Unit – 5

SQL

(Structured Query Language)

Background/Introduction:

- **SQL** also known as SEQUEL, stands for **Structured Query Language** which is a database computer language for storing, manipulating and retrieving data stored in a relational database.
- SQL was developed in the 1970s by IBM Computer Scientists and became a standard of the American National Standards Institute (ANSI) in 1986, and the International Organization for Standardization (ISO) in 1987.
- SQL is the standard language to communicate with Relational Database Systems. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their Standard Database Language.

Why SQL?

SQL is widely popular because it offers the following advantages −

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

Database Languages

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, sore and update the data in the database.

Types of Database Languages:

There are mainly four types of database languages which are listed below:

1. Data Definition Language (DDL)
2. Data Manipulation Language (DML)
3. Data Control Language (DCL)
4. Transaction Control Language (TCL)

Data Definition Language (DDL):

- DDL is a set of SQL commands **used to create, modify, and delete database structures but not data**.
- These commands are normally not used by a general user, who should be accessing the database via an application.
- **Data Definition Language (DDL) commands:**
    - ✓ **CREATE** to create a new table or database.
    - ✓ **ALTER** for alteration.
    - ✓ **TRUNCATE** to delete data from the table.
    - ✓ **DROP** to drop a table.
    - ✓ **RENAME** to rename a table.

Data Manipulation Language (DML):

- DDL is a set of SQL commands **which enables insert, delete, update and retrieve operation on data from database.**
- DML statements are converted into equivalent low-level statement by DML compiler.
- **SQL supports following DML commands:**
    - ✓ **SELECT** : It is used to retrieve data from tables..
    - ✓ **INSERT** : It is used to insert data into a table.
    - ✓ **UPDATE** : It is used to update existing data withing a table.
    - ✓ **DELETE** : It is used to delete records from a table.

Data Control Language (DCL):

- DCL commands are used to retrieve the stored or saved data.
- DCL execution is transactional.
- **DCL commands:**
  - ✓ **GRANT :** It is used to give user access privileges to a database.
  - ✓ **REVOKE :** It is used to take back permission from the user.

Transaction Control Language (TCL):

- TCL commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements.
- **TCL commands:**
  - ✓ **COMMIT :** It is used to save the transaction on the database.
  - ✓ **ROLLBACK :** It is used to restore the database to original since the last commit.

Difference between Rollback and Commit Commands

Rollback	Commit
Rollback command is used to undo the changes made by the DML commands	The commit command is use to save the modifications done to the database values by the DML commands.
It rollbacks all the changes of the current transaction.	It will make all the changes permanent that cannot be rolled back.
Syntax: DELETE FROM table_name ROLLBACK	**Syntax**: COMMIT:

Domain Types in SQL

- Char
- Varchar
- Int
- Decimal
- Date
- Text

Schema Definition in SQL:

- A schema is a collection of database objects like tables, triggers, stored procedures, etc.
- A schema is connected with a user which is known as the schema owner.
- Database may have one or more schema. SQL Server have some built-in schema, for example: **dbo, guest, sys, and INFORMATION_SCHEMA**.

Shadow Copy Scheme :

In the shadow-copy scheme, a transaction that wants to update the database first creates a complete copy of the database. All updates are done on the new database copy, leaving the original copy, the shadow copy, untouched. If at any point the transaction has to be aborted, the system merely deletes the new copy.

Joins in SQL

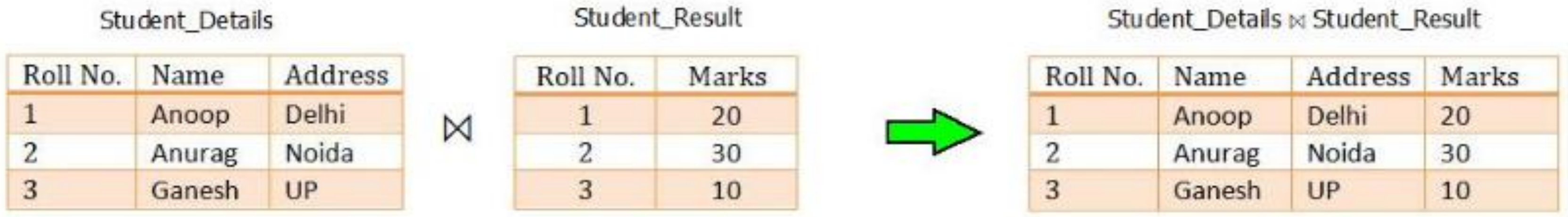
- A JOIN clause is used to combine multiple tables present in a database on the basis of some common attributes present in those tables.
- The purpose of JOINs in SQL is to access data from multiple tables based on logical relationships between them. JOINS are used to fetch data from database tables and represent the result dataset as a separate table.

Syntax : SELECT Column_Name From Table_Name1 NATURAL JOIN Table_Name2;

For Example : Consider two tables student_details and student_result.

Query : SELECT * From student_details NATURAL JOIN student_result;

Output



Student_Details

Roll No.	Name	Address
1	Anoop	Delhi
2	Anurag	Noida
3	Ganesh	UP

⋈

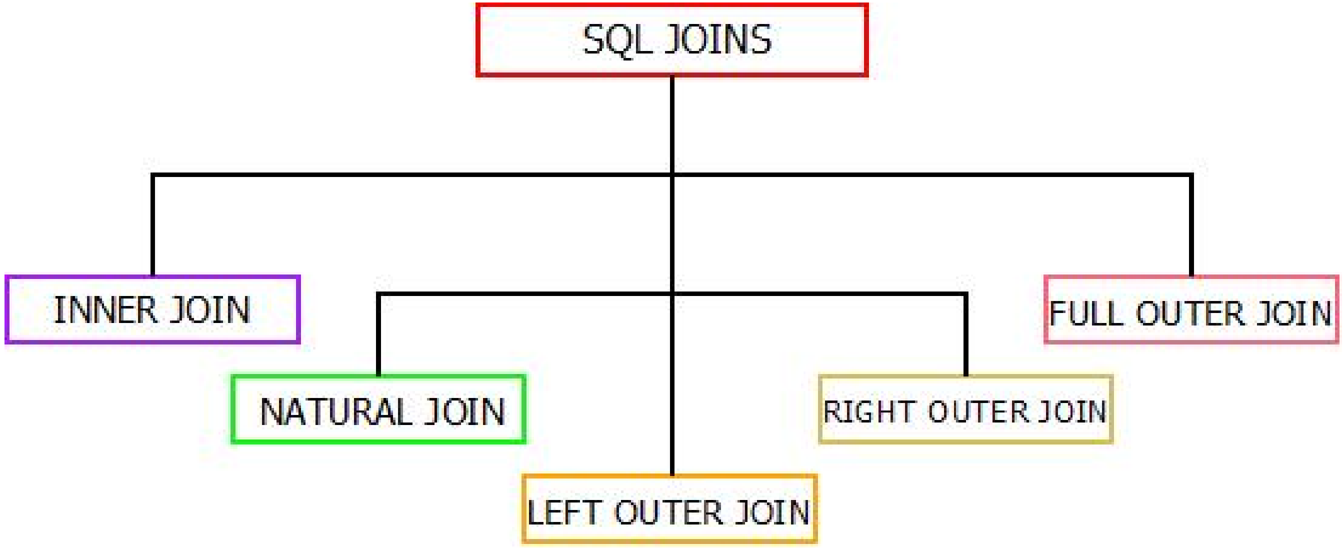
Student_Result

Roll No.	Marks
1	20
2	30
3	10

Student_Details ⋈ Student_Result

Roll No.	Name	Address	Marks
1	Anoop	Delhi	20
2	Anurag	Noida	30
3	Ganesh	UP	10

Types of Joins in SQL



SQL Joins : Types

SQL Basic Queries #1

- path : xampp/mysql/bin
- mysql -u root -p
- password:

- show databases; → To display the database list
- create database database_name; → To create new database
- use database_name; → To use the database
- show tables; → To display all the availabe tables
- desc/describe table_name; → To see the structure(columns) of the table
- drop database database_name; → To delete the database

SQL Basic Queries #2

- To create table:

Syntax:

```
CREATE TABLE table_name(column1 datatype(size) contraints,column2
        datatype(size) contraints, ..);
```

```
Create table student(
        id int(10),
        name varchar(15),
        address varchar(40)
        );
```

SQL Basic Queries #2

- To create table:

    Syntax:

    CREATE TABLE table_name(column1 datatype(size) contraints,column2 datatype(size) contraints, ..);

- To Insert data in table:

    Syntax:

    INSERT INTO table_name(columns...) values (value1,value2,..);

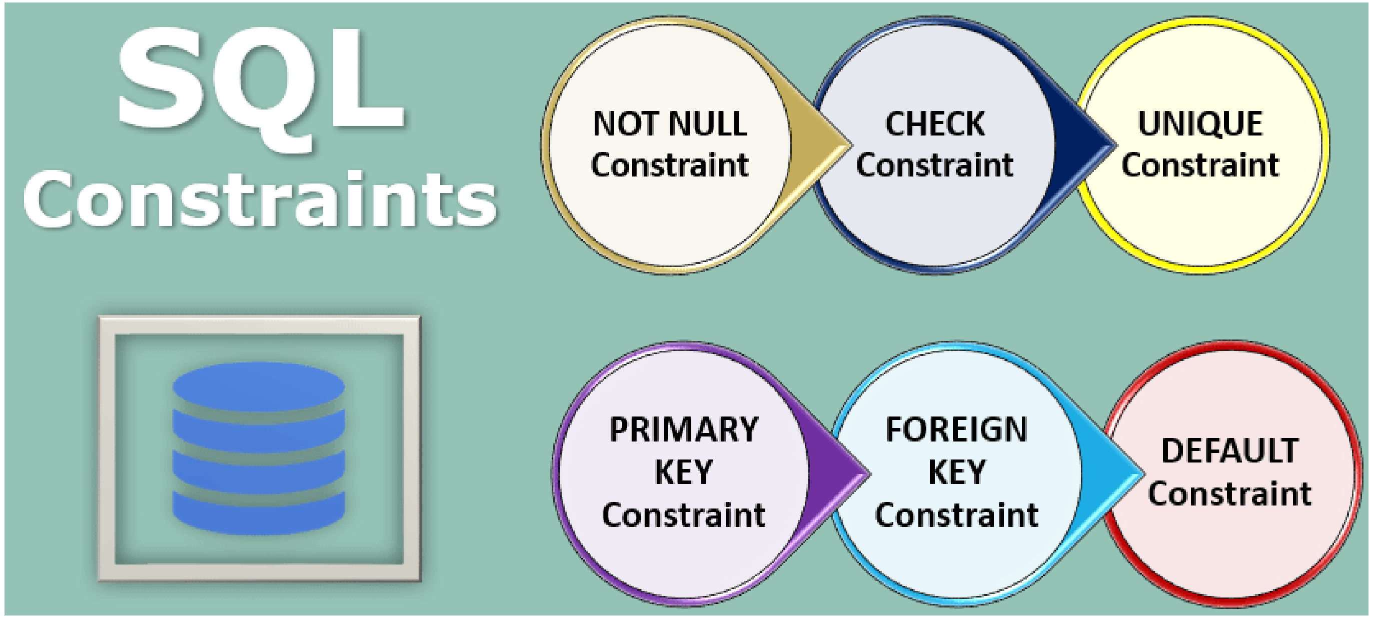
- To view the inserted data of the table:

    Syntax:

    SELECT *FROM table_name;

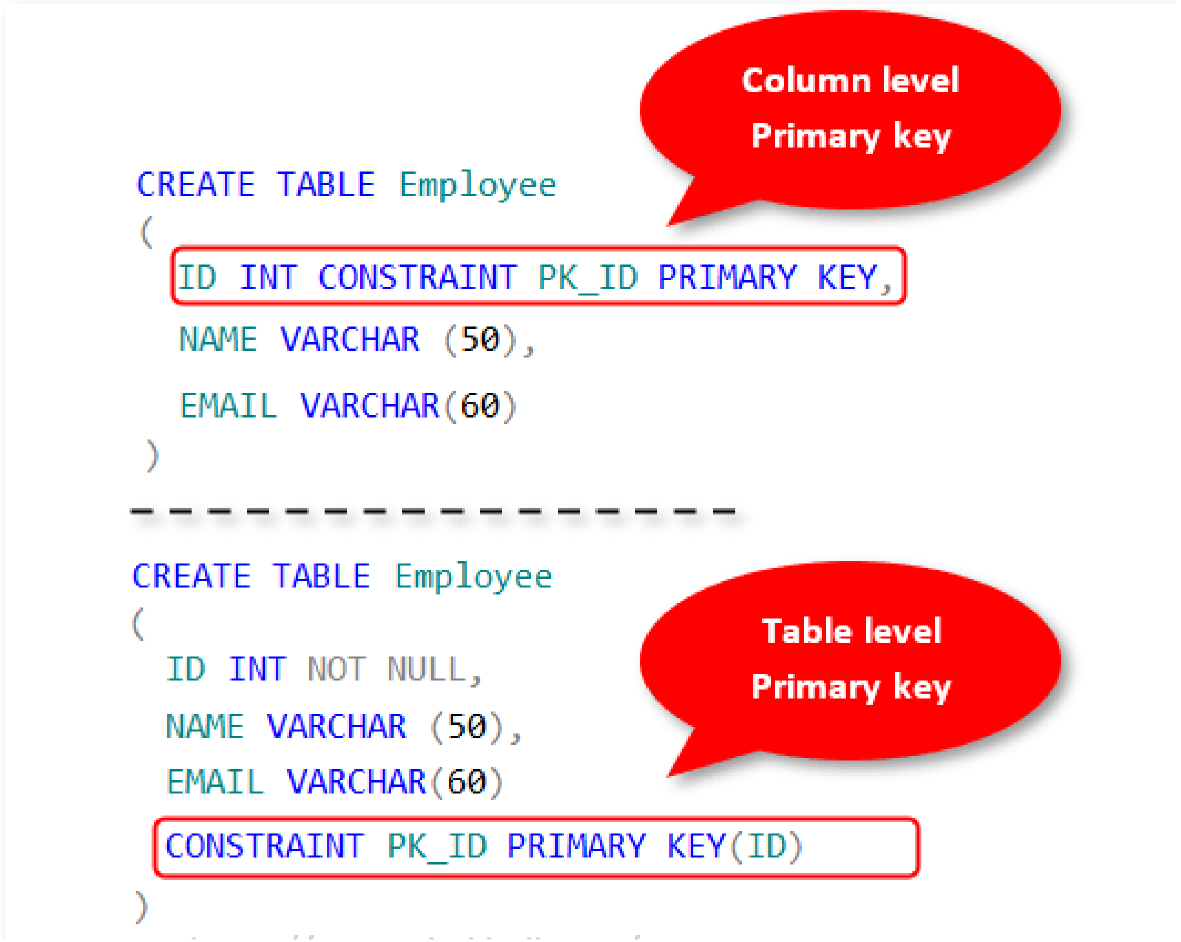
SQL Constraints

- ✓ Constraints are used to limit the type of data that can go into a table.
- ✓ Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).
- ✓ Here are the most important constraints:



PRIMARY KEY :

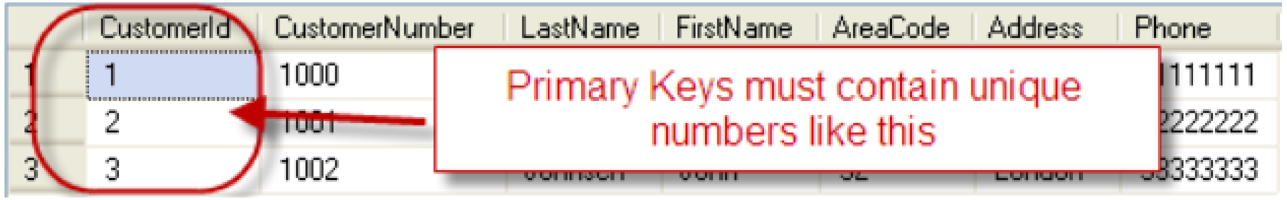
The PRIMARY KEY constraint uniquely identifies each record in a database table



```
CREATE TABLE Employee
(
  ID INT CONSTRAINT PK_ID PRIMARY KEY,
  NAME VARCHAR (50),
  EMAIL VARCHAR(60)
)
-----------------
CREATE TABLE Employee
(
  ID INT NOT NULL,
  NAME VARCHAR (50),
  EMAIL VARCHAR(60)
  CONSTRAINT PK_ID PRIMARY KEY(ID)
)
```

```
CREATE TABLE [CUSTOMER]
(
        CustomerId int IDENTITY(1,1) PRIMARY KEY,
        CustomerNumber int NOT NULL UNIQUE,
        LastName varchar(50) NOT NULL,
        FirstName varchar(50) NOT NULL,
        AreaCode int NULL,
        Address varchar(50) NULL,
        Phone varchar(50) NULL,
)
GO
```

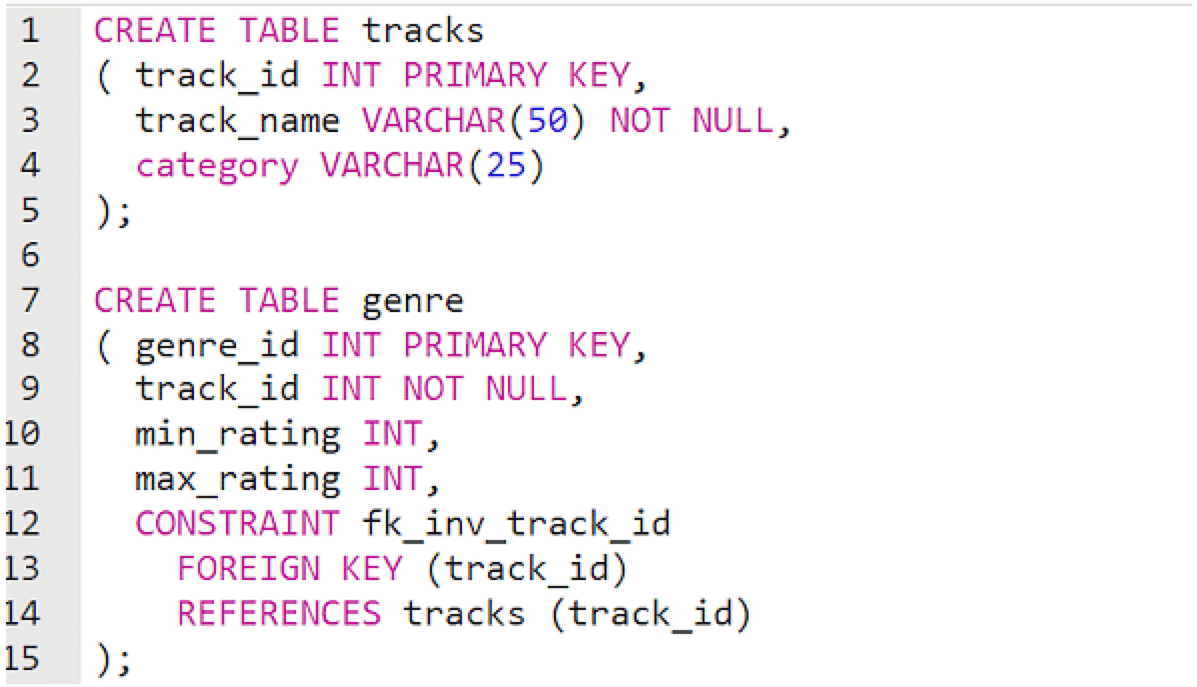
As you see we use the "Primary Key" keyword to specify that a column should be the Primary Key.



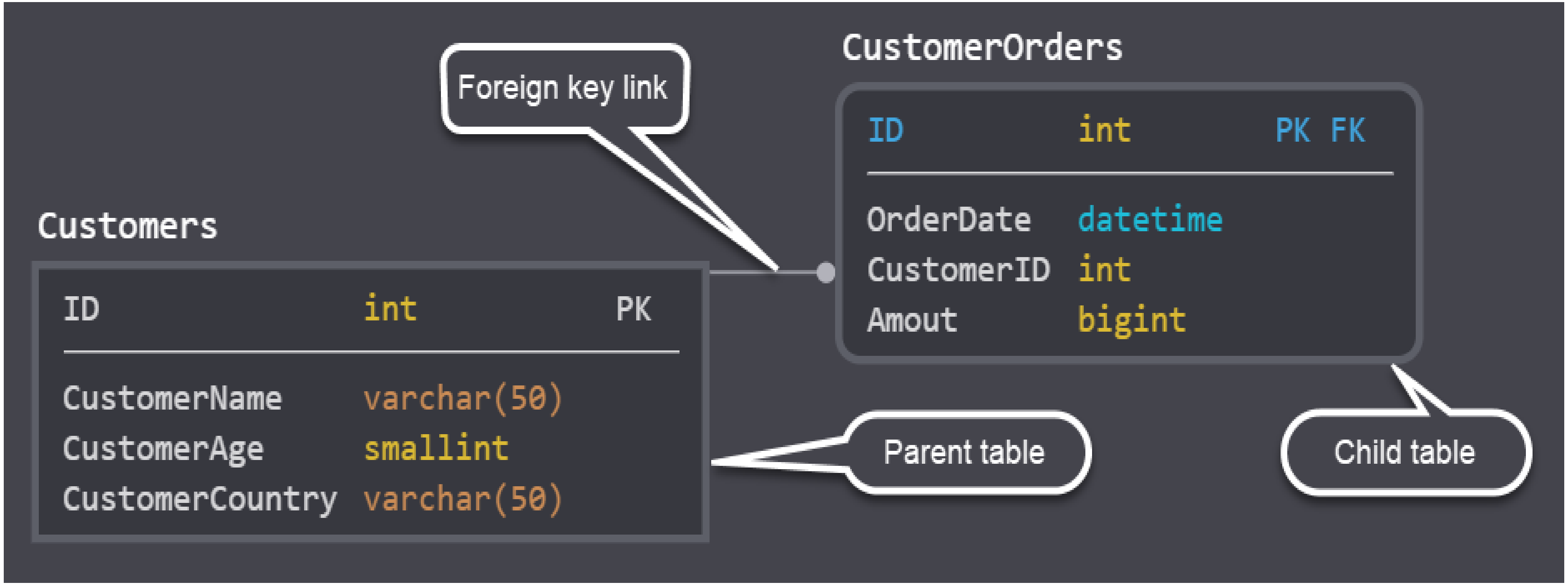
	CustomerId	CustomerNumber	LastName	FirstName	AreaCode	Address	Phone
1	1	1000					1111111
2	2	1001					2222222
3	3	1002	Johnsen	John	32	London	3333333

FOREIGN KEY :

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.



```
CREATE TABLE tracks
( track_id INT PRIMARY KEY,
  track_name VARCHAR(50) NOT NULL,
  category VARCHAR(25)
);

CREATE TABLE genre
( genre_id INT PRIMARY KEY,
  track_id INT NOT NULL,
  min_rating INT,
  max_rating INT,
  CONSTRAINT fk_inv_track_id
    FOREIGN KEY (track_id)
    REFERENCES tracks (track_id)
);
```


CustomerOrders
Foreign key link
ID int PK FK
OrderDate datetime
CustomerID int
Amout bigint
Customers
ID int PK
CustomerName varchar(50)
CustomerAge smallint
CustomerCountry varchar(50)
Parent table
Child table

SCHOOL:

```
CREATE TABLE SCHOOL
(
	SchoolId int IDENTITY(1,1) PRIMARY KEY,
	SchoolName varchar(50) NOT NULL UNIQUE,
	Description varchar(1000) NULL,
	Address varchar(50) NULL,
	Phone varchar(50) NULL,
	PostCode varchar(50) NULL,
	PostAddress varchar(50) NULL,
)
GO
```

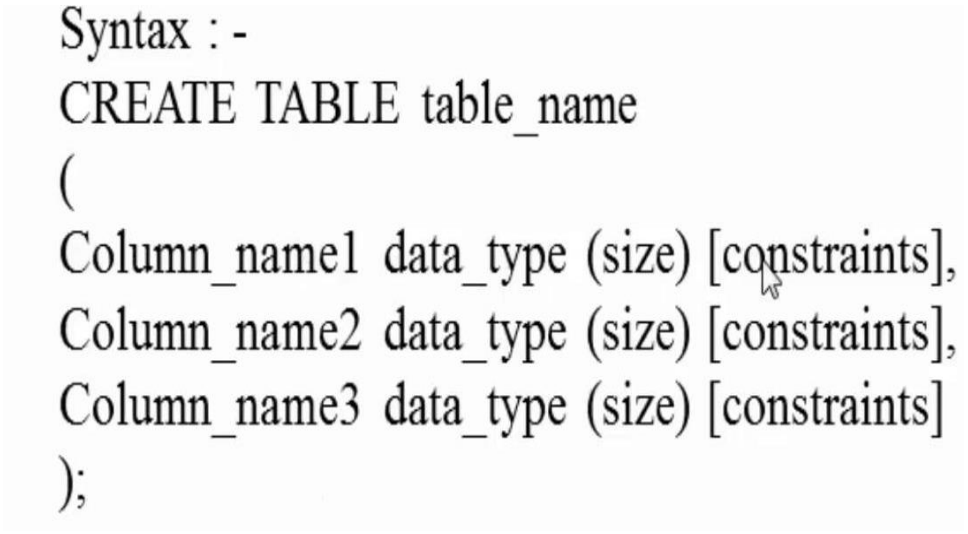
CLASS:

```
CREATE TABLE CLASS
(
	ClassId int IDENTITY(1,1) PRIMARY KEY,
	SchoolId int NOT NULL FOREIGN KEY REFERENCES SCHOOL (SchoolId),
	ClassName varchar(50) NOT NULL UNIQUE,
	Description varchar(1000) NULL,
)
GO
```

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

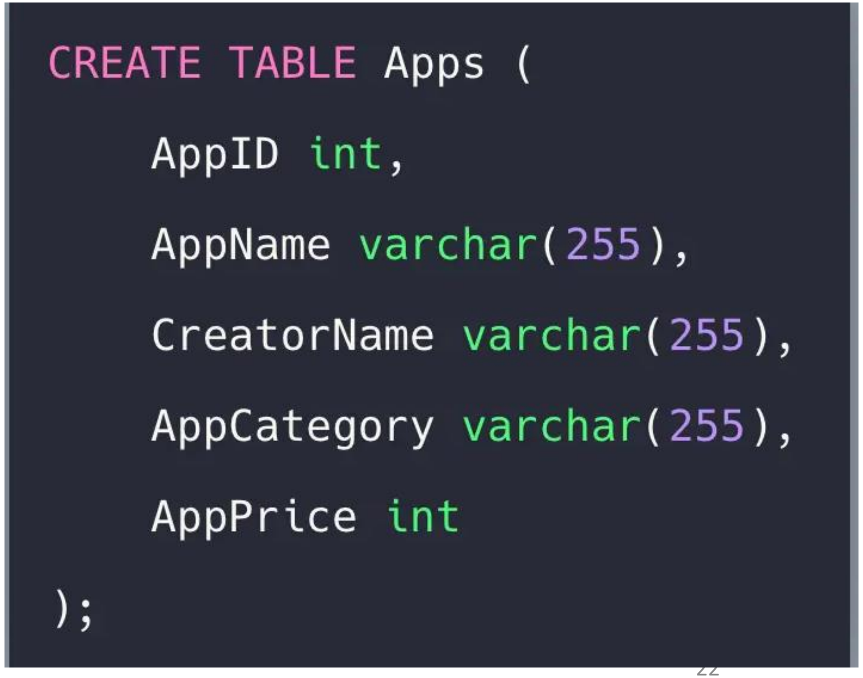
The FOREIGN KEY constraint also prevents that invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

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



Syntax : -

```
CREATE TABLE table_name
(
Column_name1 data_type (size) [constraints],
Column_name2 data_type (size) [constraints],
Column_name3 data_type (size) [constraints]
);
```



```
CREATE TABLE Apps (
    AppID int,
    AppName varchar(255),
    CreatorName varchar(255),
    AppCategory varchar(255),
    AppPrice int
);
```

The data type specifies what type of data the column can hold.

ALTER command:

Alter command can perform following task:

- Add column/s
- Remove column/s
- Modify data type
- Modify datatype length
- Add constraints
- Remove constraints
- Rename column/table

```
Create table employee
(
        id int,
        name varchar(10)
);
```

Now, we can make changes in the above table using ALTER command as follows:

- ALTER table employee add address varchar(10); **To add etc . column**
- ALTER table employee drop column address; **To delete the column**
- ALTER table employee modify id varchar(30); **To change the data type**
- ALTER table employee change column id eid int; **To rename the column for older version**

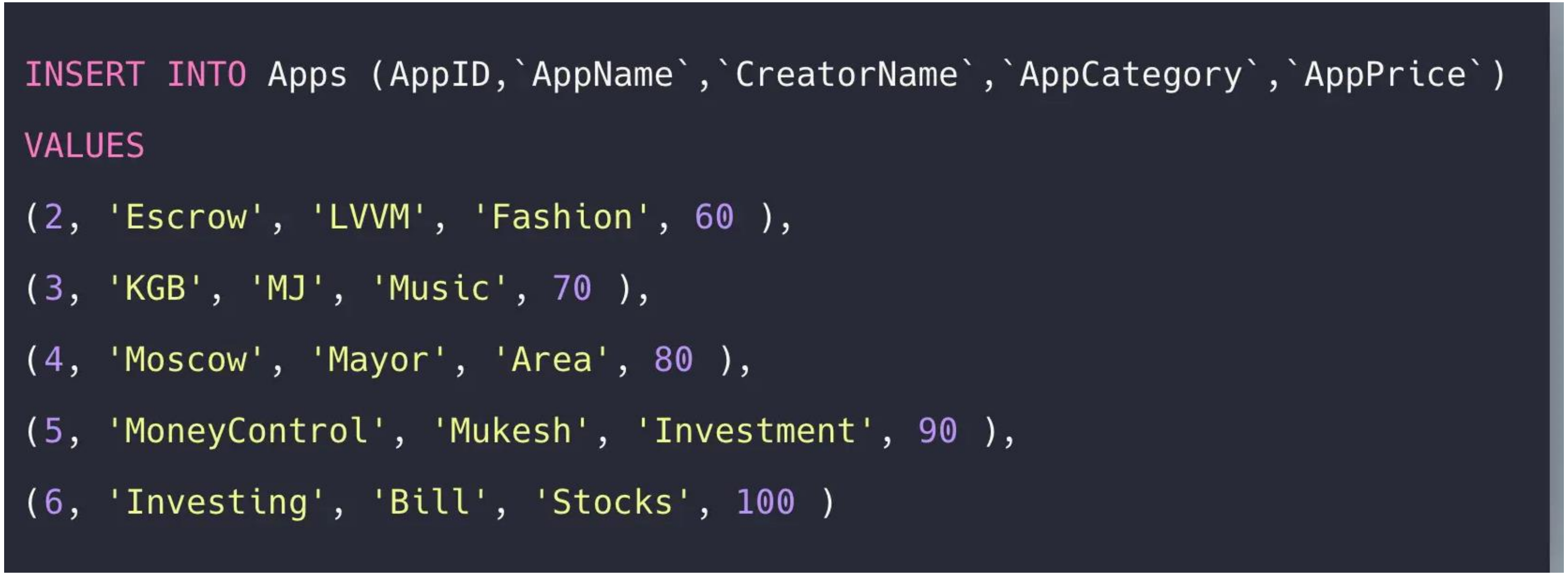
(ALTER table employee rename column id to eid; will work for higher version of mariadb)

- ALTER table employee rename to emp; **To rename the table name**
- ALTER table employee add primary key (roll_no); **To add constraints**

INSERT Command:

Syntax:

INSERT INTO table_name (column1, column2, column3,etc) VALUES (value1, value2, value3, etc);



```sql
INSERT INTO Apps (AppID,`AppName`,`CreatorName`,`AppCategory`,`AppPrice`)
VALUES
(2, 'Escrow', 'LVVM', 'Fashion', 60 ),
(3, 'KGB', 'MJ', 'Music', 70 ),
(4, 'Moscow', 'Mayor', 'Area', 80 ),
(5, 'MoneyControl', 'Mukesh', 'Investment', 90 ),
(6, 'Investing', 'Bill', 'Stocks', 100 )
```

SELECT

The SELECT statement is used to select data from a database and retrieve the information.

1. **Select all columns from the table**

   Syntax: - SELECT * FROM table_name;
   Ex: - SELECT * FROM my_tab;

Name	Roll	Mobile
Rahul	01	8753
John	02	6534

2. **Select Particular columns from the table**

```
Syntax: - SELECT column_name1, column_name1,….
                    FROM table_name;
Ex: - SELECT name, mobile
           FROM my_tab;
```

DDL & DML

Commands With Examples

DDL (Data Definition Language)

- It is used to define database structure.
- It work on database and table.
- It is used by DBA (Database Administrator).
- Some DDL commands :

  CREATE

  ALTER

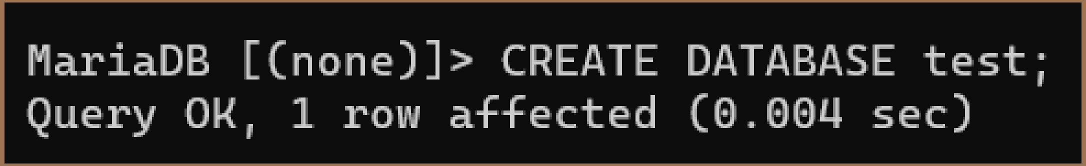
  TRUNCATE

  DROP

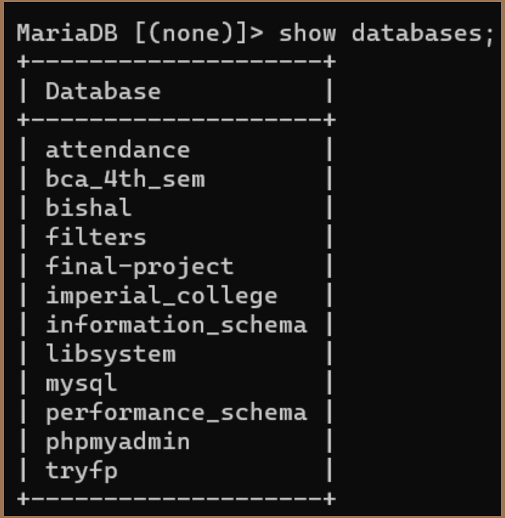
  RENAME

CREATE

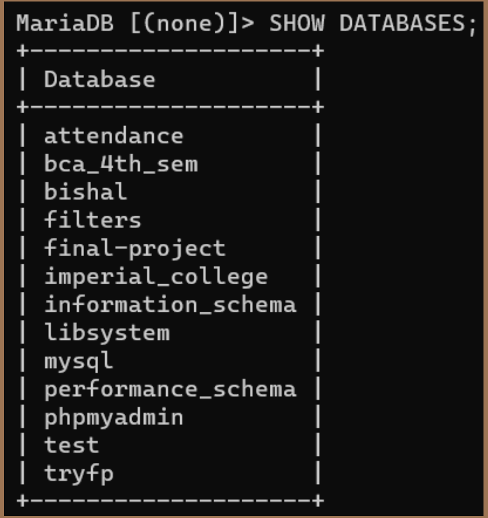
- It is use to create database and table.
- Syntax:
  CREATE DATABASE DATABASE_NAME;



```
MariaDB [(none)]> CREATE DATABASE test;
Query OK, 1 row affected (0.004 sec)
```



```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| attendance         |
| bca_4th_sem        |
| bishal             |
| filters            |
| final-project      |
| imperial_college   |
| information_schema |
| libsystem          |
| mysql              |
| performance_schema |
| phpmyadmin         |
| tryfp              |
+--------------------+
```

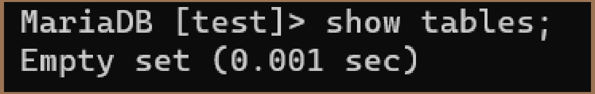


```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| attendance         |
| bca_4th_sem        |
| bishal             |
| filters            |
| final-project      |
| imperial_college   |
| information_schema |
| libsystem          |
| mysql              |
| performance_schema |
| phpmyadmin         |
| test               |
| tryfp              |
+--------------------+
```

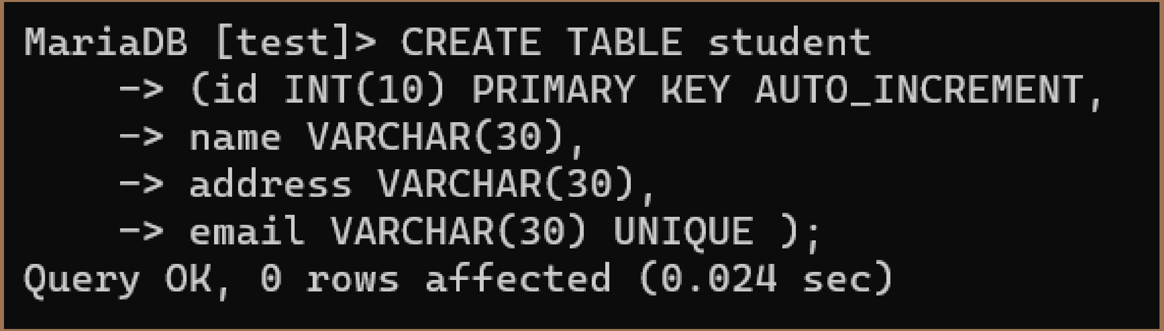
CREATE

- Syntax

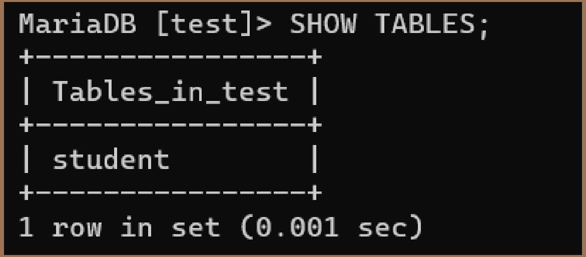
  CRATE TABLE TABLE_NAME
  (Column 1 Data_type(size) constraint,
  Column 2 Data_type(size) constraint,
  Column 3 Data_type(size) constraint);



```
MariaDB [test]> show tables;
Empty set (0.001 sec)
```



```
MariaDB [test]> CREATE TABLE student
    -> (id INT(10) PRIMARY KEY AUTO_INCREMENT,
    -> name VARCHAR(30),
    -> address VARCHAR(30),
    -> email VARCHAR(30) UNIQUE );
Query OK, 0 rows affected (0.024 sec)
```



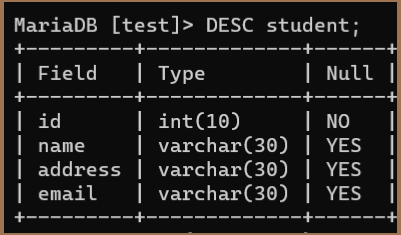
```
MariaDB [test]> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| student        |
+----------------+
1 row in set (0.001 sec)
```

ALTER

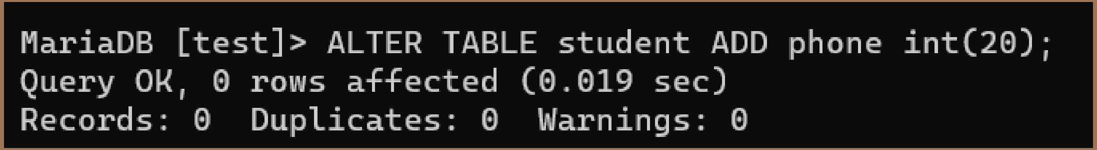
- It is use to modify existing database objects.

Add Column

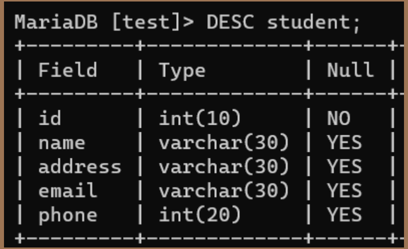
- Syntax
  ALTER TABLE TABLE_NAME ADD COLUMN_NAME DATATYPE(SIZE);



```
MariaDB [test]> DESC student;
+---------+-------------+------+
| Field   | Type        | Null |
+---------+-------------+------+
| id      | int(10)     | NO   |
| name    | varchar(30) | YES  |
| address | varchar(30) | YES  |
| email   | varchar(30) | YES  |
+---------+-------------+------+
```



```
MariaDB [test]> ALTER TABLE student ADD phone int(20);
Query OK, 0 rows affected (0.019 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

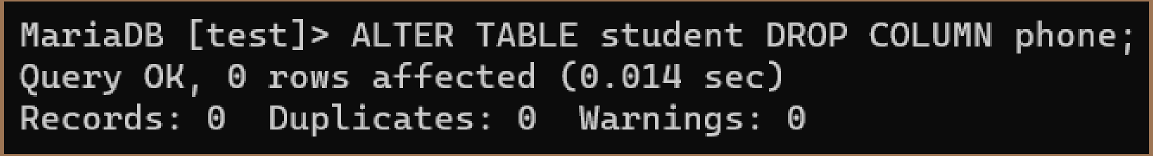


```
MariaDB [test]> DESC student;
+---------+-------------+------+
| Field   | Type        | Null |
+---------+-------------+------+
| id      | int(10)     | NO   |
| name    | varchar(30) | YES  |
| address | varchar(30) | YES  |
| email   | varchar(30) | YES  |
| phone   | int(20)     | YES  |
+---------+-------------+------+
```

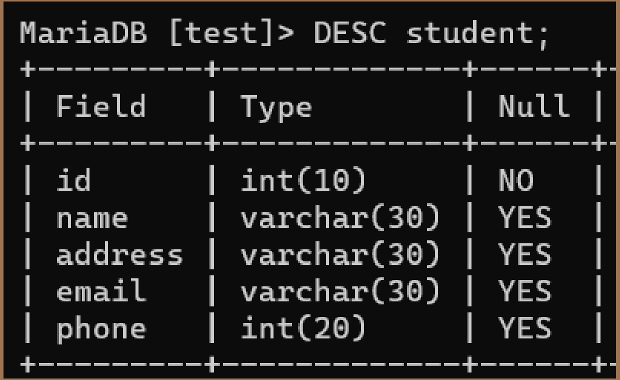
Drop Column

- Syntax

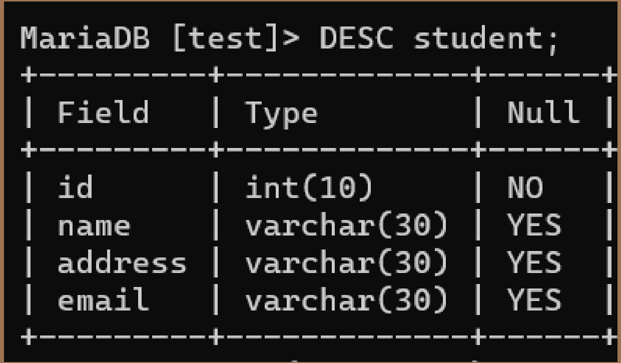
  ALTER TABLE TABLE_NAME DROP COLUMN COLUMN_NAME;



```
MariaDB [test]> ALTER TABLE student DROP COLUMN phone;
Query OK, 0 rows affected (0.014 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



```
MariaDB [test]> DESC student;
+---------+-------------+------+-
| Field   | Type        | Null |
+---------+-------------+------+-
| id      | int(10)     | NO   |
| name    | varchar(30) | YES  |
| address | varchar(30) | YES  |
| email   | varchar(30) | YES  |
| phone   | int(20)     | YES  |
+---------+-------------+------+-
```

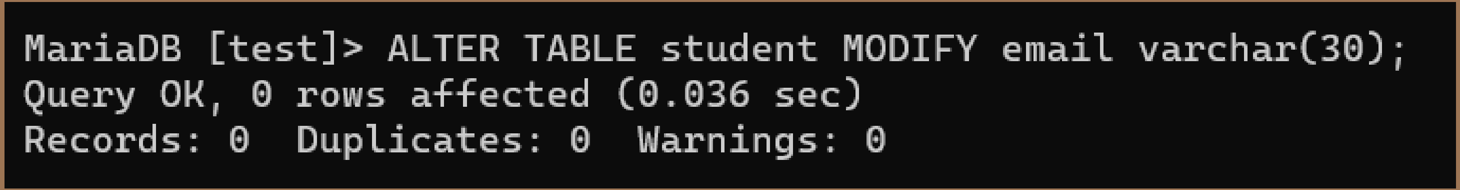


```
MariaDB [test]> DESC student;
+---------+-------------+------+
| Field   | Type        | Null |
+---------+-------------+------+
| id      | int(10)     | NO   |
| name    | varchar(30) | YES  |
| address | varchar(30) | YES  |
| email   | varchar(30) | YES  |
+---------+-------------+------+
```

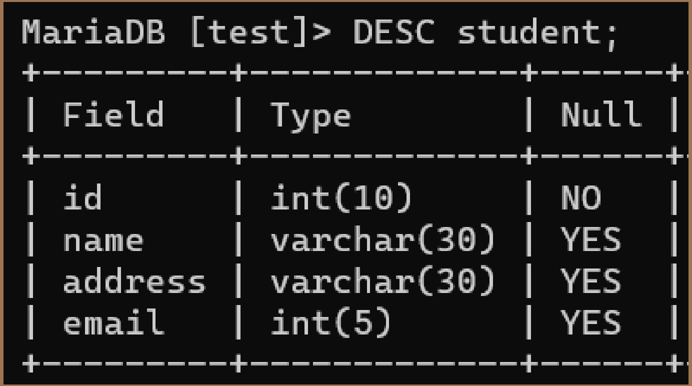
Modify Column

- Syntax

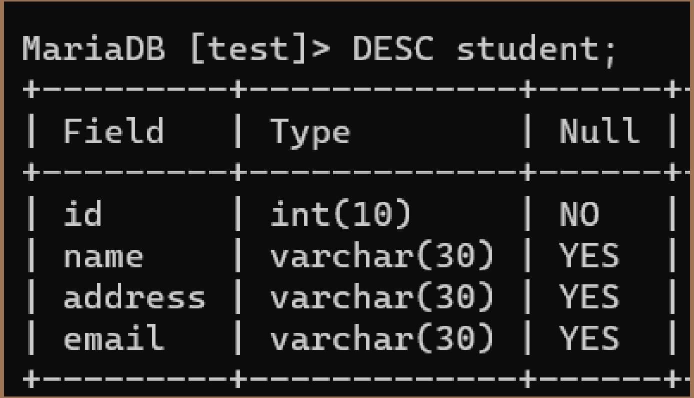
  ALTER TABLE TABLE_NAME MODIFY COLUMN_NAME DATATYPE(SIZE);



```
MariaDB [test]> ALTER TABLE student MODIFY email varchar(30);
Query OK, 0 rows affected (0.036 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



```
MariaDB [test]> DESC student;
+---------+-------------+------+
| Field   | Type        | Null |
+---------+-------------+------+
| id      | int(10)     | NO   |
| name    | varchar(30) | YES  |
| address | varchar(30) | YES  |
| email   | int(5)      | YES  |
+---------+-------------+------+
```

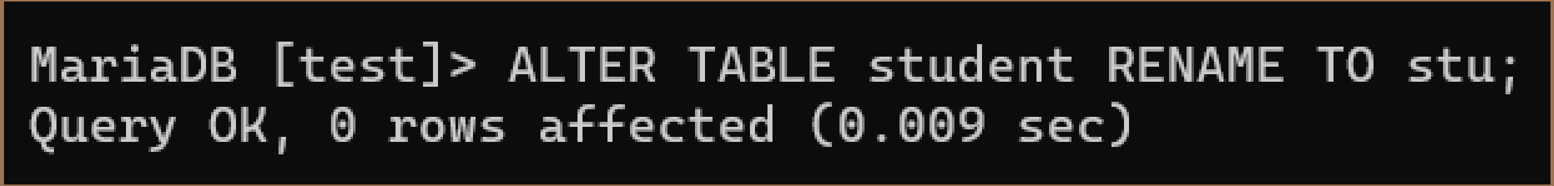


```
MariaDB [test]> DESC student;
+---------+-------------+------+
| Field   | Type        | Null |
+---------+-------------+------+
| id      | int(10)     | NO   |
| name    | varchar(30) | YES  |
| address | varchar(30) | YES  |
| email   | varchar(30) | YES  |
+---------+-------------+------+
```

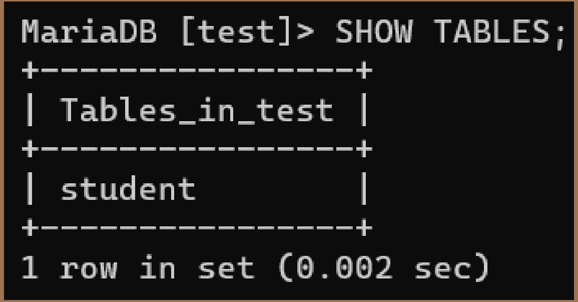
Rename Table Name

- Syntax

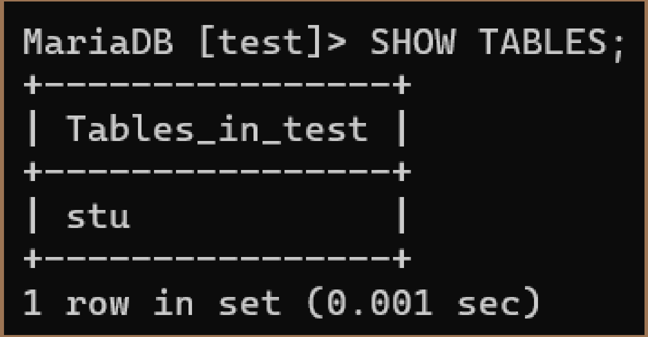
  ALTER TABLE TABLE_NAME(OLD) RENAME TO TABLE_NAME(NEW);



```
MariaDB [test]> ALTER TABLE student RENAME TO stu;
Query OK, 0 rows affected (0.009 sec)
```



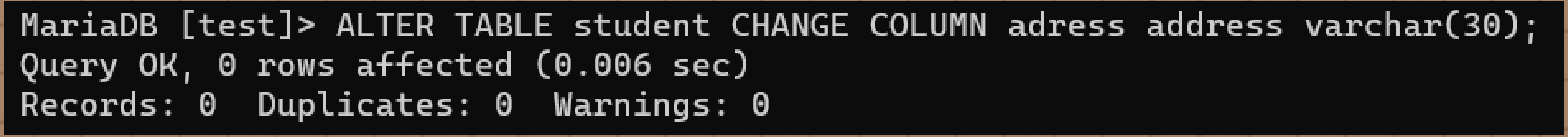
```
MariaDB [test]> SHOW TABLES;
+-----------------+
| Tables_in_test |
+-----------------+
| student         |
+-----------------+
1 row in set (0.002 sec)
```



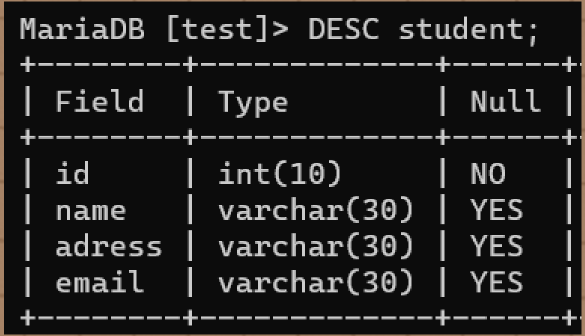
```
MariaDB [test]> SHOW TABLES;
+-----------------+
| Tables_in_test |
+-----------------+
| stu             |
+-----------------+
1 row in set (0.001 sec)
```

Rename Column Name

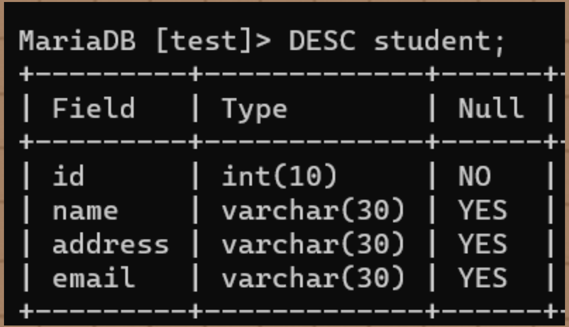
- Syntax
  ALTER TABLE TABLE_NAME CHANGE COLUMN COLUMN_NAME(OLD) COLUMN_NAME(NEW) DATATYPE(SIZE);



```
MariaDB [test]> ALTER TABLE student CHANGE COLUMN adress address varchar(30);
Query OK, 0 rows affected (0.006 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



```
MariaDB [test]> DESC student;
+--------+-------------+------+
| Field  | Type        | Null |
+--------+-------------+------+
| id     | int(10)     | NO   |
| name   | varchar(30) | YES  |
| adress | varchar(30) | YES  |
| email  | varchar(30) | YES  |
+--------+-------------+------+
```



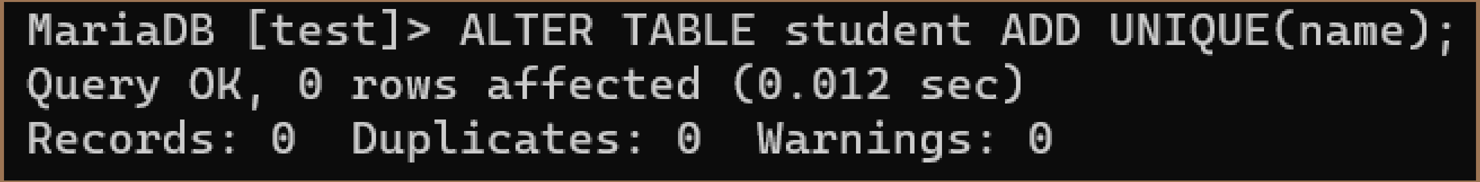
```
MariaDB [test]> DESC student;
+---------+-------------+------+
| Field   | Type        | Null |
+---------+-------------+------+
| id      | int(10)     | NO   |
| name    | varchar(30) | YES  |
| address | varchar(30) | YES  |
| email   | varchar(30) | YES  |
+---------+-------------+------+
```

ALTER TABLE TABLE_NAME RENAME COLUMN COLUMN_NAME(OLD) TO COLUMN_NAME(NEW); (Work only on higher version of mariadb)

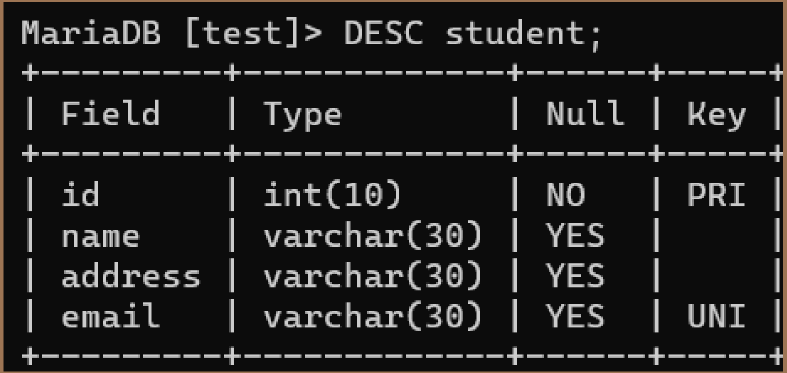
Add Constraint

- Syntax

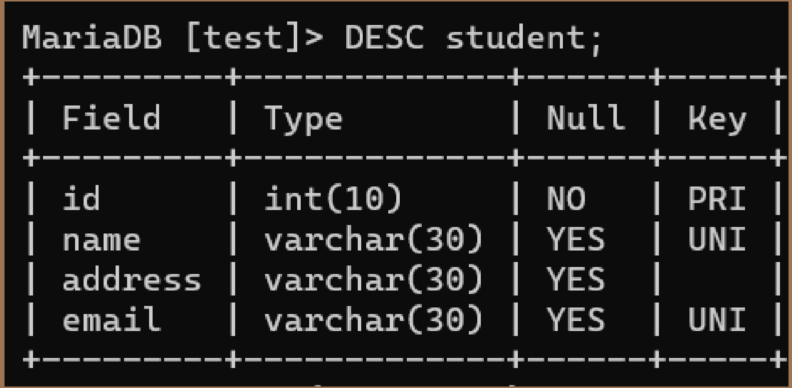
  ALTER TABLE TABLE_NAME ADD CONSTRAINT_NAME (COLUMN_NAME);



```
MariaDB [test]> ALTER TABLE student ADD UNIQUE(name);
Query OK, 0 rows affected (0.012 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



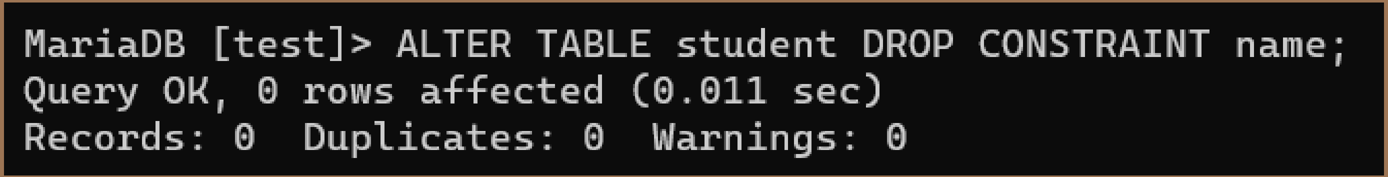
```
MariaDB [test]> DESC student;
+---------+-------------+------+-----+
| Field   | Type        | Null | Key |
+---------+-------------+------+-----+
| id      | int(10)     | NO   | PRI |
| name    | varchar(30) | YES  |     |
| address | varchar(30) | YES  |     |
| email   | varchar(30) | YES  | UNI |
+---------+-------------+------+-----+
```



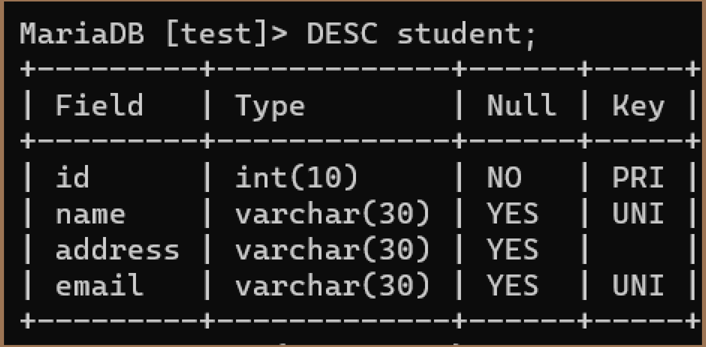
```
MariaDB [test]> DESC student;
+---------+-------------+------+-----+
| Field   | Type        | Null | Key |
+---------+-------------+------+-----+
| id      | int(10)     | NO   | PRI |
| name    | varchar(30) | YES  | UNI |
| address | varchar(30) | YES  |     |
| email   | varchar(30) | YES  | UNI |
+---------+-------------+------+-----+
```

Drop Constraint

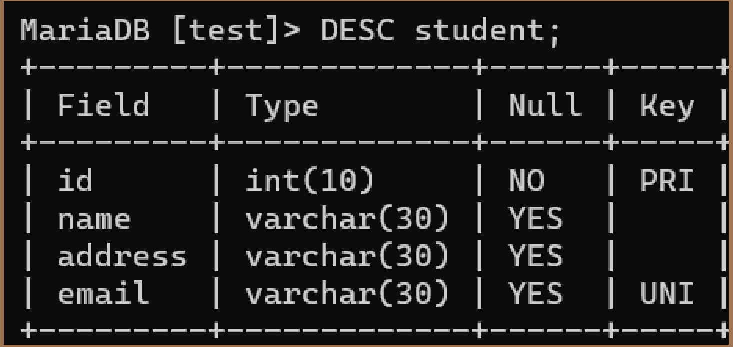
- Syntax
  ALTER TABLE TABLE_NAME DROP CONSTRAINT COLUMN_NAME;



```
MariaDB [test]> ALTER TABLE student DROP CONSTRAINT name;
Query OK, 0 rows affected (0.011 sec)
Records: 0  Duplicates: 0  Warnings: 0
```



```
MariaDB [test]> DESC student;
+---------+-------------+------+-----+
| Field   | Type        | Null | Key |
+---------+-------------+------+-----+
| id      | int(10)     | NO   | PRI |
| name    | varchar(30) | YES  | UNI |
| address | varchar(30) | YES  |     |
| email   | varchar(30) | YES  | UNI |
+---------+-------------+------+-----+
```

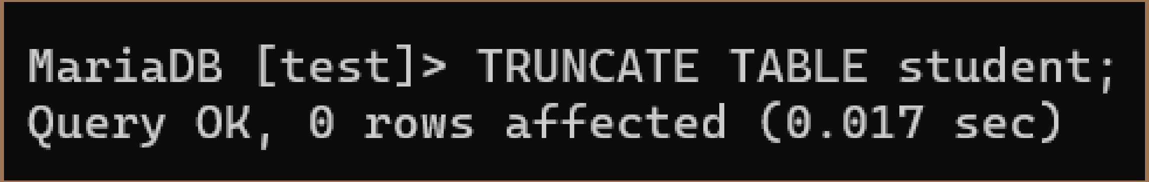


```
MariaDB [test]> DESC student;
+---------+-------------+------+-----+
| Field   | Type        | Null | Key |
+---------+-------------+------+-----+
| id      | int(10)     | NO   | PRI |
| name    | varchar(30) | YES  |     |
| address | varchar(30) | YES  |     |
| email   | varchar(30) | YES  | UNI |
+---------+-------------+------+-----+
```

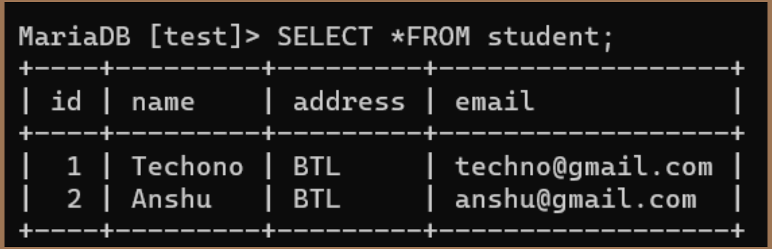
TRUNCATE

- It is use to quickly remove all rows from a table.
- Syntax:

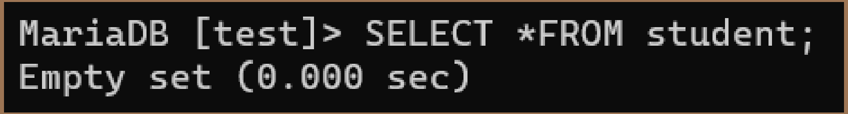
  TRUNCATE TABLE TABLE_NAME;



```
MariaDB [test]> TRUNCATE TABLE student;
Query OK, 0 rows affected (0.017 sec)
```



```
MariaDB [test]> SELECT *FROM student;
+----+---------+---------+-----------------+
| id | name    | address | email           |
+----+---------+---------+-----------------+
|  1 | Techono | BTL     | techno@gmail.com |
|  2 | Anshu   | BTL     | anshu@gmail.com |
+----+---------+---------+-----------------+
```

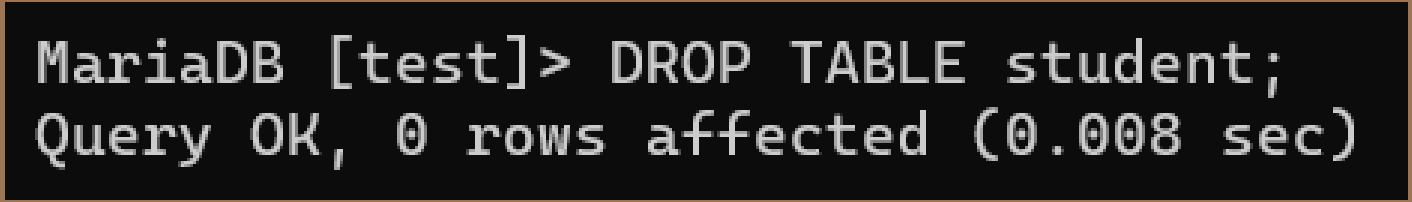


```
MariaDB [test]> SELECT *FROM student;
Empty set (0.000 sec)
```

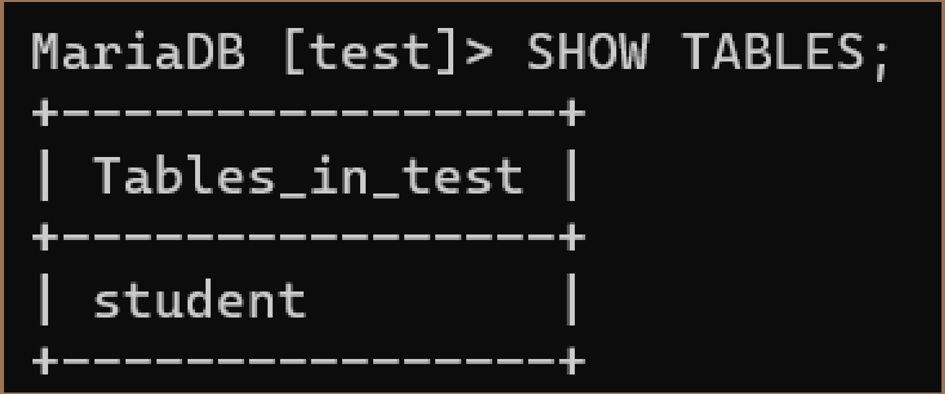
DROP

- It is use to delete database and tables.
- Syntax:

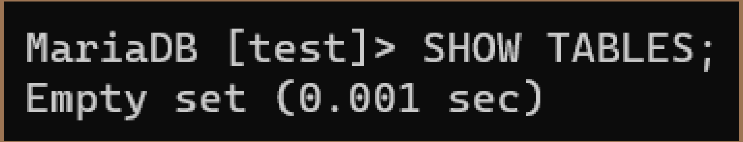
  DROP TABLE TABLE_NAME;



```
MariaDB [test]> DROP TABLE student;
Query OK, 0 rows affected (0.008 sec)
```



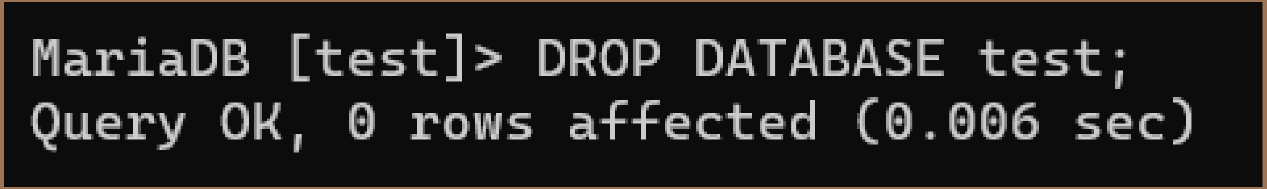
```
MariaDB [test]> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| student        |
+----------------+
```



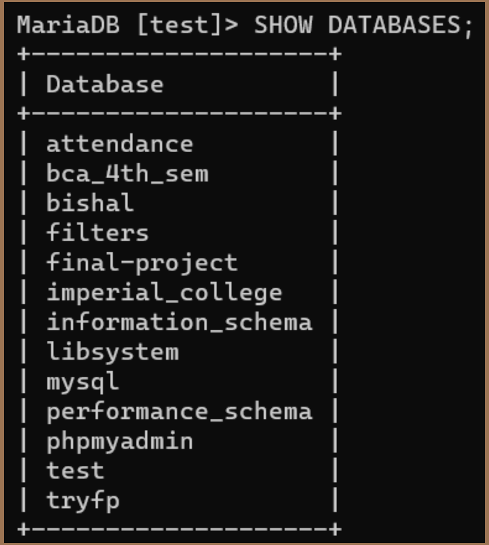
```
MariaDB [test]> SHOW TABLES;
Empty set (0.001 sec)
```

- Syntax

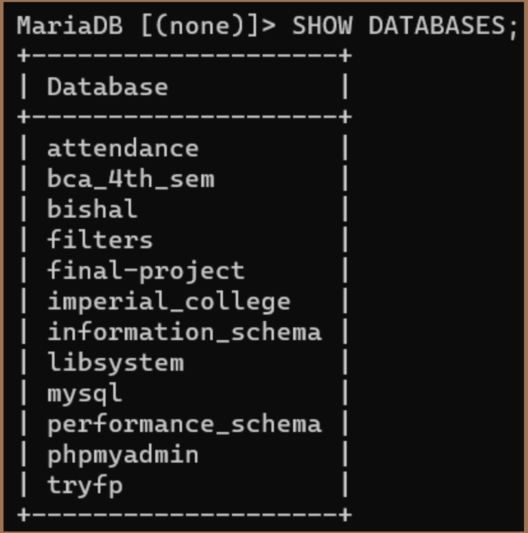
  DROP DATABASE DATABASE_NAME;



```
MariaDB [test]> DROP DATABASE test;
Query OK, 0 rows affected (0.006 sec)
```



```
MariaDB [test]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| attendance         |
| bca_4th_sem        |
| bishal             |
| filters            |
| final-project      |
| imperial_college   |
| information_schema |
| libsystem          |
| mysql              |
| performance_schema |
| phpmyadmin         |
| test               |
| tryfp              |
+--------------------+
```



```
MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| attendance         |
| bca_4th_sem        |
| bishal             |
| filters            |
| final-project      |
| imperial_college   |
| information_schema |
| libsystem          |
| mysql              |
| performance_schema |
| phpmyadmin         |
| tryfp              |
+--------------------+
```

DML (Data Manipulation Language

- It is used to manage data in table.
- It work on tables data.
- Some DDL commands :

    SELECT

    INSERT

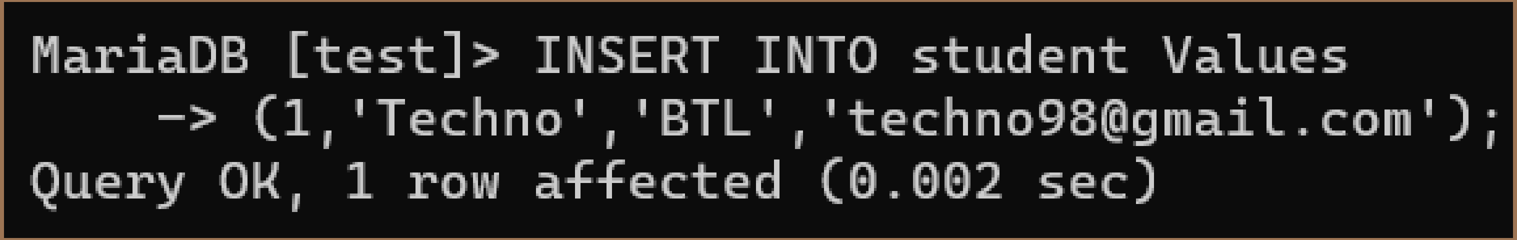
    UPDATE

    DELETE

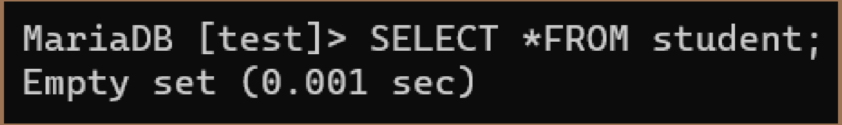
INSERT

- It is use to insert data into table.
- Syntax

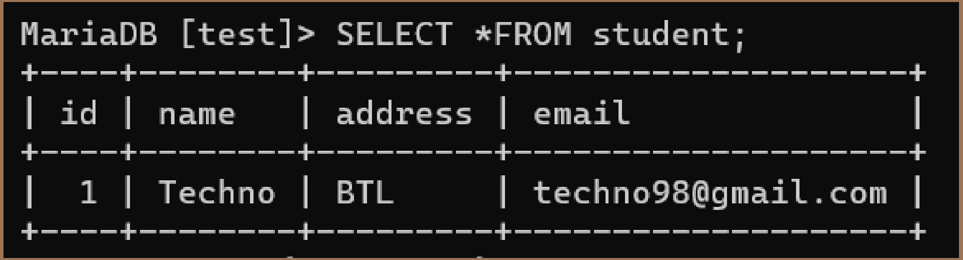
  INSERT INTO TABLE_NAME VALUES ('value-1','value-2','value-3');



```
MariaDB [test]> INSERT INTO student Values
    -> (1,'Techno','BTL','techno98@gmail.com');
Query OK, 1 row affected (0.002 sec)
```



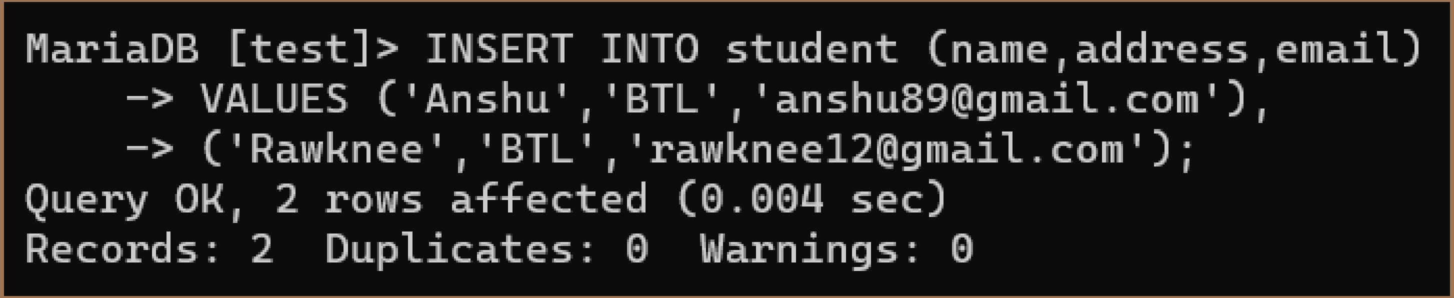
```
MariaDB [test]> SELECT *FROM student;
Empty set (0.001 sec)
```



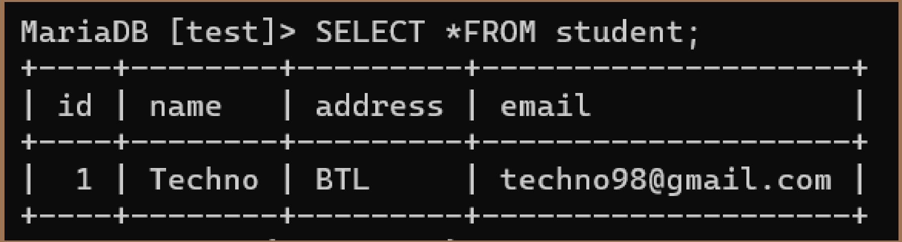
```
MariaDB [test]> SELECT *FROM student;
+----+--------+---------+--------------------+
| id | name   | address | email              |
+----+--------+---------+--------------------+
|  1 | Techno | BTL     | techno98@gmail.com |
+----+--------+---------+--------------------+
```

- Syntax

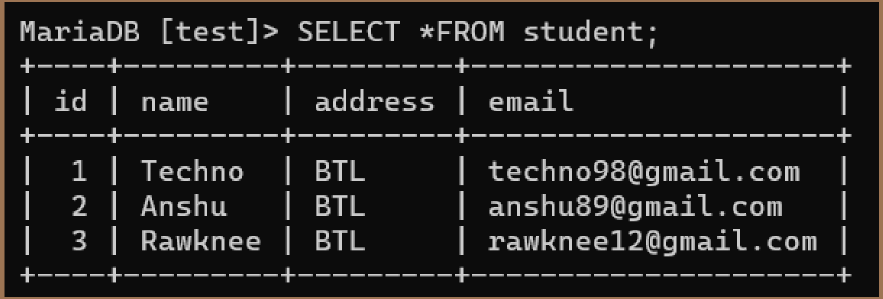
  INSERT INTO TABLE_NAME (‘column-1’, ‘column-2’, ‘column-3’)
  VALUES (‘value-1’,‘value-2’,‘value-3’),
  (‘value-1’,‘value-2’,‘value-3’);



```
MariaDB [test]> INSERT INTO student (name,address,email)
    -> VALUES ('Anshu','BTL','anshu89@gmail.com'),
    -> ('Rawknee','BTL','rawknee12@gmail.com');
Query OK, 2 rows affected (0.004 sec)
Records: 2  Duplicates: 0  Warnings: 0
```



```
MariaDB [test]> SELECT *FROM student;
+----+--------+---------+-------------------+
| id | name   | address | email             |
+----+--------+---------+-------------------+
|  1 | Techno | BTL     | techno98@gmail.com |
+----+--------+---------+-------------------+
```

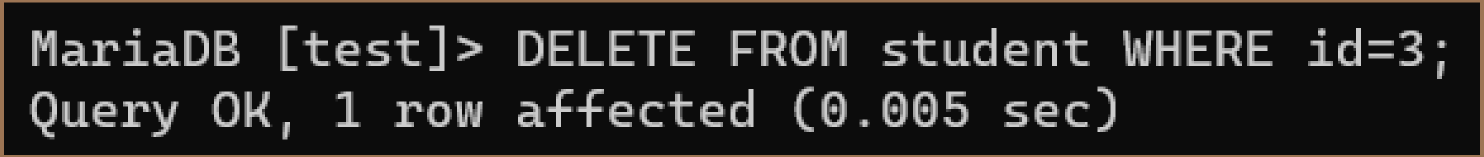


```
MariaDB [test]> SELECT *FROM student;
+----+---------+---------+---------------------+
| id | name    | address | email               |
+----+---------+---------+---------------------+
|  1 | Techno  | BTL     | techno98@gmail.com  |
|  2 | Anshu   | BTL     | anshu89@gmail.com   |
|  3 | Rawknee | BTL     | rawknee12@gmail.com |
+----+---------+---------+---------------------+
```

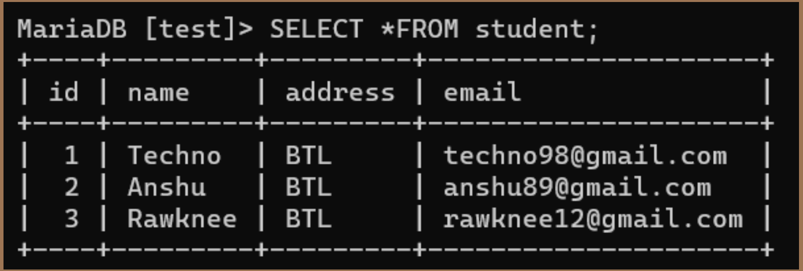
DELETE

- It is use to delete data from table.
- Syntax

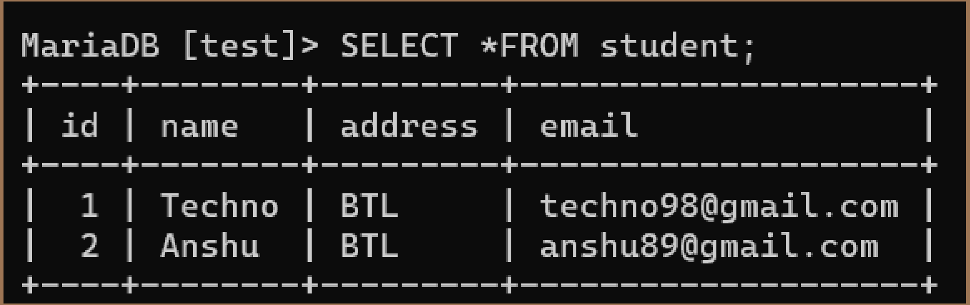
DELETE FROM TABLE_NAME WHERE CONDITION;



```
MariaDB [test]> DELETE FROM student WHERE id=3;
Query OK, 1 row affected (0.005 sec)
```



```
MariaDB [test]> SELECT *FROM student;
+----+---------+---------+---------------------+
| id | name    | address | email               |
+----+---------+---------+---------------------+
|  1 | Techno  | BTL     | techno98@gmail.com  |
|  2 | Anshu   | BTL     | anshu89@gmail.com   |
|  3 | Rawknee | BTL     | rawknee12@gmail.com |
+----+---------+---------+---------------------+
```

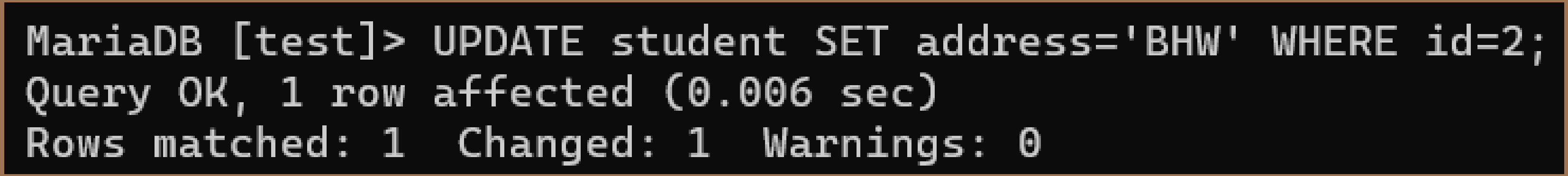


```
MariaDB [test]> SELECT *FROM student;
+----+--------+---------+--------------------+
| id | name   | address | email              |
+----+--------+---------+--------------------+
|  1 | Techno | BTL     | techno98@gmail.com |
|  2 | Anshu  | BTL     | anshu89@gmail.com  |
+----+--------+---------+--------------------+
```

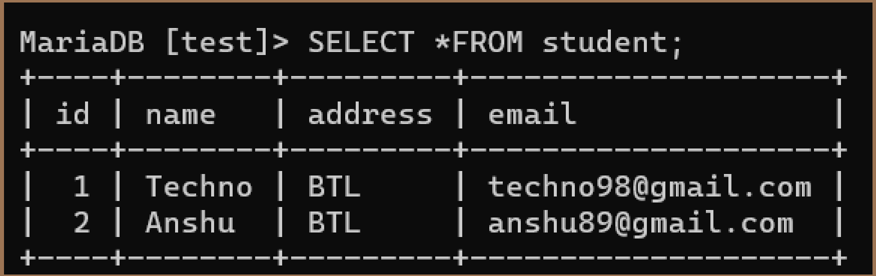
UPDATE

- It is use to update data from table.
- Syntax

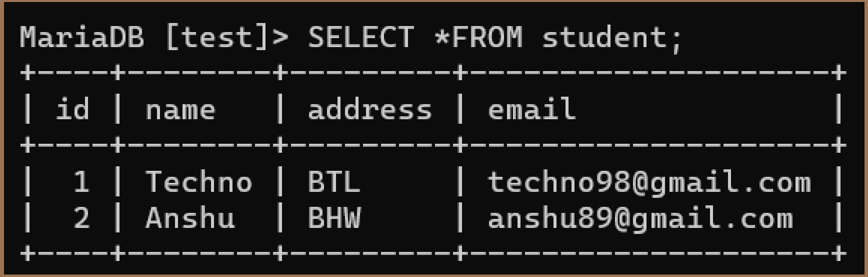
UPDATE TABLE_NAME SET COLUMN1 = VALUE1, COLUMN2 = VALUE2 WHERE CONDITION;



```
MariaDB [test]> UPDATE student SET address='BHW' WHERE id=2;
Query OK, 1 row affected (0.006 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```



```
MariaDB [test]> SELECT *FROM student;
+----+--------+---------+---------------------+
| id | name   | address | email               |
+----+--------+---------+---------------------+
|  1 | Techno | BTL     | techno98@gmail.com |
|  2 | Anshu  | BTL     | anshu89@gmail.com  |
+----+--------+---------+---------------------+
```

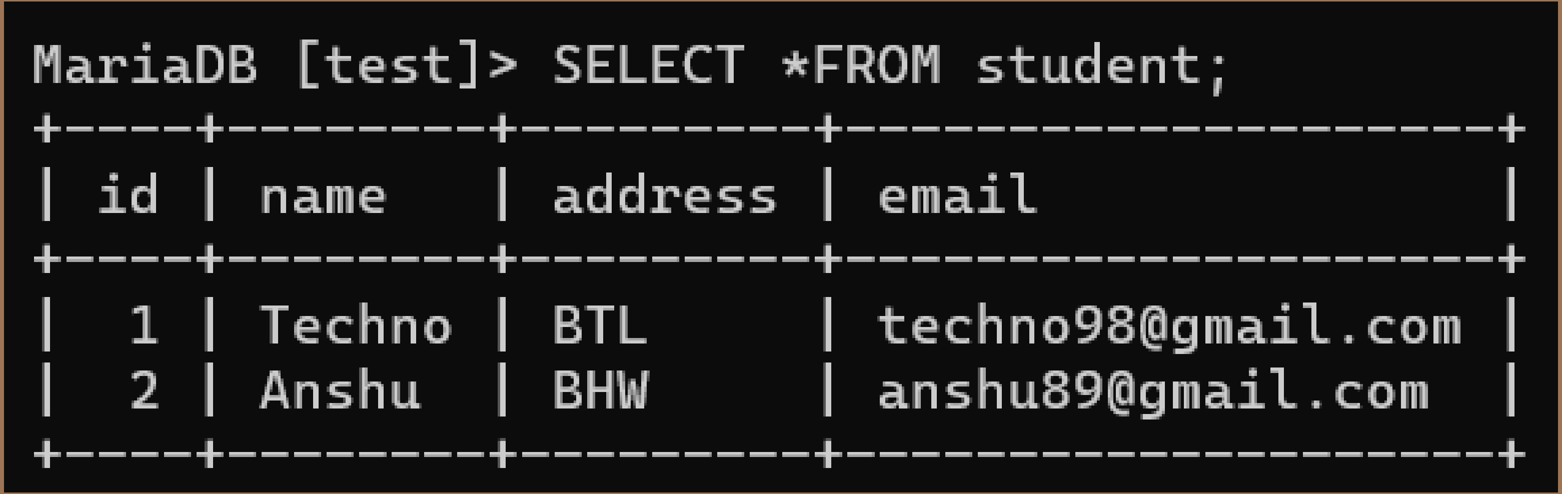


```
MariaDB [test]> SELECT *FROM student;
+----+--------+---------+---------------------+
| id | name   | address | email               |
+----+--------+---------+---------------------+
|  1 | Techno | BTL     | techno98@gmail.com |
|  2 | Anshu  | BHW     | anshu89@gmail.com  |
+----+--------+---------+---------------------+
```

SELECT

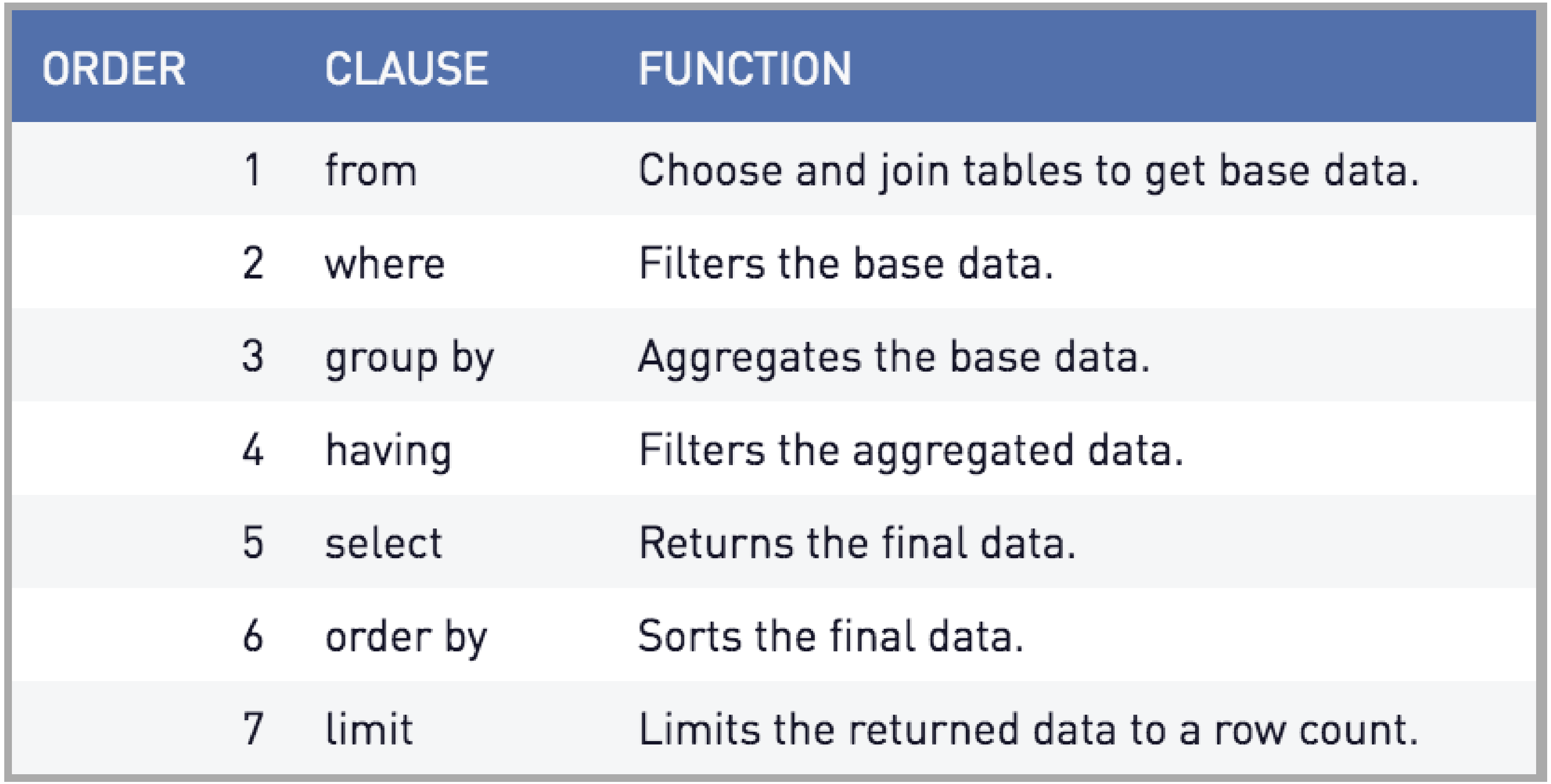
- It is use fetch data from table.
- Syntax

  SELECT *FROM TABLE_NAME;



```
MariaDB [test]> SELECT *FROM student;
+----+---------+----------+--------------------+
| id | name    | address  | email              |
+----+---------+----------+--------------------+
|  1 | Techno  | BTL      | techno98@gmail.com |
|  2 | Anshu   | BHW      | anshu89@gmail.com  |
+----+---------+----------+--------------------+
```

Some important CLAUSES in SQL are:



ORDER	CLAUSE	FUNCTION
1	from	Choose and join tables to get base data.
2	where	Filters the base data.
3	group by	Aggregates the base data.
4	having	Filters the aggregated data.
5	select	Returns the final data.
6	order by	Sorts the final data.
7	limit	Limits the returned data to a row count.

SQL AGGREGATE FUNCTIONS

- SQL provides serval built-in numeric functions
- **COUNT:** The number of rows containing non-null values
- **MIN:** The minimum attribute value encountered in a given column
- **MAX:** The maximum attribute value encountered in a given column
- **SUM:** The sum of all values for a givenColumn
- **AVG:** The arithmetic mean (average) for a specified column

COUNT() function

The SQL COUNT function returns the number of rows in a table satisfying the criteria specified in the WHERE clause.

Eg: SELECT COUNT(*) FROM product WHERE price>=50;

SUM() function

The SQL SUM() function returns the sum of all selected column.
Example : SELECT SUM(salary) FROM employee;

AVG() function

The SQL AVG function calculates the average value of a column of numeric type. It returns the average of all non NULL values.
Example : SELECT AVG(salary) FROM employee;

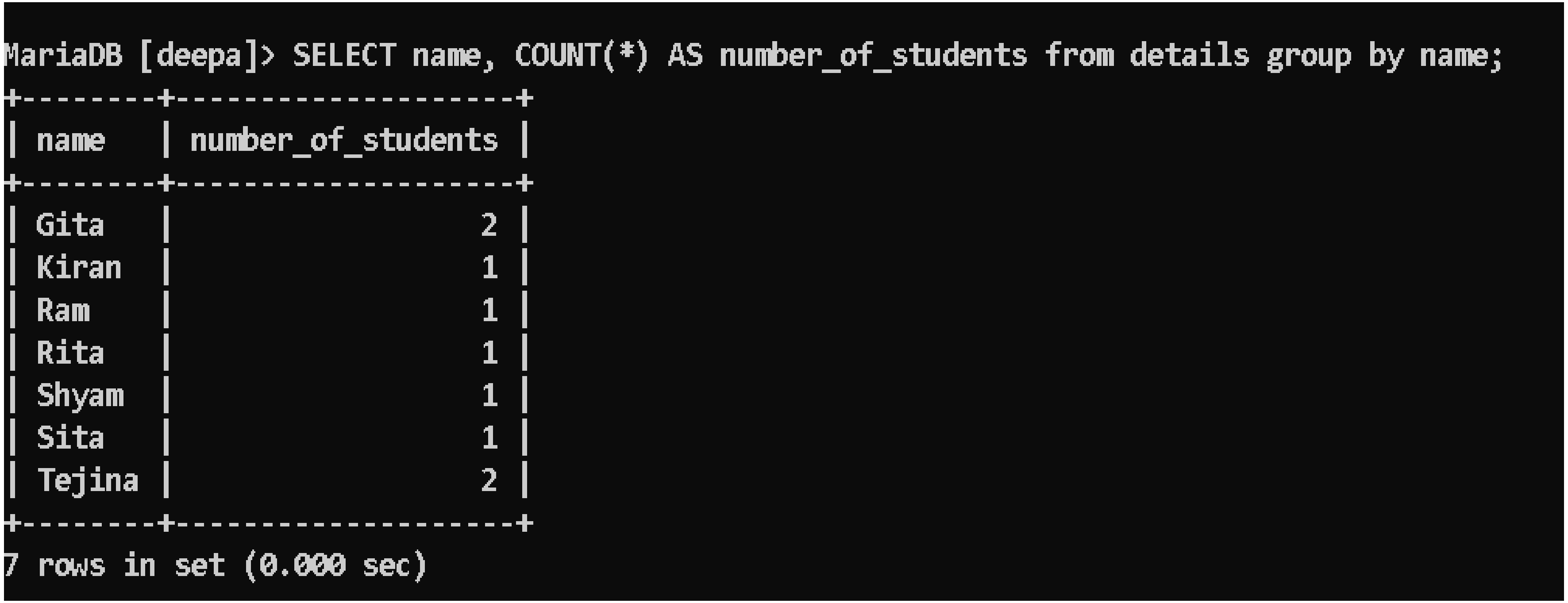
MAX() function

- The aggregate function SQL MAX() is used to find the maximum value or highest value of a certain column.
- This function is useful to determine the largest of all selected values of a column.
- Example : SELECT MAX(age) FROM employee;

MIN() function

- The aggregate function SQL MIN() is used to find the minimum value or lowest value of a column or expression.
- This function is useful to determine the smallest of all selected values of a column.
- Example : SELECT MIN(age) FROM employee;

SELECT name, COUNT(*) AS number_of_students
FROM details GROUP BY name;



```
MariaDB [deepa]> SELECT name, COUNT(*) AS number_of_students from details group by name;
+--------+--------------------+
| name   | number_of_students |
+--------+--------------------+
| Gita   |                  2 |
| Kiran  |                  1 |
| Ram    |                  1 |
| Rita   |                  1 |
| Shyam  |                  1 |
| Sita   |                  1 |
| Tejina |                  2 |
+--------+--------------------+
7 rows in set (0.000 sec)
```

MySQL v/s Oracle

Comparison Basis	MySQL	Oracle
Definition	It is an open-source, cross-platform relational database management system built by Swedish Company MYSQL AB and currently supported by the Oracle.	Oracle is a relational database system (RDBMS) that implements object-oriented features. It allows to store and retrieve data quickly and safely. It can handle a large amount of data.
Release	It was released in 1995.	It was released in 1980.
Cost	It is free and open-source. It is licensed under the GNU.	It is licensed for commercial purposes, but it provides the express edition for free which is recommended for students only.
Scalability	MySQL database is used for small and big businesses.	Oracle database is used for very large-scale deployments.
Security	It requires a username, password, and host to access the database.	It requires a username, password, and profile validation to access the database.
Null Value	MySQL supports the null value.	Oracle does not support the null value.
Character	MySQL support only two characters that are CHAR and VARCHAR.	Oracle supports four different characters that are CHAR, VARCHAR2, NCHAR, and NVARCHAR2.

Views

- Views in SQL are kind of virtual tables.
- A view also has rows and columns as they are in a real table in the database.
- It is generally used to focus, simplify, and customize the perception each user has of the database.
- Views can be used as security mechanisms by letting users access data through the view, without granting the users permissions to directly access the underlying base tables of the view.

OR

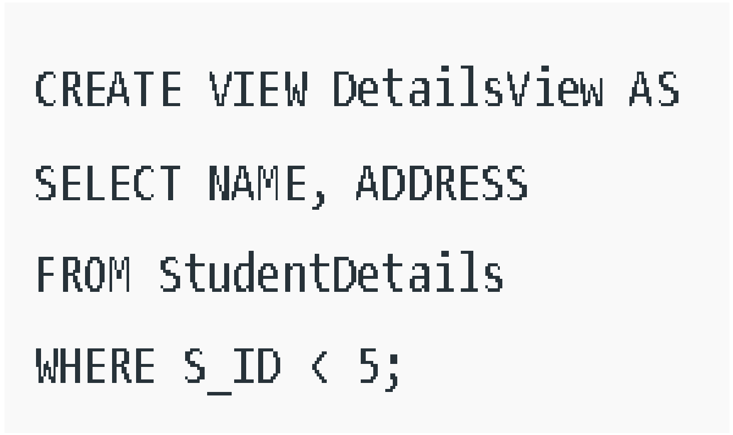
- Views can help control and restrict access to sensitive data, providing a layer of security and ensuring that users can only see the data they are authorized to access.
- We can create a view by selecting fields from one or more tables present in the database.
- A View can either have all the rows of a table or specific rows based on certain condition. Views can represent a subset of the data contained in a table.

Views

- **Syntax:**

```
CREATE VIEW view_name
AS
SELECT column1, column2.....
FROM  table_name
WHERE condition;
```

- **Example:**



```
CREATE VIEW DetailsView AS

SELECT NAME, ADDRESS

FROM StudentDetails

WHERE S_ID < 5;
```

Create a view that hides sensitive data

Example:

```
CREATE VIEW employee_public_info AS
SELECT id, first_name, last_name, email
FROM employee_details;
```

To retieve data of view: (same as normal table)

```
SELECT * FROM employee_public_info;
```

Alter view: To modify an existing view, you can use the CREATE OR REPLACE VIEW statement

Example:

```
CREATE OR REPLACE VIEW employee_summary AS
SELECT id, first_name, last_name, email, salary, address
FROM employee_details;
```

To Drop View:

Example:

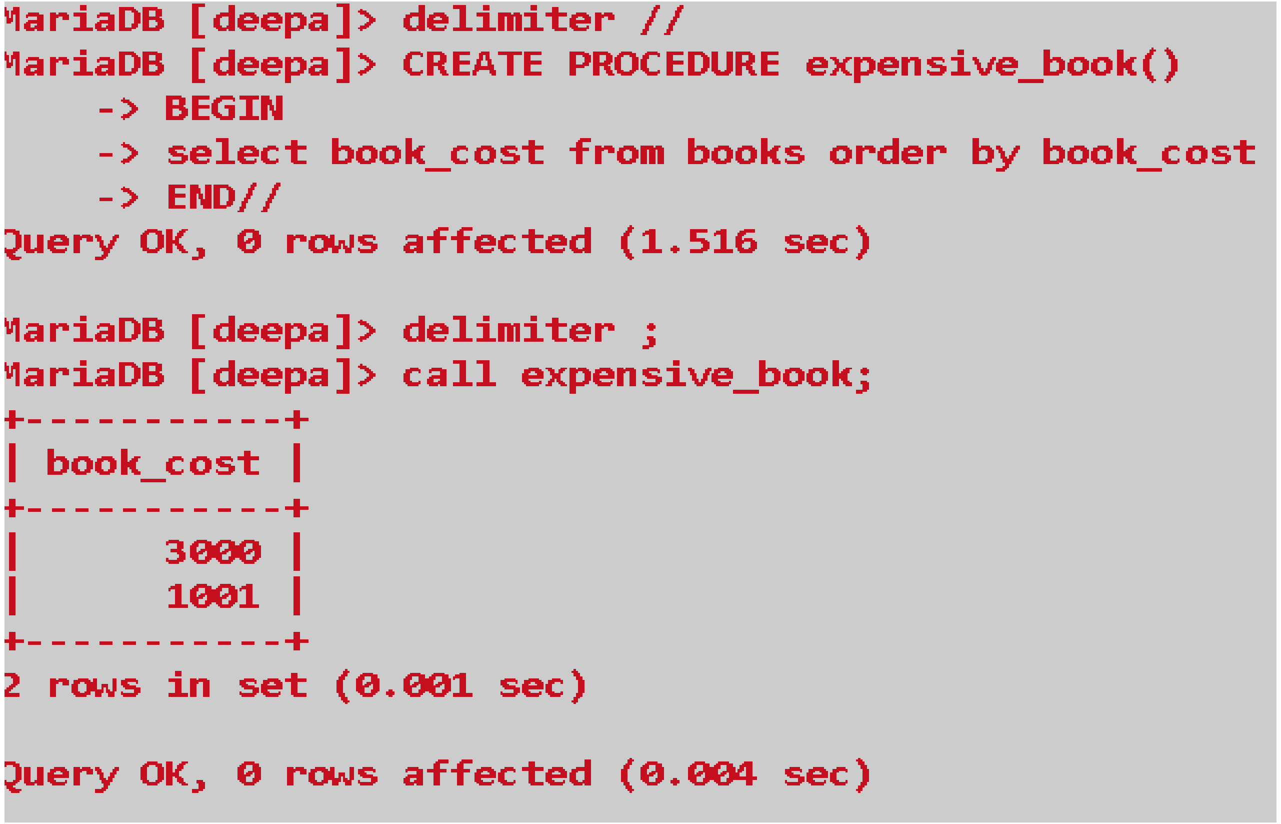
```
DROP VIEW employee_summary;
```

Other Important Concepts of SQL

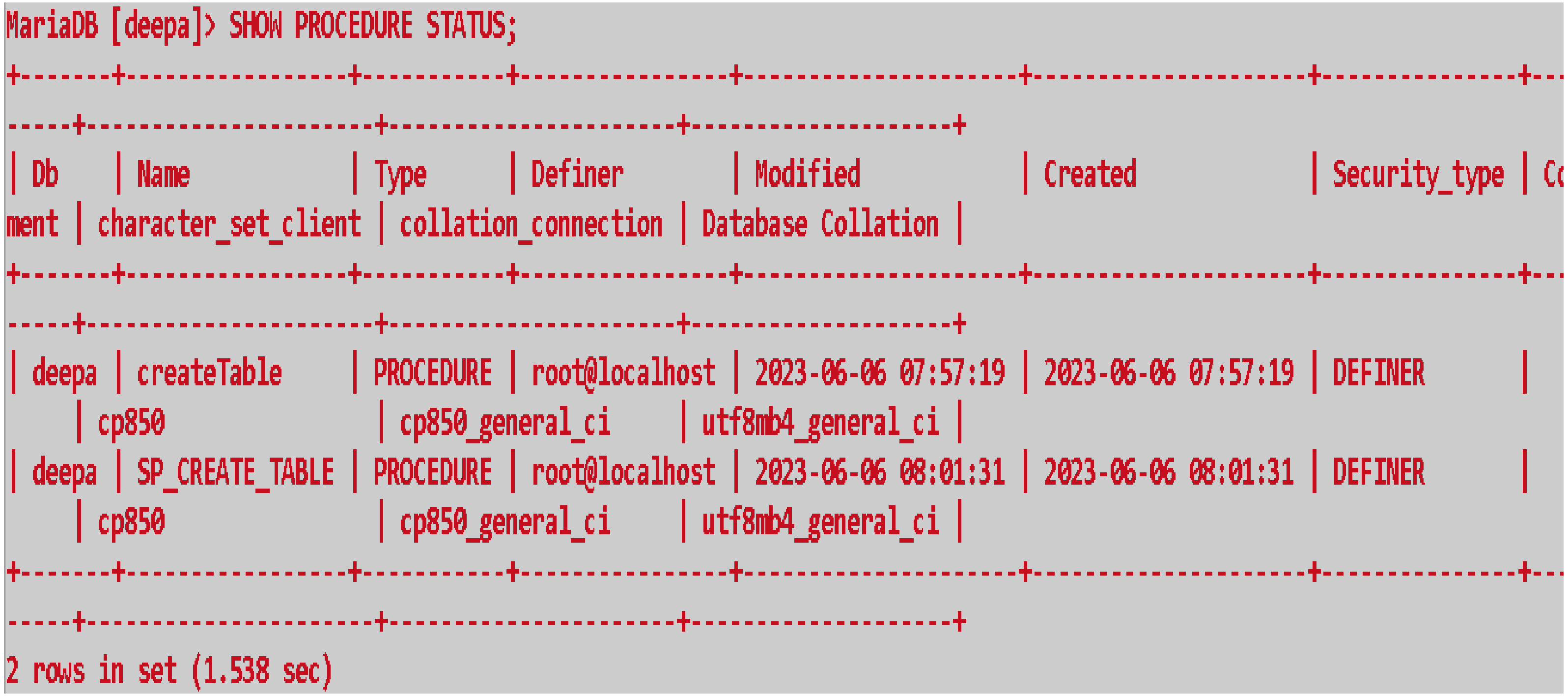
Developing Stored Procedures, DML Triggers and Indexing

Stored Procedures

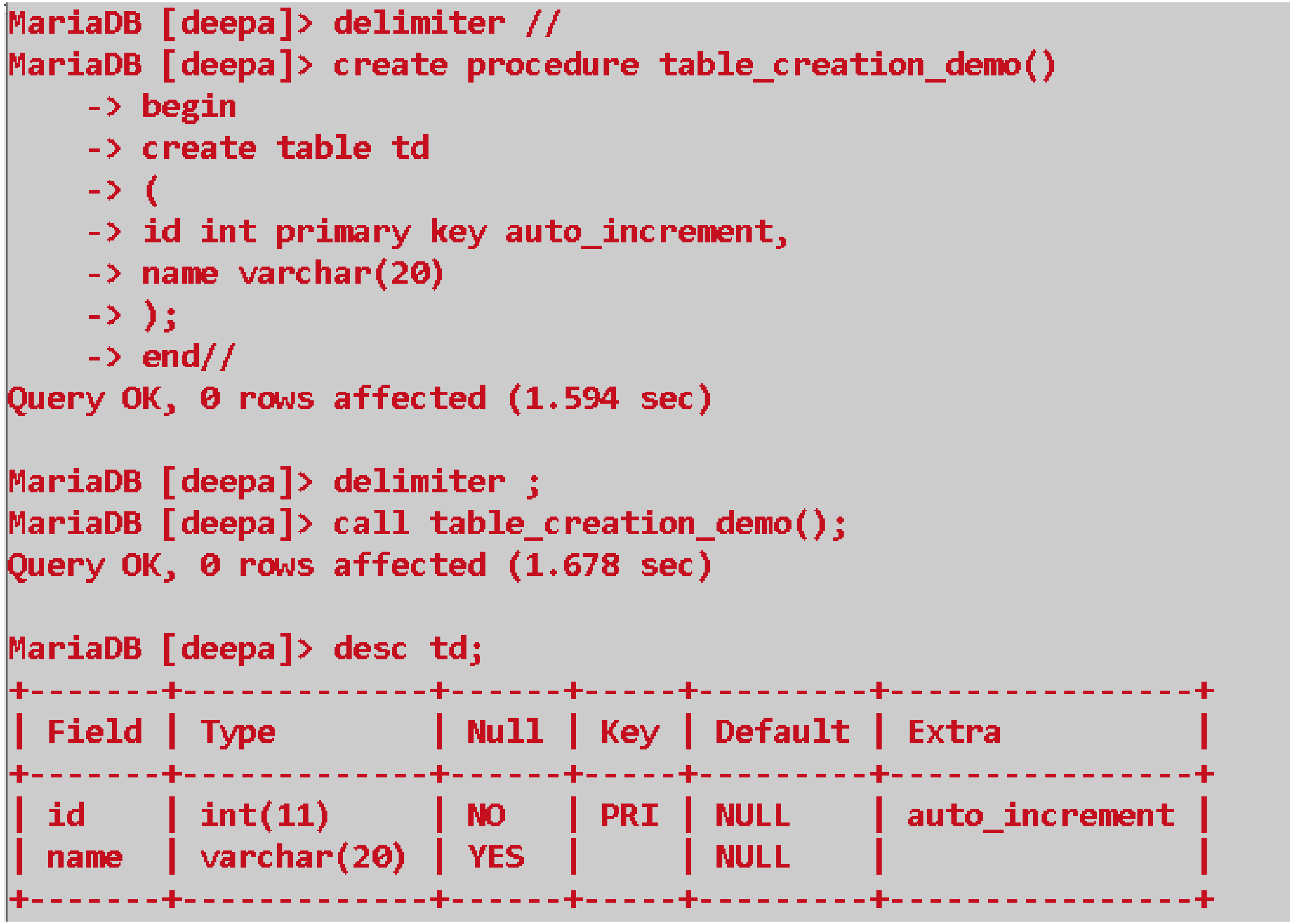
- A stored procedure is a collection of pre-compiled SQL statements stored inside the database.
- It is a subroutine or a subprogram in the regular computing language.
- A procedure always contains a name, parameter lists, and SQL statements.



```
MariaDB [deepa]> delimiter //
MariaDB [deepa]> CREATE PROCEDURE expensive_book()
    -> BEGIN
    -> select book_cost from books order by book_cost
    -> END//
Query OK, 0 rows affected (1.516 sec)

MariaDB [deepa]> delimiter ;
MariaDB [deepa]> call expensive_book;
+-----------+
| book_cost |
+-----------+
|      3000 |
|      1001 |
+-----------+
2 rows in set (0.001 sec)

Query OK, 0 rows affected (0.004 sec)
```

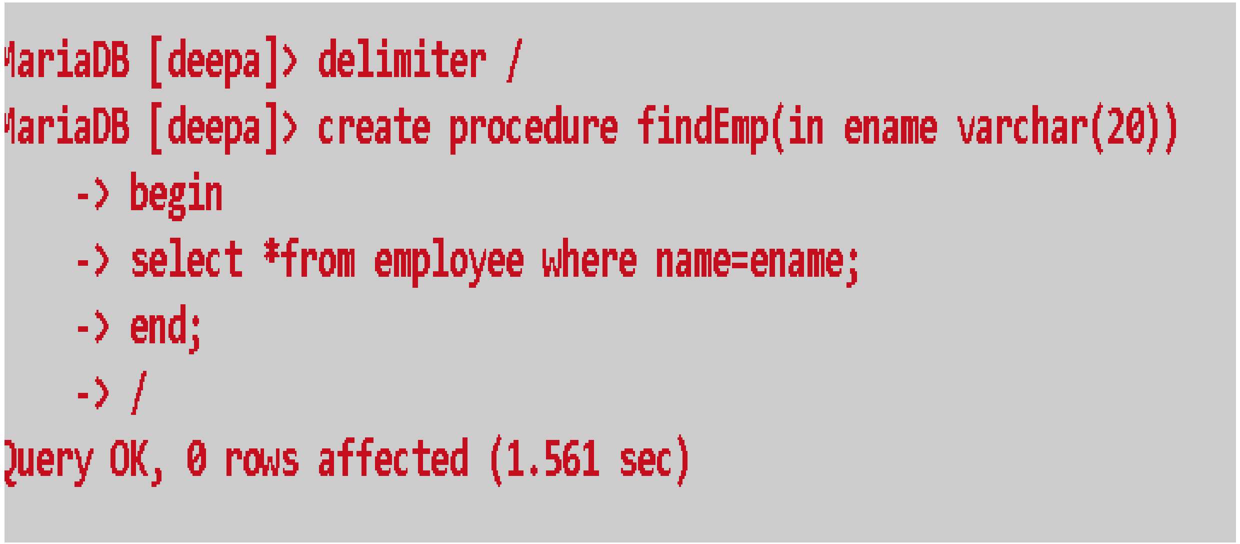


```
MariaDB [deepa]> SHOW PROCEDURE STATUS;
+-------+-----------------+-----------+----------------+---------------------+---------------------+---------------+---
-----+---------------------+---------------------+--------------------+
| Db    | Name            | Type      | Definer        | Modified            | Created             | Security_type | Co
ment | character_set_client | collation_connection | Database Collation |
+-------+-----------------+-----------+----------------+---------------------+---------------------+---------------+---
-----+---------------------+---------------------+--------------------+
| deepa | createTable     | PROCEDURE | root@localhost | 2023-06-06 07:57:19 | 2023-06-06 07:57:19 | DEFINER       |
     | cp850                | cp850_general_ci     | utf8mb4_general_ci |
| deepa | SP_CREATE_TABLE | PROCEDURE | root@localhost | 2023-06-06 08:01:31 | 2023-06-06 08:01:31 | DEFINER       |
     | cp850                | cp850_general_ci     | utf8mb4_general_ci |
+-------+-----------------+-----------+----------------+---------------------+---------------------+---------------+---
-----+---------------------+---------------------+--------------------+
2 rows in set (1.538 sec)
```

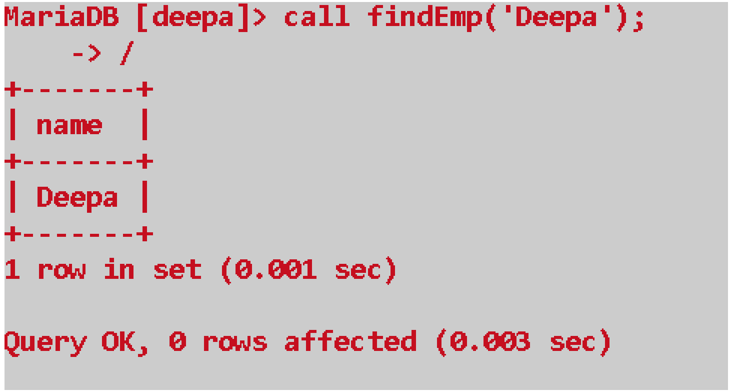


```
MariaDB [deepa]> delimiter //
MariaDB [deepa]> create procedure table_creation_demo()
    -> begin
    -> create table td
    -> (
    -> id int primary key auto_increment,
    -> name varchar(20)
    -> );
    -> end//
Query OK, 0 rows affected (1.594 sec)

MariaDB [deepa]> delimiter ;
MariaDB [deepa]> call table_creation_demo();
Query OK, 0 rows affected (1.678 sec)

MariaDB [deepa]> desc td;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| name  | varchar(20) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
delimiter //
MariaDB [deepa]> CREATE PROCEDURE expensive_book()
    -> BEGIN
    -> select book_cost from books order by book_cost desc;
    -> END//

Query OK, 0 rows affected (1.516 sec)


MariaDB [deepa]> delimiter ;

MariaDB [deepa]> call expensive_book;
```

```
delimiter //
MariaDB [deepa]> create procedure table_creation_demo()
    -> begin
    -> create table td
    -> (
    -> id int primary key auto_increment,
    -> name varchar(20)
    -> );
    -> end//
Query OK, 0 rows affected (1.594 sec)
```



```
ariaDB [deepa]> delimiter /
ariaDB [deepa]> create procedure findEmp(in ename varchar(20))
    -> begin
    -> select *from employee where name=ename;
    -> end;
    -> /
uery OK, 0 rows affected (1.561 sec)
```



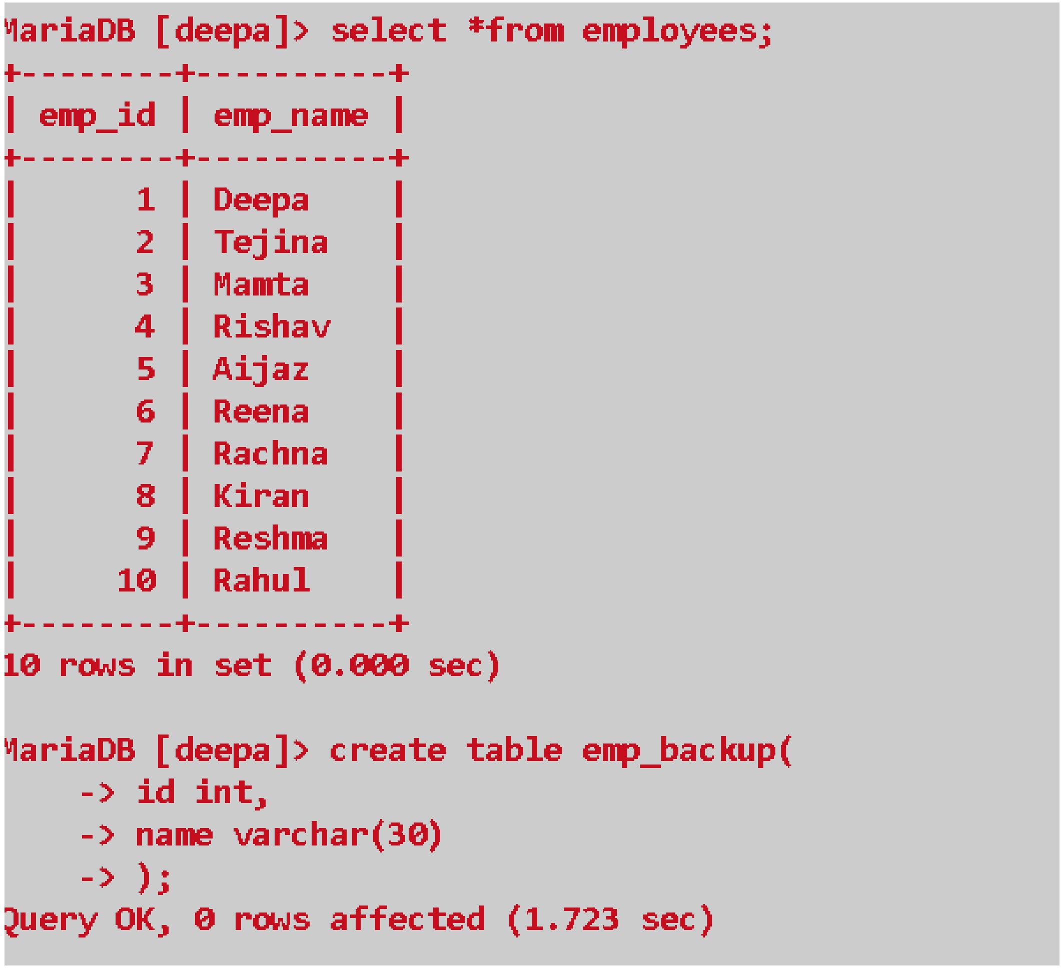
```
MariaDB [deepa]> call findEmp('Deepa');
    -> /
+-------+
| name  |
+-------+
| Deepa |
+-------+
1 row in set (0.001 sec)

Query OK, 0 rows affected (0.003 sec)
```

```
create procedure findEmp(in ename varchar(20))
    -> begin
    -> select *from employee where name=ename;
    -> end;
    -> /
```

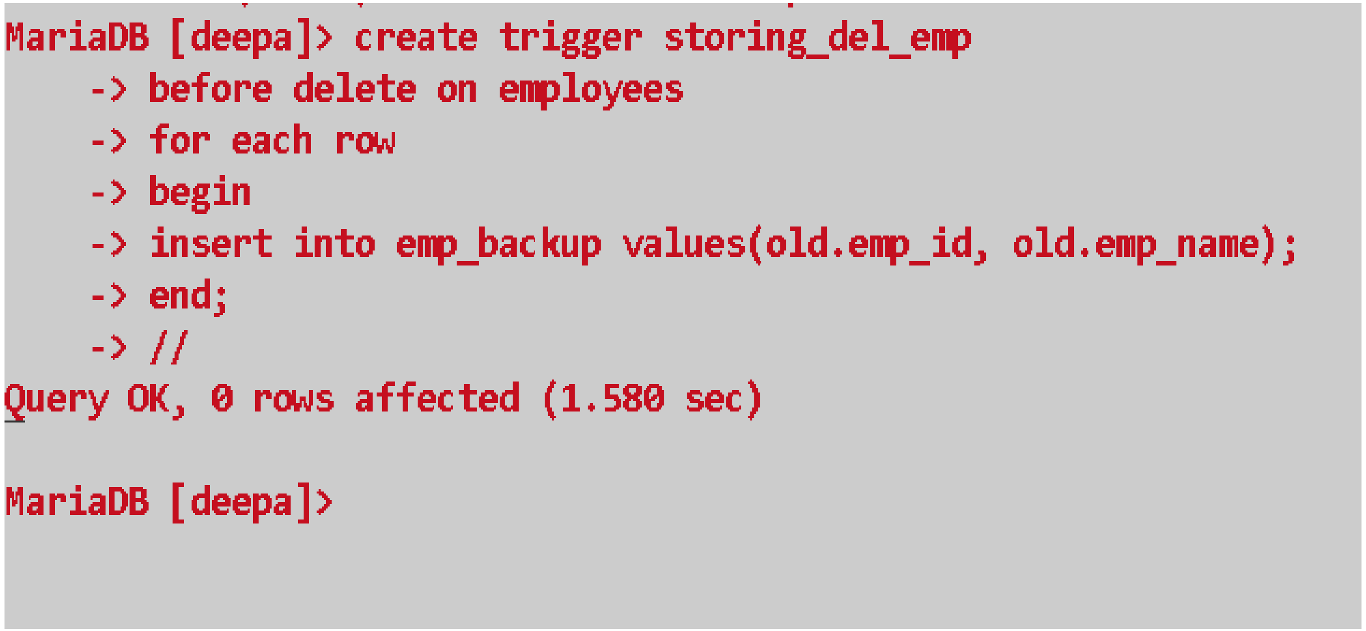
```
call findEmp('Deepa');
```

Creating Triggers

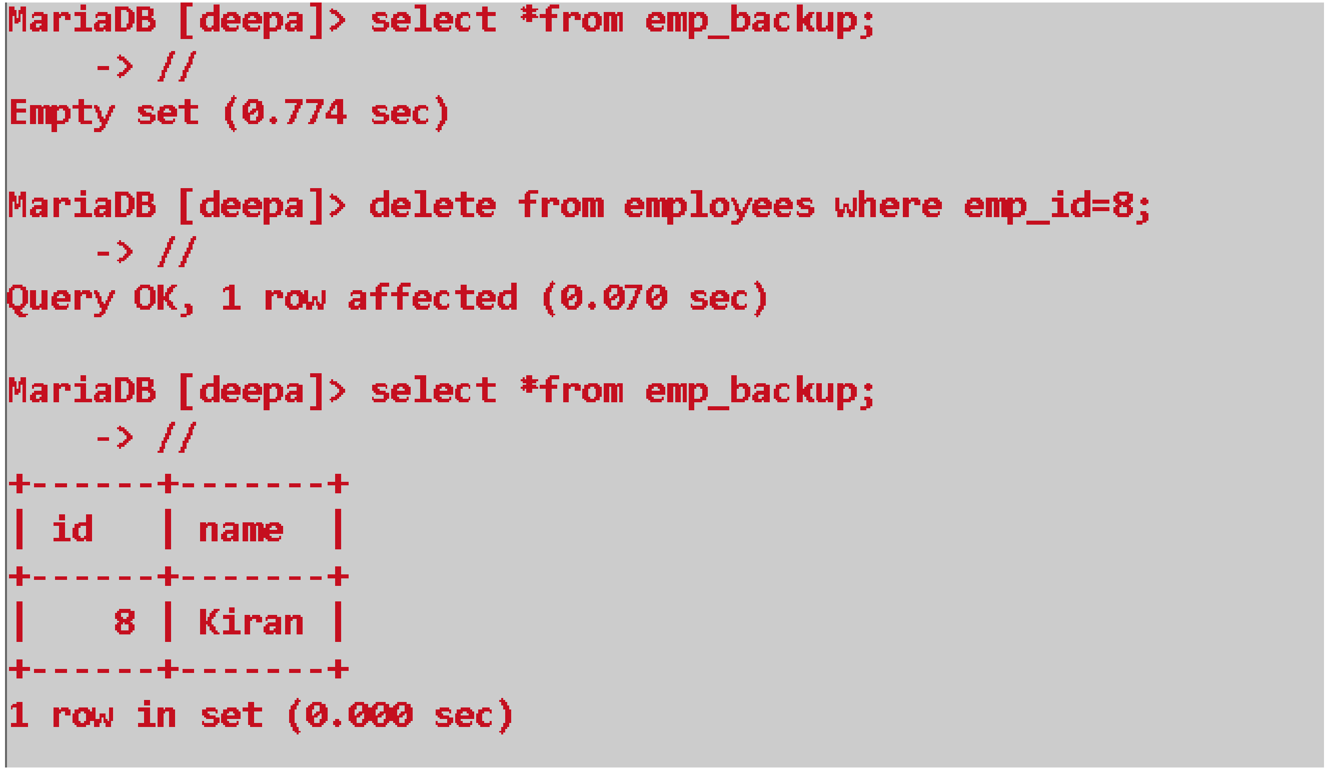
- An SQL trigger is a database object that is associated with a table and automatically executes a set of SQL statements when a specific event occurs on that table.
- Triggers are used to enforce business rules, maintain data integrity, and automate certain actions within a database.



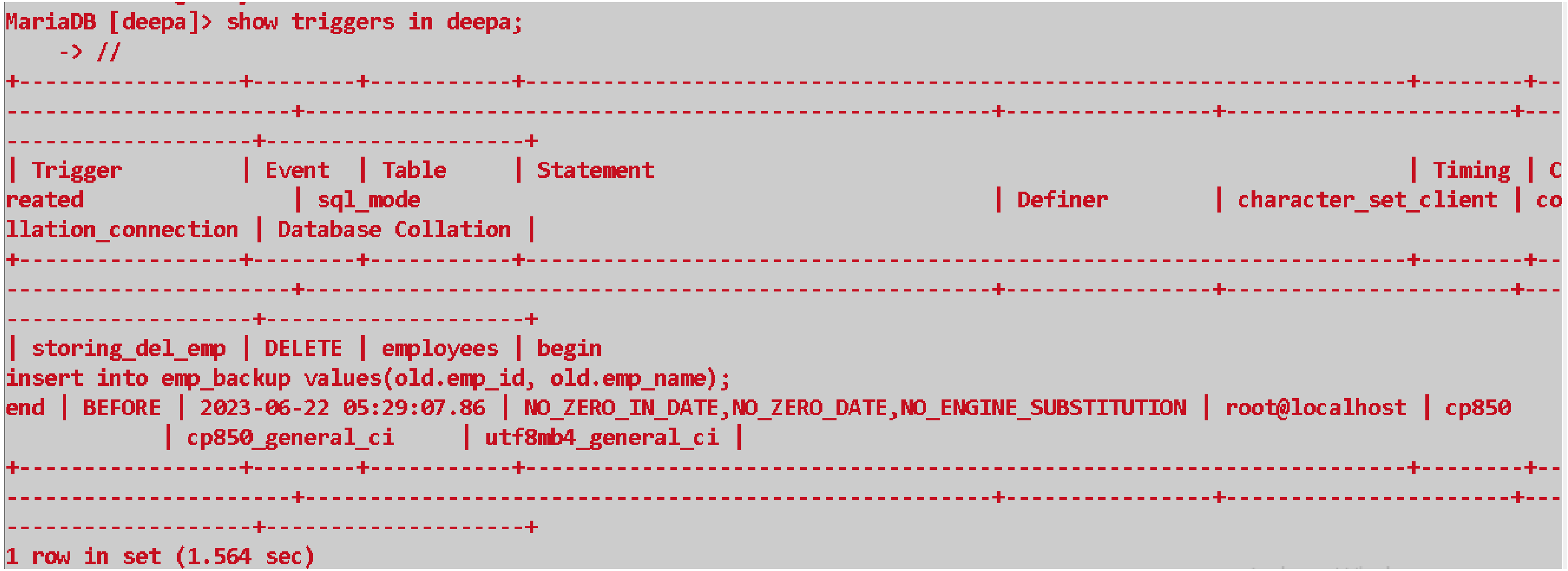
```
1ariaDB [deepa]> select *from employees;
+--------+----------+
| emp_id | emp_name |
+--------+----------+
|      1 | Deepa    |
|      2 | Tejina   |
|      3 | Mamta    |
|      4 | Rishav   |
|      5 | Aijaz    |
|      6 | Reena    |
|      7 | Rachna   |
|      8 | Kiran    |
|      9 | Reshma   |
|     10 | Rahul    |
+--------+----------+
10 rows in set (0.000 sec)

1ariaDB [deepa]> create table emp_backup(
    -> id int,
    -> name varchar(30)
    -> );
Query OK, 0 rows affected (1.723 sec)
```



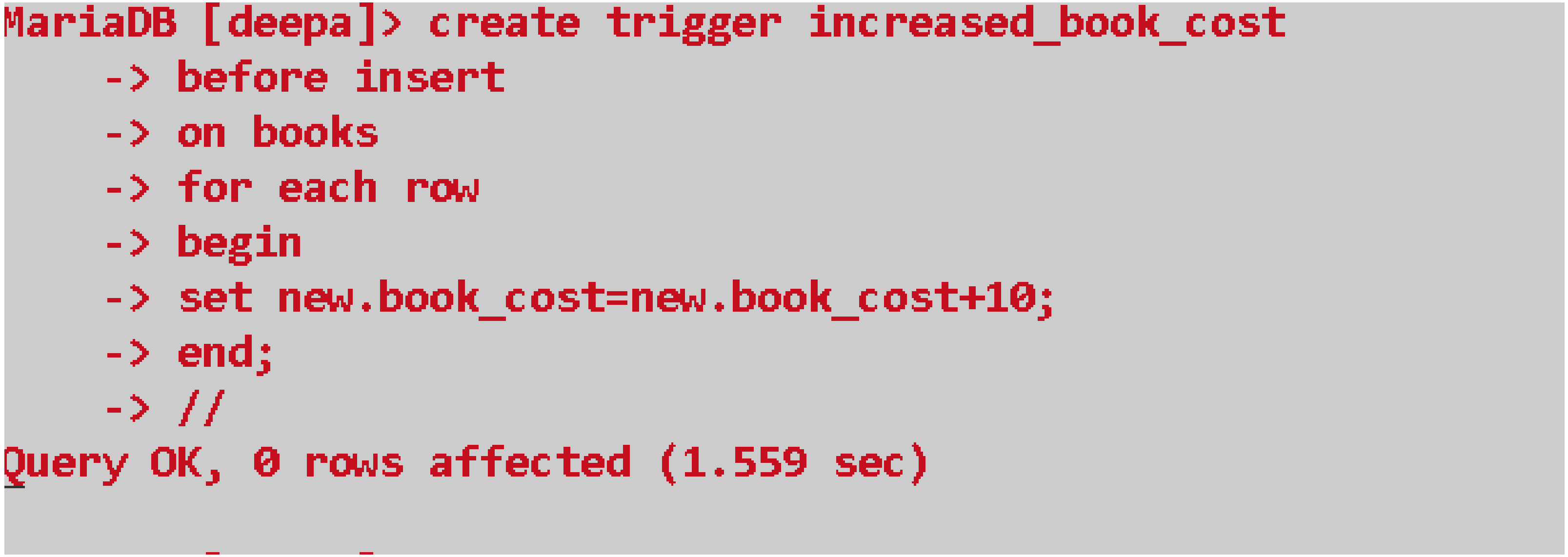
```
MariaDB [deepa]> create trigger storing_del_emp
    -> before delete on employees
    -> for each row
    -> begin
    -> insert into emp_backup values(old.emp_id, old.emp_name);
    -> end;
    -> //
Query OK, 0 rows affected (1.580 sec)

MariaDB [deepa]>
```



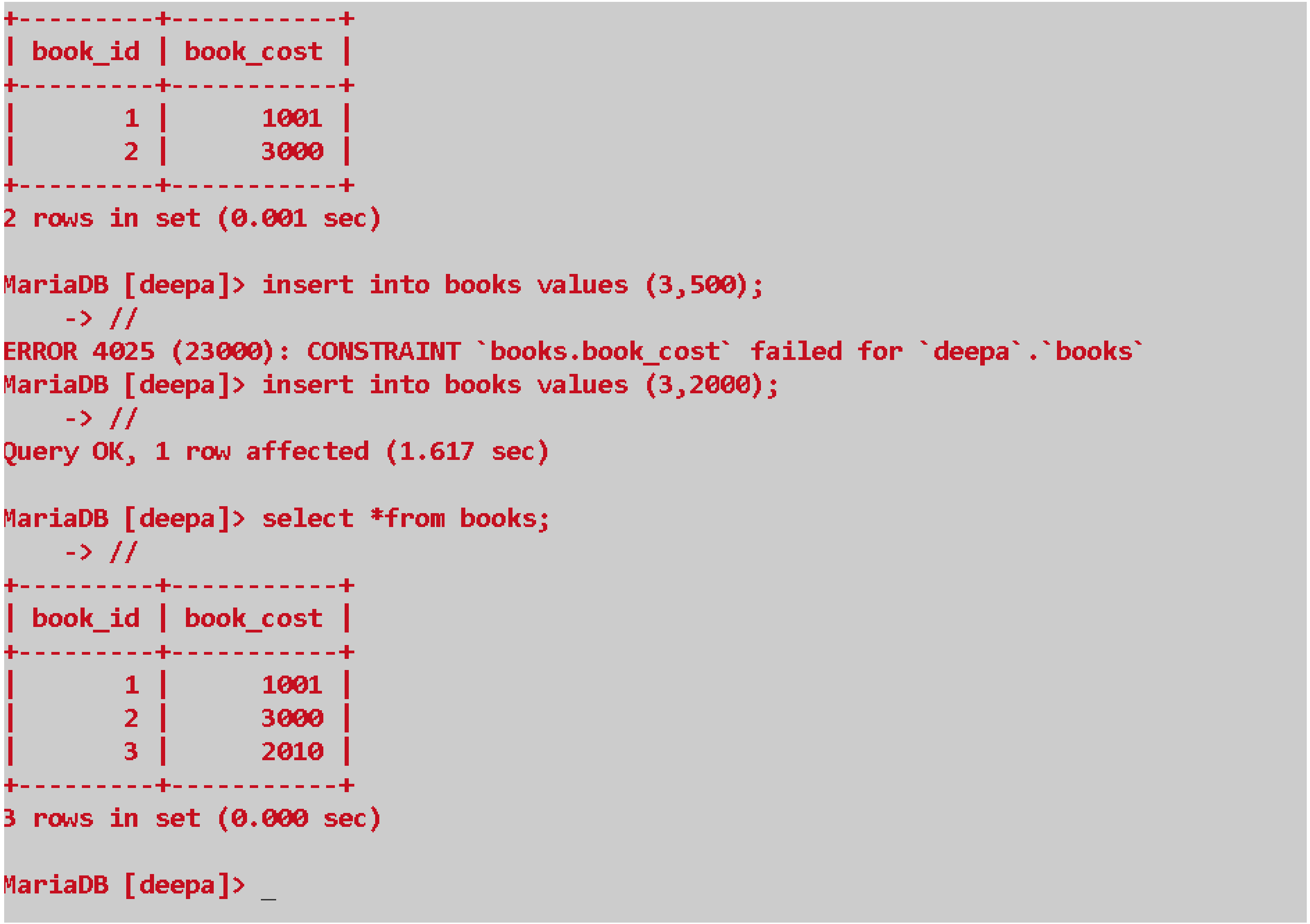
```
MariaDB [deepa]> select *from emp_backup;
    -> //
Empty set (0.774 sec)

MariaDB [deepa]> delete from employees where emp_id=8;
    -> //
Query OK, 1 row affected (0.070 sec)

MariaDB [deepa]> select *from emp_backup;
    -> //
+------+-------+
| id   | name  |
+------+-------+
|    8 | Kiran |
+------+-------+
1 row in set (0.000 sec)
```



```
MariaDB [deepa]> show triggers in deepa;
    -> //
+-----------------+--------+-----------+--------------------------------------------------------------------+--------+--
----------------------+-----------------------------------------------------+----------------+----------------------+---
-------------------+--------------------+
| Trigger         | Event  | Table      | Statement                                                          | Timing | C
reated                | sql_mode                                            | Definer        | character_set_client | co
llation_connection | Database Collation |
+-----------------+--------+-----------+--------------------------------------------------------------------+--------+--
----------------------+-----------------------------------------------------+----------------+----------------------+---
-------------------+--------------------+
| storing_del_emp | DELETE | employees | begin
insert into emp_backup values(old.emp_id, old.emp_name);
end | BEFORE | 2023-06-22 05:29:07.86 | NO_ZERO_IN_DATE,NO_ZERO_DATE,NO_ENGINE_SUBSTITUTION | root@localhost | cp850
            | cp850_general_ci     | utf8mb4_general_ci |
+-----------------+--------+-----------+--------------------------------------------------------------------+--------+--
----------------------+-----------------------------------------------------+----------------+----------------------+---
-------------------+--------------------+
1 row in set (1.564 sec)
```



```
MariaDB [deepa]> create trigger increased_book_cost
     -> before insert
     -> on books
     -> for each row
     -> begin
     -> set new.book_cost=new.book_cost+10;
     -> end;
     -> //
Query OK, 0 rows affected (1.559 sec)
```



```
+---------+-----------+
| book_id | book_cost |
+---------+-----------+
|       1 |      1001 |
|       2 |      3000 |
+---------+-----------+
2 rows in set (0.001 sec)

MariaDB [deepa]> insert into books values (3,500);
    -> //
ERROR 4025 (23000): CONSTRAINT `books.book_cost` failed for `deepa`.`books`
MariaDB [deepa]> insert into books values (3,2000);
    -> //
Query OK, 1 row affected (1.617 sec)

MariaDB [deepa]> select *from books;
    -> //
+---------+-----------+
| book_id | book_cost |
+---------+-----------+
|       1 |      1001 |
|       2 |      3000 |
|       3 |      2010 |
+---------+-----------+
3 rows in set (0.000 sec)

MariaDB [deepa]> _
```

Creating Indexes

- Indexes are used to retrieve data from the database more quickly than otherwise.
- The users cannot see the indexes, they are just used to speed up searches/queries

To Create the index:

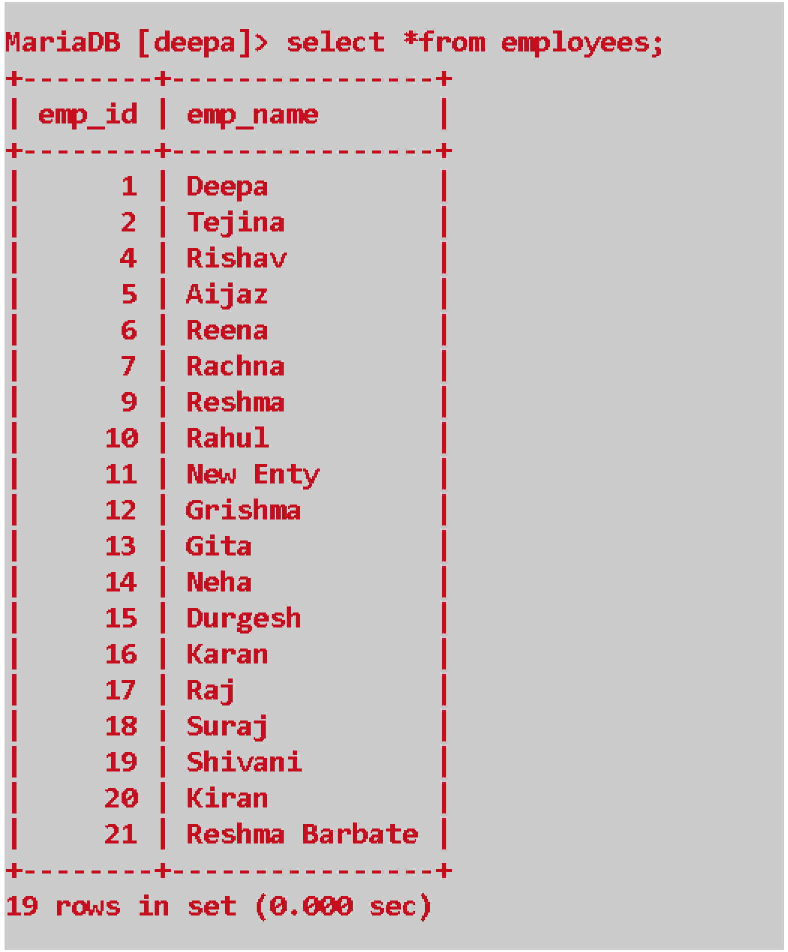
```
CREATE INDEX index_name ON
table_name(column_list);
```

To Show the indexes:

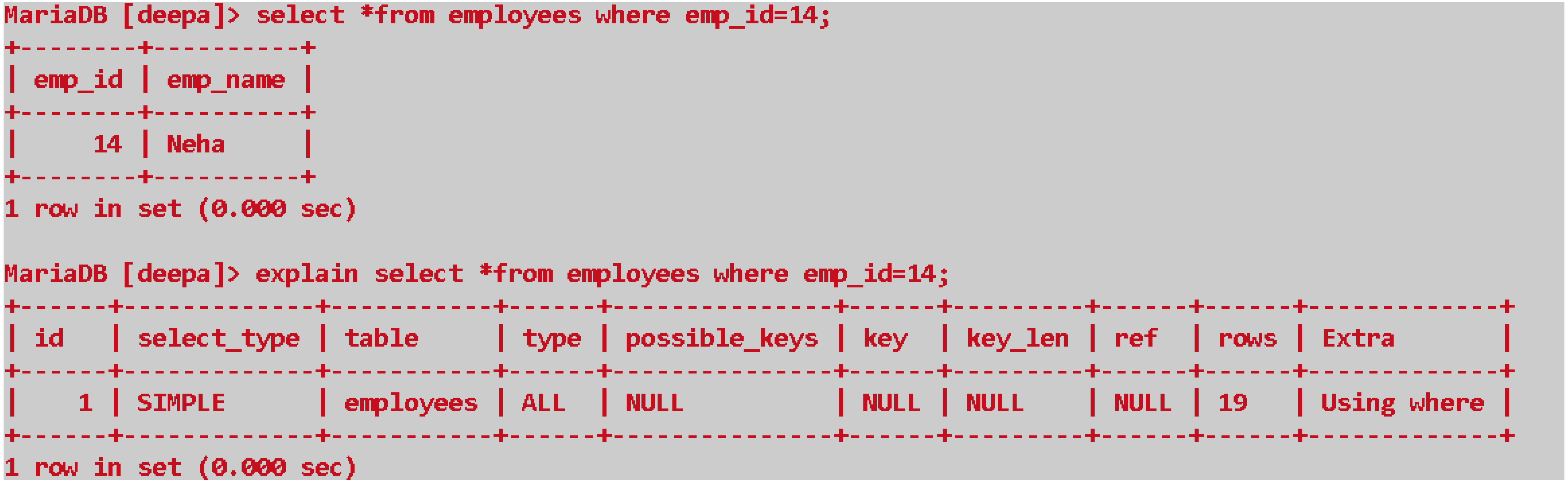
```
SHOW INDEXES FROM table_name;
```

To Delete the indexes:

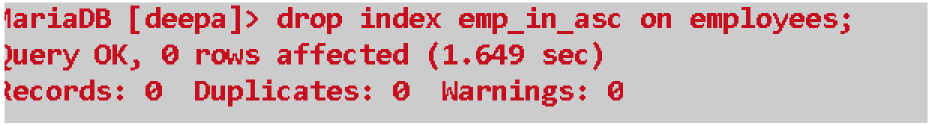
```
DROP INDEX emp_in_asc ON employees;

ALTER TABLE table_name DROP INDEX index_name;
```



```
MariaDB [deepa]> select *from employees;
+--------+----------------+
| emp_id | emp_name       |
+--------+----------------+
|      1 | Deepa          |
|      2 | Tejina         |
|      4 | Rishav         |
|      5 | Aijaz          |
|      6 | Reena          |
|      7 | Rachna         |
|      9 | Reshma         |
|     10 | Rahul          |
|     11 | New Enty       |
|     12 | Grishma        |
|     13 | Gita           |
|     14 | Neha           |
|     15 | Durgesh        |
|     16 | Karan          |
|     17 | Raj            |
|     18 | Suraj          |
|     19 | Shivani        |
|     20 | Kiran          |
|     21 | Reshma Barbate |
+--------+----------------+
19 rows in set (0.000 sec)
```



```
MariaDB [deepa]> select *from employees where emp_id=14;
+--------+----------+
| emp_id | emp_name |
+--------+----------+
|     14 | Neha     |
+--------+----------+
1 row in set (0.000 sec)

MariaDB [deepa]> explain select *from employees where emp_id=14;
+------+-------------+-----------+------+---------------+------+---------+------+------+-------------+
| id   | select_type | table     | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+------+-------------+-----------+------+---------------+------+---------+------+------+-------------+
|    1 | SIMPLE      | employees | ALL  | NULL          | NULL | NULL    | NULL | 19   | Using where |
+------+-------------+-----------+------+---------------+------+---------+------+------+-------------+
1 row in set (0.000 sec)
```



```
ariaDB [deepa]> drop index emp_in_asc on employees;
uery OK, 0 rows affected (1.649 sec)
ecords: 0  Duplicates: 0  Warnings: 0
```

Other SQL Queries

How to Test for NULL Values?

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the `IS NULL` and `IS NOT NULL` operators instead.

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

```
SELECT CustomerName,
ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

```
SELECT CustomerName,
ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

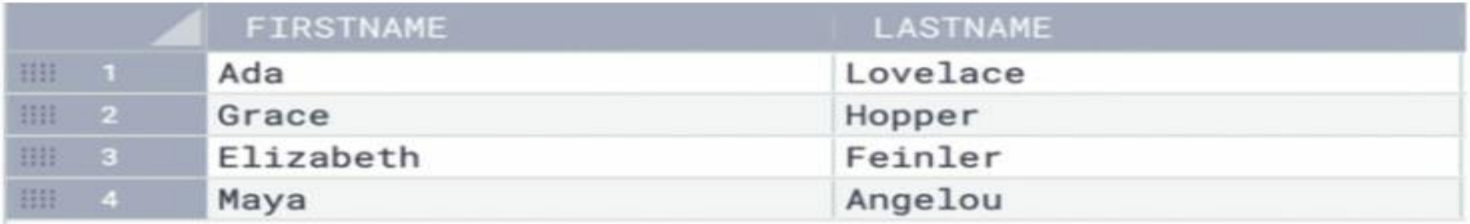
String Functions in SQL

- CONCAT: Concatenates two or more strings together.
- SUBSTRING: Extracts a substring from a larger string.
- LENGTH: Returns the length of a string.
- LOWER: Converts a string to lowercase.
- UPPER: Converts a string to uppercase
- REPLACE : To replace a string with new string.

CONCAT

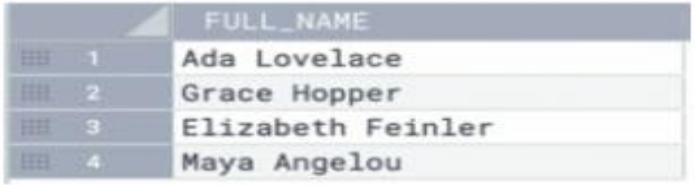
Suppose we have a table named customers with columns firstName and lastName.

We want to create a new column called full_name that combines each customer's first and last names into a single string.



	FIRSTNAME	LASTNAME
1	Ada	Lovelace
2	Grace	Hopper
3	Elizabeth	Feinler
4	Maya	Angelou

SELECT CONCAT(firstName, ' ', lastName) AS fullName FROM bookshop_customer;



	FULL_NAME
1	Ada Lovelace
2	Grace Hopper
3	Elizabeth Feinler
4	Maya Angelou

The result is a new column called "fullName" that contains the concatenated strings.

SUBSTRING

The SUBSTRING SQL string function extracts a substring or a portion of a string from a larger string.

It takes three arguments: the string to be removed from, the substring's starting position, and the substring's length. We can use the SUBSTRING function in our SQL query:

```
SELECT SUBSTRING(lastName, 1, 3) AS firstName FROM
bookshop_customer;
```

The SUBSTRING function extracts the first three characters of the "lastName" column.

LOWER / UPPER

- The LOWER SQL string function converts all the characters of a string to lowercase.
- It takes a single-string argument and returns a new string with all the characters in lowercase.

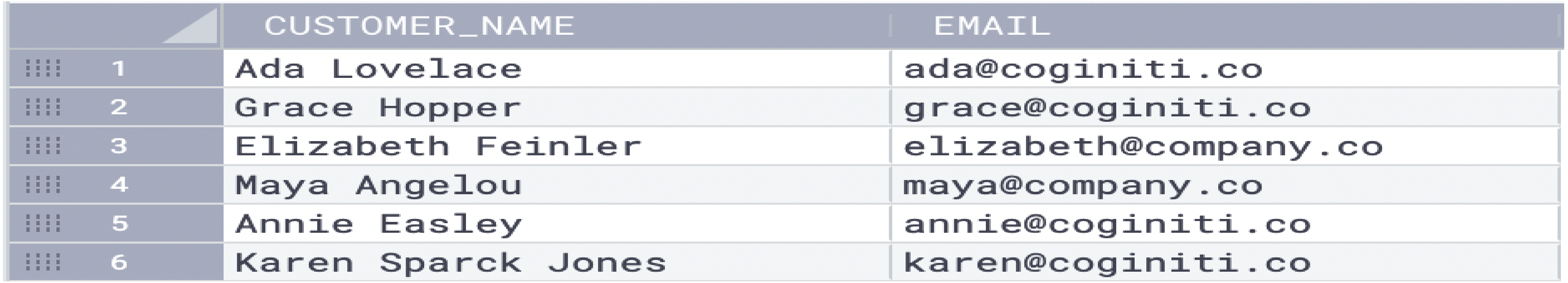
Here is an example of using the LOWER function:

We will look at the column email in the bookshop_customer table to convert all the email addresses to lowercase.

The LOWER function in our SQL query is as follows:

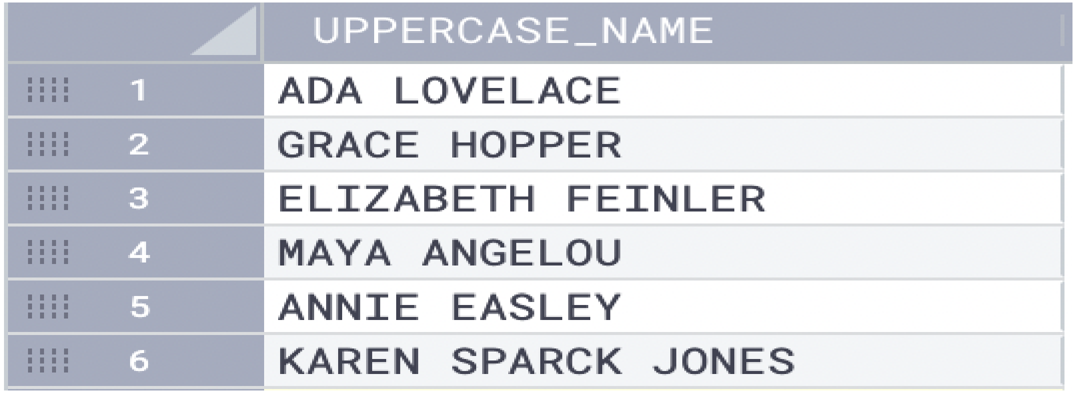
SELECT LOWER(email) AS lowercase_email FROM bookshop_customer;

In this example, the LOWER function converts all the characters in the email column to lowercase. The result is a new column called lowercase_email containing lowercase email addresses.



	CUSTOMER_NAME	EMAIL
1	Ada Lovelace	ada@coginiti.co
2	Grace Hopper	grace@coginiti.co
3	Elizabeth Feinler	elizabeth@company.co
4	Maya Angelou	maya@company.co
5	Annie Easley	annie@coginiti.co
6	Karen Sparck Jones	karen@coginiti.co

```
SELECT UPPER(title) AS uppercase_name FROM
bookshop;
```



```
   UPPERCASE_NAME
1  ADA LOVELACE
2  GRACE HOPPER
3  ELIZABETH FEINLER
4  MAYA ANGELOU
5  ANNIE EASLEY
6  KAREN SPARCK JONES
```

CHAR_LENGTH String Function

This string function returns the length of the specified word. It shows the number of characters from the word.

Example 1: This example shows the number of characters of the JavaTpoint word:

```
SELECT CHAR_LENGTH('JavaTpoint');
```

Output: 10

Example 2: This example uses CHAR_LENGTH() with the Faculty_Last_Name column of the above Faculty_Info table.

SELECT Faculty_Last_Name, CHAR_LENGTH(Faculty_Last_Name) AS Length_of_Last_Namecolumn FROM Faculty_Info;

This query shows the total number of characters of the last name of each faculty.

Faculty_Last_Name	Length_of_Last_Namecolumn
Sharma	6
Roy	3
Roy	3
Singhania	9
Sharma	6
Besas	5

CHARACTER_LENGTH String Function

This string function returns the length of the given string. It shows the number of all characters and spaces from the sentence.

Example 1: The following SELECT query shows the total number of characters and spaces of the specified string:

```
SELECT CHARACTER_LENGTH('JavaTpoint is a good company');
```

Output : 28

Example 2: The following SELECT query uses CHARACTER_LENGTH() with the Faculty_Addresss column of the above Faculty_Info table.

SELECT Faculty_Address, CHARACTER_LENGTH(Faculty_Address) AS Length_of_Addr ess_column FROM Faculty_Info;

This SQL statement shows the total number of characters and spaces of the address of each faculty.

Output:

Faculty_Address	**Length_of_Address_column**
Aman Vihar	10
Nirman Vihar	12
Sector 128	10
Vivek Vihar	11
Sarvodya Calony	15
Krishna Nagar	13

REPLACE String Function

- This string function replaces all occurrences of a substring within a string, with a new substring.
- This function performs a case-sensitive replacement.

Example 1: This example uses REPLACE to replace the word “Engineer” with “Eng” in the post column of a table.

```
SELECT REPLACE(post, “Engineer”, “Eng” FROM emp_details;
```

Nested Queries:

```
SELECT StudentName
FROM Students
WHERE
StudentID IN (SELECT StudentID FROM Grades
WHERE Subject = ‘Mathematics’ AND Score > 90
);
```

End!