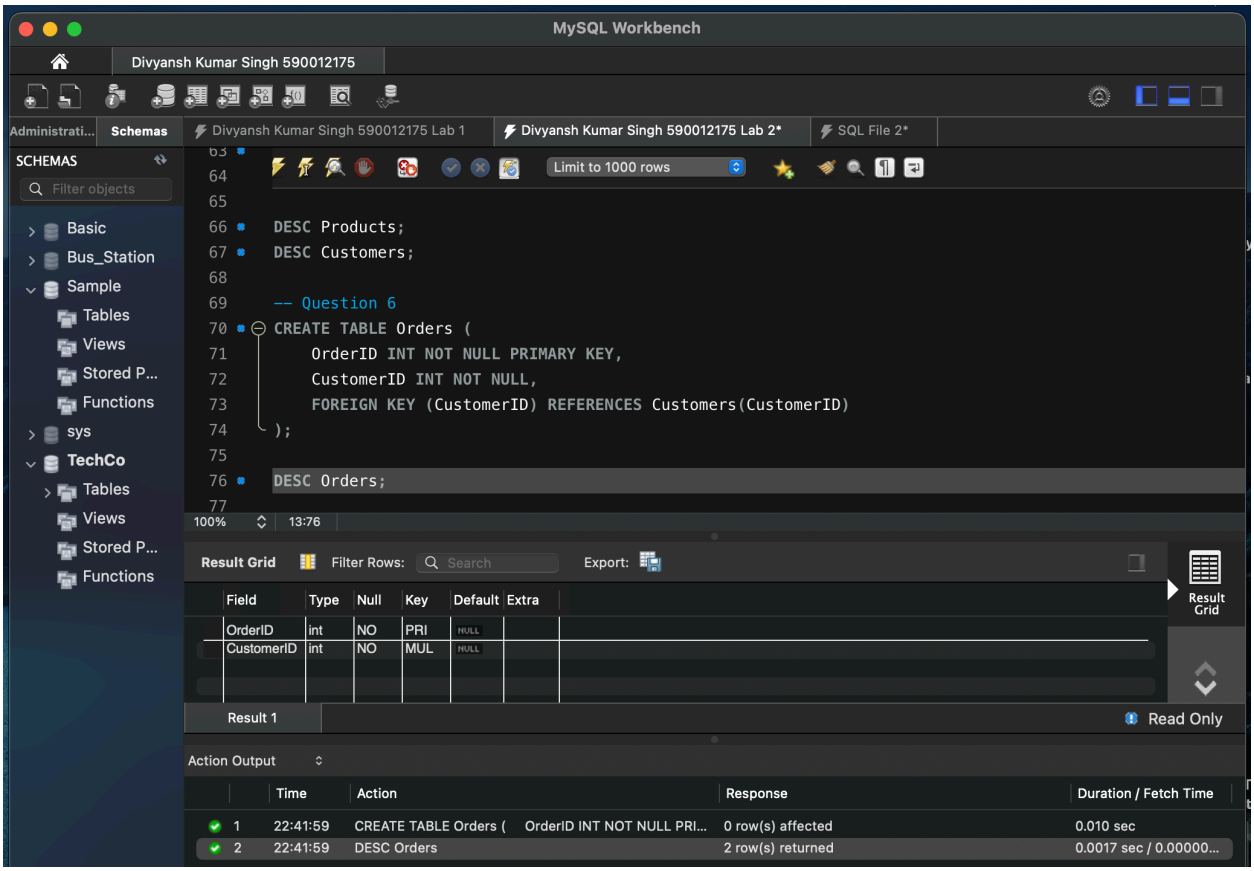
5. Set the `ProductID` column as the primary key for the `Products` table. Also, define `CustomerID` as the primary key for the `Customers` table.





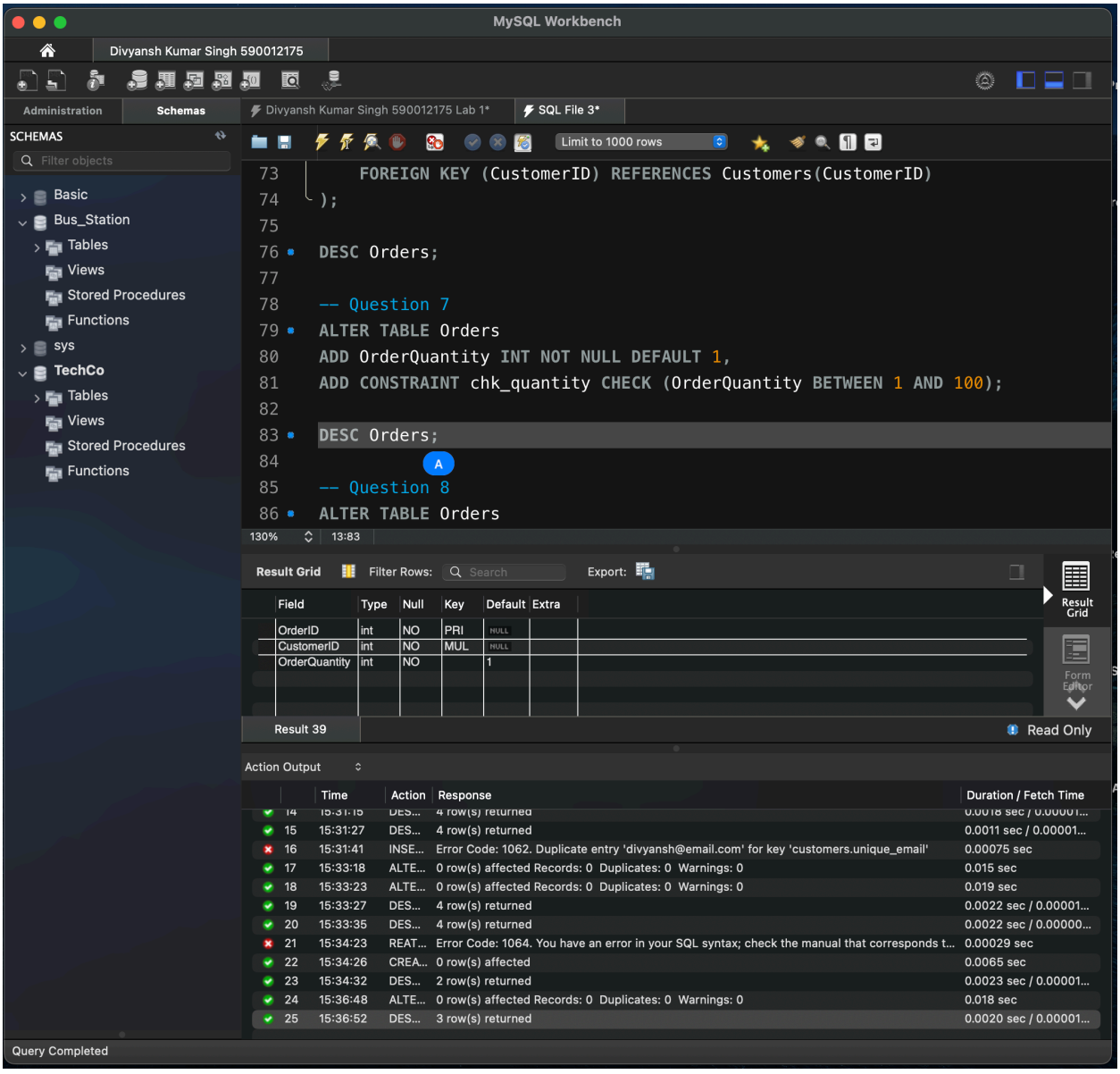
- **PRIMARY KEY** → Combination of NOT NULL + UNIQUE, ensures entity integrity.

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.



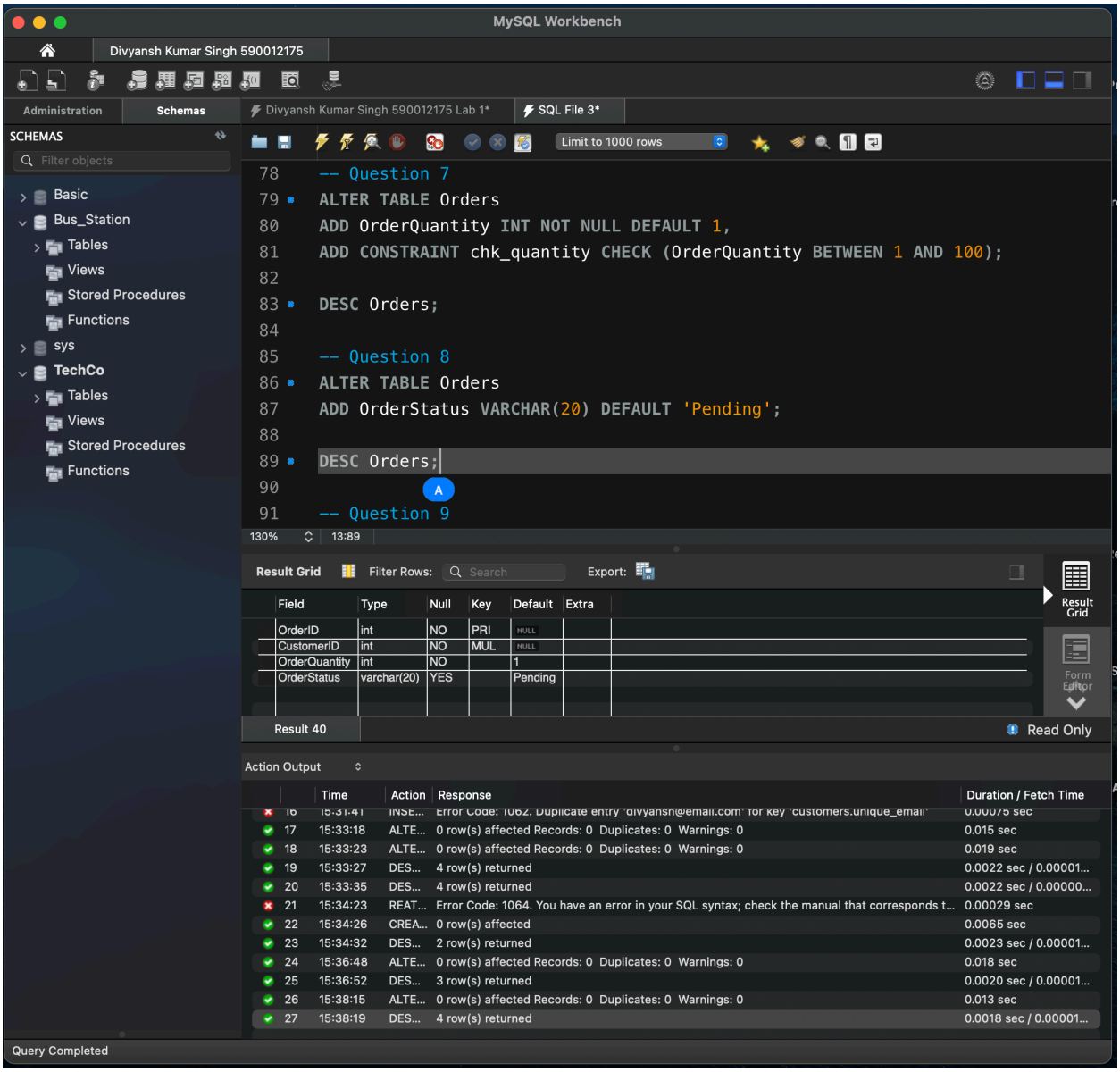
- **FOREIGN KEY** → Links a column to a primary key in another table.
- **REFERENCES** → Defines which table/column the foreign key points to.
- **Referential Integrity** → Ensures relationships between tables remain valid.

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.



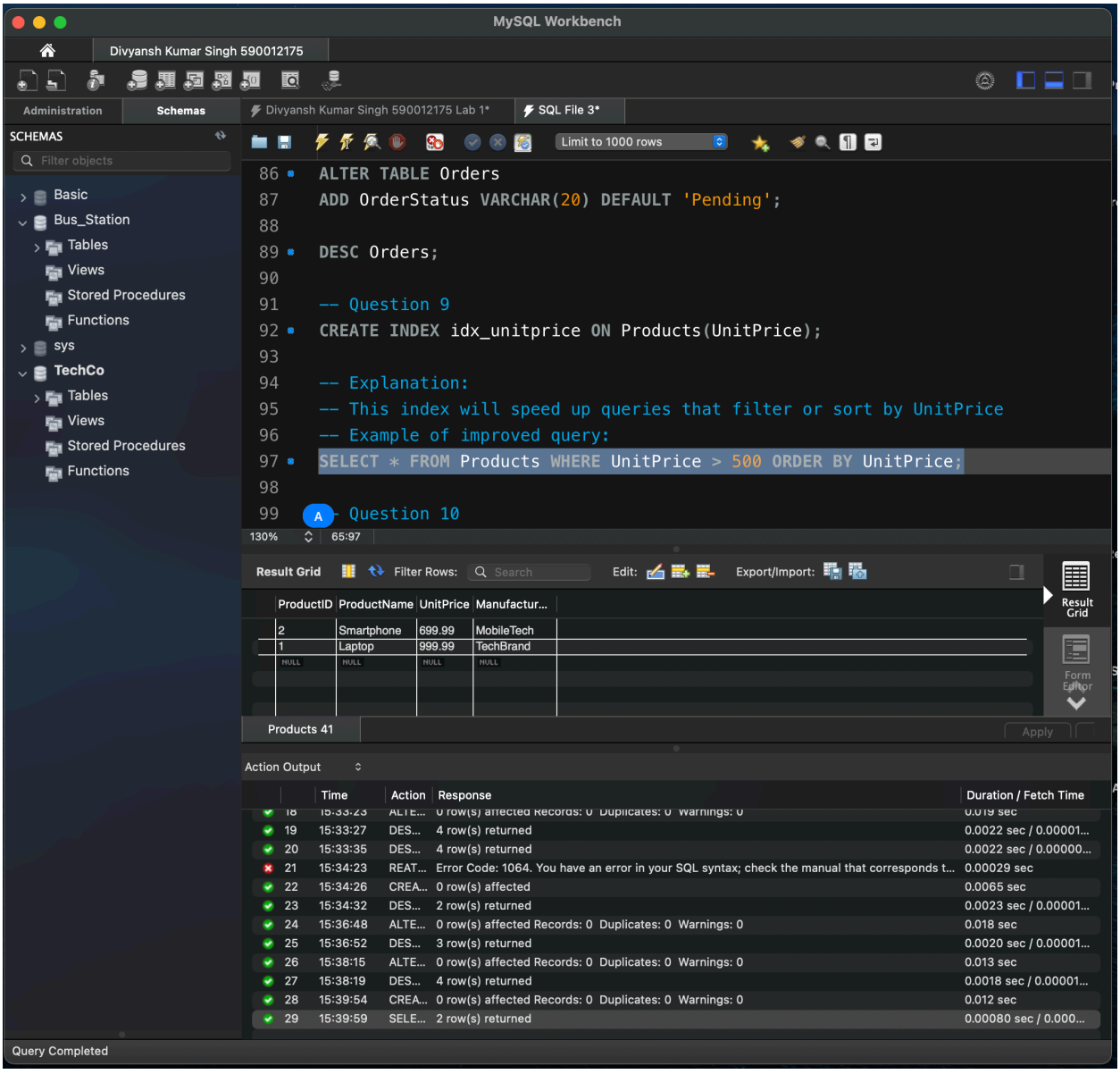
- **CHECK** → Restricts values in a column based on a condition (e.g., between 1–100).

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.'



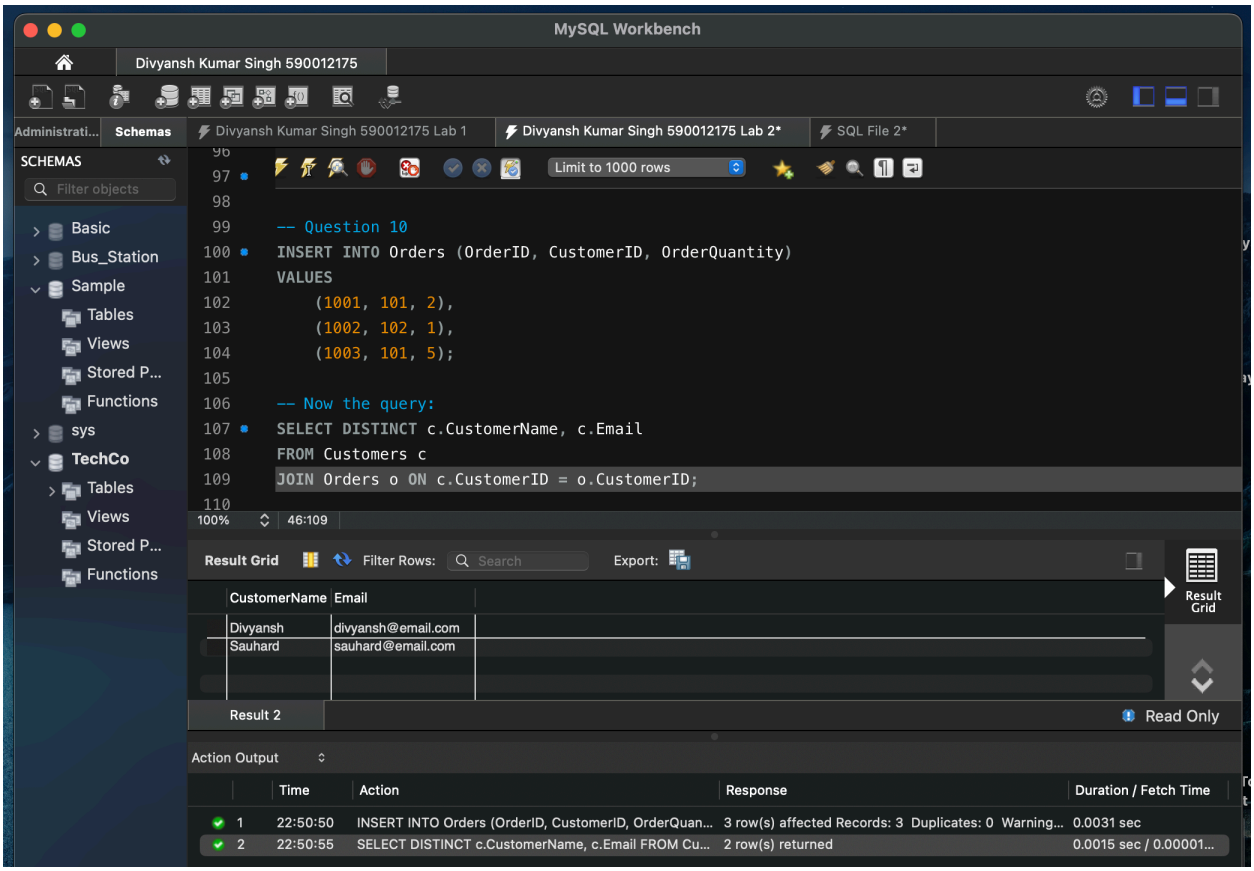
- **DEFAULT** → Inserts a preset value when no value is provided (e.g., 'Pending').

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



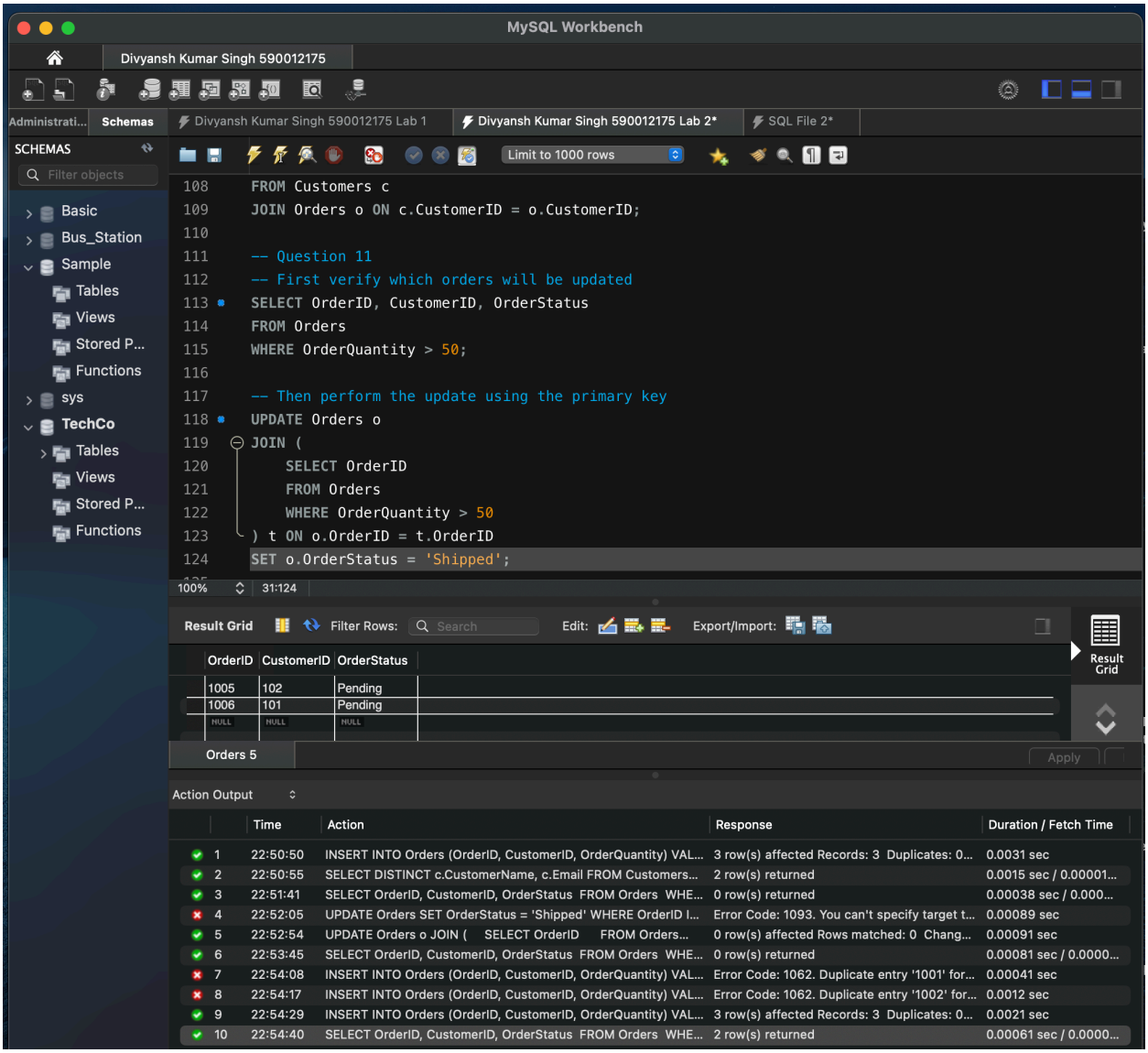
- **CREATE INDEX** → Improves search and query performance on a column.

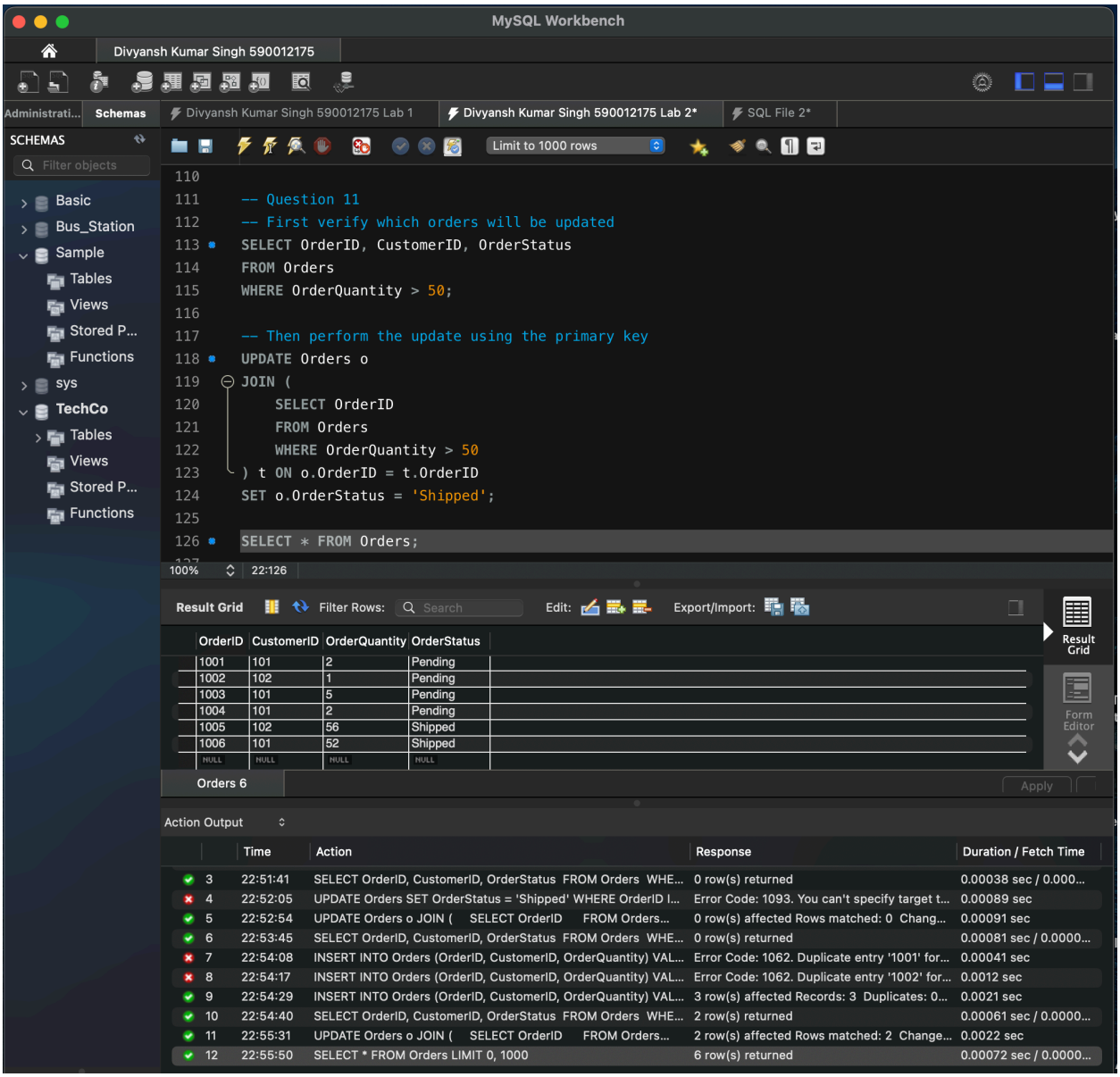
10. Write an SQL query to retrieve the names and email addresses of all customers who placed orders.
Join the `Customers` and `Orders` tables to achieve this.



- **SELECT** → Retrieves data from tables.
- **JOIN** → Combines rows from multiple tables based on related columns.
- **ON** → Defines the join condition.

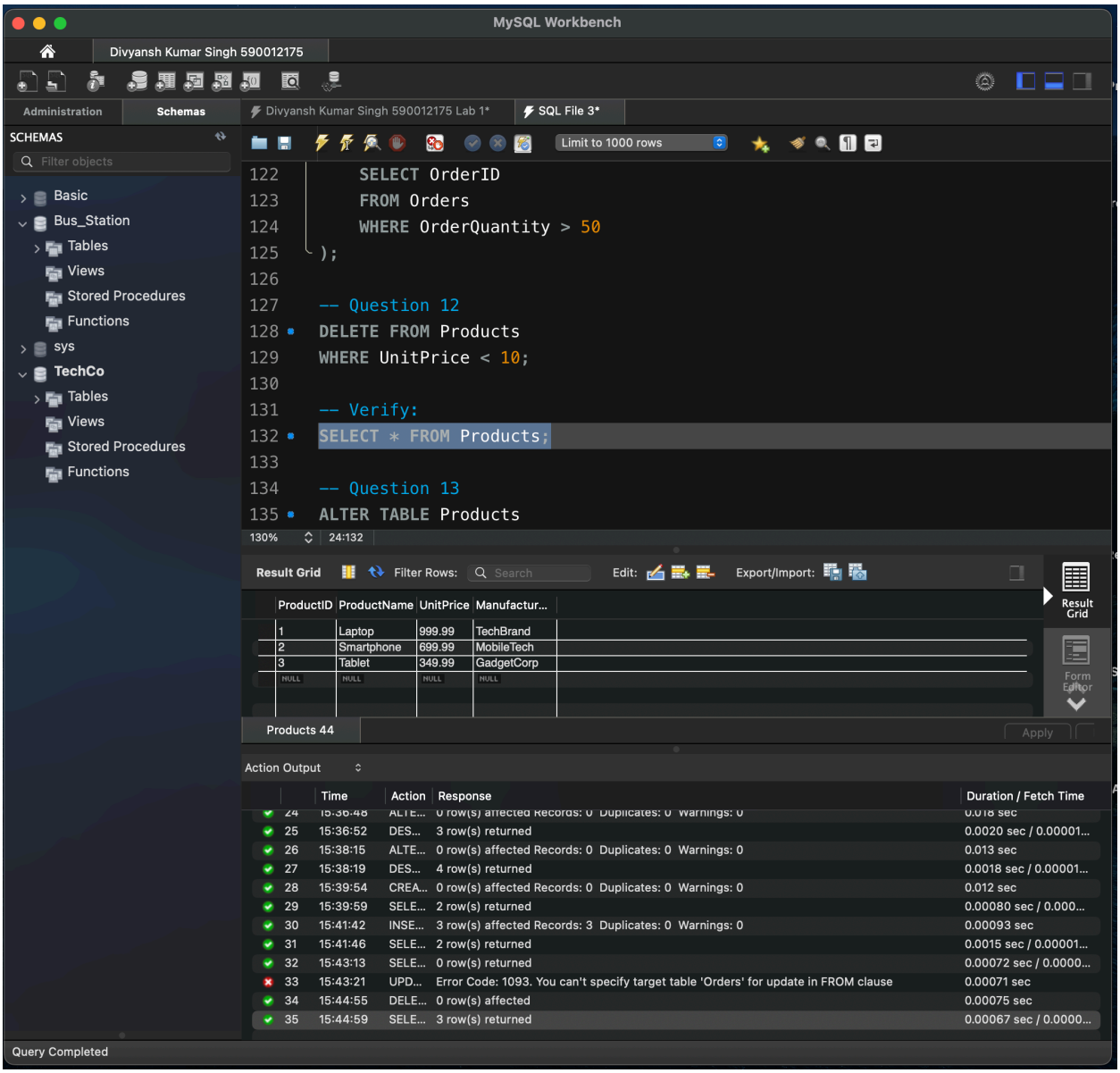
11. Modify the `OrderStatus` of all orders with a quantity greater than 50 to 'Shipped.' Provide the SQL statement for this update.





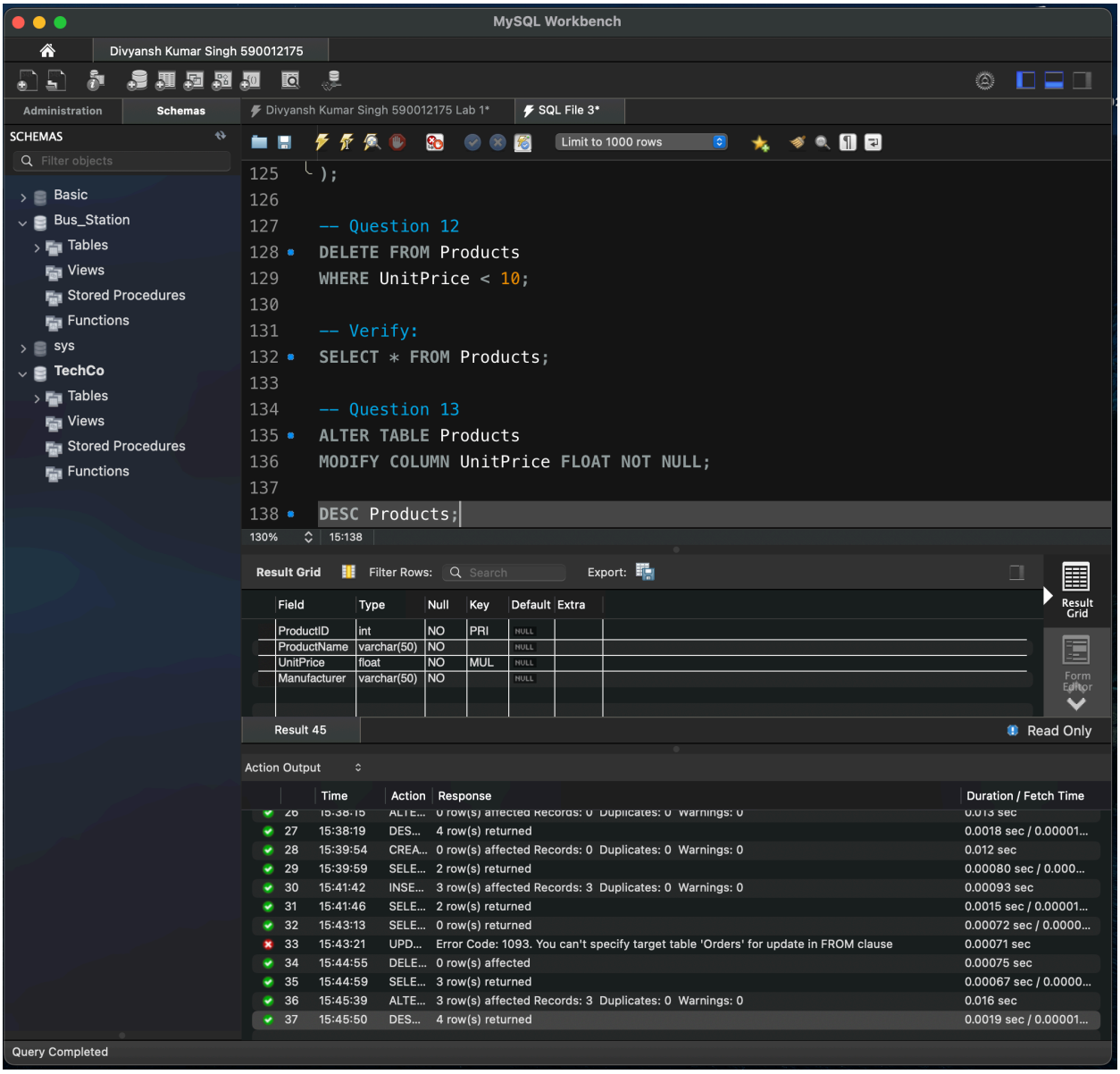
- **UPDATE** → Modifies existing rows in a table.
- **SET** → Assigns new values to columns.
- **WHERE** → Applies a condition to restrict which rows are updated.

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



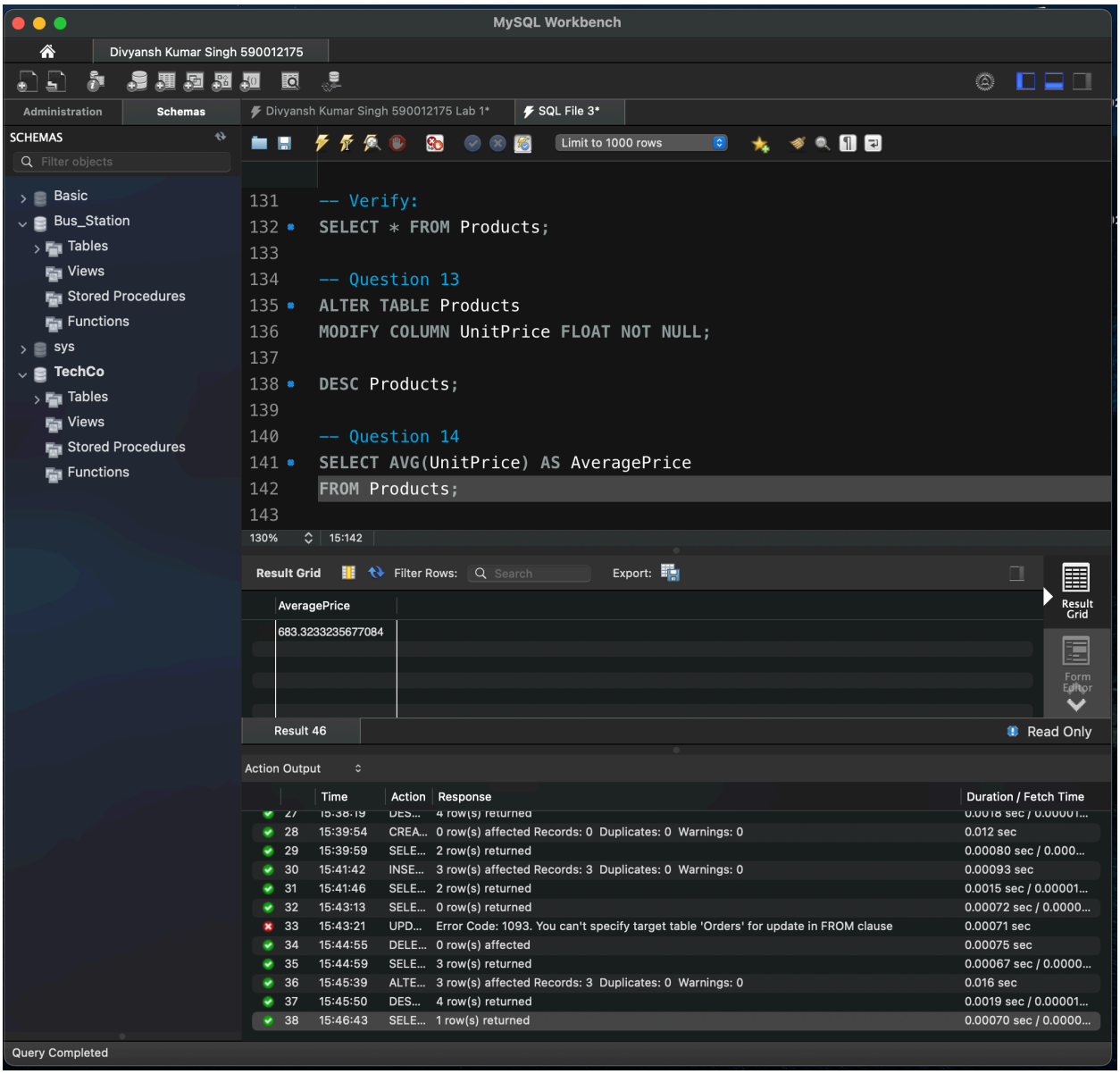
- **DELETE FROM** → Removes rows from a table.
- **WHERE** → Restricts deletion to matching rows.

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.



- **ALTER TABLE** → Changes table structure.
- **MODIFY / ALTER COLUMN** → Alters column definition.
- **FLOAT** → Data type for approximate decimal values.

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



- **SELECT AVG()** → Aggregate function to calculate the average of a column.
- **FROM** → Specifies the table to query.

CONCLUSION

This DBMS lab helped me understand the importance of SQL constraints in ensuring data accuracy and integrity. I practiced using constraints like **NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT, and INDEX** to manage and validate data effectively.

Along with this, I worked on **DDL and DML queries** to create, modify, and manipulate tables. Overall, the lab improved my practical knowledge of database design and strengthened my ability to build reliable and consistent databases.