جامعة القاهرة الجديدة التكنولوجية
NEW CAIRO TECHNOLOGICAL UNIVERSITY

كلية تكنولوجيا الصناعة والطاقة

PEARSON BTEC International Standards Verifier

ICT Program

Dr. Amany AbdElSamea Saeed

Lec. 6 Procedures, Triggers, Views and Index

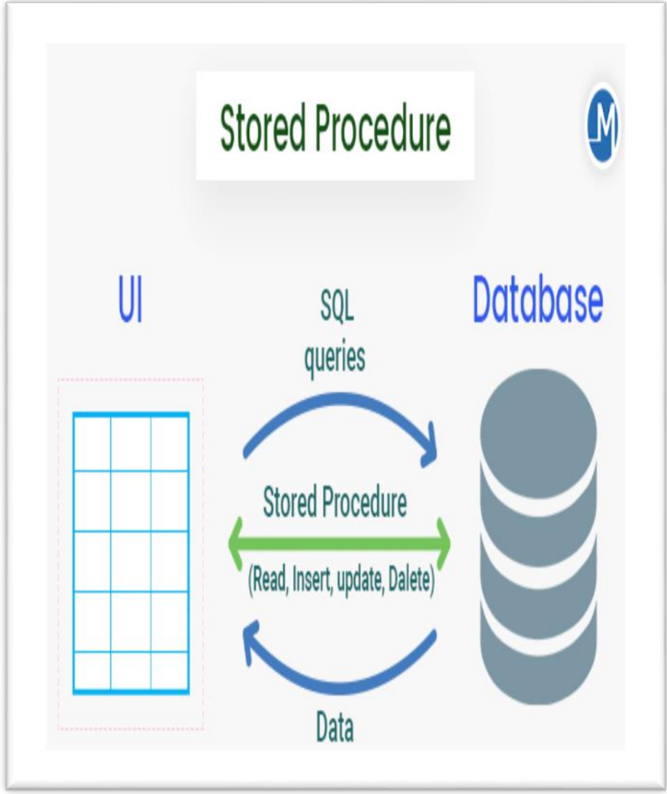
4/27/2024 12:34 PM

Outline

MYSQL Stored Procedure

MySQL Stored Procedure

- A procedure (often called a stored procedure) is a **collection of pre-compiled SQL statements** stored inside the database.
- **A procedure always contains a name, parameter lists, and SQL statements**. We can invoke the procedures by using triggers, other procedures and applications such as Java, Python, PHP, etc.
- It was first introduced in MySQL **version 5**. Presently, it can be supported by almost all relational database systems.
- A procedure is called a **recursive stored procedure** when it calls itself. Most database systems support recursive stored procedures. But, it is not supported well in MySQL.



Stored Procedure Syntax

DELIMITER &&

CREATE PROCEDURE procedure_name [[IN | **OUT** | INOUT] parameter_name datatype [, parameter datatype])]

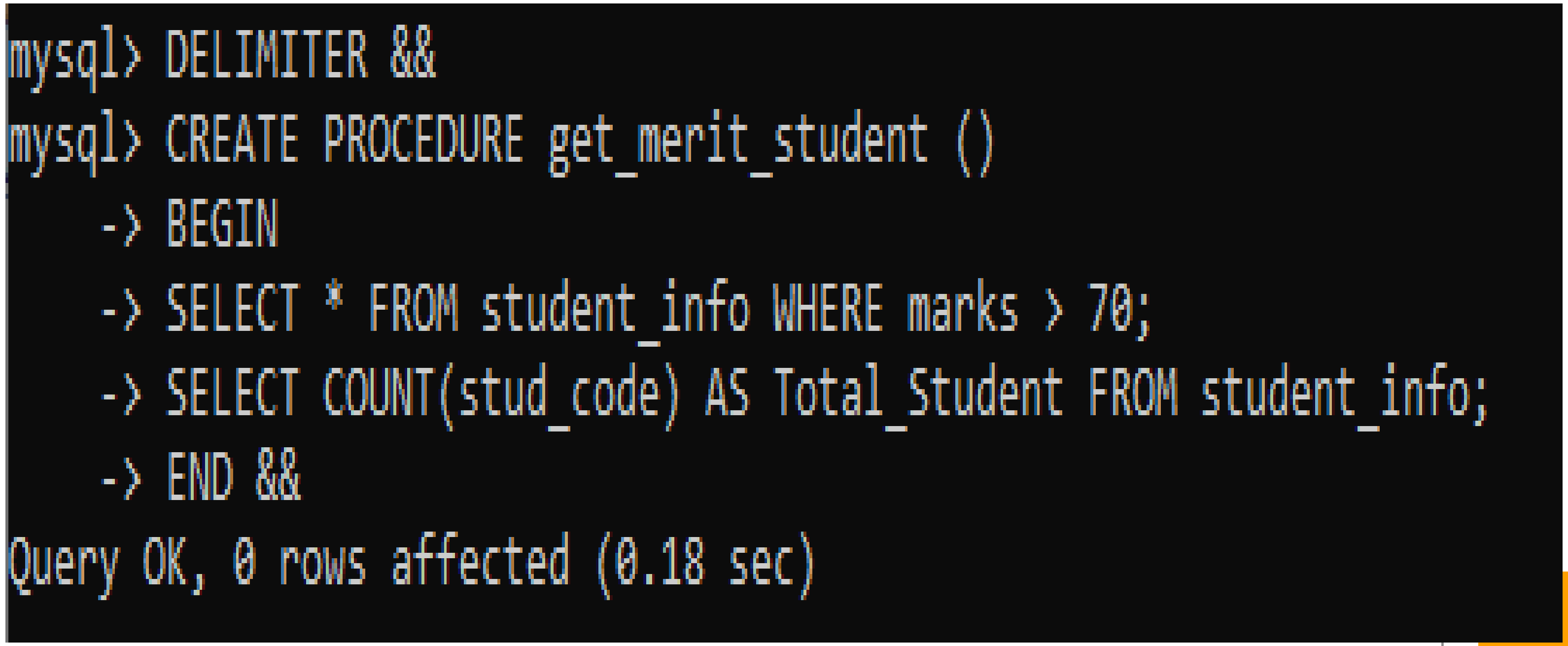
BEGIN

Declaration_section

Executable_section

END &&

DELIMITER ;



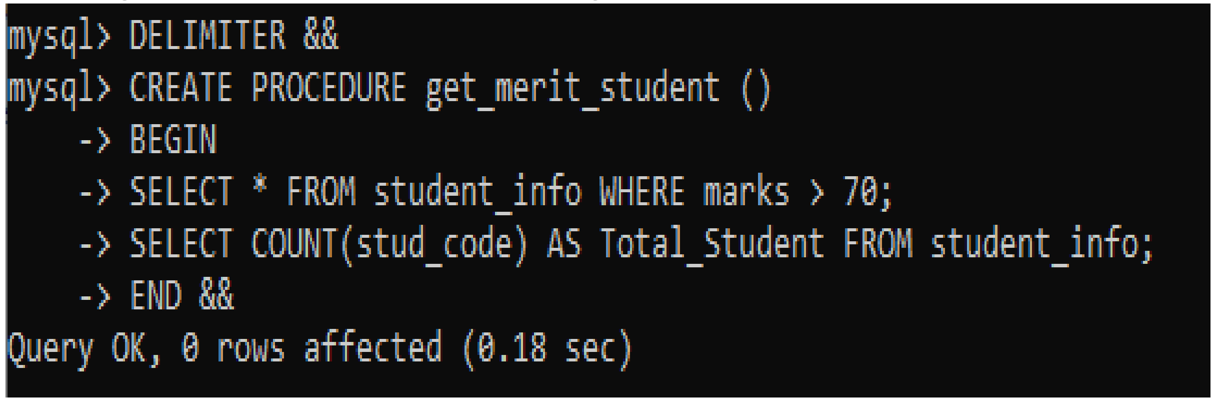
```
mysql> DELIMITER &&
mysql> CREATE PROCEDURE get_merit_student ()
    -> BEGIN
    -> SELECT * FROM student_info WHERE marks > 70;
    -> SELECT COUNT(stud_code) AS Total_Student FROM student_info;
    -> END &&
Query OK, 0 rows affected (0.18 sec)
```

Call A Procedure

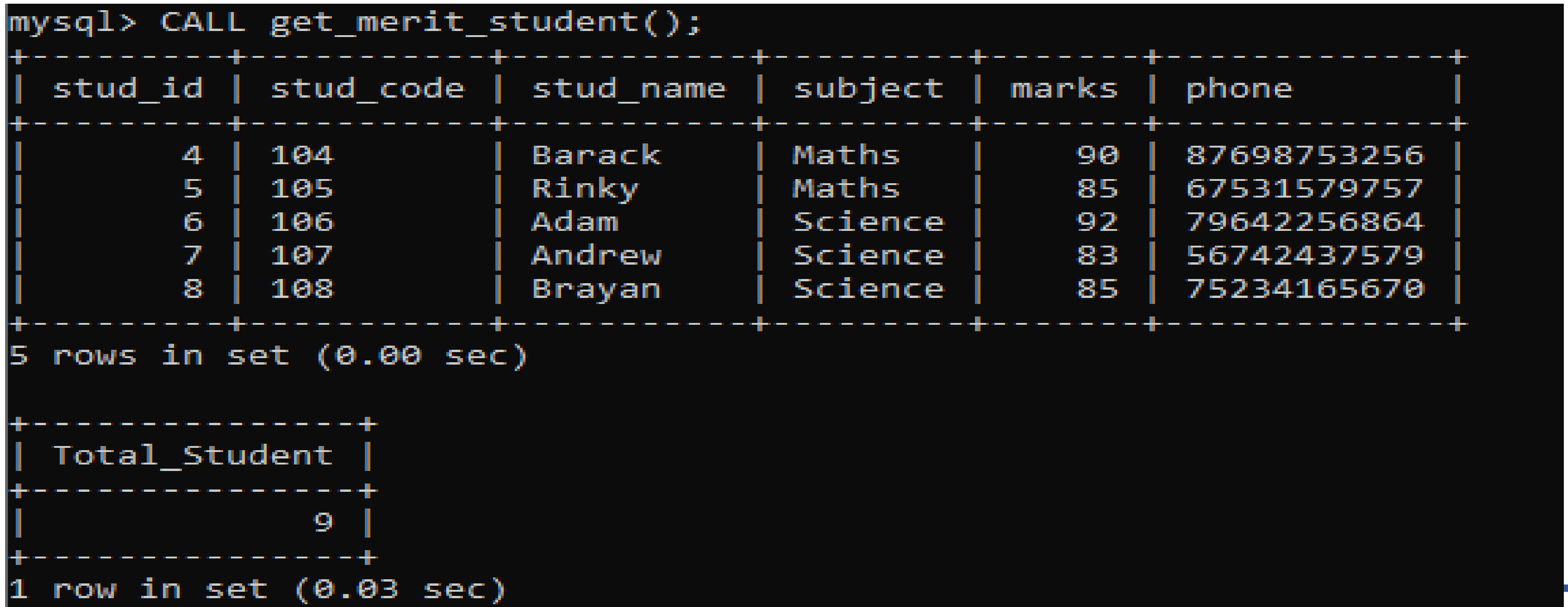
- The following syntax is used to call the stored procedure in MySQL:

```
CALL procedure_name ( parameter(s))
```

- Example of Procedure without Parameter



```
mysql> DELIMITER &&
mysql> CREATE PROCEDURE get_merit_student ()
    -> BEGIN
    -> SELECT * FROM student_info WHERE marks > 70;
    -> SELECT COUNT(stud_code) AS Total_Student FROM student_info;
    -> END &&
Query OK, 0 rows affected (0.18 sec)
```



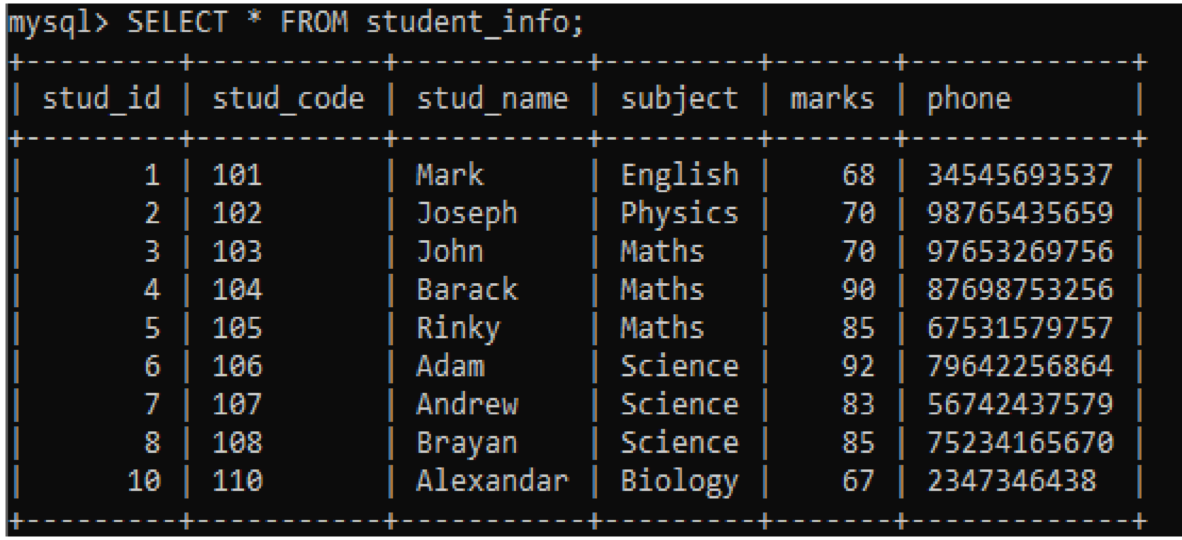
```
mysql> CALL get_merit_student();
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       4 | 104       | Barack    | Maths   |    90 | 87698753256 |
|       5 | 105       | Rinky     | Maths   |    85 | 67531579757 |
|       6 | 106       | Adam      | Science |    92 | 79642256864 |
|       7 | 107       | Andrew    | Science |    83 | 56742437579 |
|       8 | 108       | Brayan    | Science |    85 | 75234165670 |
+---------+-----------+-----------+---------+-------+-------------+
5 rows in set (0.00 sec)

+---------------+
| Total_Student |
+---------------+
|             9 |
+---------------+
1 row in set (0.03 sec)
```

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY

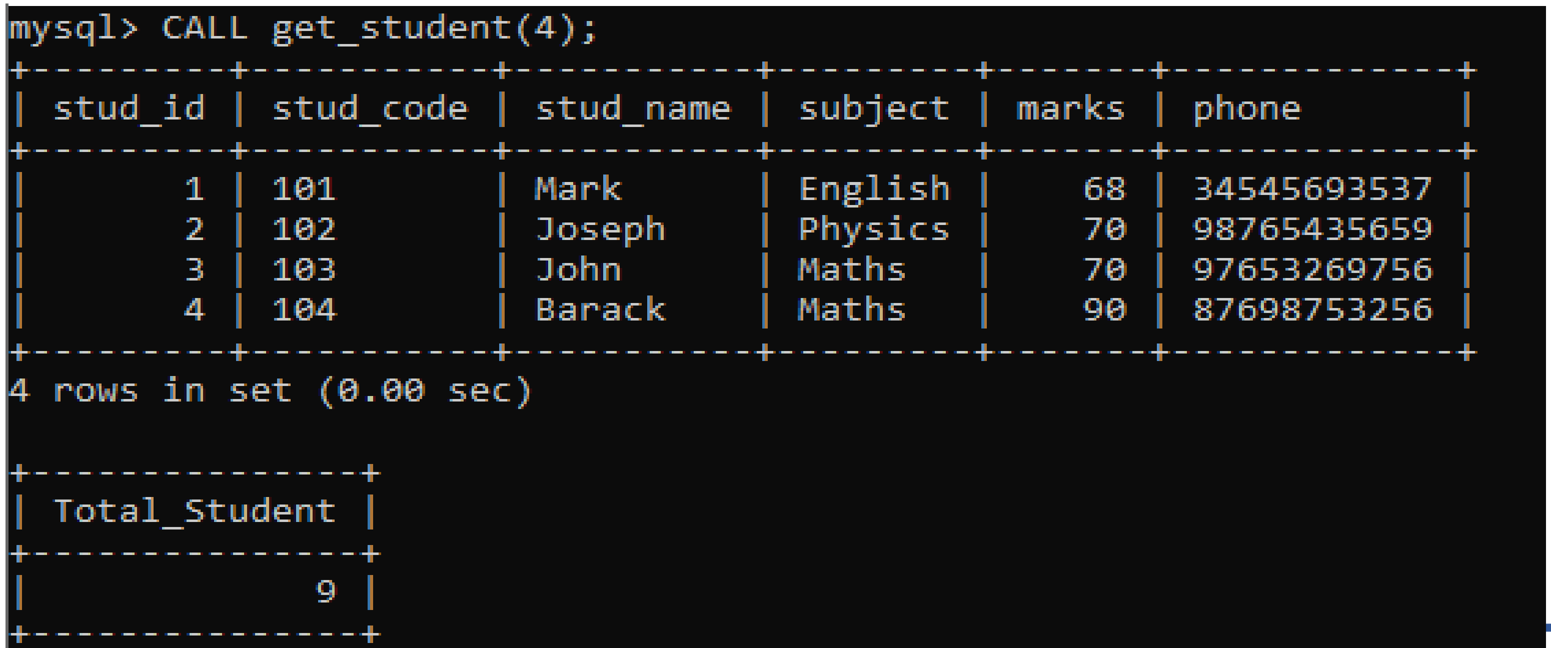
كلية تكنولوجيا الصناعة والطاقة

Procedures with IN Parameter

```
DELIMITER &&
CREATE PROCEDURE get_student (IN var1 INT)
BEGIN
   SELECT * FROM student_info LIMIT var1;
   SELECT COUNT(stud_code) AS Total_Student FROM student_info;
END &&
DELIMITER ;
```



```
mysql> SELECT * FROM student_info;
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       1 | 101       | Mark      | English |    68 | 34545693537 |
|       2 | 102       | Joseph    | Physics |    70 | 98765435659 |
|       3 | 103       | John      | Maths   |    70 | 97653269756 |
|       4 | 104       | Barack    | Maths   |    90 | 87698753256 |
|       5 | 105       | Rinky     | Maths   |    85 | 67531579757 |
|       6 | 106       | Adam      | Science |    92 | 79642256864 |
|       7 | 107       | Andrew    | Science |    83 | 56742437579 |
|       8 | 108       | Brayan    | Science |    85 | 75234165670 |
|      10 | 110       | Alexandar | Biology |    67 | 2347346438  |
+---------+-----------+-----------+---------+-------+-------------+
```

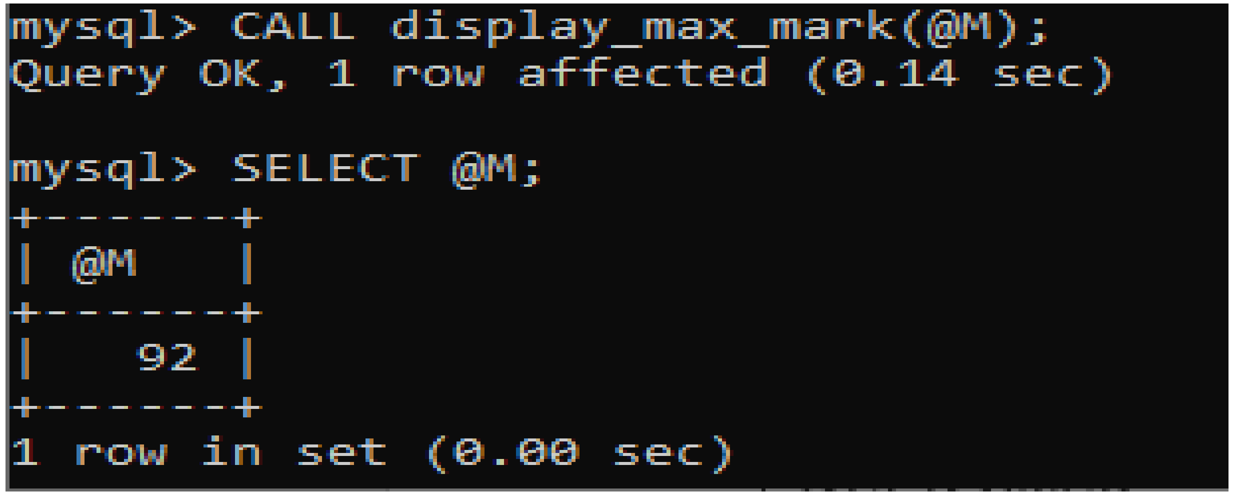


```
mysql> CALL get_student(4);
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       1 | 101       | Mark      | English |    68 | 34545693537 |
|       2 | 102       | Joseph    | Physics |    70 | 98765435659 |
|       3 | 103       | John      | Maths   |    70 | 97653269756 |
|       4 | 104       | Barack    | Maths   |    90 | 87698753256 |
+---------+-----------+-----------+---------+-------+-------------+
4 rows in set (0.00 sec)

+---------------+
| Total_Student |
+---------------+
|             9 |
+---------------+
```

Procedures with OUT Parameter

```
DELIMITER &&
CREATE PROCEDURE display_max_mark (OUT highestmark INT)
BEGIN
   SELECT MAX(marks) INTO highestmark FROM student_info;
END &&
DELIMITER ;
```

- The OUT parameter tells the database systems that its value goes out from the procedures. Now, we will pass its value to a session variable **@M** in the CALL statement as follows:

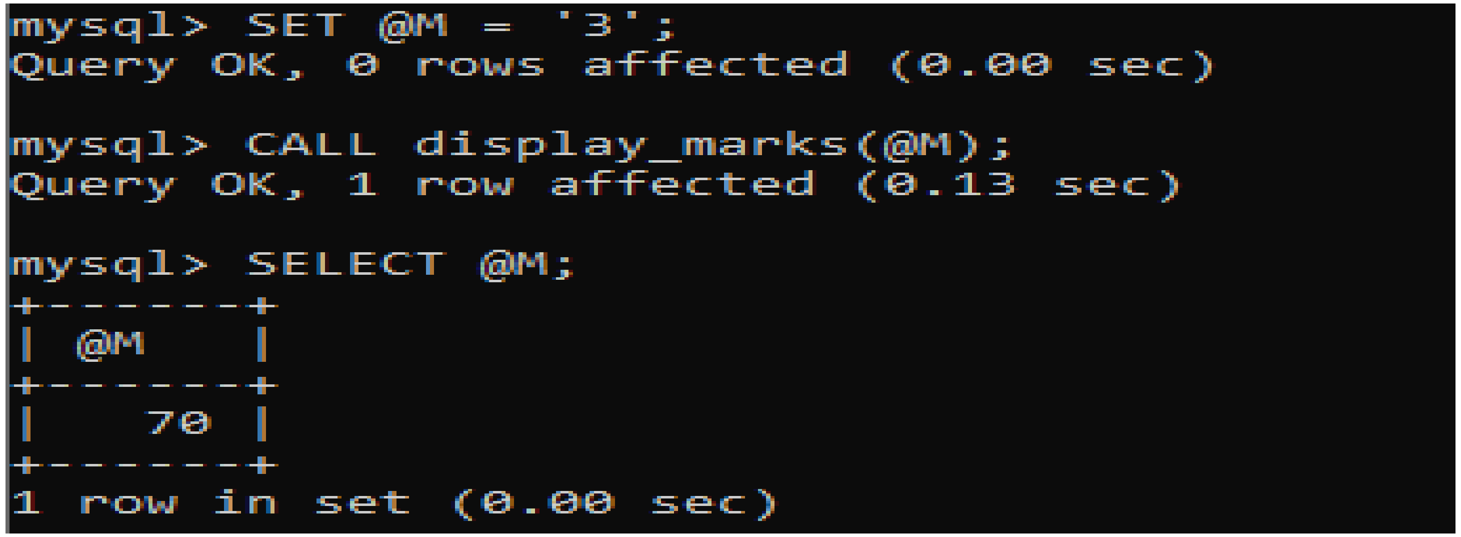


```
mysql> CALL display_max_mark(@M);
Query OK, 1 row affected (0.14 sec)

mysql> SELECT @M;
+-------+
| @M    |
+-------+
|    92 |
+-------+
1 row in set (0.00 sec)
```

جامعة القاهرة الجديدة التكنولوجية
NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

Procedures with INOUT Parameter

```
DELIMITER &&
CREATE PROCEDURE display_marks (INOUT var1 INT)
BEGIN
   SELECT marks INTO var1 FROM student_info WHERE stud_id = var1;
END &&
DELIMITER ;
```



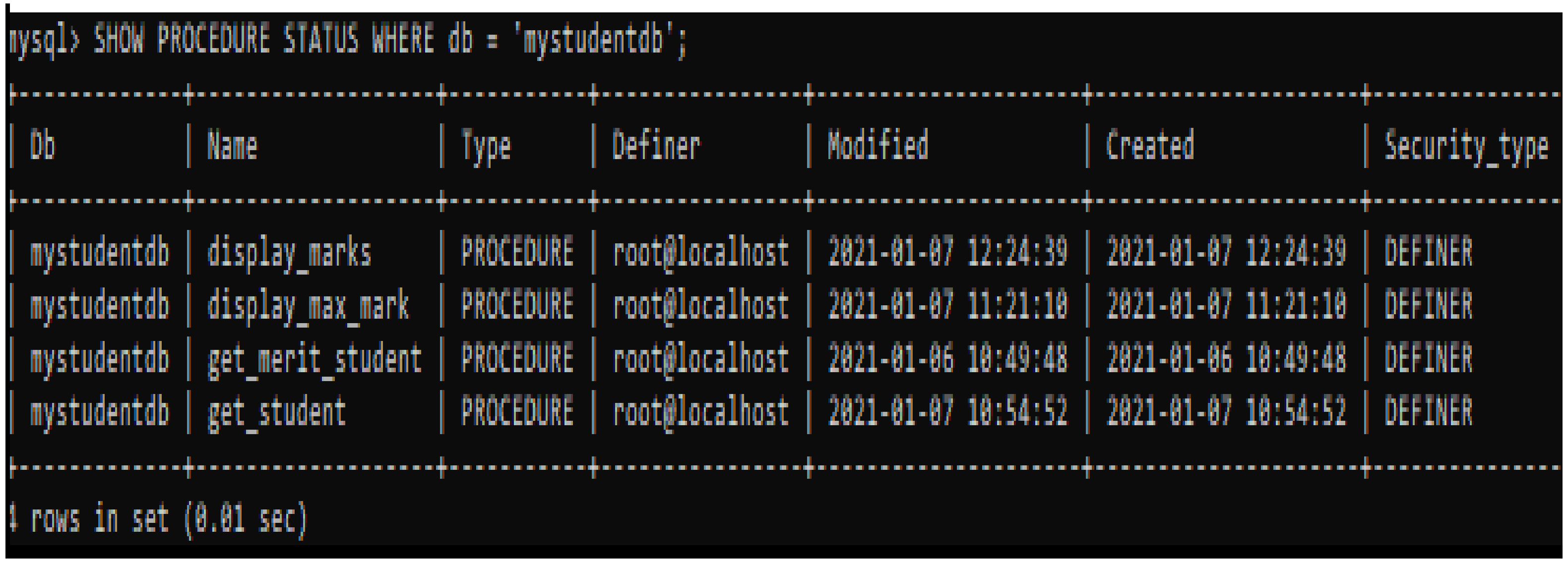
```
mysql> SET @M = '3';
Query OK, 0 rows affected (0.00 sec)

mysql> CALL display_marks(@M);
Query OK, 1 row affected (0.13 sec)

mysql> SELECT @M;
+------+
| @M   |
+------+
|   70 |
+------+
1 row in set (0.00 sec)
```

جامعة القاهرة الجديدة التكنولوجية NCT NEW CAIRO TECHNOLOGICAL UNIVERSITY

كلية تكنولوجيا الصناعة والطاقة

Show or List Stored Procedures in MySQL

```
SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE search_condition]
```



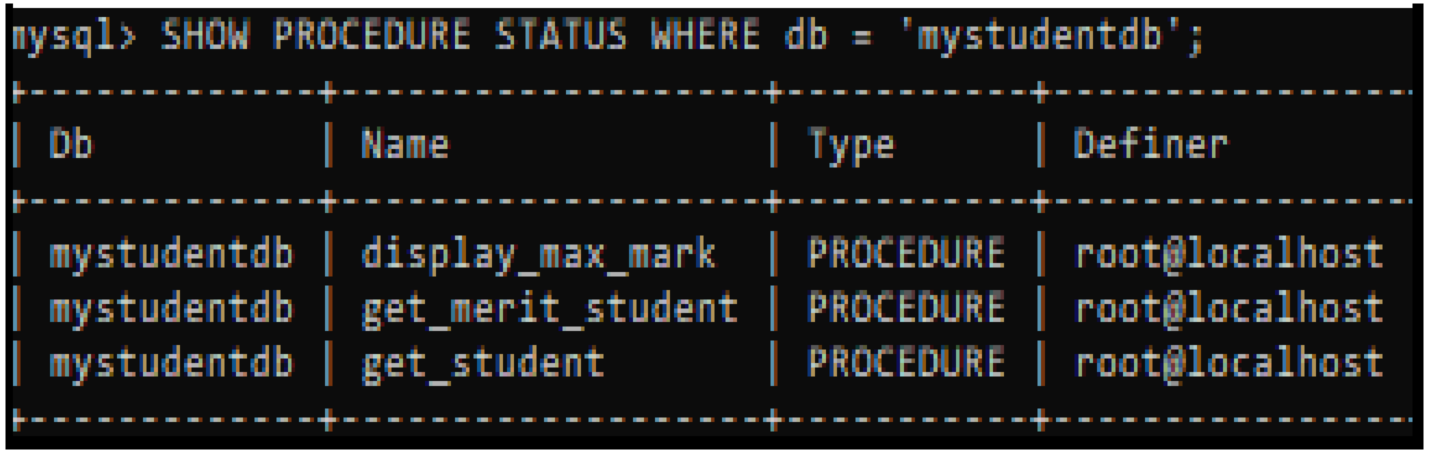
```
nysql> SHOW PROCEDURE STATUS WHERE db = 'mystudentdb';
+-------------+-------------------+-----------+----------------+---------------------+---------------------+---------------
| Db          | Name              | Type      | Definer        | Modified            | Created             | Security_type
+-------------+-------------------+-----------+----------------+---------------------+---------------------+---------------
| mystudentdb | display_marks     | PROCEDURE | root@localhost | 2021-01-07 12:24:39 | 2021-01-07 12:24:39 | DEFINER
| mystudentdb | display_max_mark  | PROCEDURE | root@localhost | 2021-01-07 11:21:10 | 2021-01-07 11:21:10 | DEFINER
| mystudentdb | get_merit_student | PROCEDURE | root@localhost | 2021-01-06 10:49:48 | 2021-01-06 10:49:48 | DEFINER
| mystudentdb | get_student       | PROCEDURE | root@localhost | 2021-01-07 10:54:52 | 2021-01-07 10:54:52 | DEFINER
+-------------+-------------------+-----------+----------------+---------------------+---------------------+---------------
4 rows in set (0.01 sec)
```

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

Delete/Drop Stored Procedures

```
DROP PROCEDURE [ IF EXISTS ] procedure_name;
```

```
mysql>  DROP PROCEDURE [IF EXISTS] display_mark;
```

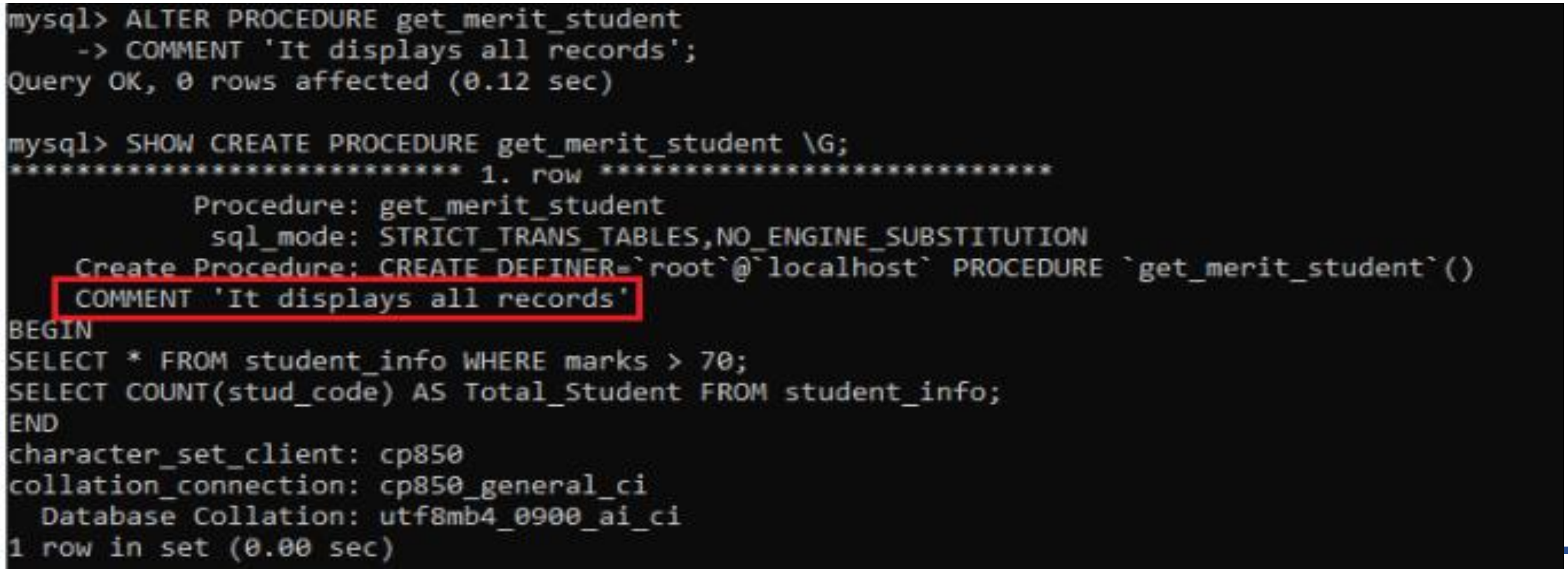


```
mysql> SHOW PROCEDURE STATUS WHERE db = 'mystudentdb';
+-------------+-------------------+-----------+-----------------
| Db          | Name              | Type      | Definer
+-------------+-------------------+-----------+-----------------
| mystudentdb | display_max_mark  | PROCEDURE | root@localhost
| mystudentdb | get_merit_student | PROCEDURE | root@localhost
| mystudentdb | get_student       | PROCEDURE | root@localhost
+-------------+-------------------+-----------+-----------------
```

Alter Procedure

The below statement is used to change the characteristics of a procedure but not the actual procedure

```
ALTER PROCEDURE procedure_name [characteristics ...]


characteristics: {
   COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}
```



```
mysql> ALTER PROCEDURE get_merit_student
    -> COMMENT 'It displays all records';
Query OK, 0 rows affected (0.12 sec)

mysql> SHOW CREATE PROCEDURE get_merit_student \G;
*************************** 1. row ***************************
           Procedure: get_merit_student
            sql_mode: STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION
    Create Procedure: CREATE DEFINER=`root`@`localhost` PROCEDURE `get_merit_student`()
    COMMENT 'It displays all records'
BEGIN
SELECT * FROM student_info WHERE marks > 70;
SELECT COUNT(stud_code) AS Total_Student FROM student_info;
END
character_set_client: cp850
collation_connection: cp850_general_ci
  Database Collation: utf8mb4_0900_ai_ci
1 row in set (0.00 sec)
```

MySQL Trigger

Introduction to Triggers

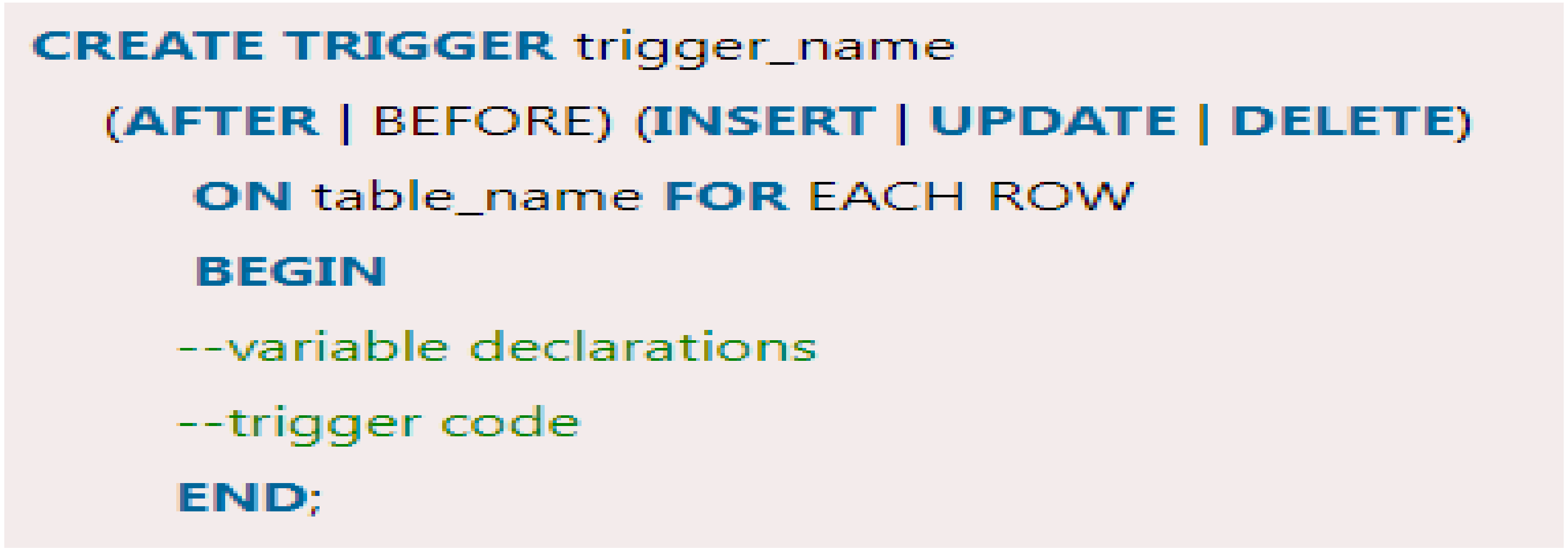
- **Trigger is a special type of stored procedure that is invoked automatically in response to an event**.

- The main difference between the trigger and procedure is that a trigger is called automatically when a data modification event is made against a table. In contrast, a stored procedure must be called explicitly.
- One drawback to using triggers instead of stored procedures is that they cannot accept parameters.

- When the user wants to modify data using a DML event then the DML trigger is executed. In other words, a DML trigger is used for INSERT, DELETE and UPDATE statements of a table or view.

Create Trigger

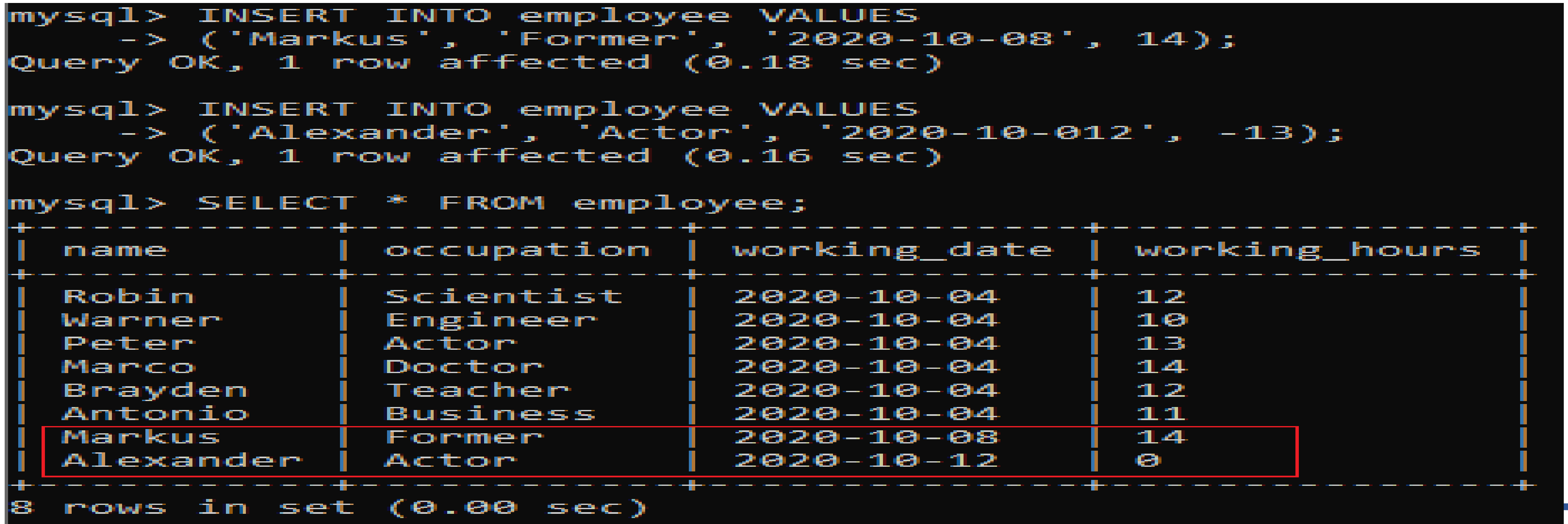
- **CREATE TRIGGER** statement for creating a new trigger in MySQL. Below is the syntax of creating a trigger in MySQL:



```
CREATE TRIGGER trigger_name
    (AFTER | BEFORE) (INSERT | UPDATE | DELETE)
        ON table_name FOR EACH ROW
        BEGIN
       --variable declarations
       --trigger code
       END;
```

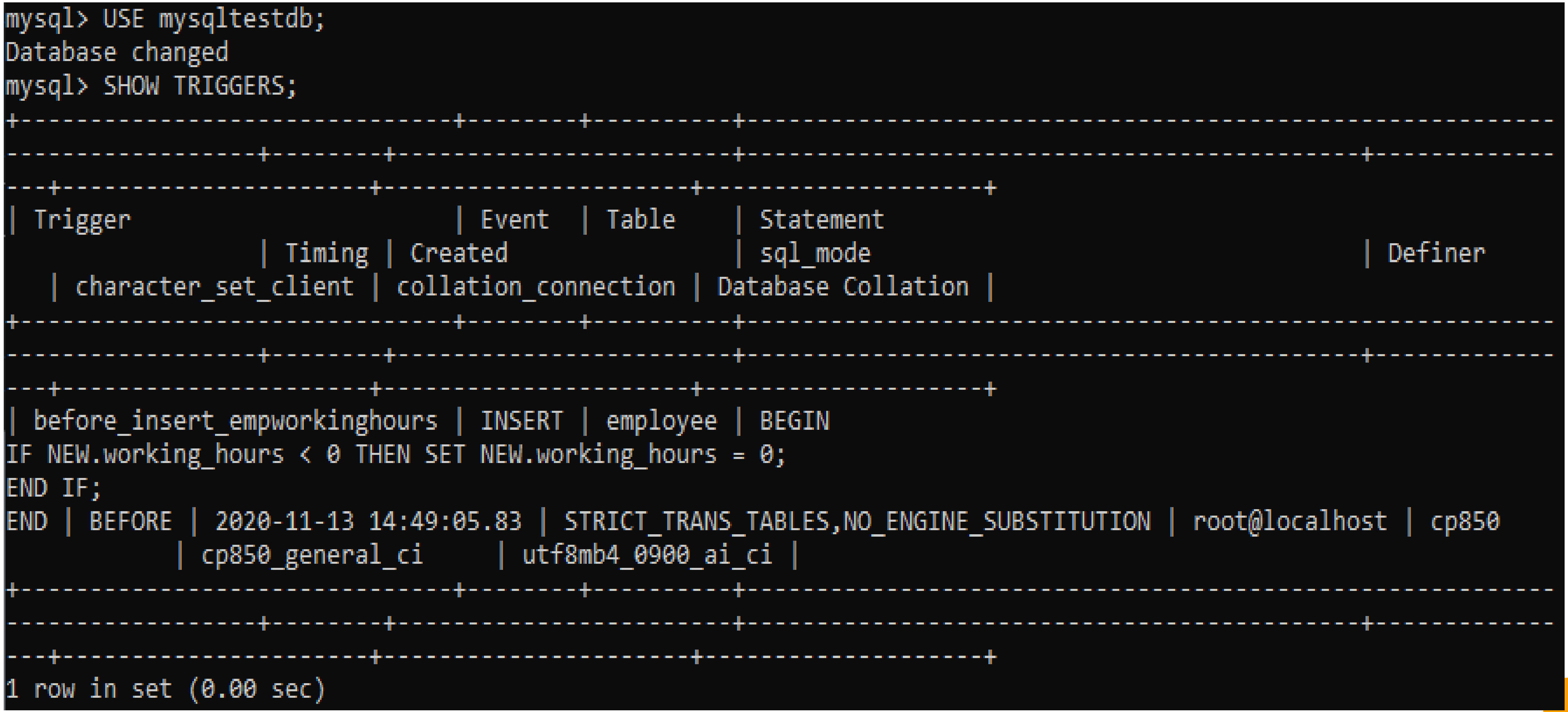
Trigger Example

```
mysql> DELIMITER //
mysql> Create Trigger before_insert_empworkinghours
BEFORE INSERT ON employee FOR EACH ROW
BEGIN
IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;
END IF;
END //
```



```
mysql> INSERT INTO employee VALUES
    -> ('Markus', 'Former', '2020-10-08', 14);
Query OK, 1 row affected (0.18 sec)

mysql> INSERT INTO employee VALUES
    -> ('Alexander', 'Actor', '2020-10-012', -13);
Query OK, 1 row affected (0.16 sec)

mysql> SELECT * FROM employee;
+-----------+------------+--------------+---------------+
| name      | occupation | working_date | working_hours |
+-----------+------------+--------------+---------------+
| Robin     | Scientist  | 2020-10-04   | 12            |
| Warner    | Engineer   | 2020-10-04   | 10            |
| Peter     | Actor      | 2020-10-04   | 13            |
| Marco     | Doctor     | 2020-10-04   | 14            |
| Brayden   | Teacher    | 2020-10-04   | 12            |
| Antonio   | Business   | 2020-10-04   | 11            |
| Markus    | Former     | 2020-10-08   | 14            |
| Alexander | Actor      | 2020-10-12   | 0             |
+-----------+------------+--------------+---------------+
8 rows in set (0.00 sec)
```

Show/List Triggers



```
mysql> USE mysqltestdb;
Database changed
mysql> SHOW TRIGGERS;
+--------------------------------+--------+----------+-----------------------------------------------------------
-------------------+--------+------------------------+----------------------------------------------+------------
---+---------------------+----------------------+--------------------+
| Trigger                        | Event  | Table    | Statement
                   | Timing | Created                | sql_mode                                     | Definer
   | character_set_client | collation_connection | Database Collation |
+--------------------------------+--------+----------+-----------------------------------------------------------
-------------------+--------+------------------------+----------------------------------------------+------------
---+---------------------+----------------------+--------------------+
| before_insert_empworkinghours | INSERT | employee | BEGIN
IF NEW.working_hours < 0 THEN SET NEW.working_hours = 0;
END IF;
END | BEFORE | 2020-11-13 14:49:05.83 | STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION | root@localhost | cp850
            | cp850_general_ci      | utf8mb4_0900_ai_ci |
+--------------------------------+--------+----------+-----------------------------------------------------------
-------------------+--------+------------------------+----------------------------------------------+------------
---+---------------------+----------------------+--------------------+
1 row in set (0.00 sec)
```

Show Triggers Using Pattern Matching

- The syntax to use pattern matching with show trigger command:

```
mysql> SHOW TRIGGERS LIKE pattern;
OR,
mysql> SHOW TRIGGERS FROM database_name LIKE pattern;
```

- If we want to list/show trigger names based on specific search condition, we can use the WHERE clause as follows:

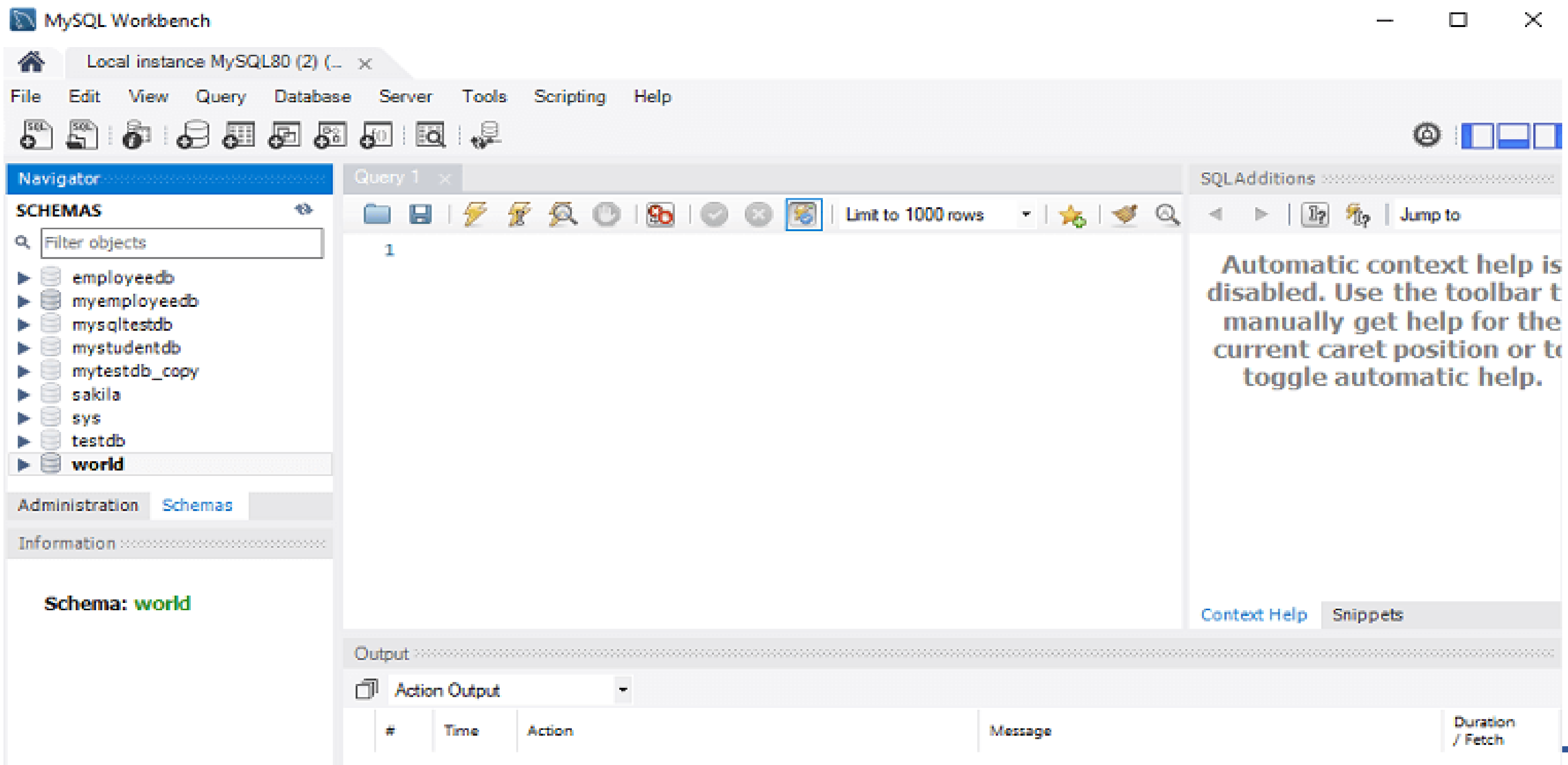
```
mysql> SHOW TRIGGERS WHERE search_condition;
OR,
mysql> SHOW TRIGGERS FROM database_name WHERE search_condition;
```

NOTE: It is to note that we must have a SUPER privilege to execute the SHOW TRIGGERS statement.

Show Triggers in MySQL Workbench

جامعة القاهرة الجديدة التكنولوجية NCT NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

- Launch the MySQL Workbench and log in using the username and password

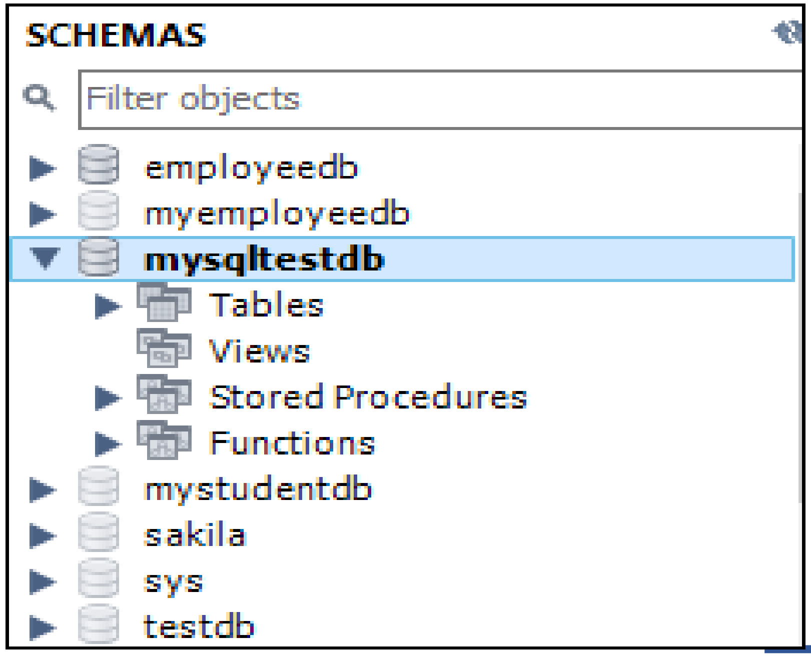


Schema: world

Automatic context help is disabled. Use the toolbar t
manually get help for the
current caret position or t
toggle automatic help.

Show Triggers in MySQL Workbench

- Go to the Navigation tab and click on the **Schema menu** that contains all the databases available in the MySQL server.
- Select the database (for example, **mysqltestdb**), double click on it, and show the **sub-menu** containing Tables, Views, Functions, and Stored Procedures. See the below screen.

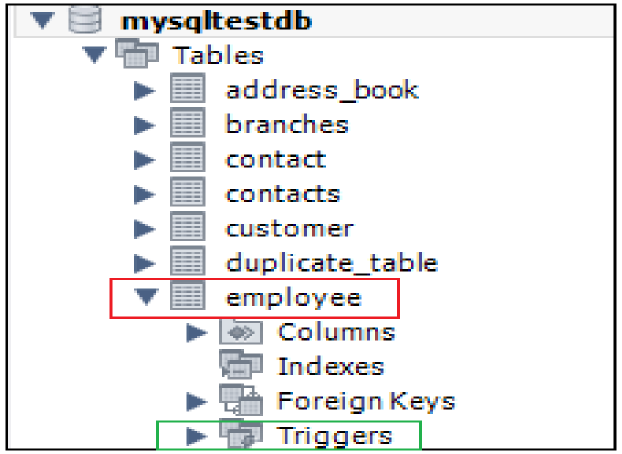


جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY

كلية تكنولوجيا الصناعة والطاقة

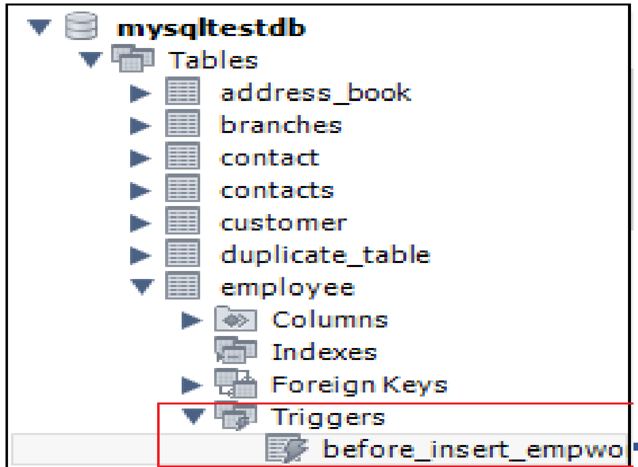
Show Triggers in MySQL Workbench

- Click on the **Tables sub-menu** and select the table on which you have created a trigger.



```
▼ mysqltestdb
  ▼ Tables
    ▶ address_book
    ▶ branches
    ▶ contact
    ▶ contacts
    ▶ customer
    ▶ duplicate_table
    ▼ employee
      ▶ Columns
        Indexes
      ▶ Foreign Keys
      ▶ Triggers
```

- Clicking on the **Triggers sub-menu**, we can see all triggers associated with the selected table.



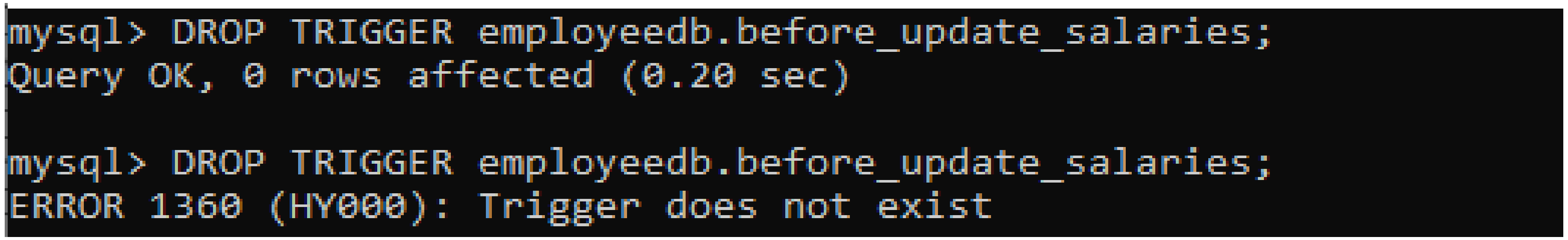
```
▼ mysqltestdb
  ▼ Tables
    ▶ address_book
    ▶ branches
    ▶ contact
    ▶ contacts
    ▶ customer
    ▶ duplicate_table
    ▼ employee
      ▶ Columns
        Indexes
      ▶ Foreign Keys
      ▼ Triggers
          before_insert_empwo
```

Drop Trigger

- We can drop an existing trigger from the database by using the DROP TRIGGER statement with the below syntax:

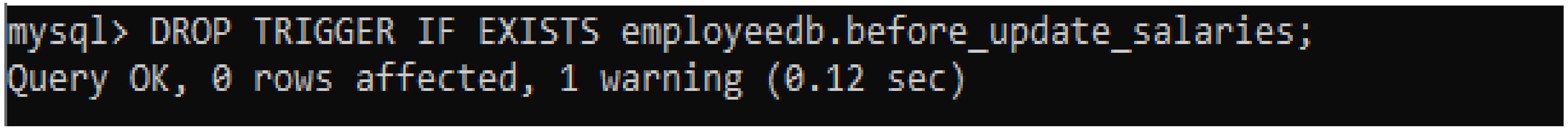
```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;
```

- Example:



```
mysql> DROP TRIGGER employeedb.before_update_salaries;
Query OK, 0 rows affected (0.20 sec)

mysql> DROP TRIGGER employeedb.before_update_salaries;
ERROR 1360 (HY000): Trigger does not exist
```

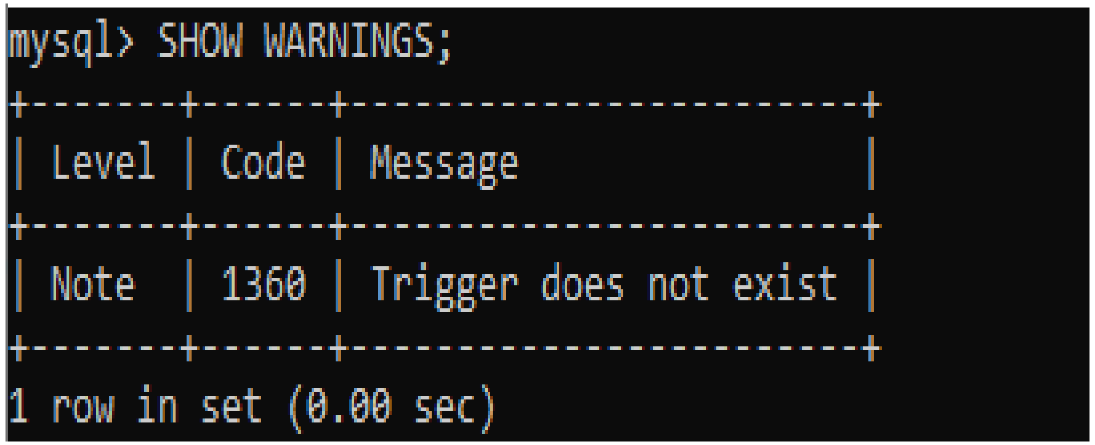
- If we execute the above statement again with an **IF EXISTS** clause, it will return the warning message instead of producing an error



```
mysql> DROP TRIGGER IF EXISTS employeedb.before_update_salaries;
Query OK, 0 rows affected, 1 warning (0.12 sec)
```

Show Warning

- We can execute the **SHOW WARNING** statement that generates a **NOTE** for a non-existent trigger when using IF EXISTS



```
mysql> SHOW WARNINGS;
+-------+------+-------------------------+
| Level | Code | Message                 |
+-------+------+-------------------------+
| Note  | 1360 | Trigger does not exist |
+-------+------+-------------------------+
1 row in set (0.00 sec)
```

BEFORE INSERT TRIGGER

- Syntax

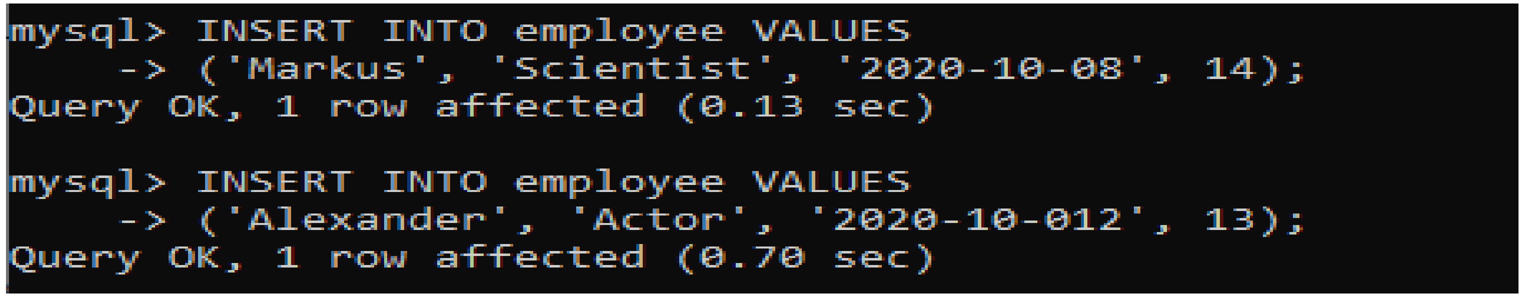
```
CREATE TRIGGER trigger_name
BEFORE INSERT
ON table_name FOR EACH ROW
Trigger_body ;
```

- If we want to execute multiple statements, we will use the **BEGIN END** block that contains a set of queries to define the logic for the trigger

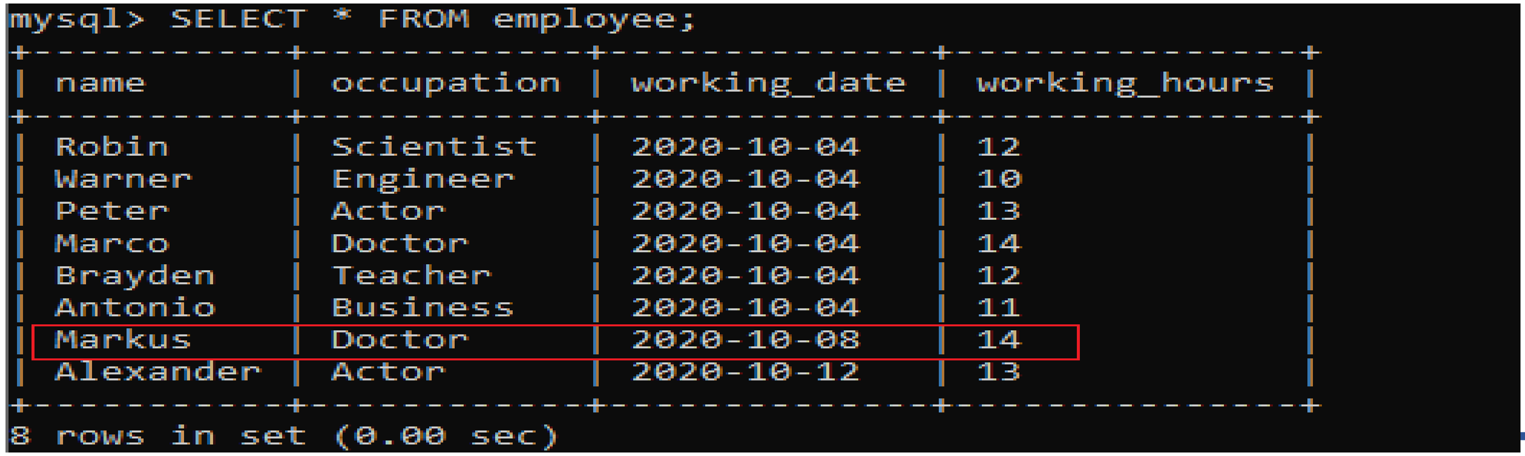
```
DELIMITER $$
CREATE TRIGGER trigger_name BEFORE INSERT
ON table_name FOR EACH ROW
BEGIN
  variable declarations
  trigger code
END$$
DELIMITER ;
```

Example

```
mysql> DELIMITER //
mysql> Create Trigger before_insert_occupation
BEFORE INSERT ON employee FOR EACH ROW
BEGIN
IF NEW.occupation = 'Scientist' THEN SET NEW.occupation = 'Doctor';
END IF;
END //
```



```
mysql> INSERT INTO employee VALUES
    -> ('Markus', 'Scientist', '2020-10-08', 14);
Query OK, 1 row affected (0.13 sec)

mysql> INSERT INTO employee VALUES
    -> ('Alexander', 'Actor', '2020-10-012', 13);
Query OK, 1 row affected (0.70 sec)
```



```
mysql> SELECT * FROM employee;
+-----------+------------+--------------+---------------+
| name      | occupation | working_date | working_hours |
+-----------+------------+--------------+---------------+
| Robin     | Scientist  | 2020-10-04   | 12            |
| Warner    | Engineer   | 2020-10-04   | 10            |
| Peter     | Actor      | 2020-10-04   | 13            |
| Marco     | Doctor     | 2020-10-04   | 14            |
| Brayden   | Teacher    | 2020-10-04   | 12            |
| Antonio   | Business   | 2020-10-04   | 11            |
| Markus    | Doctor     | 2020-10-08   | 14            |
| Alexander | Actor      | 2020-10-12   | 13            |
+-----------+------------+--------------+---------------+
8 rows in set (0.00 sec)
```

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY

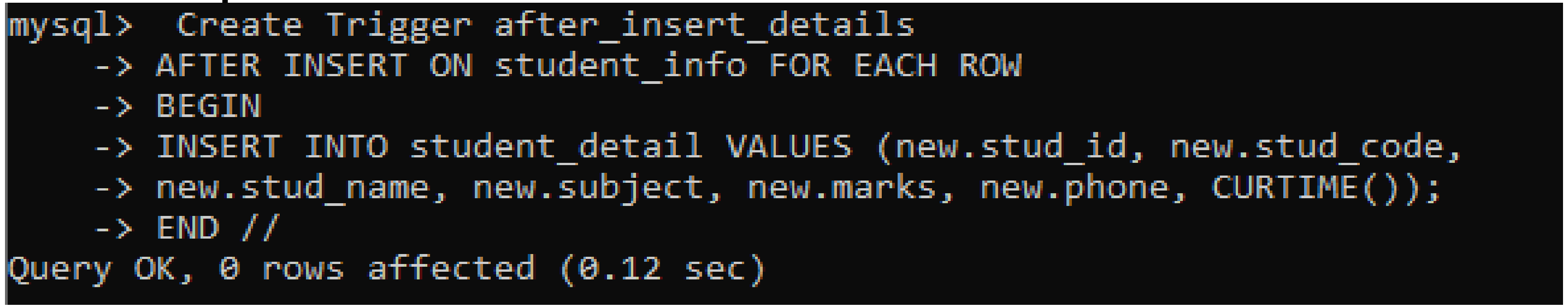
كلية تكنولوجيا الصناعة والطاقة

AFTER INSERT Trigger

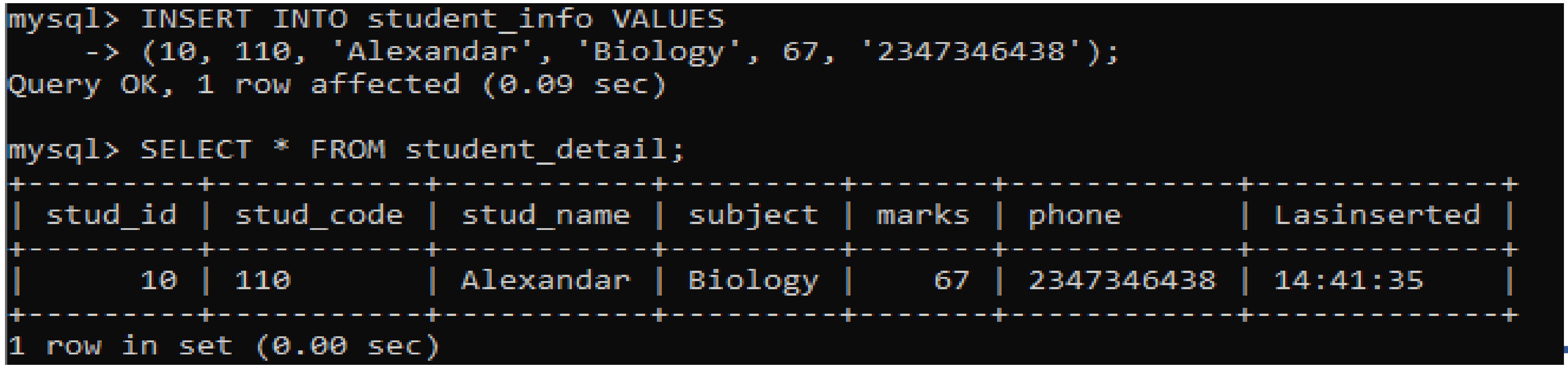
- Syntax

```
CREATE TRIGGER trigger_name
AFTER INSERT
ON table_name FOR EACH ROW
trigger_body ;
```

- Example:



```
mysql>  Create Trigger after_insert_details
    -> AFTER INSERT ON student_info FOR EACH ROW
    -> BEGIN
    -> INSERT INTO student_detail VALUES (new.stud_id, new.stud_code,
    -> new.stud_name, new.subject, new.marks, new.phone, CURTIME());
    -> END //
Query OK, 0 rows affected (0.12 sec)
```

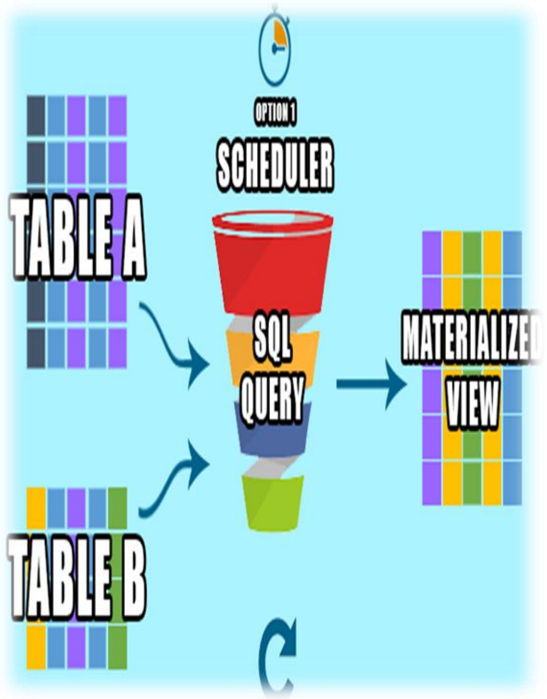


```
mysql> INSERT INTO student_info VALUES
    -> (10, 110, 'Alexandar', 'Biology', 67, '2347346438');
Query OK, 1 row affected (0.09 sec)

mysql> SELECT * FROM student_detail;
+---------+-----------+-----------+---------+-------+------------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone      | Lasinserted |
+---------+-----------+-----------+---------+-------+------------+-------------+
|      10 | 110       | Alexandar | Biology |    67 | 2347346438 | 14:41:35    |
+---------+-----------+-----------+---------+-------+------------+-------------+
1 row in set (0.00 sec)
```

MySQL Views

Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- Views, which are a type of virtual tables allow users to do the following −
  - Structure data.
  - Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
  - Summarize data from various tables which can be used to generate reports.



Creating Views

- Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.
- The basic **CREATE VIEW** syntax is as follows –

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

- You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

Example Views

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS;
```

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table.

```
SQL > SELECT * FROM CUSTOMERS_VIEW;
```

This would produce the following result.

```
+----------+-----+
| name     | age |
+----------+-----+
| Ramesh   |  32 |
| Khilan   |  25 |
| kaushik  |  23 |
| Chaitali |  25 |
| Hardik   |  27 |
| Komal    |  22 |
| Muffy    |  24 |
+----------+-----+
```

The WITH CHECK OPTION

- The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.
- If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

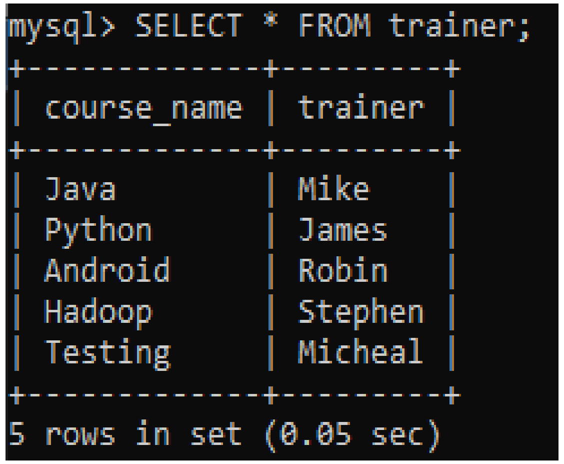
```
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM  CUSTOMERS
WHERE age IS NOT NULL
WITH CHECK OPTION;
```

- The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

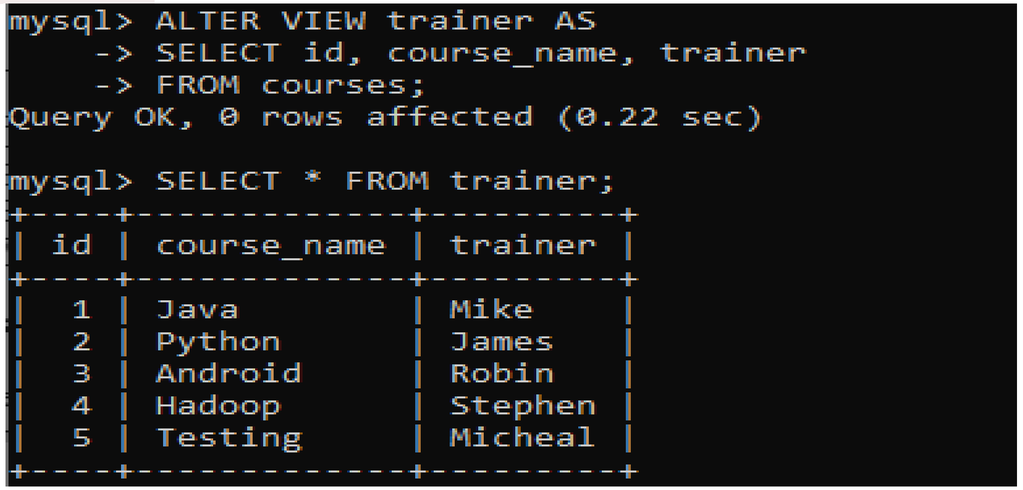
Update VIEW

- In MYSQL, the ALTER VIEW statement is used to modify or update the already created VIEW without dropping it.
- **Syntax:**

```
ALTER VIEW view_name AS
SELECT columns
FROM table
WHERE conditions;
```



```
mysql> SELECT * FROM trainer;
+-------------+---------+
| course_name | trainer |
+-------------+---------+
| Java        | Mike    |
| Python      | James   |
| Android     | Robin   |
| Hadoop      | Stephen |
| Testing     | Micheal |
+-------------+---------+
5 rows in set (0.05 sec)
```

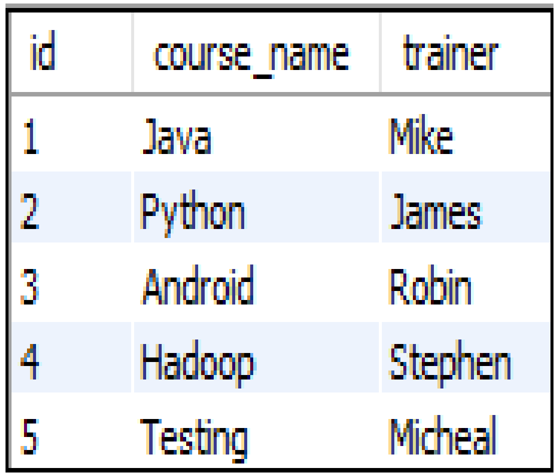


```
mysql> ALTER VIEW trainer AS
    -> SELECT id, course_name, trainer
    -> FROM courses;
Query OK, 0 rows affected (0.22 sec)

mysql> SELECT * FROM trainer;
+----+-------------+---------+
| id | course_name | trainer |
+----+-------------+---------+
|  1 | Java        | Mike    |
|  2 | Python      | James   |
|  3 | Android     | Robin   |
|  4 | Hadoop      | Stephen |
|  5 | Testing     | Micheal |
+----+-------------+---------+
```

Create View with Multiple Tables

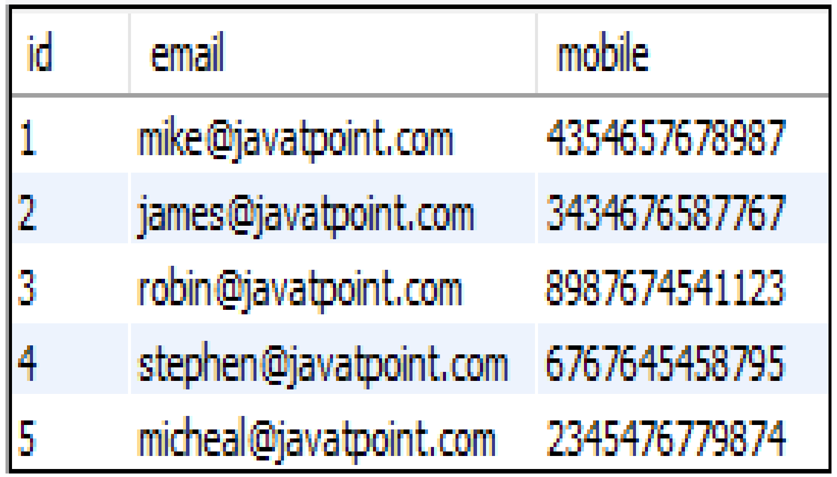
- Here, we will see the complex example of view creation that involves multiple tables and uses a **join** clause.
- Suppose we have two sample table as shown below:

Table: course



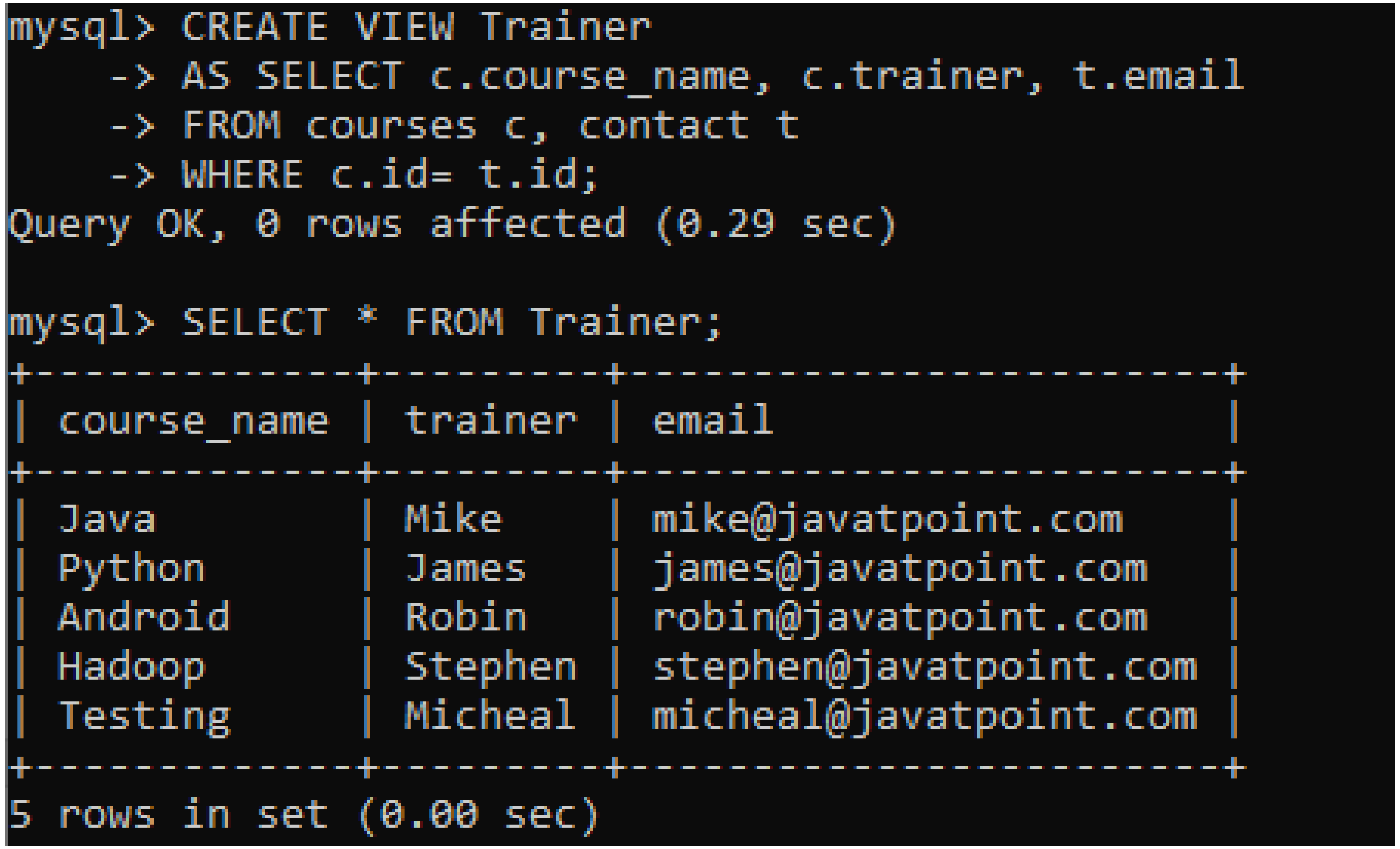
id	course_name	trainer
1	Java	Mike
2	Python	James
3	Android	Robin
4	Hadoop	Stephen
5	Testing	Micheal

Table: contact



id	email	mobile
1	mike@javatpoint.com	4354657678987
2	james@javatpoint.com	3434676587767
3	robin@javatpoint.com	8987674541123
4	stephen@javatpoint.com	6767645458795
5	micheal@javatpoint.com	2345476779874

Example



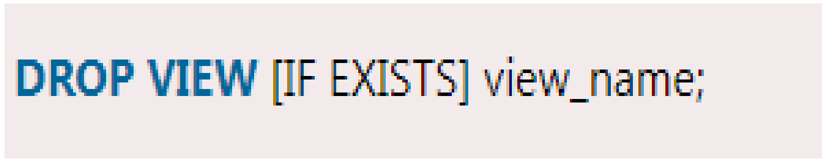
```
mysql> CREATE VIEW Trainer
    -> AS SELECT c.course_name, c.trainer, t.email
    -> FROM courses c, contact t
    -> WHERE c.id= t.id;
Query OK, 0 rows affected (0.29 sec)

mysql> SELECT * FROM Trainer;
+-------------+---------+------------------------+
| course_name | trainer | email                  |
+-------------+---------+------------------------+
| Java        | Mike    | mike@javatpoint.com    |
| Python      | James   | james@javatpoint.com   |
| Android     | Robin   | robin@javatpoint.com   |
| Hadoop      | Stephen | stephen@javatpoint.com |
| Testing     | Micheal | micheal@javatpoint.com |
+-------------+---------+------------------------+
5 rows in set (0.00 sec)
```

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY

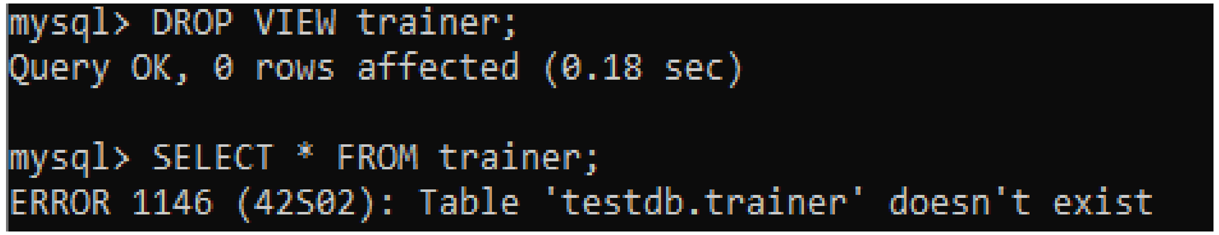
كلية تكنولوجيا الصناعة والطاقة

Drop VIEW

- We can drop the existing VIEW by using the **DROP VIEW** statement.
- **Syntax:**



```
DROP VIEW [IF EXISTS] view_name;
```



```
mysql> DROP VIEW trainer;
Query OK, 0 rows affected (0.18 sec)

mysql> SELECT * FROM trainer;
ERROR 1146 (42S02): Table 'testdb.trainer' doesn't exist
```

MYSQL Index

Index

- Indexes on a table is very similar to an index that we find in a book.
- If you don’t have an index, and I ask you to locate a specific chapter in the book, you will have to look at every page starting from the first page of the book
- On the other hand, if you have the index, you lookup the page number of the chapter in the index, and then directly go to that page number to locate the chapter.
- Obviously the book index is helping to drastically reduce the time it takes to find the chapter.
- In a similar way. Table and view indexes, can help the query to find data quickly.

Create Index

- An index enables you to improve the faster retrieval of records on a database table. It creates an **entry** for each value of the indexed columns.
- We use it to quickly find the record without searching each row in a database table whenever the table is accessed.
- We can create an index by using one or more **columns** of the table for efficient access to the records.
- When a table is created with a primary key or unique key, it automatically creates a special index named **PRIMARY**. We called this index as a clustered index. All indexes other than PRIMARY indexes are known as a non-clustered index or secondary index.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

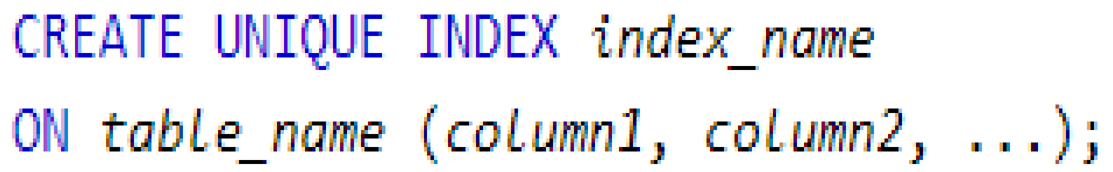
```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

جامعة القاهرة الجديدة التكنولوجية
NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

CREATE UNIQUE INDEX

CREATE UNIQUE INDEX Syntax:

Creates a unique index on a table. Duplicate values are not allowed:

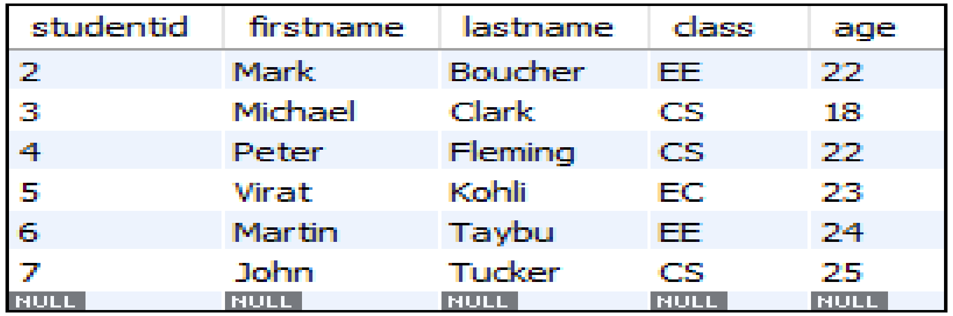


```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY

كلية تكنولوجيا الصناعة والطاقة

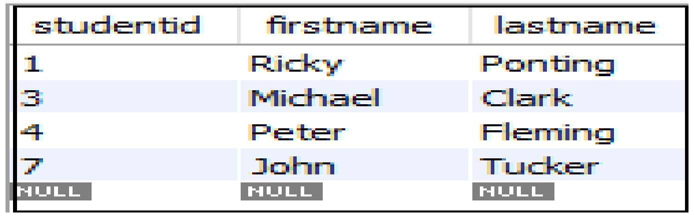
Example



studentid	firstname	lastname	class	age
2	Mark	Boucher	EE	22
3	Michael	Clark	CS	18
4	Peter	Fleming	CS	22
5	Virat	Kohli	EC	23
6	Martin	Taybu	EE	24
7	John	Tucker	CS	25
NULL	NULL	NULL	NULL	NULL

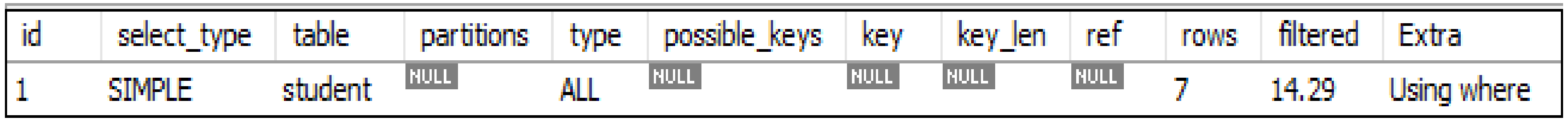
```
mysql> SELECT studentid, firstname, lastname FROM student WHERE class = 'CS';
```

This statement will give the following output:



studentid	firstname	lastname
1	Ricky	Ponting
3	Michael	Clark
4	Peter	Fleming
7	John	Tucker
NULL	NULL	NULL

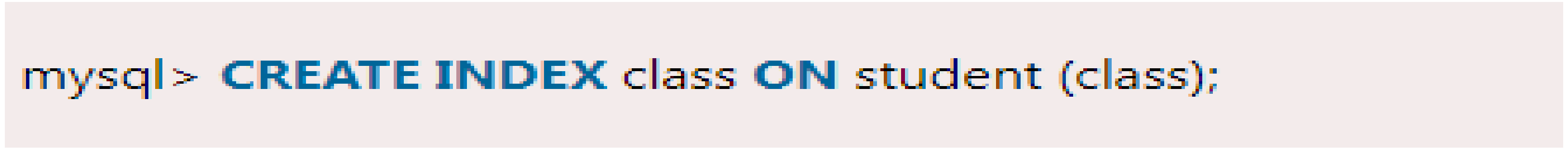
```
mysql> EXPLAIN SELECT studentid, firstname, lastname FROM student WHERE class = 'CS';
```



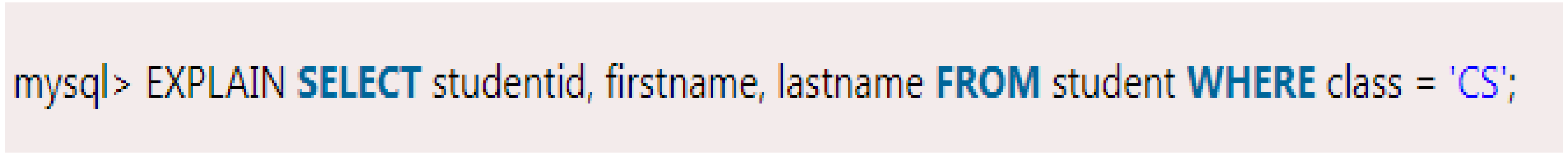
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	student	NULL	ALL	NULL	NULL	NULL	NULL	7	14.29	Using where

MySQL scans the whole table that contains **seven** rows to find the student whose class is the CS branch

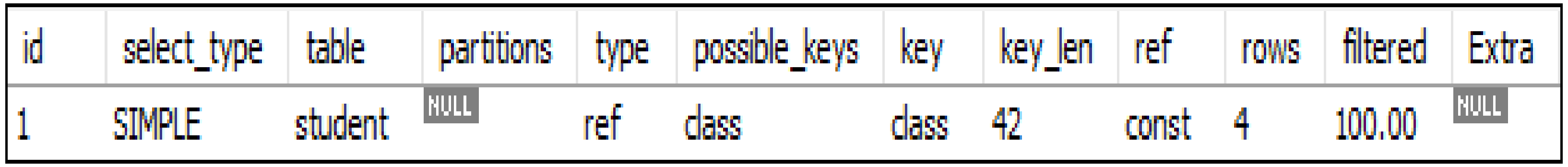
Example cont.,



```
mysql> CREATE INDEX class ON student (class);
```



```
mysql> EXPLAIN SELECT studentid, firstname, lastname FROM student WHERE class = 'CS';
```



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	student	NULL	ref	class	class	42	const	4	100.00	NULL

In this output, MySQL finds **four** rows from the class index without scanning the whole table. Hence, it increases the speed of retrieval of records on a database table.

Show Indexes

If you want to **show** the indexes of a table, execute the following statement:

```
mysql> SHOW INDEXES FROM student;
```

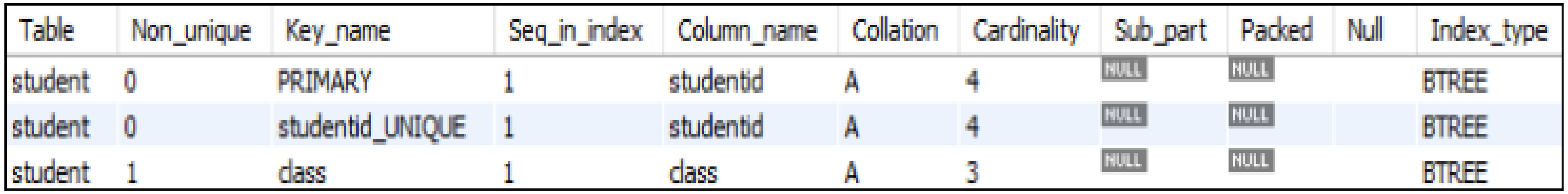
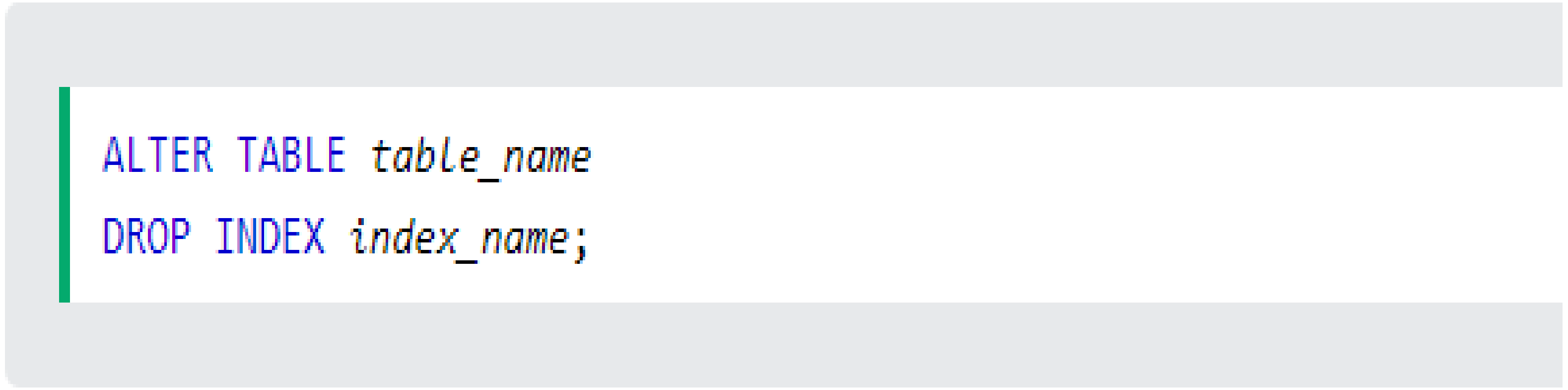


Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
student	0	PRIMARY	1	studentid	A	4	NULL	NULL		BTREE
student	0	studentid_UNIQUE	1	studentid	A	4	NULL	NULL		BTREE
student	1	class	1	class	A	3	NULL	NULL		BTREE

DROP Indexes

The `DROP INDEX` statement is used to delete an index in a table.



```
ALTER TABLE table_name
DROP INDEX index_name;
```

جامعة القاهرة الجديدة التكنولوجية NEW CAIRO TECHNOLOGICAL UNIVERSITY

كلية تكنولوجيا الصناعة والطاقة

جامعة القاهرة الجديدة التكنولوجية
NCT
NEW CAIRO TECHNOLOGICAL UNIVERSITY

Thank you