Chandigarh College of Engineering & Technology (Degree Wing)

Department of Computer Science and Engineering

Database Systems (Practical)

CS 352

Practical - 12

DOP: - 11/10/2024 **DOS:-**18/10/2024

Submitted By:

Abhay Pratap Singh
(Roll No: - CO23306)

Submitted To:

Dr. Dheerendra Singh
CSE Department

AIM :- To implement procedure and triggers in SQL with two examples each on database Project assigned to each student, by running on WAMP/ LAMP /XAMPP /SQL server.

DATABASE PROJECT NUMBER :- 7
"Computational data of all Staff"

1. Stored Procedure

A **stored procedure** is a precompiled collection of SQL statements stored in the database, which can be executed as a single unit. They help automate repetitive tasks and improve performance by reducing the amount of information sent between the client and the database.

Key Characteristics:

- **Reusable**: Once created, a stored procedure can be called and executed multiple times.
- **Encapsulate Logic**: It allows encapsulation of complex logic like calculations, data validations, and transformations.
- **Parameters**: Stored procedures can accept input parameters (input arguments) and return values (output).
- **Improves Performance**: As the SQL code is precompiled, it reduces the need for repeated parsing and execution, leading to faster performance.
- **Security**: Stored procedures allow restricting direct access to underlying tables and enforcing business logic through predefined commands.

Example Use Case:

- Automating a complex query.
- Inserting or updating data across multiple tables in one operation.
- Performing data validation before committing data to the database.

EXAMPLES :-

a. Procedure to Add a New Teaching Staff Member

This stored procedure adds a new teaching staff member to the details_teaching_contacts table.

Procedure 1 Query **:-**

```
 DELIMITER $$
CREATE PROCEDURE AddTeachingStaff(
   IN p_name VARCHAR(100),
   IN p_age INT,
   IN p_gender VARCHAR(10),
   IN p_phone_number VARCHAR(15),
   IN p_email VARCHAR(100)
)
BEGIN
   INSERT INTO details_teaching_contacts (NAME, AGE, GENDER, PHONE_NUMBER, E_MAIL)
   VALUES (p_name, p_age, p_gender, p_phone_number, p_email);
END $$

DELIMITER ;
```

Run SQL query/queries on table staff_data_ccet.details_teaching_contacts:

```
DELIMITER $$

CREATE PROCEDURE AddTeachingStaff(
    IN p_name VARCHAR(100),
    IN p_age INT,
    IN p_gender VARCHAR(10),
    IN p_phone_number VARCHAR(15),
    IN p_email VARCHAR(100)
)
BEGIN
    INSERT INTO details_teaching_contacts (NAME, AGE, GENDER,
PHONE_NUMBER, E_MAIL)
    VALUES (p_name, p_age, p_gender, p_phone_number, p_email);
END $$

DELIMITER ;
```

To call the procedure and add a new staff member:

Query : - CALL AddTeachingStaff('Jane Smith', 35, 'Female', '9876543210', 'jane.smith@ccet.ac.in');

Run SQL query/queries on table staff_data_ccet.details_teaching_contacts:

```
CALL AddTeachingStaff('Jane Smith', 35, 'Female', '9876543210',
'jane.smith@ccet.ac.in');
```

Browse Structure SQL Search Insert Export Import Privileges Operations Trigge

	NAME	AGE	GENDER	PHONE_NUMBER	E_MAIL
Edit Copy Delete	Ashwani Kumar	40	Male	9872823250	ashwanikumar@ccet.ac.i
Edit Copy Delete	Bhasker Gupta	42	Male	9855908643	bgupta@ccet.ac.in
Edit Copy Delete	Davinder Singh Saini	46	Male	8146730369	dssaini@ccet.ac.in
Edit Copy Delete	Dheerendra Singh	45	Male	9876439071	dsingh@ccet.ac.in
Edit Copy Delete	Dinesh Sharma	39	Male	9671721850	dsharma@ccet.ac.in
Edit Copy Delete	Dr Parvinder Kaur	40	Female	8295688911	pkaur@ccet.ac.in
Edit Copy Delete	Gulshan Goyal	41	Male	9417506206	gulshangoyal@ccet.ac.in
Edit Copy Delete	Hardeep Saini	36	Male	9914611106	hsaini@ccet.ac.in
Edit Copy Delete	Irfan Ahmad Khan	43	Male	7835847022	iakhan@ccet.ac.in
Edit Copy Delete	Jane Smith	35	Female	9876543210	jane.smith@ccet.ac.in
Edit Copy Delete	Jatinder Madan	52	Male	9041291970	jatindermadan@ccet.ac.i
Edit Copy Delete	Karuna Sharma	41	Female	8283833589	karunasharma@ccet.ac.i
Edit Copy Delete	Krishan Gopal Sharma	44	Male	9414403565	kgsharma@ccet.ac.in
Edit Copy Delete	Manveen Kaur	35	Female	9988957007	manveenkaur@ccet.ac.in

b. Procedure to Retrieve Non-Teaching Staff by Department

This stored procedure retrieves all non-teaching staff in a given department.

Procedure 2 Query : -

```
DELIMITER $$

CREATE PROCEDURE GetTeachingStaffFromDepartment(
   IN dept_name VARCHAR(50)
)
BEGIN
   SELECT *
   FROM details_teaching_official
   WHERE DEPARTMENT = dept_name;
END $$

DELIMITER ;
```

Run SQL query/queries on table staff_data_ccet.details_teaching_official:

```
1  DELIMITER $$
2
3  CREATE PROCEDURE GetTeachingStaffFromDepartment(
4      IN dept_name VARCHAR(50)
5  )
6  BEGIN
7      SELECT *
8      FROM details_teaching_official
9      WHERE DEPARTMENT = dept_name;
10 END $$
11
12 DELIMITER ;
```

To execute this procedure:

Query : - CALL GetTeachingStaffFromDepartment('Applied Science');

Run SQL query/queries on table staff_data_ccet.details_teaching_official:

```
1  CALL GetTeachingStaffFromDepartment('Applied Science');
```

NAME	DESIGNATION	DEPARTMENT	DesignationID	DepartmentID	ROOM_NO
Manveen Kaur	Assistant Professor	Applied Science	4	5	A-113
Neha	Assistant Professor	Applied Science	4	5	A-104 (Inside Chemistry Lab)
Parul Aggarwal	Assistant Professor	Applied Science	4	5	A-115
Varun Gupta	HOD	Applied Science	3	1	A-109

2. Trigger

A **trigger** is a special type of stored procedure that automatically runs or "fires" when a specific event (INSERT, UPDATE, DELETE) occurs on a table or view. Triggers allow you to enforce business rules or data integrity without requiring explicit action from the user or application.

Key Characteristics:

- **Automated Execution**: Triggers are automatically executed when a predefined event occurs, such as an insert, update, or delete operation.
- **Data Integrity**: Triggers can be used to enforce referential integrity, check constraints, or even audit changes to data.
- **Before or After**: Triggers can be set to execute before or after an event.
  - **BEFORE**: Executes before the triggering action (like insert, update, delete) is carried out.
  - **AFTER**: Executes after the action is completed.
- **Row-Level or Statement-Level**: Triggers can execute for each row affected (row-level) or only once for the entire statement (statement-level).

Example Use Case:

- Automatically updating related tables when data in a table changes.
- Enforcing data integrity, such as preventing duplicate records.
- Capturing changes to tables for auditing purposes.

a. Trigger to Prevent Insert if Phone Number Exists (Before Insert)

This trigger prevents inserting a new record into the details_teaching_contacts table if the phone number already exists.

Trigger 1 Query :-

```
DELIMITER $$

CREATE TRIGGER BeforeInsertTeachingContact
BEFORE INSERT ON details_teaching_contacts
FOR EACH ROW
BEGIN
   DECLARE existing_phone VARCHAR(15);

   SELECT PHONE_NUMBER INTO existing_phone
   FROM details_teaching_contacts
   WHERE PHONE_NUMBER = NEW.PHONE_NUMBER;

   IF existing_phone IS NOT NULL THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Phone number already exists!';
   END IF;
END $$

DELIMITER ;
```

Run SQL query/queries on table staff_data_ccet.details_teaching_official:

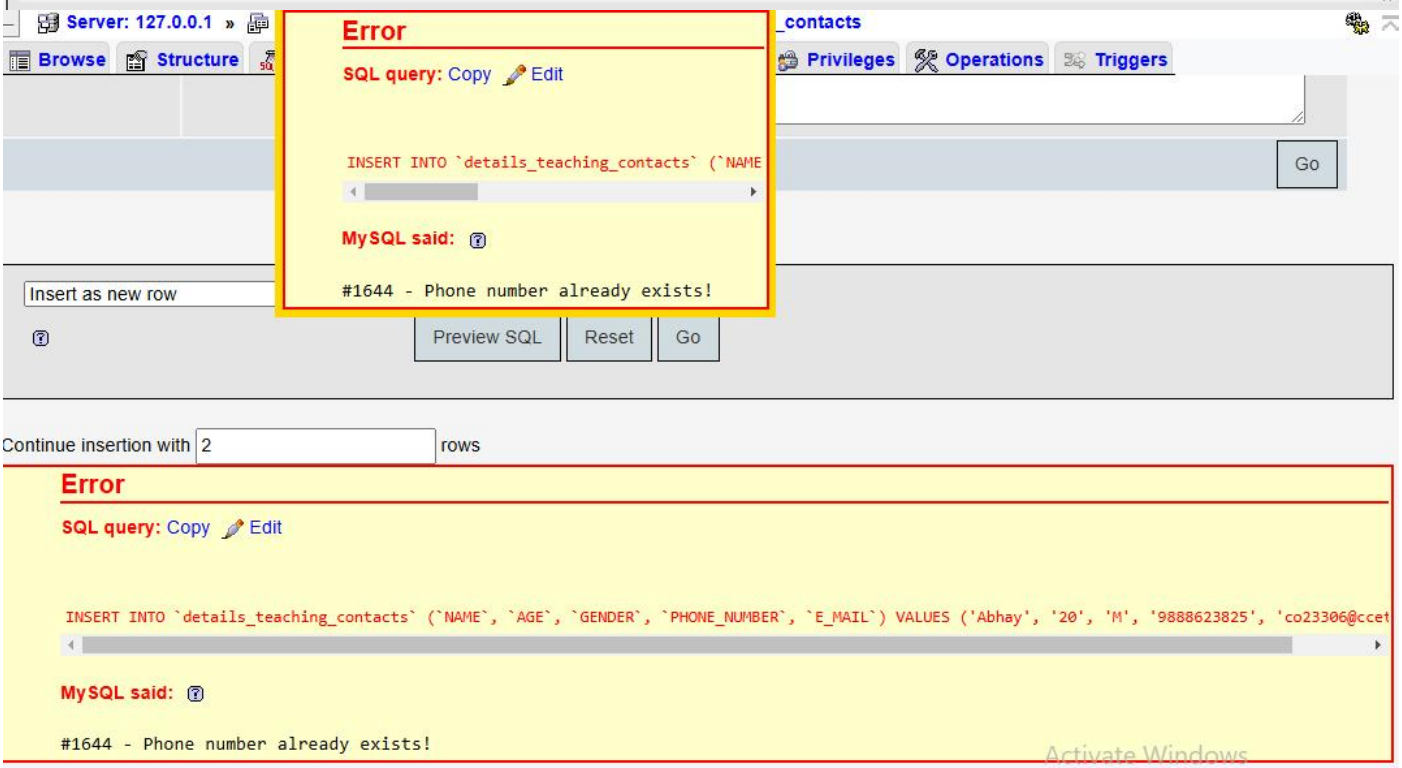
```
DELIMITER $$

CREATE TRIGGER BeforeInsertTeachingContact
BEFORE INSERT ON details_teaching_contacts
FOR EACH ROW
BEGIN
    DECLARE existing_phone VARCHAR(15);

    SELECT PHONE_NUMBER INTO existing_phone
    FROM details_teaching_contacts
    WHERE PHONE_NUMBER = NEW.PHONE_NUMBER;

    IF existing_phone IS NOT NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Phone number already exists!';
    END IF;
END $$

DELIMITER ;
```



This trigger checks if the phone number already exists before allowing the insert operation to proceed.

b. Trigger to Log Changes After Update in Printer Model (After Update)

This trigger logs the details of the printer model change in a printer_changes table after a record in the details_staff_printer table is updated.

Trigger 2 Query :-

DELIMITER $$

CREATE TRIGGER AfterUpdatePrinter
AFTER UPDATE ON details_staff_printer

```
FOR EACH ROW
BEGIN
   INSERT INTO printer_changes (staff_name, old_printer_model, new_printer_model, change_time)
   VALUES (NEW.Name, OLD.Printer_Model, NEW.Printer_Model, NOW());
END $$

DELIMITER ;
```

Run SQL query/queries on table staff_data_ccet.details_teaching_official:

```
DELIMITER $$

CREATE TRIGGER AfterUpdatePrinter
AFTER UPDATE ON details_staff_printer
FOR EACH ROW
BEGIN
    INSERT INTO printer_changes (staff_name, old_printer_model, new_printer_model, change_time)
    VALUES (NEW.Name, OLD.Printer_Model, NEW.Printer_Model, NOW());
END $$

DELIMITER ;
```

Log Table Query :-

```
CREATE TABLE printer_changes (
   change_id INT AUTO_INCREMENT PRIMARY KEY,
   staff_name VARCHAR(50),
   old_printer_model VARCHAR(50),
   new_printer_model VARCHAR(50),
   change_time DATETIME
);
```

Run SQL query/queries on table staff_data_ccet.details_teaching_official:

```
CREATE TABLE printer_changes (
    change_id INT AUTO_INCREMENT PRIMARY KEY,
    staff_name VARCHAR(50),
    old_printer_model VARCHAR(50),
    new_printer_model VARCHAR(50),
    change_time DATETIME
);
```

change_id	staff_name	old_printer_model	new_printer_model	change_time
1	Dr. Manveen Kaur	Brother_7500	Brother-7500D	2024-12-02 20:31:31