Advanced SQL – Triggers and database security

Week 5 Topic 3 - Triggers and database security, Error handling and programming issues

Course Learning Outcome (CLO) – 3

Lesson 9 and 10

Outline

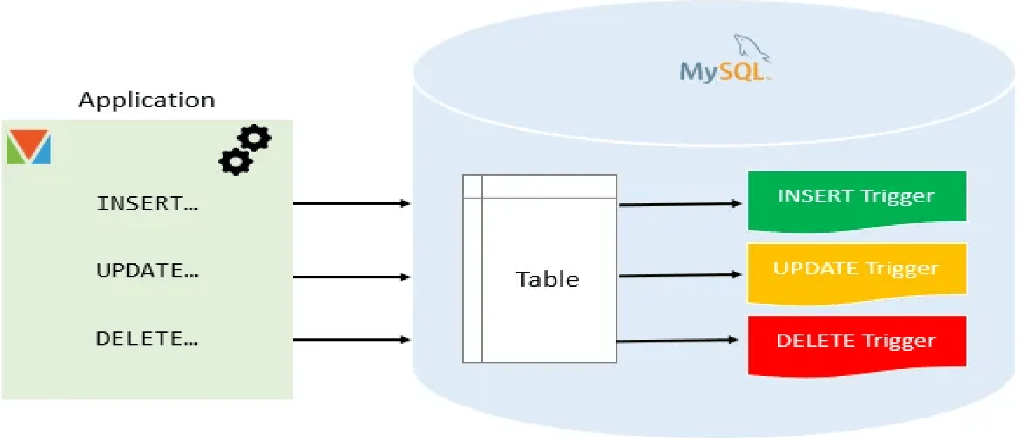
- ✓ Triggers
- ✓ security: audit, enforcement, roles, privileges, RLS
- ✓ Error handling
- ✓ programming issues: TRY/CATCH, EXCEPTION

What is a Trigger?

- ➢ A trigger is a special database object that automatically executes when a specified event occurs on a table.
- ➢ Unlike a procedure, or a function, which must be invoked explicitly, database triggers are invoked implicitly.
- ➢ Database triggers can be used to perform any of the following:
  - Audit data modification
  - Log events transparently
  - Enforce complex business rules
  - Derive column values automatically
  - Implement complex security authorizations
  - Maintain replicate tables
- ➢ A database trigger has three parts: a triggering event, an optional trigger constraint, and a trigger action.
- ➢ When an event occurs, a database trigger is fired, and an predefined SQL block will perform the necessary action
- ➢ Trigger executes automatically, without user calling it.

Events supported:

- ○ INSERT
- ○ UPDATE
- ○ DELETE



Trigger Timing & Components

Trigger Timing

- BEFORE → executes before the event
- AFTER → executes after the event

Trigger Components

- Trigger name
- Trigger timing (BEFORE / AFTER)
- Trigger event (INSERT / UPDATE / DELETE)
- Table name
- Trigger body

OLD and NEW Keywords

- NEW.column → new value (INSERT, UPDATE)
- OLD.column → old value (UPDATE, DELETE)

Trigger Syntax

```
DELIMITER $$

CREATE TRIGGER trigger_name
BEFORE | AFTER INSERT | UPDATE | DELETE
ON table_name
FOR EACH ROW
   [WHEN condition]
   DECLARE
   Declaration statements
BEGIN
  SQL statements;
END $$

DELIMITER ;
```

Important Notes

- FOR EACH ROW is mandatory
- Cannot use COMMIT or ROLLBACK inside triggers
- Triggers cannot return values

Examples –
BEFORE INSERT Trigger:

```
CREATE TRIGGER set_min_salary
BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
  IF NEW.salary < 12000 THEN
    SET NEW.salary = 12000;
  END IF;
END;
```

Examples Cont..

AFTER DELETE Trigger:

```
CREATE TRIGGER emp_delete_log
AFTER DELETE ON employee
FOR EACH ROW
BEGIN
  INSERT INTO emp_log(emp_id, action)
  VALUES (OLD.emp_id, 'Deleted');
END;
```

Hands on Practice –

Create a trigger that automatically records UPDATE operations performed on a product table into a product_log table.

- If the tables do not exist, create the required tables.
- Insert sample data into the product table.
- Create an AFTER UPDATE trigger to store old and new values in the product_log table.
- Test the trigger by updating a record and display the log table.

HANDLE ERRORS (Exception Handling)

What is Error Handling?

- Error handling allows MySQL to manage runtime errors during execution.
- Prevents unexpected termination of programs.

Where Used?

- Stored Procedures
- Functions
- Triggers

Common Errors

- Duplicate primary key
- Invalid input values
- Constraint violations

Types of Handlers –

MySQL Error Handling Mechanism

- DECLARE HANDLER

Types of Handlers

- CONTINUE → execution continues
- EXIT → execution stops

Error Conditions

- MySQL error code (e.g., 1062)
- SQLSTATE value
- Named conditions

Error Handling Syntax

DECLARE HANDLER Syntax

```
DECLARE handler_action HANDLER FOR error_condition
BEGIN
  SQL statements;
END;
```

SIGNAL Statement -

- Used to raise user-defined errors

```
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Custom error message';
```

Test the Trigger –

```
INSERT INTO product VALUES (1, 'Keyboard', -500);
```

Outcome. -

Error message will be displayed:

Prevent Invalid Data Using SIGNAL Example

Use case - Do not allow product price ≤ 0 during INSERT or UPDATE.

```
CREATE TABLE product (
   product_id INT PRIMARY KEY,
   product_name VARCHAR(50),
   price DECIMAL(10,2) );

DELIMITER $$

CREATE TRIGGER check_product_price
BEFORE INSERT ON product
FOR EACH ROW
BEGIN
   IF NEW.price <= 0 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Price must be greater than zero';
   END IF;
END $$

DELIMITER ;
```

Example Error Handling During UPDATE in Trigger

Use case – Do not allow quantity less than 1 when updating stock.

```
CREATE TABLE stock (
   item_id INT PRIMARY KEY,
   item_name VARCHAR(50),
   quantity INT
);
```

Insert data –

```
INSERT INTO stock VALUES (1, 'Pen', 20);
```

Create BEFORE UPDATE Trigger:

```
DELIMITER $$
CREATE TRIGGER check_stock_quantity
BEFORE UPDATE ON stock
FOR EACH ROW
BEGIN
   IF NEW.quantity < 1 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Quantity cannot be less than 1';
   END IF;
END $$
DELIMITER ;
```

Test the Trigger –

```
UPDATE stock SET quantity = 0 WHERE item_id = 1;
```

- Output: Error message shown - Quantity cannot be less than 1

Using DECLARE HANDLER Inside Trigger

Use case - Handle duplicate entry attempt gracefully.

```
CREATE TABLE users (
   user_id INT PRIMARY KEY,
   username VARCHAR(30) UNIQUE
);
```

Test the Trigger –

```
INSERT INTO users VALUES (1, 'admin');
INSERT INTO users VALUES (2, 'admin’);
```

Output –

```
Duplicate username not allowed
```

Create Trigger with CONTINUE HANDLER –

```
DELIMITER $$

CREATE TRIGGER handle_duplicate_user
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
   DECLARE CONTINUE HANDLER FOR 1062
   BEGIN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Duplicate username not allowed';
   END;
END $$

DELIMITER ;
```

Note - DECLARE HANDLER is mostly used in stored procedures, but can appear in triggers for controlled handling.

Database Security

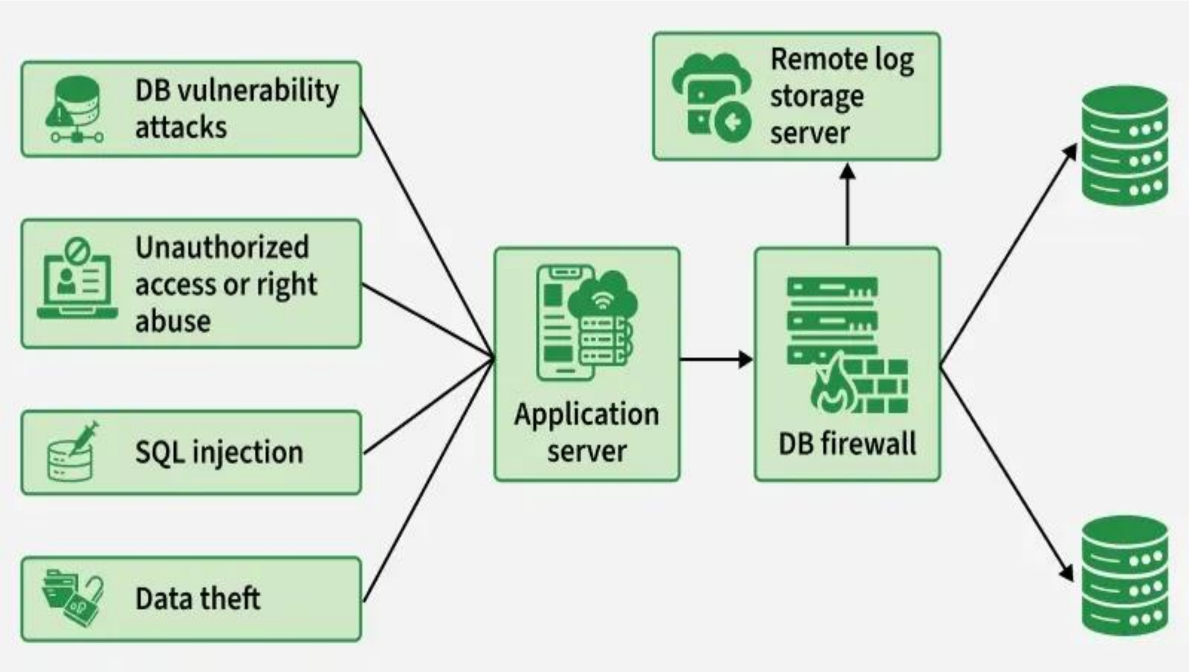
Database security protects data from unauthorized access, invalid changes, and data misuse.

Ensures:

- Data integrity
- Data accuracy
- Controlled operations

Role of Triggers & Error Handling

- Triggers automatically monitor database actions
- Error handling blocks unauthorized or invalid operations
- Both act as security layers inside the database



Database security structure

Database Security….

- Authentication: who are you? (login, password, SSO)
- Authorization: what can you do? (GRANT/REVOKE permissions)
- Least privilege: give only what is needed, nothing more
- Use roles / groups; grant to roles, then add users to roles
- Separate schemas for app vs reporting; avoid using superuser in apps

`Example` Create a role and grant read-only access to one schema.

```
-- Create a database role
CREATE ROLE reporting_reader;

-- Create user (assumes login exists)
CREATE USER analyst FOR LOGIN analyst;
ALTER ROLE reporting_reader ADD MEMBER analyst;

-- Grant read access on a schema
GRANT SELECT ON SCHEMA::reporting TO
reporting_reader;

-- Optionally block writes explicitly
DENY INSERT, UPDATE, DELETE ON SCHEMA::reporting TO
reporting_reader;
```

Example - create a read-only reporting role

- □ Step 1: Create role (reporting_reader)
- □ Step 2: GRANT USAGE on schema + GRANT SELECT on tables/views
- □ Step 3: Add users to role
- □ Step 4: Test with a “deny-by-default” mindset

Goal - An “analyst” can read reporting schema but nothing else.

```
------- Test cases ----------

Should succeed:

SELECT * FROM reporting.monthly_revenue LIMIT 5;

-- Should fail:
UPDATE reporting.monthly_revenue SET revenue = 0;
SELECT * FROM private_schema.secrets;
```

Example - Row-Level Security (RLS)

Use a predicate function + security policy to filter rows automatically.

```
Sketch only (keep it simple) -----

•  CREATE FUNCTION dbo.fn_ordersPredicate(@customer_id int) RETURNS TABLE ...
•  CREATE SECURITY POLICY ... ADD FILTER PREDICATE ...
•  App sets CONTEXT_INFO/SESSION_CONTEXT with customer_id
```

Goal: users can only see “their” rows.

Use of Triggers for security

How Triggers Improve Security

- Restrict invalid data insertion or updates
- Prevent unauthorized changes to sensitive columns
- Maintain audit logs of user activities

Examples

- Prevent negative salary insertion
- Restrict deletion of important records
- Log UPDATE or DELETE operations automatically

Sample Trigger Example

```
IF NEW.salary < 15000 THEN
  SIGNAL SQLSTATE '45000'
  SET MESSAGE_TEXT = 'Unauthorized salary value';
END IF;
```

Error Handling for Security Enforcement

Why Error Handling is Important

- Stops execution when security rules are violated
- Displays meaningful error messages
- Prevents data corruption

Security with SIGNAL

- Custom error messages guide users
- Prevents unauthorized data manipulation

Example –

```
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT = 'Access denied: Invalid operation';
```

Combined Use of Triggers & Error Handling

How They Work Together

- Trigger detects unsafe operation
- Error handling blocks the operation
- Logs the event if required

Security Benefits

- Automatic rule enforcement
- Reduced dependency on application-level security
- Better control over database actions

Use Cases

- Banking systems
- Student management systems
- Inventory and payroll systems

Case Study 1 - Payroll System : Prevent Invalid Salary

Problem Description

- In a company payroll system, employees must not have salary below 500,000.
- HR users may accidentally enter wrong salary values.

Security Requirement

- Prevent insertion of invalid salary
- Display a meaningful error message
- Enforce rule at database level

Idea -

Tables Used

employee(emp_id, emp_name, salary)

Trigger Logic

- Trigger executes BEFORE INSERT
- Checks salary value
- Raises error if salary < 15000

```
CREATE TABLE employee (
   emp_id INT PRIMARY KEY,
   emp_name VARCHAR(50),
   salary DECIMAL(10,2)
);

DELIMITER $$

CREATE TRIGGER salary_validation
BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
   IF NEW.salary < 15000 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Salary below minimum limit not
allowed';
   END IF;
END $$

DELIMITER ;
```

Security Achieved

- Invalid salary blocked
- Database-level protection

Case Study 4: Student Database – Audit Logging

Scenario (Real World) -

- Universities must track who changed student marks.

Security Requirement

- Automatically log updates
- Store old and new values

Tables will be Used

- student
- student_log

Trigger Logic

- AFTER UPDATE trigger
- Records changes for audit

```
CREATE TABLE student (
    student_id INT PRIMARY KEY,
    name VARCHAR(50),
    marks INT
);

CREATE TABLE student_log (
    log_id INT AUTO_INCREMENT PRIMARY KEY,
    student_id INT,
    old_marks INT,
    new_marks INT,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Case Study 2: Student Database – Audit Logging…

```
DELIMITER $$

CREATE TRIGGER marks_audit
AFTER UPDATE ON student
FOR EACH ROW
BEGIN
    INSERT INTO student_log
    (student_id, old_marks, new_marks)
    VALUES
    (OLD.student_id, OLD.marks, NEW.marks);
END $$

DELIMITER ;
```

Security Achieved

- Audit trail maintained
- Unauthorized changes detectable

Further reading and Practice

- Silberschatz et al., 2020, Ch.5
- Triggers Syntax and Examples - https://dev.mysql.com/doc/refman/8.4/en/trigger-syntax.html
- MySQL Create Trigger - https://www.geeksforgeeks.org/mysql/mysql-create-trigger/