BENAZIR BHUTTO SHAHEED UNIVERSITY LYARI KARACHI

Faculty of Computing Science and Information Technology



Department of Computer Science

**Lab Manual**

**CS-441 DATABASE SYSTEMS**

**(5th Semester)**

**Spring– 2025**

**Prepared by**

Hamza Yaseen

B2362018

CONTENT PAGE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Weeks** | **Topic** | **P: No** | **Date** | **Signature** |
| **1** | MySQL Database Installation, Login with Default User and Password, SQL CREATE DATABASE Statement, SQL DROP DATABASE Statement, SQL BACKUP DATABASE for SQL Server, |  |  |  |
| **2** | SQL CREATE TABLE Statement, SQL DROP TABLE Statement, SQL ALTER TABLE Statement, SQL NOT NULL Constraint, SQL UNIQUE Constraint, SQL DEFAULT Constraint |  |  |  |
| **3** | SQL INSERT INTO Statement, SQL SELECT Statement, SQL SELECT DISTINCT Statement, SQL WHERE Clause, SQL AND, OR and NOT Operators, SQL ORDER BY Keyword |  |  |  |
| **4** | SQL UPDATE Statement, SQL DELETE Statement, SQL PRIMARY KEY Constraint, SQL FOREIGN KEY Constraint, SQL AUTO INCREMENT Field |  |  |  |
| **5** | SQL TOP, LIMIT, FETCH FIRST or ROWNUM Clause, SQL MIN () and MAX() Functions, SQL COUNT(), AVG() and SUM() Functions, |  |  |  |
| **6** | SQL LIKE Operator, SQL Wildcards, SQL IN Operator, SQL BETWEEN Operators |  |  |  |
| **7** | SQL Joins, SQL INNER JOIN Keyword, SQL LEFT JOIN Keyword, SQL RIGHT JOIN Keyword |  |  |  |
| **8** | **Mid Term Examination** | | | |
| **9** | SQL FULL OUTER JOIN Keyword, SQL Self Join, SQL UNION Operator |  |  |  |
| **10** | SQL GROUP BY Statement, SQL HAVING Clause, SQL EXISTS Operator, SQL ANY and ALL Operators |  |  |  |
| **11** | SQL SELECT INTO Statement, SQL INSERT INTO SELECT Statement, |  |  |  |
| **12** | Sub Queries |  |  |  |
| **13** | Sub Queries |  |  |  |
| **14** | Revision |  |  |  |
| **15** | Final Semester Project Discussion |  |  |  |
| **16** | Final Lab |  |  |  |

Lab 01

Topic: MySQL Database Installation, Login with Default User and Password, SQL CREATE DATABASE Statement, SQL DROP DATABASE Statement, SQL BACKUP DATABASE for SQL Server,

**MySQL**

MySQL is an open-source relational database management system (RDBMS) that is widely used for various applications, from small-scale projects to large-scale websites and enterprise-level solutions. It is a popular choice for web applications because it is fast, reliable, and scalable.

MySQL uses Structured Query Language (SQL) to manage data inside a database. SQL is a standard language for interacting with databases, and it is supported by most RDBMSs.

Here are some of the benefits of using MySQL:

* It is open source, so it is free to use and distribute.
* It is fast and reliable.
* It is scalable, so it can be used for small-scale projects or large-scale enterprise applications.
* It is easy to use and administer.
* It is well-supported by a large community of developers.

**XAMPP**

XAMPP is a free and open-source cross-platform web server solution stack package developed by Apache Friends, consisting mainly of the Apache HTTP Server, MariaDB database, and interpreters for scripts written in the PHP and Perl programming languages.

XAMPP is a popular choice for web developers because it allows them to set up a local web server on their own computer. This makes it possible to test web applications and websites without having to deploy them to a live web server.

**XAMPP Installation**

1. Go to the XAMPP website: https://www.apachefriends.org/index.html and download the latest version of XAMPP for your operating system.
2. Once the download is complete, run the installer.
3. Follow the on-screen instructions to install XAMPP.
4. Once XAMPP is installed, open the XAMPP Control Panel.
5. In the XAMPP Control Panel, start the Apache and MySQL modules.
6. Open a web browser and go to http://localhost.
7. You should see the XAMPP welcome page.

**Mysql login with Default login and password**

1. Open a command prompt window.
2. Navigate to the XAMPP installation directory. The default installation directory is C:\xampp\mysql\bin.
3. Type the following command and press Enter:

mysql -u root -p

1. Enter the MySQL root password when prompted.
2. You will be logged in to the MySQL command line.

**SQL CREATE DATABASE Statement**

CREATE DATABASE databasename;

**SQL Drop Database Statement**

DROP DATABASE databasename;

**SQL Backup Database for SQL Server**

BACKUP DATABASE databasename  
TO DISK = 'filepath';

**Lab Task**

* Objective: To learn about the basic concepts of MySQL database management system and how to use SQL statements to create, drop, and backup databases.
* Instructions:
  1. Install XAMPP on your computer.
  2. Start the Apache and MySQL modules in the XAMPP Control Panel.
  3. Open a command prompt window and navigate to the XAMPP installation directory.
  4. Use the mysql -u root -p command to log in to the MySQL command line.
  5. Create a new database using the CREATE DATABASE statement.
  6. Drop the database using the DROP DATABASE statement.
  7. Back up the database using the BACKUP DATABASE statement.
* Expected Outcome:
  1. You should be able to create, drop, and backup databases using SQL statements.
  2. You should be familiar with the basic concepts of MySQL database management system.

Here are some additional resources that you may find helpful:

* MySQL Documentation: https://dev.mysql.com/doc/
* SQL Tutorial: https://www.w3schools.com/sql/
* XAMPP Documentation: https://www.apachefriends.org/index.html

Lab 02

Topic: SQL CREATE TABLE Statement, SQL DROP TABLE Statement, SQL ALTER TABLE Statement, SQL NOT NULL Constraint, SQL UNIQUE Constraint, SQL DEFAULT Constraint

**SQL CREATE TABLE Statement**

The CREATE TABLE statement is used to create a new table in a database.

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

### **Syntax**

CREATE TABLE table\_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
   ....  
);

**SQL CREATE TABLE Example**

CREATE TABLE Persons (

PersonID int,

LastName varchar(255),

FirstName varchar(255),

Address varchar(255),

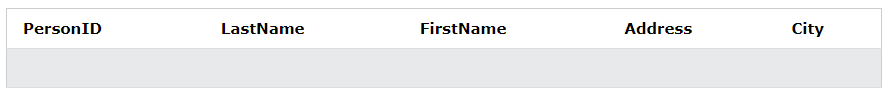
City varchar(255)

);

The PersonID column is of type int and will hold an integer.

The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters.

The empty "Persons" table will now look like this:



**Create Table Using Another Table**

A copy of an existing table can also be created using CREATE TABLE.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

### **Syntax**

CREATE TABLE new\_table\_name AS  
    SELECT column1, column2,...  
    FROM existing\_table\_name  
    WHERE ....;

The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

CREATE TABLE TestTable AS  
SELECT customername, contactname  
FROM customers;

**SQL DROP TABLE Statement**

The DROP TABLE statement is used to drop an existing table in a database.

### **Syntax**

DROP TABLE table\_name;

**Note:** Be careful before dropping a table. Deleting a table will result in loss of complete information stored in the table!

The following SQL statement drops the existing table "Shippers":

DROP TABLE Shippers;

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

### **Syntax**

TRUNCATE TABLE table\_name;

**SQL ALTER TABLE Statement**

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

**ALTER TABLE - ADD Column**

To add a column in a table, use the following syntax:

ALTER TABLE table\_name  
ADD column\_name datatype;

The following SQL adds an "Email" column to the "Customers" table:

ALTER TABLE Customers  
ADD Email varchar(255);

**ALTER TABLE - DROP COLUMN**

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table\_name  
DROP COLUMN column\_name;

The following SQL deletes the "Email" column from the "Customers" table:

ALTER TABLE Customers  
DROP COLUMN Email;

**ALTER TABLE - RENAME COLUMN**

To rename a column in a table, use the following syntax:

ALTER TABLE table\_name  
RENAME COLUMN old\_name to new\_name;

**ALTER TABLE - ALTER/MODIFY DATATYPE**

To change the data type of a column in a table, use the following syntax:

ALTER TABLE table\_name  
MODIFY COLUMN column\_name datatype;

**SQL NOT NULL Constraint**

By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

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

**SQL NOT NULL on CREATE TABLE**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int  
);

**SQL NOT NULL on ALTER TABLE**

To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

ALTER TABLE Persons  
MODIFY COLUMN Age int NOT NULL;

**SQL UNIQUE Constraint.**

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table

**SQL UNIQUE Constraint on CREATE TABLE**

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    UNIQUE (ID)  
);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

**SQL UNIQUE Constraint on ALTER TABLE**

ALTER TABLE Persons  
ADD UNIQUE (ID);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

ALTER TABLE Persons  
ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName);

**DROP a UNIQUE Constraint**

To drop a UNIQUE constraint, use the following SQL:

ALTER TABLE Persons  
DROP INDEX UC\_Person;

**SQL DEFAULT Constraint**

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

**SQL DEFAULT on CREATE TABLE**

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255) DEFAULT 'Sandnes'  
);

The DEFAULT constraint can also be used to insert system values, by using functions like [GETDATE()](https://www.w3schools.com/sql/func_sqlserver_getdate.asp)

CREATE TABLE Orders (  
    ID int NOT NULL,  
    OrderNumber int NOT NULL,  
    OrderDate date DEFAULT GETDATE()  
);

**SQL DEFAULT on ALTER TABLE**

To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

ALTER TABLE Persons  
ALTER City SET DEFAULT 'Sandnes';

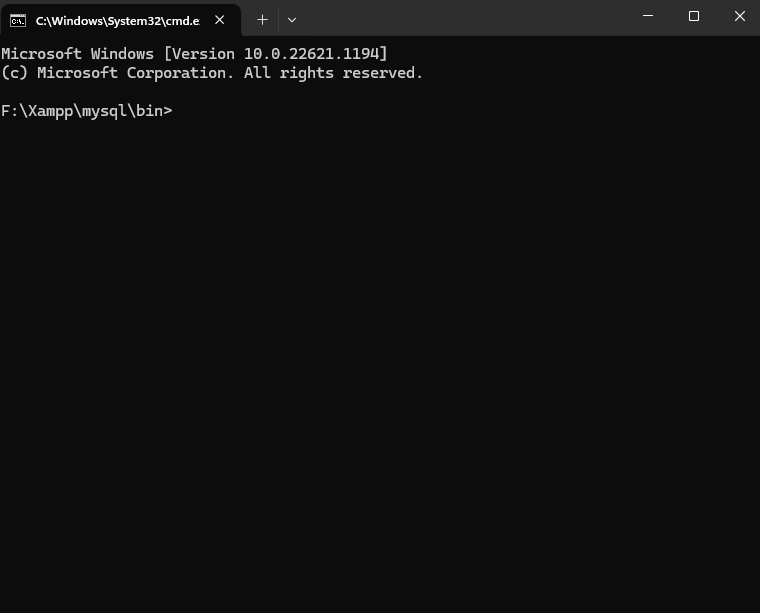
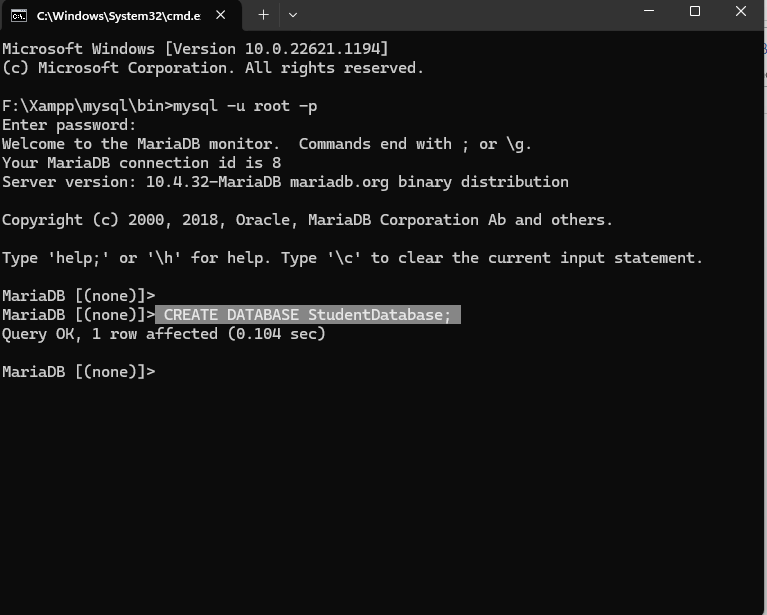
**DROP a DEFAULT Constraint**

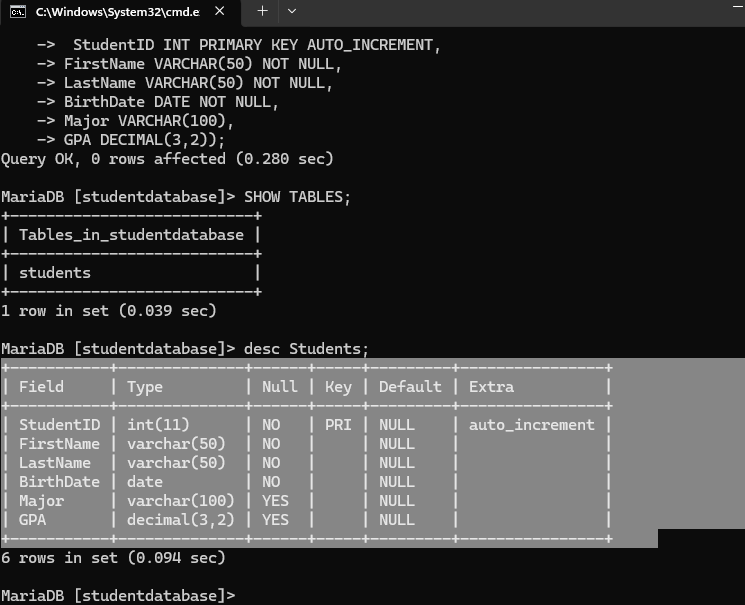
To drop a DEFAULT constraint, use the following SQL:

ALTER TABLE Persons  
ALTER City DROP DEFAULT;

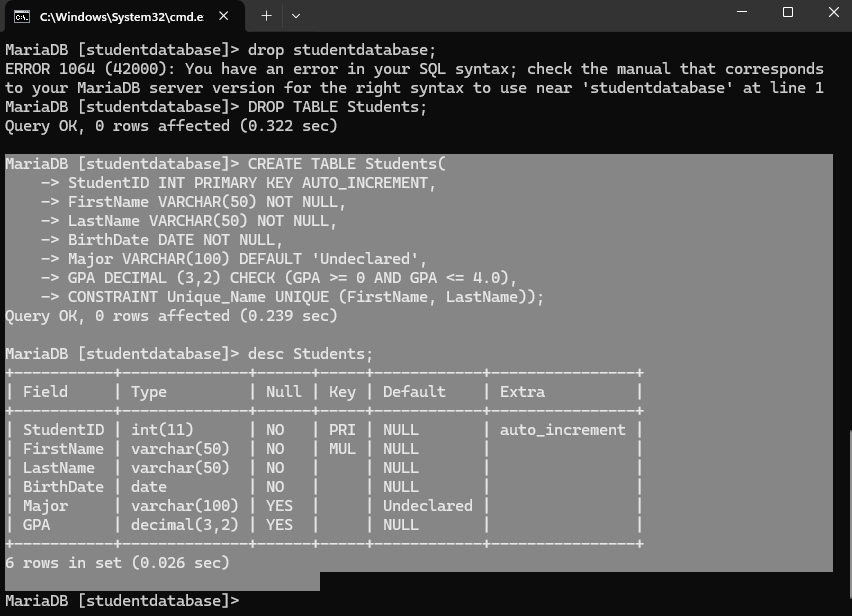
**Lab Task:** **SQL CREATE TABLE Statement**

**Objective:** To create a well-structured database table for storing student information using the SQL CREATE TABLE statement.

1. Open your preferred SQL client or interface (e.g., MySQL Workbench, SQLite Studio).
2. Create a new database named "StudentDatabase." 
3. Design a table named "Students" to store student information with the following columns:
   1. StudentID (INT): A unique identifier for each student.
   2. FirstName (VARCHAR): The first name of the student.
   3. LastName (VARCHAR): The last name of the student.
   4. BirthDate (DATE): The date of birth of the student.
   5. Major (VARCHAR): The major or field of study of the student.
   6. GPA (DECIMAL): The Grade Point Average of the student.



1. Specify appropriate constraints for the columns:
   1. Set StudentID as the primary key.
   2. Ensure FirstName and LastName cannot be NULL.
   3. Set a unique constraint on the combination of FirstName and LastName.
   4. Set a default value of 'Undeclared' for the Major column.
   5. Limit GPA values to the range of 0 to 4.0.



**Expected Outcome:**

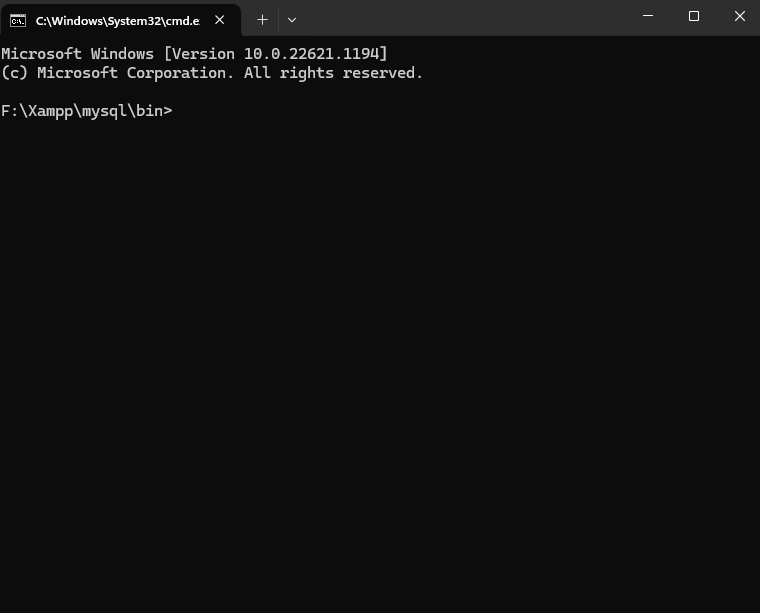
1. You should have successfully created a "Students" table within the "StudentDatabase."
2. The table's columns should have appropriate data types and constraints.
3. After inserting sample data into the table, you should be able to verify the constraints' enforcement.
4. You will gain practical experience in designing and implementing a database table using SQL CREATE TABLE statements.

**Lab Task: SQL DROP TABLE Statement**

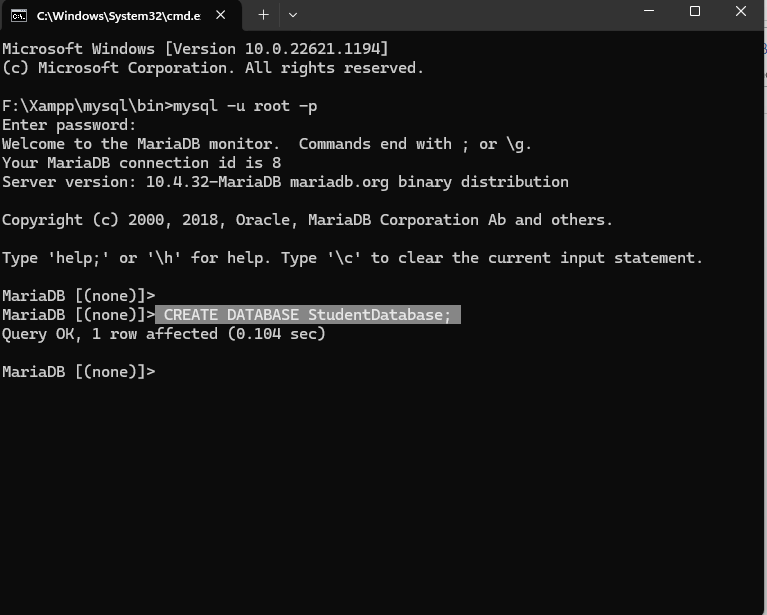
**Objective:** To understand the SQL DROP TABLE statement and practice using it to remove a table from a database.

**Task:**

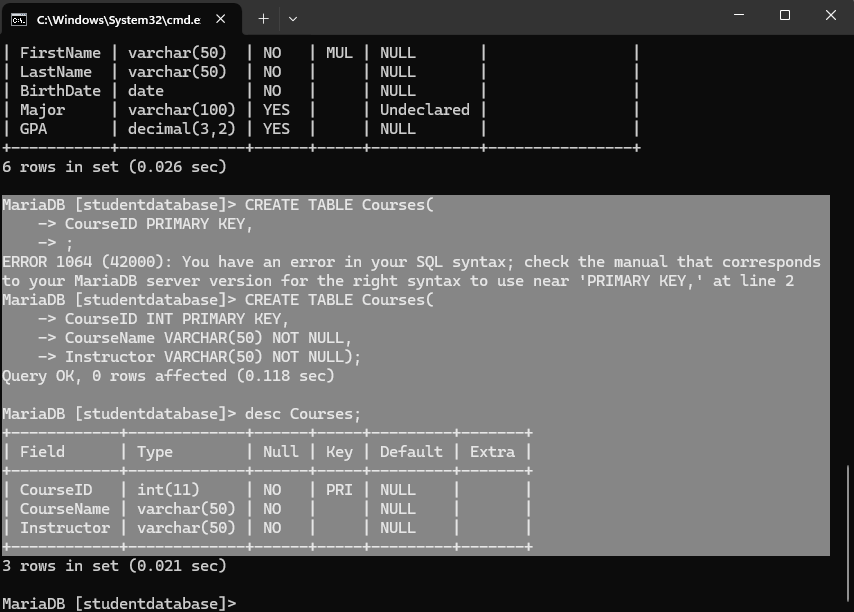
1. Open your preferred SQL client or interface (e.g., MySQL Workbench, SQLite Studio).



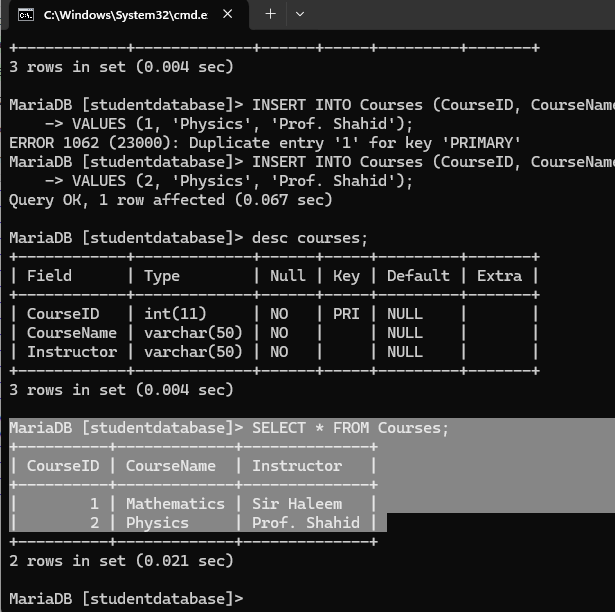
1. Use the database "StudentDatabase" that you created in a previous lab or create a new one for this task.



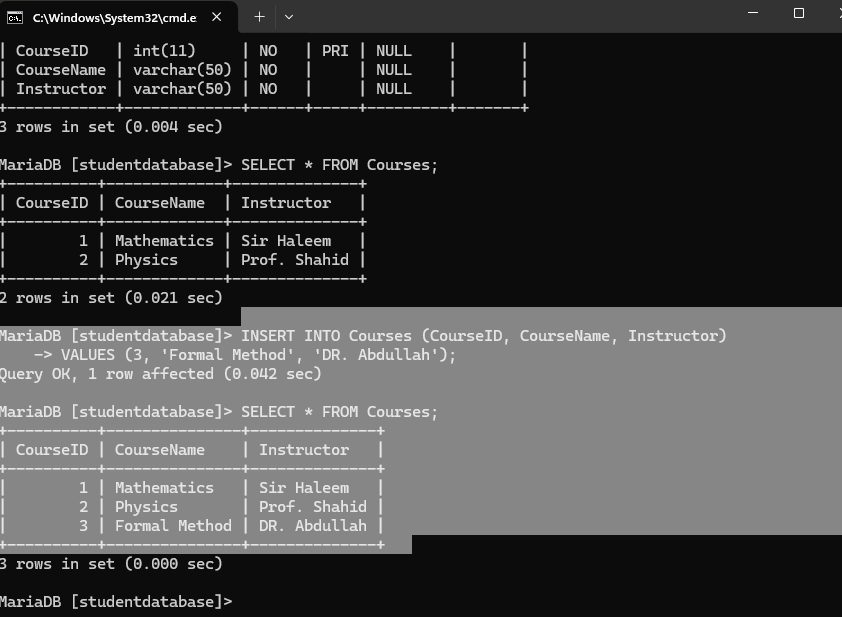
1. Within the database, create a table named "Courses" with the following columns:
   1. CourseID (INT): A unique identifier for each course.
   2. CourseName (VARCHAR): The name of the course.
   3. Instructor (VARCHAR): The name of the course instructor.



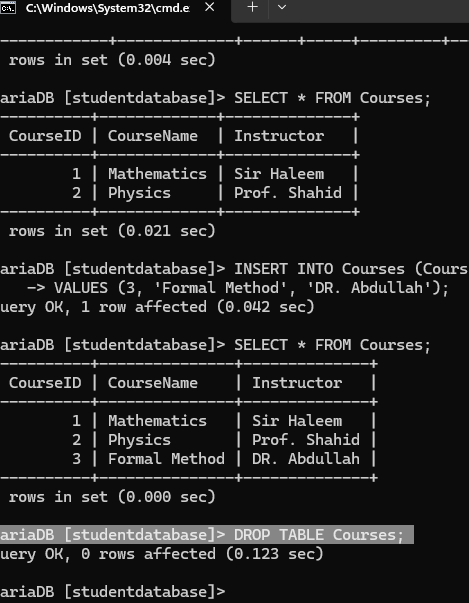
1. Insert sample data into the "Courses" table.



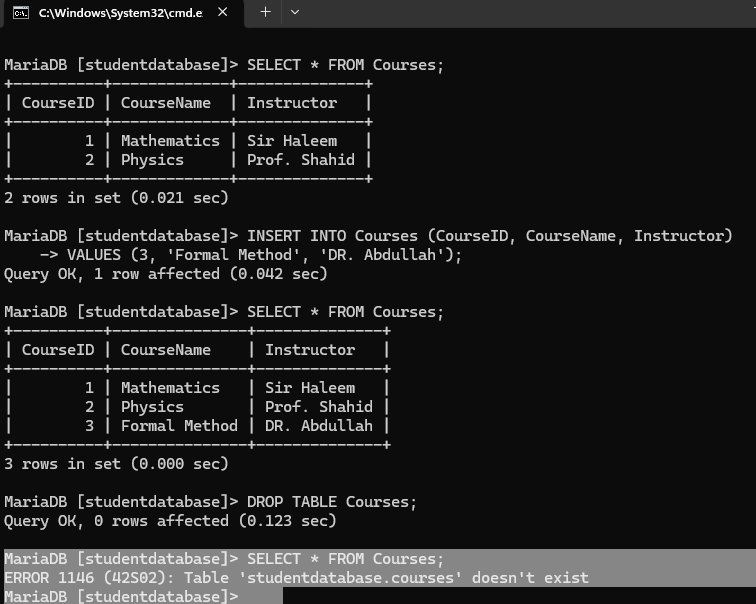
1. Write a SQL query to retrieve and display all records from the "Courses" table to verify the data.



1. Use the SQL DROP TABLE statement to remove the "Courses" table from the database.



1. Attempt to run the same query from step 5 again and observe the result.



**Expected Outcome:**

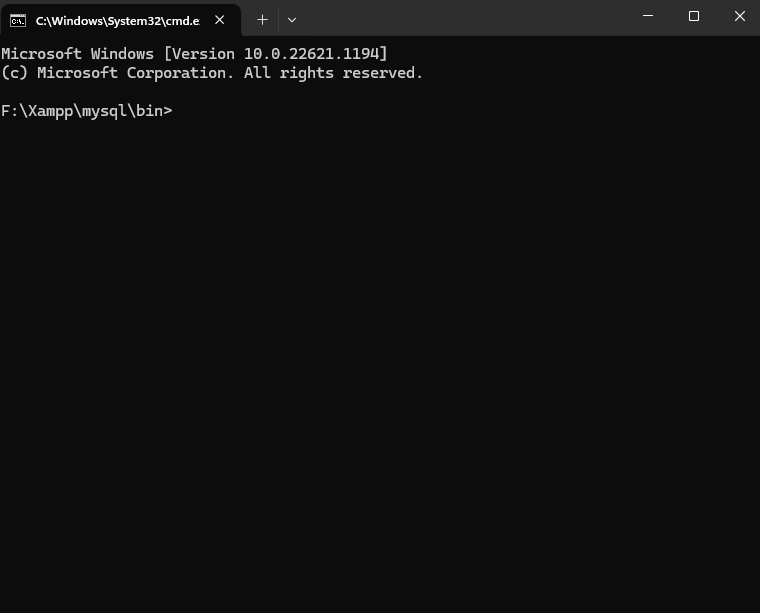
1. You should have successfully created the "Courses" table within the specified database and inserted sample data.
2. The query in step 5 should display the sample data from the "Courses" table.
3. After using the DROP TABLE statement, the table should be removed from the database.
4. Running the query from step 5 after dropping the table should result in an error indicating that the table doesn't exist.

**Lab Task: SQL ALTER TABLE Statement**

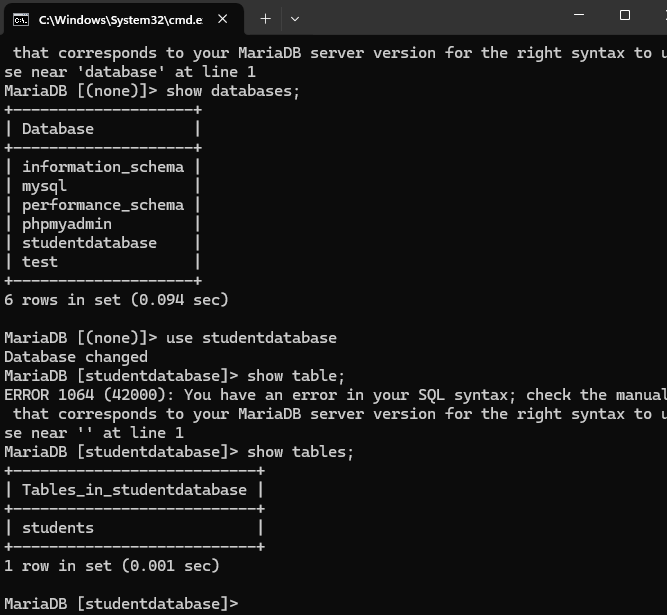
**Objective**: To practice using the SQL ALTER TABLE statement to modify the structure of an existing database table.

**Task:**

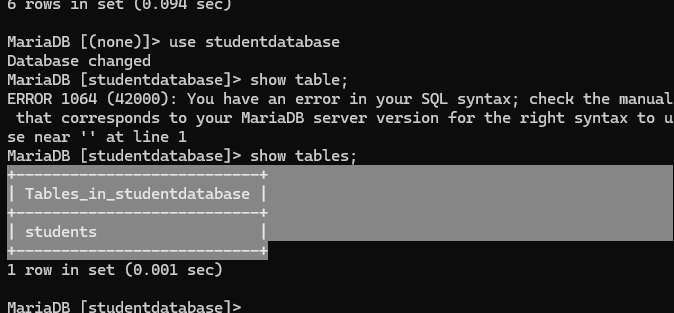
1. Open your preferred SQL client or interface (e.g., MySQL Workbench, SQLite Studio).



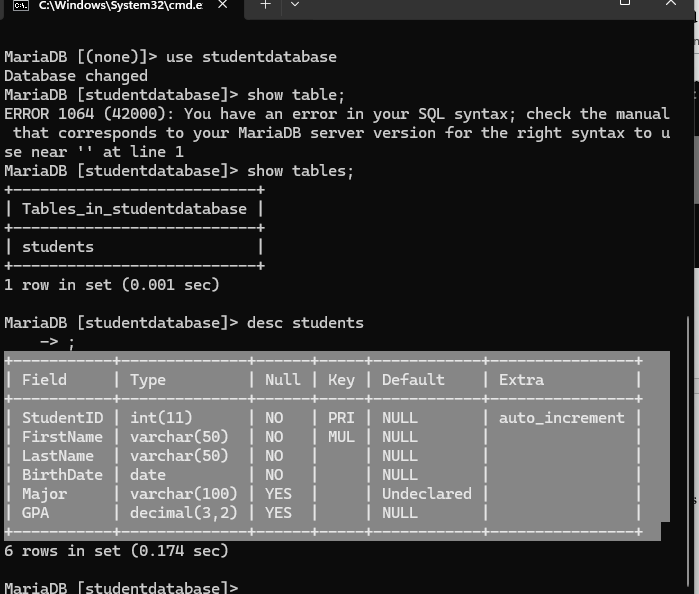
1. Use the database "StudentDatabase" that you've been working with in previous labs or create a new one for this task.



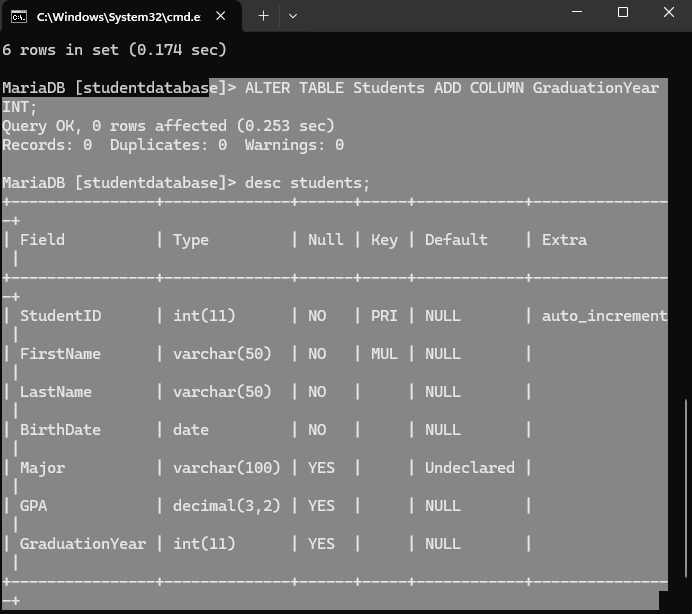
1. Within the database, you should have a "Students" table created in an earlier lab.



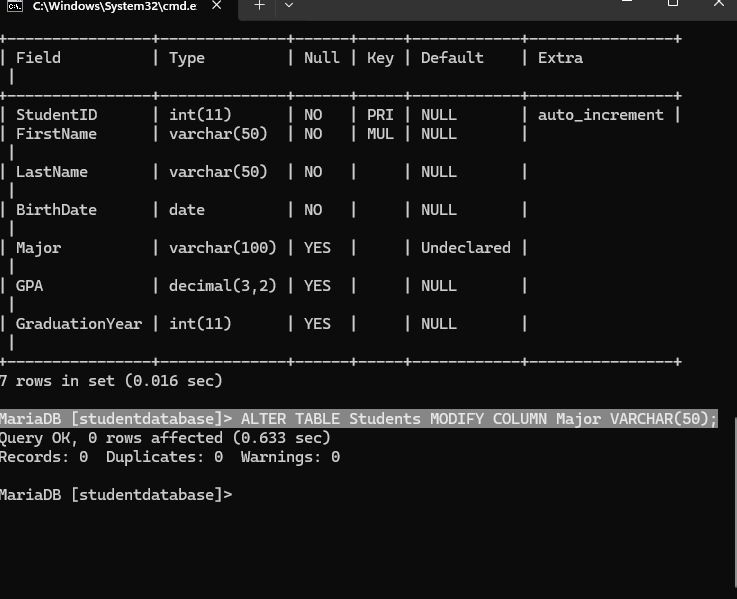
1. Review the "Students" table structure and data. It should have columns such as StudentID, FirstName, LastName, BirthDate, Major, and GPA.



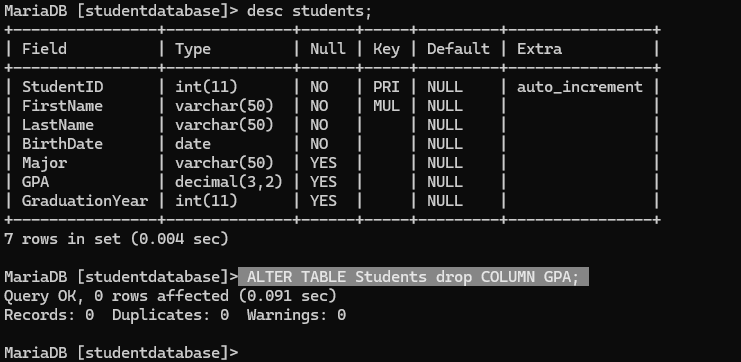
1. Use the SQL ALTER TABLE statement to add a new column named "GraduationYear" (INT) to the "Students" table.



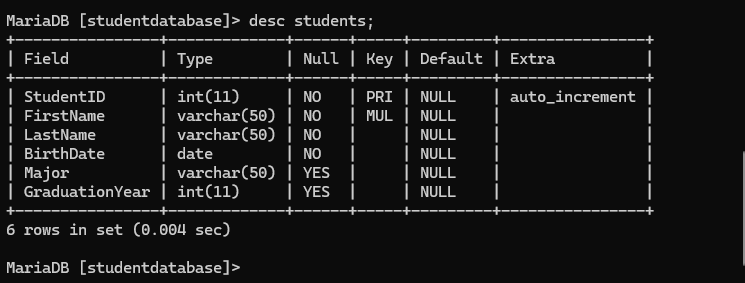
1. Modify the "Major" column's data type to VARCHAR(50) to accommodate longer field entries.



1. Drop the "GPA" column from the "Students" table.



1. Write an SQL query to retrieve and display all records from the "Students" table after the alterations.



**Expected Outcome:**

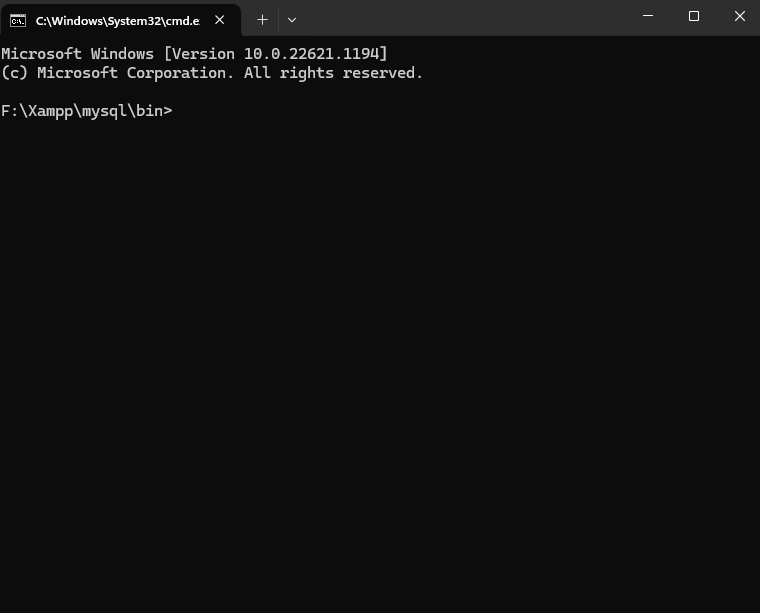
1. You should have the "Students" table available within the specified database.
2. The table should initially have columns like StudentID, FirstName, LastName, BirthDate, Major, and GPA.
3. After completing the lab, the "Students" table should have a new "GraduationYear" column and the "Major" column's data type should be modified.
4. The "GPA" column should no longer be present in the table.
5. Running the query to retrieve data from the "Students" table should display the altered structure.

**Lab Tasl: SQL NOT NULL Constraint**

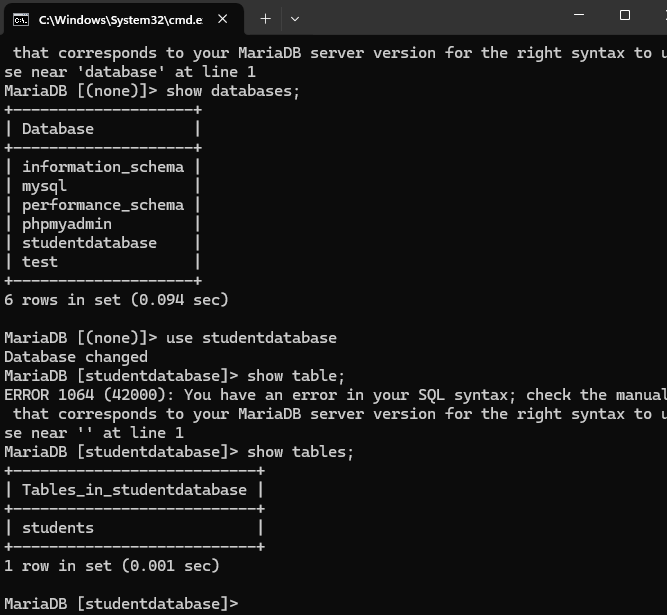
**Objective:** To understand the SQL NOT NULL constraint and practice applying it to enforce data integrity in a database table.

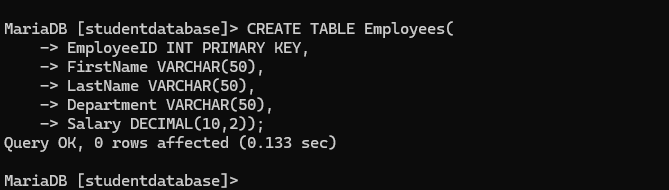
**Task:**

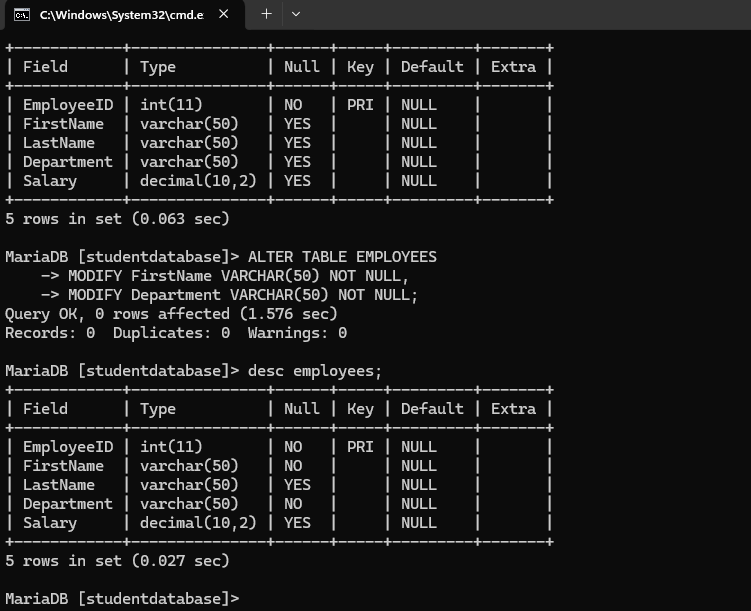
1. Open your preferred SQL client or interface (e.g., MySQL Workbench, SQLite Studio).



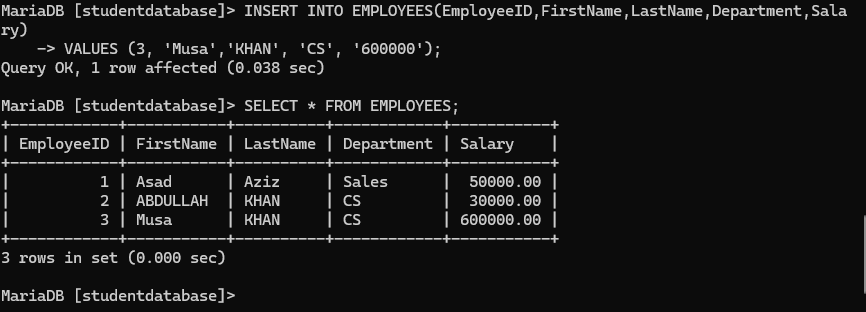
1. Use the database "StudentDatabase" that you've been working with in previous labs or create a new one for this task.



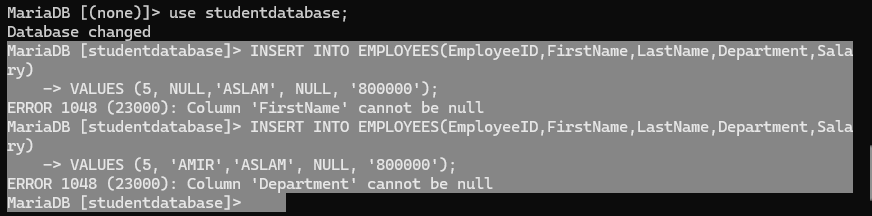
1. Create a new table named "Employees" with the following columns:
   1. EmployeeID (INT): A unique identifier for each employee.
   2. FirstName (VARCHAR): The first name of the employee.
   3. LastName (VARCHAR): The last name of the employee.
   4. Department (VARCHAR): The department the employee belongs to.
   5. Salary (DECIMAL): The salary of the employee
2. Apply the SQL NOT NULL constraint to the "FirstName" and "Department" columns to ensure that these fields cannot have NULL values.



1. Insert sample data into the "Employees" table, ensuring you provide values for the "FirstName" and "Department" columns.



1. Attempt to insert a record with a NULL value in the "FirstName" or "Department" column and observe the result.



**Expected Outcome:**

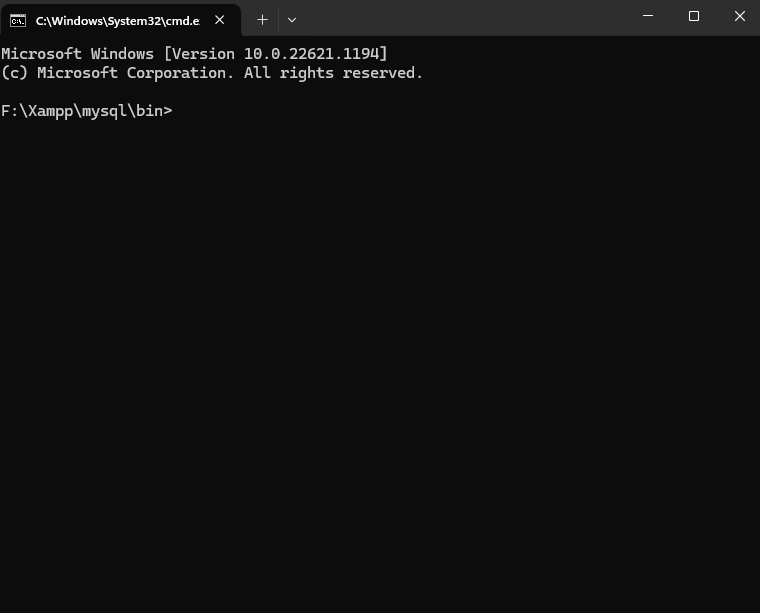
1. The "Employees" table should be successfully created within the specified database.
2. The "FirstName" and "Department" columns should have the NOT NULL constraint applied.
3. You should be able to insert sample data into the table without any issues, as long as the "FirstName" and "Department" columns have valid values.
4. Attempting to insert a record with a NULL value in the "FirstName" or "Department" column should result in an error due to the NOT NULL constraint.

**Lab Task: SQL UNIQUE Constraint**

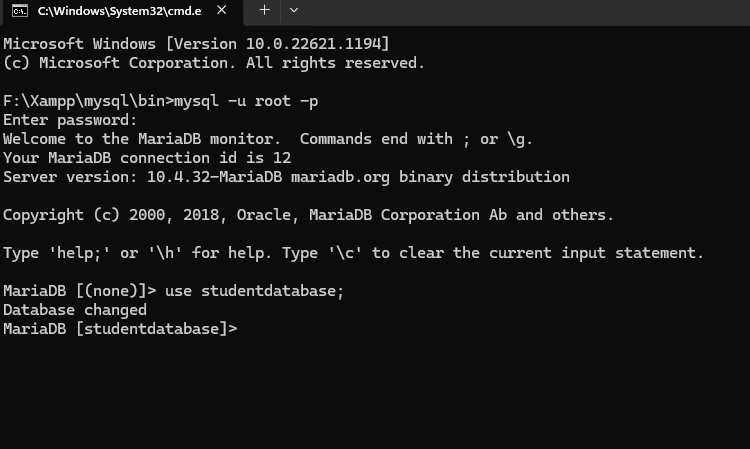
**Objective:** To comprehend the SQL UNIQUE constraint and practice implementing it to maintain data integrity within a database table.

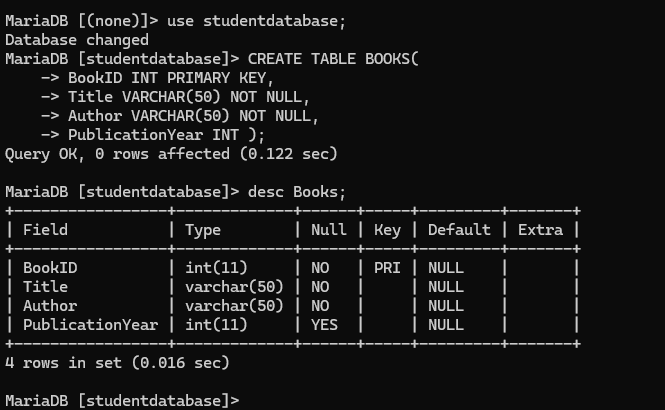
**Task:**

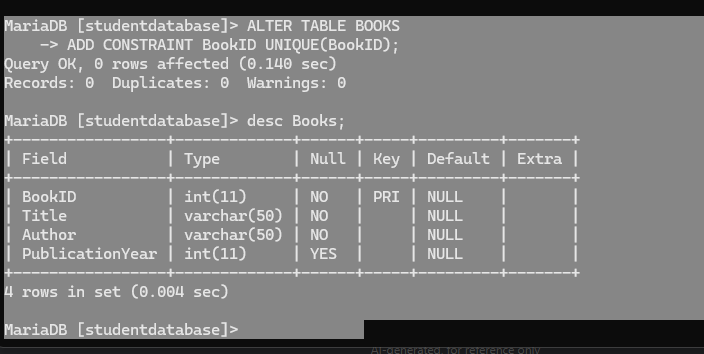
1. Open your preferred SQL client or interface (e.g., MySQL Workbench, SQLite Studio).



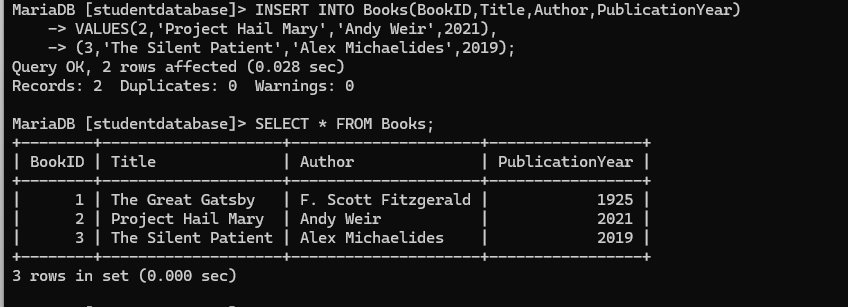
1. Utilize the existing "StudentDatabase" or create a new one for this task.



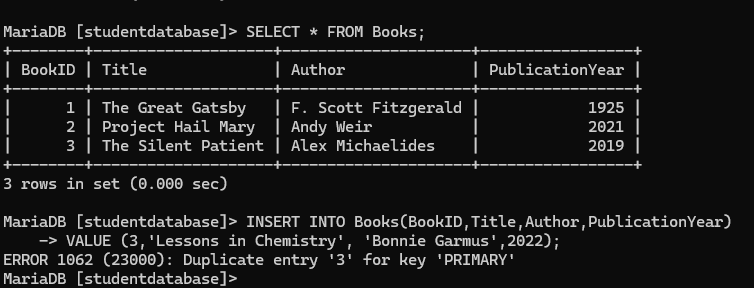
1. Create a new table named "Books" with the following columns:
   1. BookID (INT): A unique identifier for each book.
   2. Title (VARCHAR): The title of the book.
   3. Author (VARCHAR): The author of the book.
   4. PublicationYear (INT): The year the book was published.
2. Implement the SQL UNIQUE constraint on the "BookID" column to ensure each book has a distinct identifier.



1. Insert sample book records into the "Books" table, providing unique values for the "BookID" column.



1. Attempt to insert a new book record with a duplicate "BookID" and observe the outcome.



**Expected Outcome**:

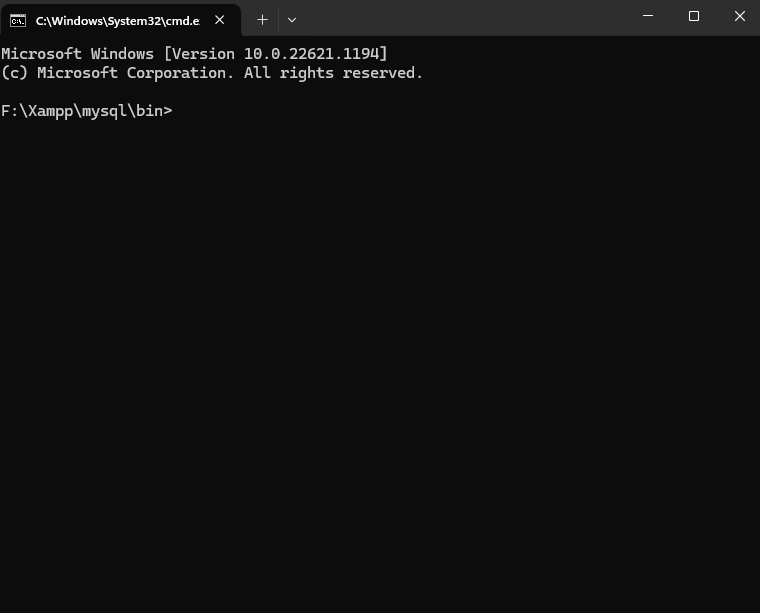
1. The "Books" table should be successfully created within the specified database.
2. The "BookID" column should have the UNIQUE constraint applied.
3. You should be able to insert sample book records into the table, adhering to the constraint's uniqueness requirement.
4. Attempting to insert a new book record with a duplicate "BookID" should result in an error due to the UNIQUE constraint.

**Lab Task: SQL DEFAULT Constraint**

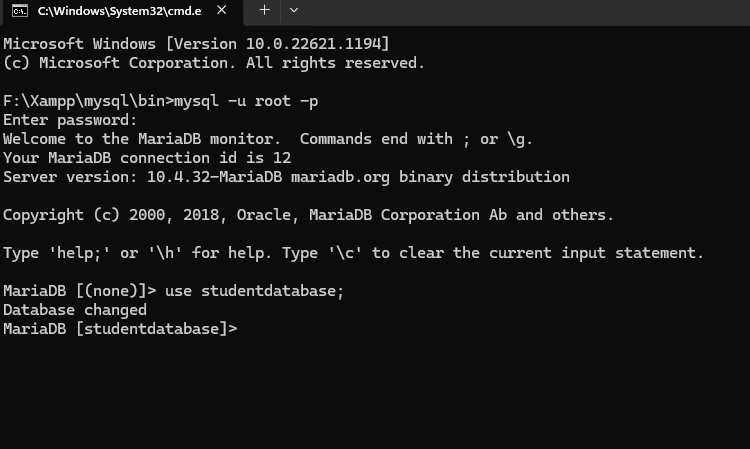
**Objective:** To comprehend the SQL DEFAULT constraint and practice implementing it to provide default values for columns during data insertion.

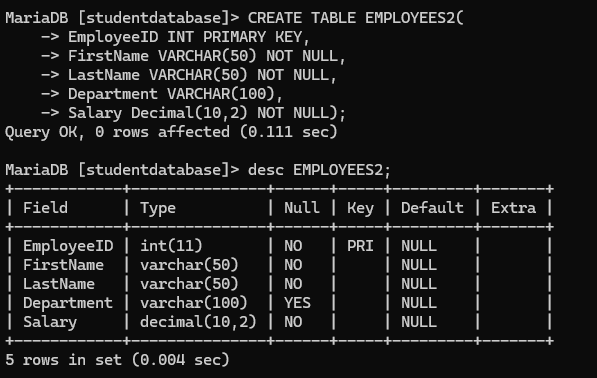
**Task:**

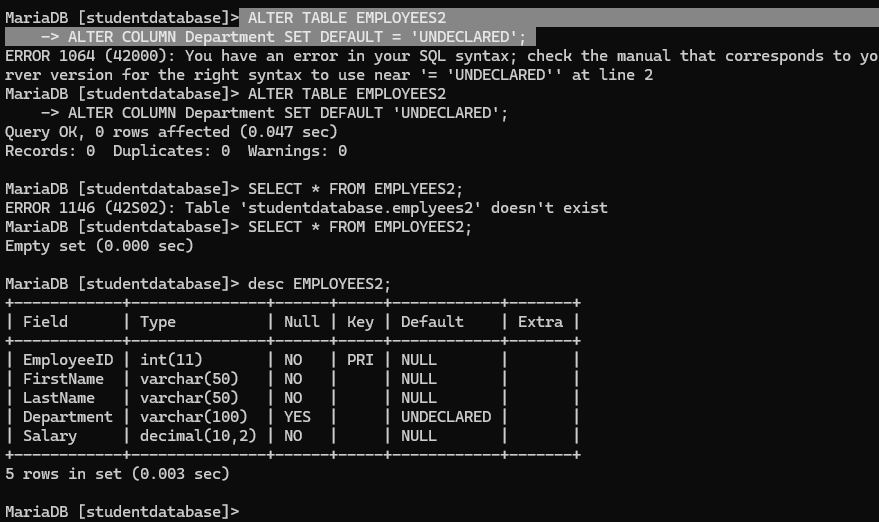
1. Open your preferred SQL client or interface (e.g., MySQL Workbench, SQLite Studio).



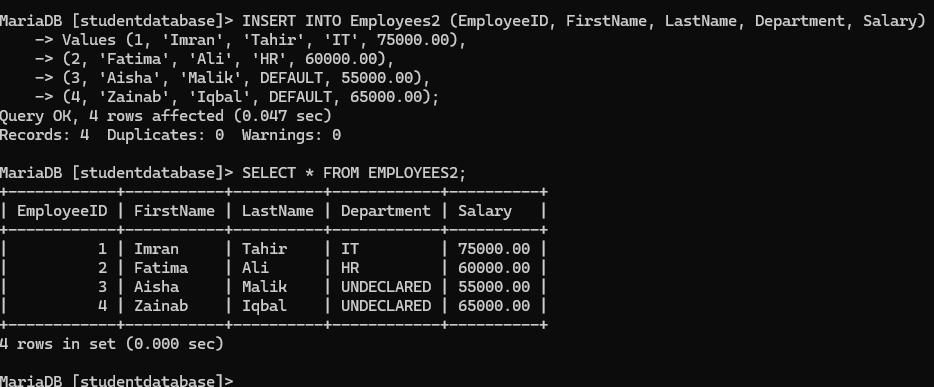
1. Utilize the existing "StudentDatabase" or create a new one for this task.



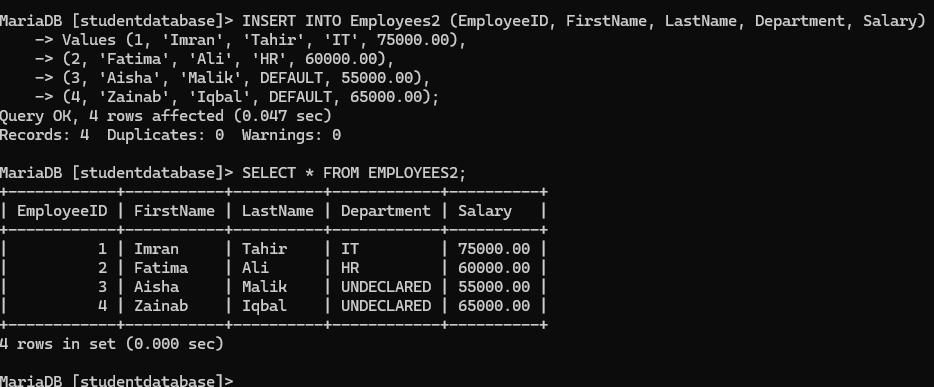
1. Create a new table named "Employees" with the following columns:
   1. EmployeeID (INT): A unique identifier for each employee.
   2. FirstName (VARCHAR): The first name of the employee.
   3. LastName (VARCHAR): The last name of the employee.
   4. Department (VARCHAR): The department the employee belongs to.
   5. Salary (DECIMAL): The salary of the employee.
2. Apply the SQL DEFAULT constraint to the "Department" column, providing a default value of 'Undeclared'.



1. Insert sample employee records into the "Employees" table, omitting the "Department" value in some cases.



1. Observe how the DEFAULT constraint applies the default value when no value is provided for the "Department" column.



**Expected Outcome:**

1. The "Employees" table should be successfully created within the specified database.
2. The "Department" column should have the DEFAULT constraint applied with a default value of 'Undeclared'.
3. You should be able to insert sample employee records into the table, allowing some records to omit the "Department" value.
4. When inserting records without specifying a "Department," the DEFAULT constraint should automatically assign the value 'Undeclared' to the column.