

Lesson Plan: MySQL Advanced Concepts with One Schema

Learning Objectives

By the end of this lesson, learners will be able to:

1. Understand and implement Stored Procedures, Triggers, Cursors, Indexes, and Views in MySQL.
 2. Apply these concepts using a consistent schema.
 3. Analyze how each feature improves database functionality, automation, and performance.
-

Schema Used: E-commerce Database

We will use the following tables:

```
CREATE TABLE Customers (  
    customer_id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(100),  
    email VARCHAR(100),  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE TABLE Orders (  
    order_id INT PRIMARY KEY AUTO_INCREMENT,  
    customer_id INT,  
    order_date DATE,  
    total_amount DECIMAL(10,2),  
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)  
);  
  
CREATE TABLE Order_Items (  
    item_id INT PRIMARY KEY AUTO_INCREMENT,  
    order_id INT,  
    product_name VARCHAR(100),  
    quantity INT,  
    price DECIMAL(10,2),  
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)  
);
```

Part 1: Stored Procedure

Concept: A stored procedure is a reusable block of SQL statements stored in the database. Example: Create a procedure to insert a new order.

```
DELIMITER //
CREATE PROCEDURE AddNewOrder(
    IN p_customer_id INT,
    IN p_order_date DATE,
    IN p_total DECIMAL(10,2)
)
BEGIN
    INSERT INTO Orders (customer_id, order_date, total_amount)
    VALUES (p_customer_id, p_order_date, p_total);
END //
DELIMITER ;
```

Usage:

```
CALL AddNewOrder(1, '2025-09-23', 250.00);
```

Part 2: Trigger

Concept: A trigger automatically executes in response to an event (INSERT, UPDATE, DELETE). Example: Create a trigger to log when a new customer is added.

```
CREATE TABLE Customer_Log (
    log_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    action VARCHAR(50),
    log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

DELIMITER //
CREATE TRIGGER after_customer_insert
AFTER INSERT ON Customers
FOR EACH ROW
BEGIN
    INSERT INTO Customer_Log(customer_id, action)
    VALUES (NEW.customer_id, 'New customer added');
END //
DELIMITER ;
```

Test:

```
INSERT INTO Customers (name, email) VALUES ('Alice', 'alice@example.com');
SELECT * FROM Customer_Log;
```

Part 3: Cursor

Concept: A cursor is used to iterate over query results row by row. Example: Calculate total sales amount by looping through orders.

```
DELIMITER //
CREATE PROCEDURE CalculateTotalSales()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE v_order_id INT;
    DECLARE v_total DECIMAL(10,2);
    DECLARE total_sales DECIMAL(10,2) DEFAULT 0.00;

    DECLARE cur CURSOR FOR SELECT order_id, total_amount FROM Orders;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

    OPEN cur;
read_loop: LOOP
    FETCH cur INTO v_order_id, v_total;
    IF done THEN
        LEAVE read_loop;
    END IF;
    SET total_sales = total_sales + v_total;
END LOOP;

    CLOSE cur;
    SELECT total_sales AS Total_Sales;
END //
DELIMITER ;
```


 Usage:

```
CALL CalculateTotalSales();
```

Part 4: Index

Concept: Indexes improve query performance by allowing faster lookups. Example: Create an index on `Orders.customer_id` for faster searching.

```
CREATE INDEX idx_customer_id ON Orders(customer_id);
```

 Benefit: Queries like

```
SELECT * FROM Orders WHERE customer_id = 1;
```

will run faster.

Part 5: Views

Concept: A view is a saved query that acts like a virtual table. Example: Create a view showing customer orders with total amounts.

```
CREATE VIEW CustomerOrderSummary AS
SELECT c.name, c.email, o.order_id, o.order_date, o.total_amount
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id;
```

 Usage:

```
SELECT * FROM CustomerOrderSummary;
```

Wrap-up

- Stored Procedure → Automates reusable SQL logic
 - Trigger → Automates actions based on events
 - Cursor → Processes query results row by row
 - Index → Speeds up data retrieval
 - View → Simplifies complex queries
-