MySQL

Demo project walkthrough from the database industry sessions by **Salman Farshi** (industry trainer, EDGE: BU-CSE); **contact**: 01680012549; **e-mail**: farshisalman.bd@gmail.com.

We'll be planning and working on a basic part of an e-commerce platform. Let's create a new database and add some tables.

Database: CREATE DATABASE demo_ecom;

Users table

```
CREATE TABLE users(

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(100) NOT NULL,

   contact VARCHAR(20) NOT NULL UNIQUE,

   mail VARCHAR(50),

   role INT DEFAULT 1,

   status INT DEFAULT 1

);
```

Shops table

```
CREATE TABLE shops(

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(100) NOT NULL UNIQUE,

   contact VARCHAR(20) NOT NULL,

   user_id INT,

   FOREIGN KEY (user_id) REFERENCES users(id)

);
```

Categories table

```
CREATE TABLE categories(

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(100) NOT NULL UNIQUE

);
```

Products table

```
CREATE TABLE products(

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(100) NOT NULL,

   price DECIMAL(10, 2) NOT NULL,

   stock INT DEFAULT 0,

   description TEXT,

   category_id INT,

   FOREIGN KEY (category _id) REFERENCES categories(id),

   shop_id INT,

   FOREIGN KEY (shop _id) REFERENCES shops(id)

);
```

Logs table

```
CREATE TABLE logs(

   id INT AUTO_INCREMENT PRIMARY KEY,

   message TEXT NOT NULL

);
```

VIEW and JOIN

Now, as the products are linked to their own shops through foreign keys, we can use joins to see all the products with their shops and owners.

Let's create a view to see and store the list (query).

```
CREATE VIEW product_shop_view AS

SELECT products.name AS product_name, shops.name AS shop_name, users.name AS owner_name

FROM products JOIN shops ON products.shop_id = shops.id

JOIN users ON shops.user_id = users.id;
```

In order to call/see the view:

```
SELECT * FROM product_shop_view;
```

STORED PROCEDURES

It's time to write procedures. A procedure is a collection of pre-compiled SQL statements stored inside the database, so that it can be reused over and over again. A procedure always contains a name, parameter lists, and SQL statements.

Let's write a simple procedure that inserts a new row into the users table.

```
DELIMITER $$

CREATE PROCEDURE addNewUser(IN u_name VARCHAR(100), IN u_contact VARCHAR(20))

BEGIN

        INSERT INTO users(name, contact) VALUES(u_name, u_contact);

END $$

DELIMITER ;
```

Now, write three more procedures to add new categories, new shops and new products respectively.

```
DELIMITER $$

CREATE PROCEDURE addNewCategory(IN c_name VARCHAR(100), OUT message VARCHAR(100))

BEGIN

   DECLARE cat INT;

   SELECT COUNT(*) INTO cat FROM categories WHERE name = c_name;

   IF cat > 0 THEN

           SET message = 'Category already exists!';

   ELSE

           INSERT INTO categories(name) VALUES(c_name);

           SET message = 'Category added!';

   END IF;

END $$

DELIMITER ;
```

```
DELIMITER $$

CREATE PROCEDURE addNewShop(IN s_name VARCHAR(100), IN s_contact VARCHAR(20), IN s_user_id
INT)

BEGIN

           INSERT INTO shops(name, contact, user_id) VALUES(s_name, s_contact, s_user_id);

END $$

DELIMITER ;
DELIMITER $$

CREATE PROCEDURE addNewProduct(

   IN p_name VARCHAR(100),

   IN p_price DECIMAL(10, 2),

   IN p_stock INT,

   IN p_category_id INT,

   IN p_shop_id INT)

BEGIN

        INSERT INTO products(name, price, stock, category_id, shop_id)

        VALUES(p_name, p_price, p_stock, p_category_id, p_shop_id);

END $$

DELIMITER ;
```

To call the addNewCategory() procedure and receive the message,

```
CALL addNewCategory('—category name--', @msg);

SELECT @msg;
```

To call the addNewShop() and addNewProduct() procedures respectively,

```
CALL addNewShop('—shop name--', '—shop contact--', --user id--);


CALL addNewProduct('—product name--', —product price--, —product stock--, —category id--,
—shop id--);
```

STORED FUNCTIONS

A stored function in MySQL is a set of SQL statements that perform a specific operation and then return a single value. Unlike procedures, functions can only have input parameters.

Let’s write a function that calculates the total price when a customer orders an item and then returns the result.

```
DELIMITER $$

CREATE FUNCTION calculateTotalPrice(p_name VARCHAR(100), p_quantity INT)

RETURNS VARCHAR(100)

BEGIN

   DECLARE p_price, p_total_price DECIMAL(10, 2);

   DECLARE p_stock INT;

   SELECT price, stock INTO p_price, p_stock FROM products WHERE name = p_name;

   IF p_stock >= p_quantity THEN

            RETURN CONCAT('Total Price: ', p_price * p_quantity);

   ELSE

            RETURN 'Product is not available!';

   END IF;

END $$

DELIMITER ;
```

To call the function,

```
SELECT calculateTotalPrice(‘—product name--’, --product quantity--);
```

<u>TRIGGERS</u>

Finally, we want to store a message in the logs table every time a new user is added. To achieve this, we’ll be using trigger. A MySQL trigger is a stored program which is executed automatically to respond to a specific event such as **insertion**, **modification**, and **deletion**.

```
DELIMITER $$

CREATE TRIGGER LogNewUser AFTER

INSERT ON users

FOR EACH ROW

BEGIN

   DECLARE msg VARCHAR(100);

   SET msg = CONCAT('New user added: ', NEW.name);

           INSERT INTO logs(message) VALUES(msg);

END $$

DELIMITER ;
```