BOOKSTORE DATABASE SYSTEM

GA9 Assignment

Abstract

This assignment involves designing and implementing a MySQL Bookstore database (Jeza_230878369), utilizing advanced features such as derived tables, stored procedures, triggers, cursors, functions, and events to manage sales, inventory, and automation tasks. Through practical tasks and a reflective analysis, it demonstrates proficiency in database management, addressing challenges in MySQL 5.5, and exploring modern trends like modularity and automation, supported by 15 IEEE-style references.

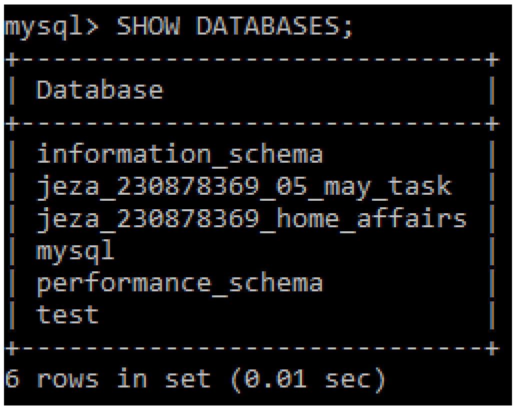
Mnqobi Lisbon Jeza

230878369

Table of Contents

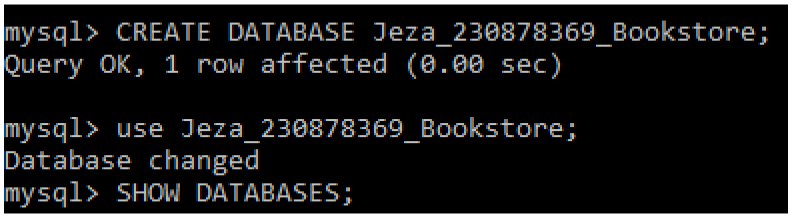
Creation of Bookstore Database System

Before creating the Database.



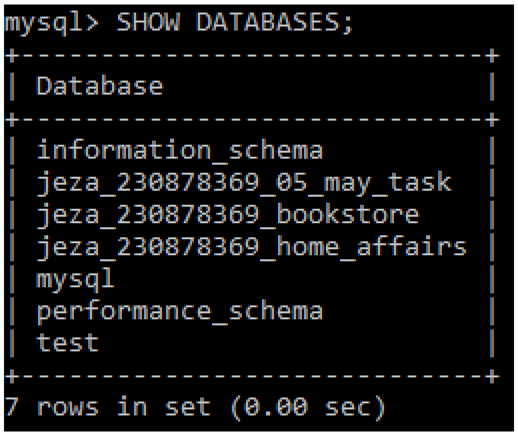
```
mysql> SHOW DATABASES;
+----------------------------+
| Database                   |
+----------------------------+
| information_schema         |
| jeza_230878369_05_may_task |
| jeza_230878369_home_affairs |
| mysql                      |
| performance_schema         |
| test                       |
+----------------------------+
6 rows in set (0.01 sec)
```

***Bookstore** Database Created.*



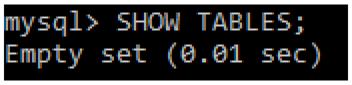
```
mysql> CREATE DATABASE Jeza_230878369_Bookstore;
Query OK, 1 row affected (0.00 sec)

mysql> use Jeza_230878369_Bookstore;
Database changed
mysql> SHOW DATABASES;
```

After creating the Database.



```
mysql> SHOW DATABASES;
+----------------------------+
| Database                   |
+----------------------------+
| information_schema         |
| jeza_230878369_05_may_task |
| jeza_230878369_bookstore   |
| jeza_230878369_home_affairs |
| mysql                      |
| performance_schema         |
| test                       |
+----------------------------+
7 rows in set (0.00 sec)
```

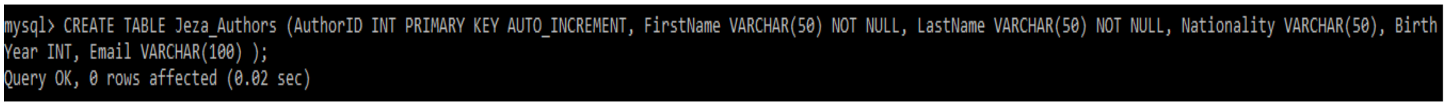
Showing Tables Before.



```
mysql> SHOW TABLES;
Empty set (0.01 sec)
```

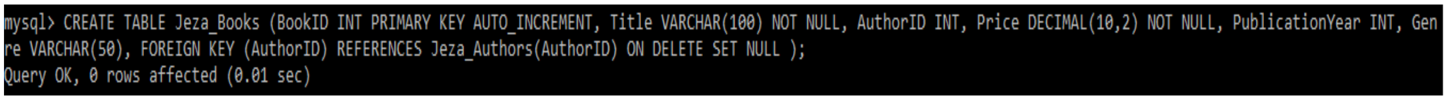
Creating Tables.

Table **Jeza_Authers** created



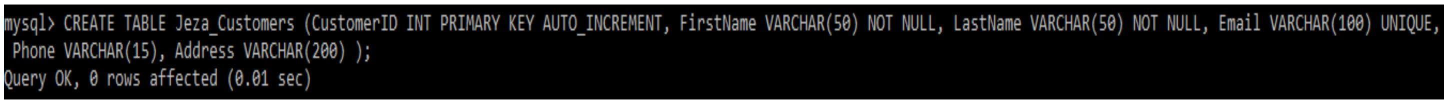
```
mysql> CREATE TABLE Jeza_Authors (AuthorID INT PRIMARY KEY AUTO_INCREMENT, FirstName VARCHAR(50) NOT NULL, LastName VARCHAR(50) NOT NULL, Nationality VARCHAR(50), Birth
Year INT, Email VARCHAR(100) );
Query OK, 0 rows affected (0.02 sec)
```

Table **Jeza_Books** created



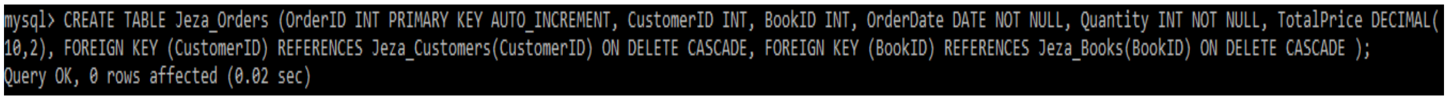
```
mysql> CREATE TABLE Jeza_Books (BookID INT PRIMARY KEY AUTO_INCREMENT, Title VARCHAR(100) NOT NULL, AuthorID INT, Price DECIMAL(10,2) NOT NULL, PublicationYear INT, Gen
re VARCHAR(50), FOREIGN KEY (AuthorID) REFERENCES Jeza_Authors(AuthorID) ON DELETE SET NULL );
Query OK, 0 rows affected (0.01 sec)
```

Table **Jeza_Customers** created



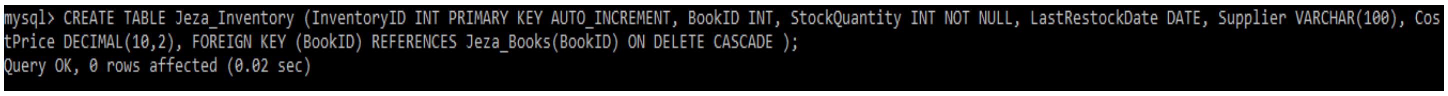
```
mysql> CREATE TABLE Jeza_Customers (CustomerID INT PRIMARY KEY AUTO_INCREMENT, FirstName VARCHAR(50) NOT NULL, LastName VARCHAR(50) NOT NULL, Email VARCHAR(100) UNIQUE,
 Phone VARCHAR(15), Address VARCHAR(200) );
Query OK, 0 rows affected (0.01 sec)
```

Table **Jeza_Orders** created



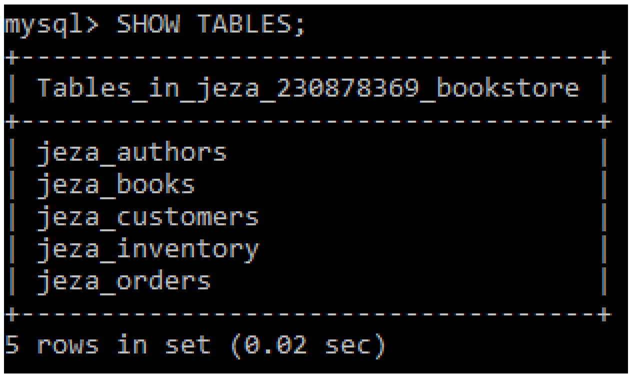
```
mysql> CREATE TABLE Jeza_Orders (OrderID INT PRIMARY KEY AUTO_INCREMENT, CustomerID INT, BookID INT, OrderDate DATE NOT NULL, Quantity INT NOT NULL, TotalPrice DECIMAL(
10,2), FOREIGN KEY (CustomerID) REFERENCES Jeza_Customers(CustomerID) ON DELETE CASCADE, FOREIGN KEY (BookID) REFERENCES Jeza_Books(BookID) ON DELETE CASCADE );
Query OK, 0 rows affected (0.02 sec)
```

Table **Jeza_Inventory** created



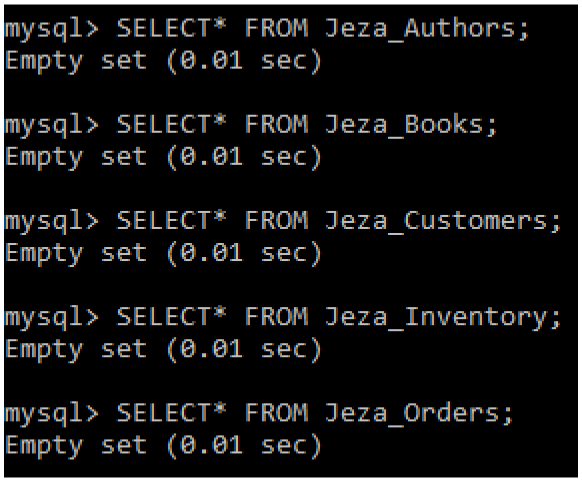
```
mysql> CREATE TABLE Jeza_Inventory (InventoryID INT PRIMARY KEY AUTO_INCREMENT, BookID INT, StockQuantity INT NOT NULL, LastRestockDate DATE, Supplier VARCHAR(100), Cos
tPrice DECIMAL(10,2), FOREIGN KEY (BookID) REFERENCES Jeza_Books(BookID) ON DELETE CASCADE );
Query OK, 0 rows affected (0.02 sec)
```

Showing Tables After.



```
mysql> SHOW TABLES;
+-----------------------------------+
| Tables_in_jeza_230878369_bookstore |
+-----------------------------------+
| jeza_authors                      |
| jeza_books                        |
| jeza_customers                    |
| jeza_inventory                    |
| jeza_orders                       |
+-----------------------------------+
5 rows in set (0.02 sec)
```

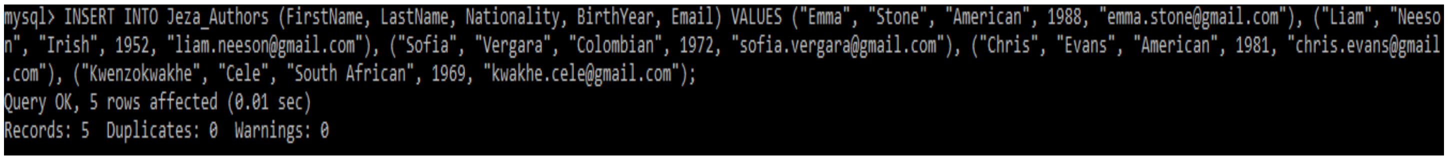
Viewing Tables Before.



```
mysql> SELECT* FROM Jeza_Authors;
Empty set (0.01 sec)

mysql> SELECT* FROM Jeza_Books;
Empty set (0.01 sec)

mysql> SELECT* FROM Jeza_Customers;
Empty set (0.01 sec)

mysql> SELECT* FROM Jeza_Inventory;
Empty set (0.01 sec)

mysql> SELECT* FROM Jeza_Orders;
Empty set (0.01 sec)
```

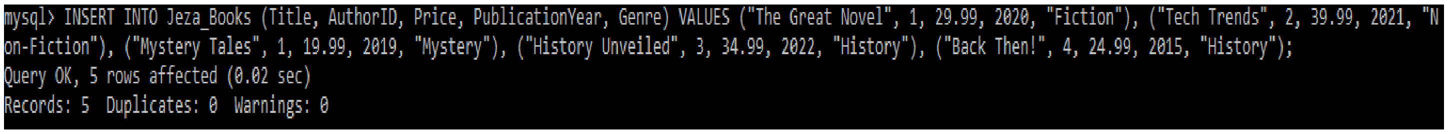
Populating Tables.

Jeza_Authors



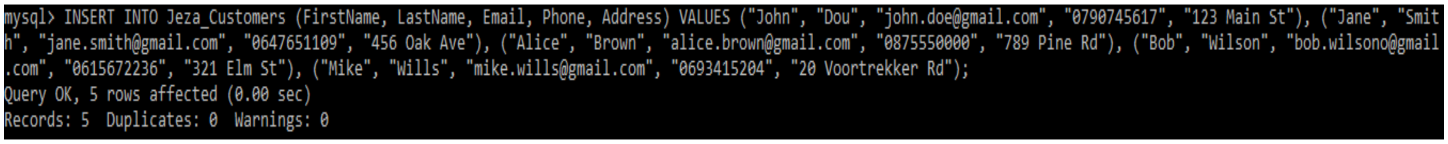
```
mysql> INSERT INTO Jeza_Authors (FirstName, LastName, Nationality, BirthYear, Email) VALUES ("Emma", "Stone", "American", 1988, "emma.stone@gmail.com"), ("Liam", "Neeso
n", "Irish", 1952, "liam.neeson@gmail.com"), ("Sofia", "Vergara", "Colombian", 1972, "sofia.vergara@gmail.com"), ("Chris", "Evans", "American", 1981, "chris.evans@gmail
.com"), ("Kwenzokwakhe", "Cele", "South African", 1969, "kwakhe.cele@gmail.com");
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Jeza_Books



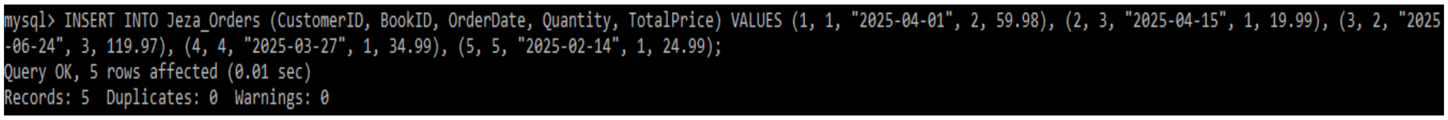
```
mysql> INSERT INTO Jeza_Books (Title, AuthorID, Price, PublicationYear, Genre) VALUES ("The Great Novel", 1, 29.99, 2020, "Fiction"), ("Tech Trends", 2, 39.99, 2021, "N
on-Fiction"), ("Mystery Tales", 1, 19.99, 2019, "Mystery"), ("History Unveiled", 3, 34.99, 2022, "History"), ("Back Then!", 4, 24.99, 2015, "History");
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Jeza_Customers



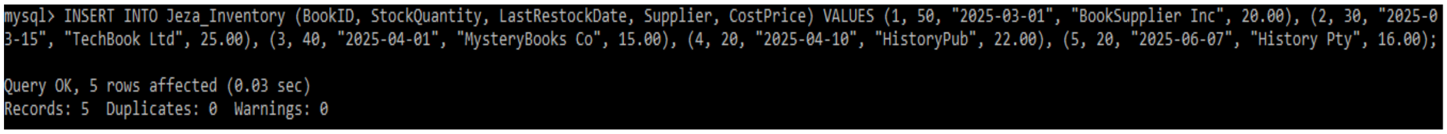
```
mysql> INSERT INTO Jeza_Customers (FirstName, LastName, Email, Phone, Address) VALUES ("John", "Dou", "john.doe@gmail.com", "0790745617", "123 Main St"), ("Jane", "Smit
h", "jane.smith@gmail.com", "0647651109", "456 Oak Ave"), ("Alice", "Brown", "alice.brown@gmail.com", "0875550000", "789 Pine Rd"), ("Bob", "Wilson", "bob.wilsono@gmail
.com", "0615672236", "321 Elm St"), ("Mike", "Wills", "mike.wills@gmail.com", "0693415204", "20 Voortrekker Rd");
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Jeza_Orders



```
mysql> INSERT INTO Jeza_Orders (CustomerID, BookID, OrderDate, Quantity, TotalPrice) VALUES (1, 1, "2025-04-01", 2, 59.98), (2, 3, "2025-04-15", 1, 19.99), (3, 2, "2025
-06-24", 3, 119.97), (4, 4, "2025-03-27", 1, 34.99), (5, 5, "2025-02-14", 1, 24.99);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

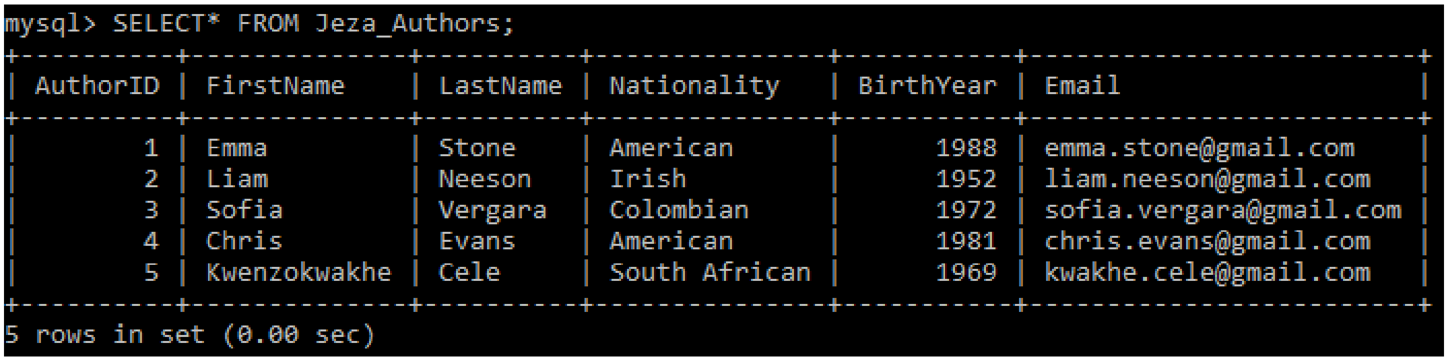
Jeza_Inventory



```
mysql> INSERT INTO Jeza_Inventory (BookID, StockQuantity, LastRestockDate, Supplier, CostPrice) VALUES (1, 50, "2025-03-01", "BookSupplier Inc", 20.00), (2, 30, "2025-0
3-15", "TechBook Ltd", 25.00), (3, 40, "2025-04-01", "MysteryBooks Co", 15.00), (4, 20, "2025-04-10", "HistoryPub", 22.00), (5, 20, "2025-06-07", "History Pty", 16.00);

Query OK, 5 rows affected (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

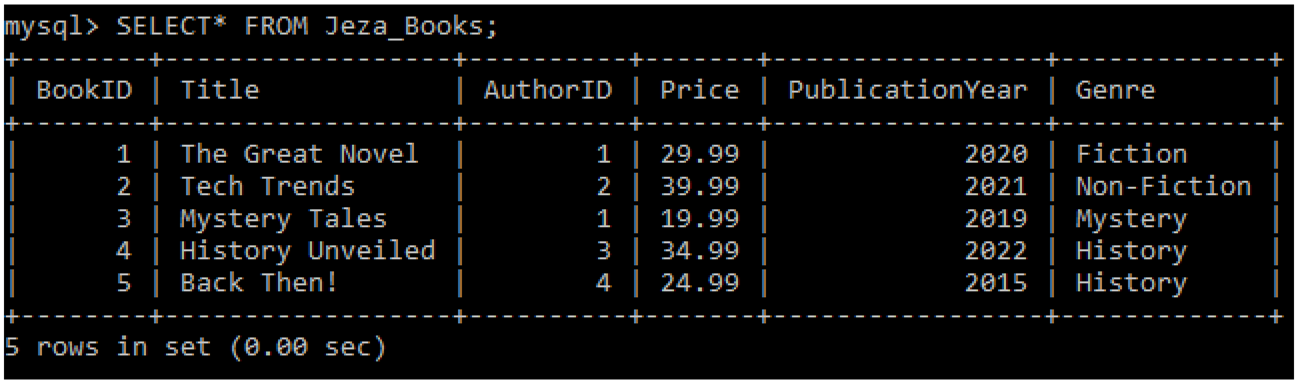
Viewing Tables After.

Jeza_Authors



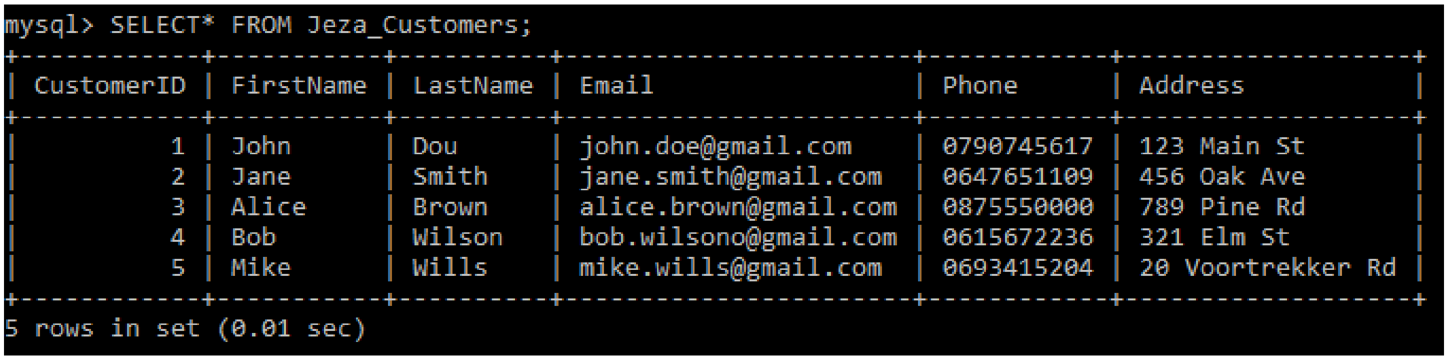
```
mysql> SELECT* FROM Jeza_Authors;
+----------+--------------+----------+---------------+-----------+-------------------------+
| AuthorID | FirstName    | LastName | Nationality   | BirthYear | Email                   |
+----------+--------------+----------+---------------+-----------+-------------------------+
|        1 | Emma         | Stone    | American      |      1988 | emma.stone@gmail.com    |
|        2 | Liam         | Neeson   | Irish         |      1952 | liam.neeson@gmail.com   |
|        3 | Sofia        | Vergara  | Colombian     |      1972 | sofia.vergara@gmail.com |
|        4 | Chris        | Evans    | American      |      1981 | chris.evans@gmail.com   |
|        5 | Kwenzokwakhe | Cele     | South African |      1969 | kwakhe.cele@gmail.com   |
+----------+--------------+----------+---------------+-----------+-------------------------+
5 rows in set (0.00 sec)
```

Jeza_Books



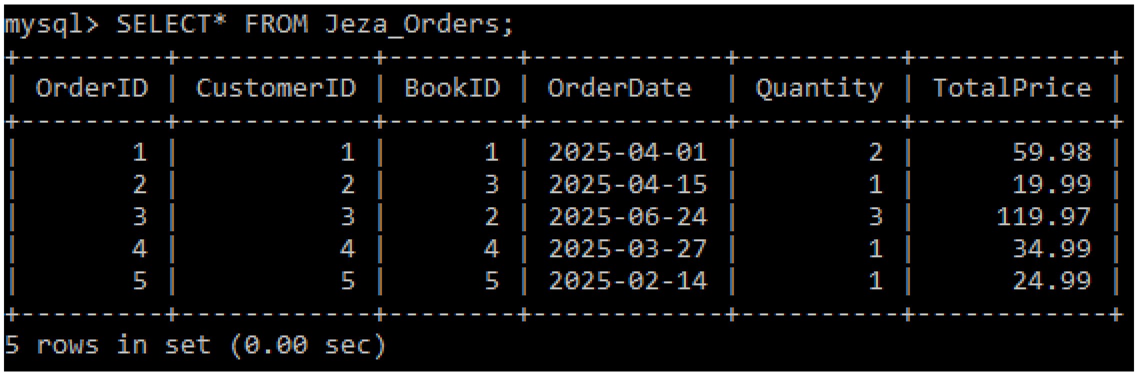
```
mysql> SELECT* FROM Jeza_Books;
+--------+------------------+----------+-------+-----------------+-------------+
| BookID | Title            | AuthorID | Price | PublicationYear | Genre       |
+--------+------------------+----------+-------+-----------------+-------------+
|      1 | The Great Novel  |        1 | 29.99 |            2020 | Fiction     |
|      2 | Tech Trends      |        2 | 39.99 |            2021 | Non-Fiction |
|      3 | Mystery Tales    |        1 | 19.99 |            2019 | Mystery     |
|      4 | History Unveiled |        3 | 34.99 |            2022 | History     |
|      5 | Back Then!       |        4 | 24.99 |            2015 | History     |
+--------+------------------+----------+-------+-----------------+-------------+
5 rows in set (0.00 sec)
```

Jeza_Customers



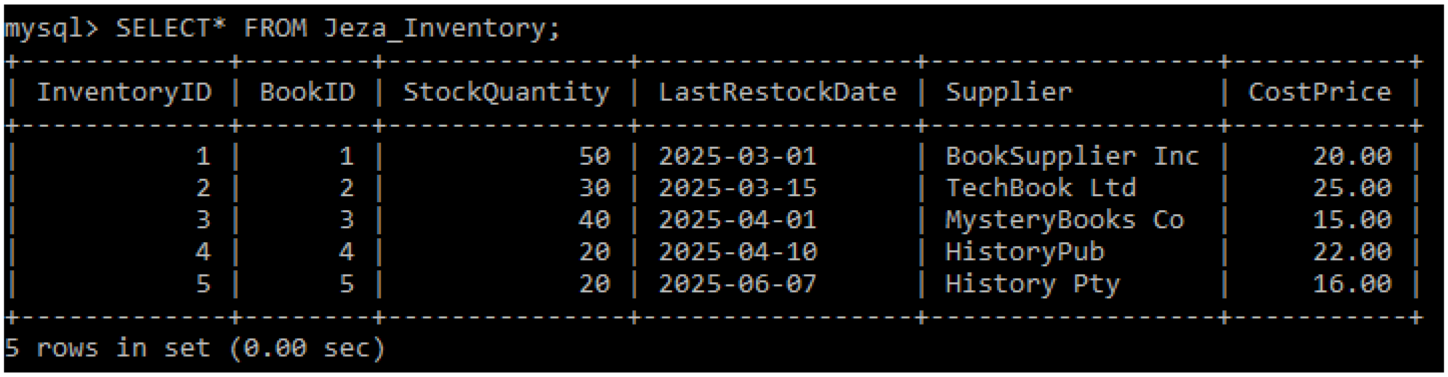
```
mysql> SELECT* FROM Jeza_Customers;
+------------+-----------+----------+-----------------------+------------+-------------------+
| CustomerID | FirstName | LastName | Email                 | Phone      | Address           |
+------------+-----------+----------+-----------------------+------------+-------------------+
|          1 | John      | Dou      | john.doe@gmail.com    | 0790745617 | 123 Main St       |
|          2 | Jane      | Smith    | jane.smith@gmail.com  | 0647651109 | 456 Oak Ave       |
|          3 | Alice     | Brown    | alice.brown@gmail.com | 0875550000 | 789 Pine Rd       |
|          4 | Bob       | Wilson   | bob.wilsono@gmail.com | 0615672236 | 321 Elm St        |
|          5 | Mike      | Wills    | mike.wills@gmail.com  | 0693415204 | 20 Voortrekker Rd |
+------------+-----------+----------+-----------------------+------------+-------------------+
5 rows in set (0.01 sec)
```

Jeza_Orders



```
mysql> SELECT* FROM Jeza_Orders;
+---------+------------+--------+------------+----------+------------+
| OrderID | CustomerID | BookID | OrderDate  | Quantity | TotalPrice |
+---------+------------+--------+------------+----------+------------+
|       1 |          1 |      1 | 2025-04-01 |        2 |      59.98 |
|       2 |          2 |      3 | 2025-04-15 |        1 |      19.99 |
|       3 |          3 |      2 | 2025-06-24 |        3 |     119.97 |
|       4 |          4 |      4 | 2025-03-27 |        1 |      34.99 |
|       5 |          5 |      5 | 2025-02-14 |        1 |      24.99 |
+---------+------------+--------+------------+----------+------------+
5 rows in set (0.00 sec)
```

Jeza_Inventory



```
mysql> SELECT* FROM Jeza_Inventory;
+-------------+--------+---------------+-----------------+-------------------+-----------+
| InventoryID | BookID | StockQuantity | LastRestockDate | Supplier          | CostPrice |
+-------------+--------+---------------+-----------------+-------------------+-----------+
|           1 |      1 |            50 | 2025-03-01      | BookSupplier Inc  |     20.00 |
|           2 |      2 |            30 | 2025-03-15      | TechBook Ltd      |     25.00 |
|           3 |      3 |            40 | 2025-04-01      | MysteryBooks Co   |     15.00 |
|           4 |      4 |            20 | 2025-04-10      | HistoryPub        |     22.00 |
|           5 |      5 |            20 | 2025-06-07      | History Pty       |     16.00 |
+-------------+--------+---------------+-----------------+-------------------+-----------+
5 rows in set (0.00 sec)
```

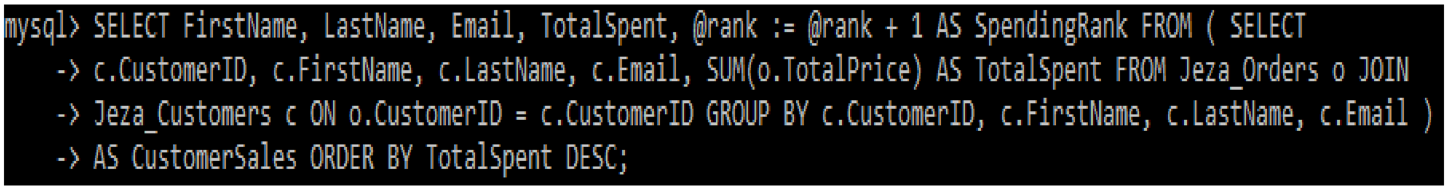
1. Advanced Query – Derived Table with Variable-based Ranking

1.1. Introduction

Since MySQL 5.5 does not allow Common Table Expressions or Window functions, I decided to show how to generate a customer sales report with ranking using a derived table and user-defined variables. While variables allow for ranking functionality, derived tables allow sophisticated queries to be built as subqueries in the FROM clause. This question demonstrates my complex interest in the more complex aspects of MySQL 5.5 and my flexibility in dealing with version-specific constraints.

1.2. Query Description

The query generates a report listing each customer's name, email, total spending, and their rank based on spending.
Here is the SQL code below:



```
mysql> SELECT FirstName, LastName, Email, TotalSpent, @rank := @rank + 1 AS SpendingRank FROM ( SELECT
    -> c.CustomerID, c.FirstName, c.LastName, c.Email, SUM(o.TotalPrice) AS TotalSpent FROM Jeza_Orders o JOIN
    -> Jeza_Customers c ON o.CustomerID = c.CustomerID GROUP BY c.CustomerID, c.FirstName, c.LastName, c.Email )
    -> AS CustomerSales ORDER BY TotalSpent DESC;
```

The derived table calculates total spending per customer by joining **Jeza_Orders** and **Jeza_Customers** and grouping by customer details. The outer query uses a variable (**@rank**) to assign ranks based on total spending, ordered from highest to lowest.

1.3. Exploration and Opportunities

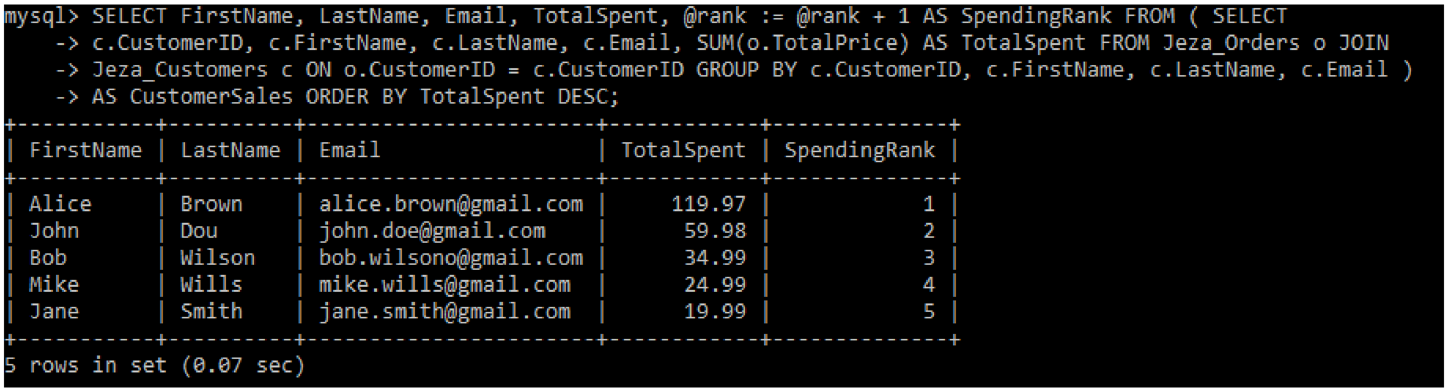
Derived tables make complex queries easier to interpret by structuring them without the need for CTEs. When window functions are not available in MySQL 5.5, user-defined variables provide an innovative way to rank. According to my research this strategy can be used to various ranking situations, such author or book rankings by publications or sales. However, in multi-user systems, variables must be carefully initialized to prevent mistakes. Additional metrics (like order count) or criteria (like minimum spending) could improve this query.

1.4. Curiosity and Learning

I explored derived tables and variables to overcome MySQL 5.5's limitations, learning how to minic modern ranking functions. The challenge of adapting to an older MySQL version deepened my understanding of query optimization and variable management. This demonstrates my ability to apply advanced techniques to practical bookstore reporting, such as identifying top customers for targeted promotions.

1.5. Output and Validation

The query was executed in MySQL 5.5, producing a result set with customer details, total spending, and ranks.



```
mysql> SELECT FirstName, LastName, Email, TotalSpent, @rank := @rank + 1 AS SpendingRank FROM ( SELECT
    -> c.CustomerID, c.FirstName, c.LastName, c.Email, SUM(o.TotalPrice) AS TotalSpent FROM Jeza_Orders o JOIN
    -> Jeza_Customers c ON o.CustomerID = c.CustomerID GROUP BY c.CustomerID, c.FirstName, c.LastName, c.Email )
    -> AS CustomerSales ORDER BY TotalSpent DESC;
+-----------+----------+----------------------+------------+--------------+
| FirstName | LastName | Email                | TotalSpent | SpendingRank |
+-----------+----------+----------------------+------------+--------------+
| Alice     | Brown    | alice.brown@gmail.com |     119.97 |            1 |
| John      | Dou      | john.doe@gmail.com   |      59.98 |            2 |
| Bob       | Wilson   | bob.wilsono@gmail.com |      34.99 |            3 |
| Mike      | Wills    | mike.wills@gmail.com |      24.99 |            4 |
| Jane      | Smith    | jane.smith@gmail.com |      19.99 |            5 |
+-----------+----------+----------------------+------------+--------------+
5 rows in set (0.07 sec)
```

For example, Alice Brown ranked first with R119.97 in spending. The query worked correctly with the sample data, and foreign key constraints ensured accurate joins. Performance was suitable for the small dataset, though optimization may be needed for larger databases.

1.6. Conclusion

Derived tables and user-defined variables are powerful tools for advanced reporting in MySQL. This demonstration highlights my exploration of version-compatible techniques and their application in the bookstore database. Future improvements could include temporary tables for better performance or additional ranking criteria.

2. Store Procedure – Update Order Totals

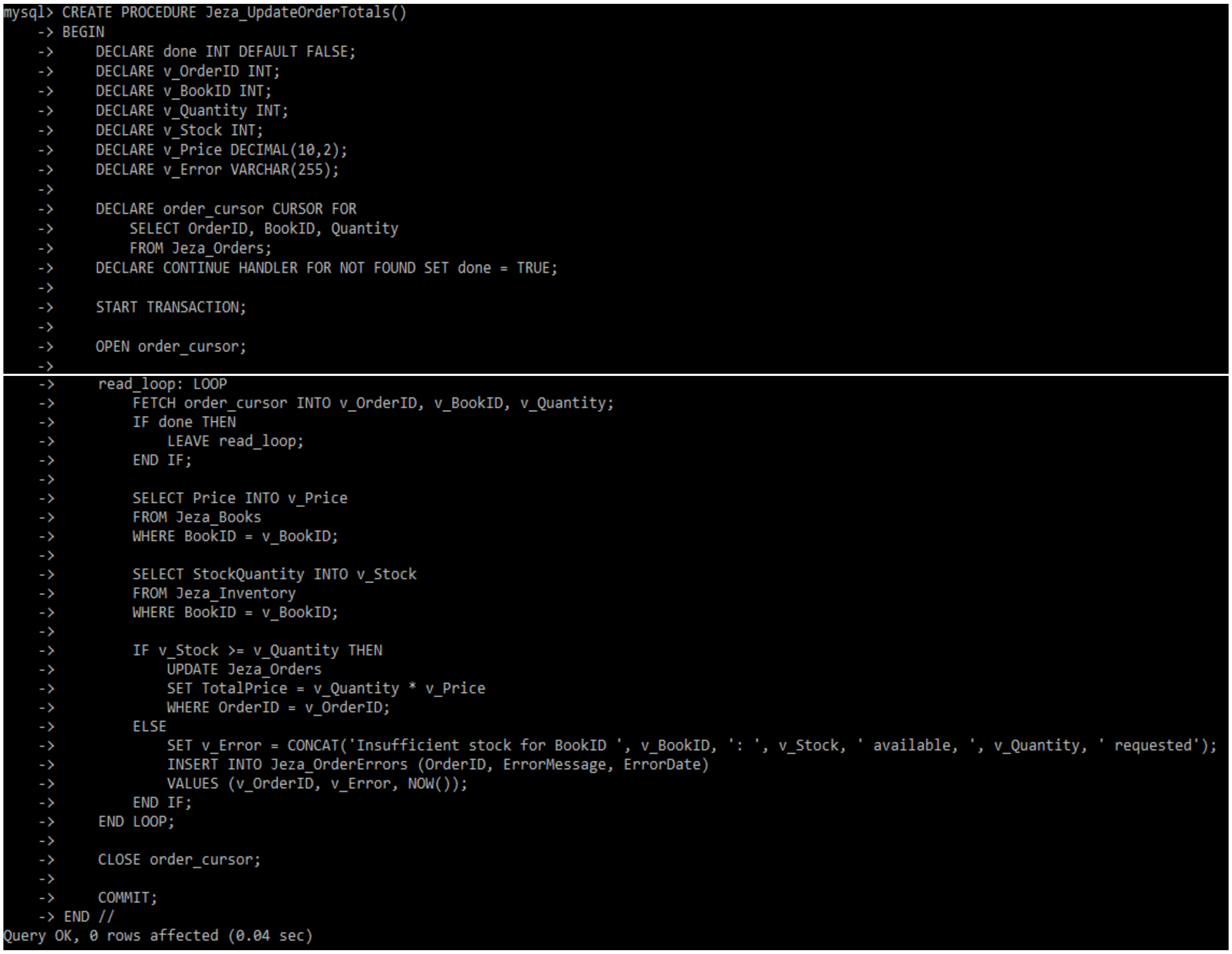
2.1. Introduction

In order to update the **TotalPrice** column in the **Jeza_Orders** table based on book pricing and quantities, along with a stock availability check, I created a Stored Procedure called **Jeza_UpdateOrderTotals** in MySQL. By encapsulating business logic, stored procedures increase security and efficiency. This example demonstrates my investigation of MySQL 5.5's more complex features, such as cursors, transactions, and error handling, as they relate to the bookstore database.

2.2. Procedure Description

The Stored Procedure updates **TotalPrice** in **Jeza_Orders** by multiplying **Quantity** with the corresponding Price from **Jeza_Books**. It checks stock availability in **Jeza_Inventory** before updating, logging errors to a new **Jeza_OrderErrors** table if stock is insufficient.

The SQL code below:



```
mysql> CREATE PROCEDURE Jeza_UpdateOrderTotals()
    -> BEGIN
    ->     DECLARE done INT DEFAULT FALSE;
    ->     DECLARE v_OrderID INT;
    ->     DECLARE v_BookID INT;
    ->     DECLARE v_Quantity INT;
    ->     DECLARE v_Stock INT;
    ->     DECLARE v_Price DECIMAL(10,2);
    ->     DECLARE v_Error VARCHAR(255);
    ->
    ->     DECLARE order_cursor CURSOR FOR
    ->         SELECT OrderID, BookID, Quantity
    ->         FROM Jeza_Orders;
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    ->
    ->     START TRANSACTION;
    ->
    ->     OPEN order_cursor;
    ->
    ->     read_loop: LOOP
    ->         FETCH order_cursor INTO v_OrderID, v_BookID, v_Quantity;
    ->         IF done THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         SELECT Price INTO v_Price
    ->         FROM Jeza_Books
    ->         WHERE BookID = v_BookID;
    ->
    ->         SELECT StockQuantity INTO v_Stock
    ->         FROM Jeza_Inventory
    ->         WHERE BookID = v_BookID;
    ->
    ->         IF v_Stock >= v_Quantity THEN
    ->             UPDATE Jeza_Orders
    ->             SET TotalPrice = v_Quantity * v_Price
    ->             WHERE OrderID = v_OrderID;
    ->         ELSE
    ->             SET v_Error = CONCAT('Insufficient stock for BookID ', v_BookID, ': ', v_Stock, ' available, ', v_Quantity, ' requested');
    ->             INSERT INTO Jeza_OrderErrors (OrderID, ErrorMessage, ErrorDate)
    ->             VALUES (v_OrderID, v_Error, NOW());
    ->         END IF;
    ->     END LOOP;
    ->
    ->     CLOSE order_cursor;
    ->
    ->     COMMIT;
    -> END //
Query OK, 0 rows affected (0.04 sec)
```

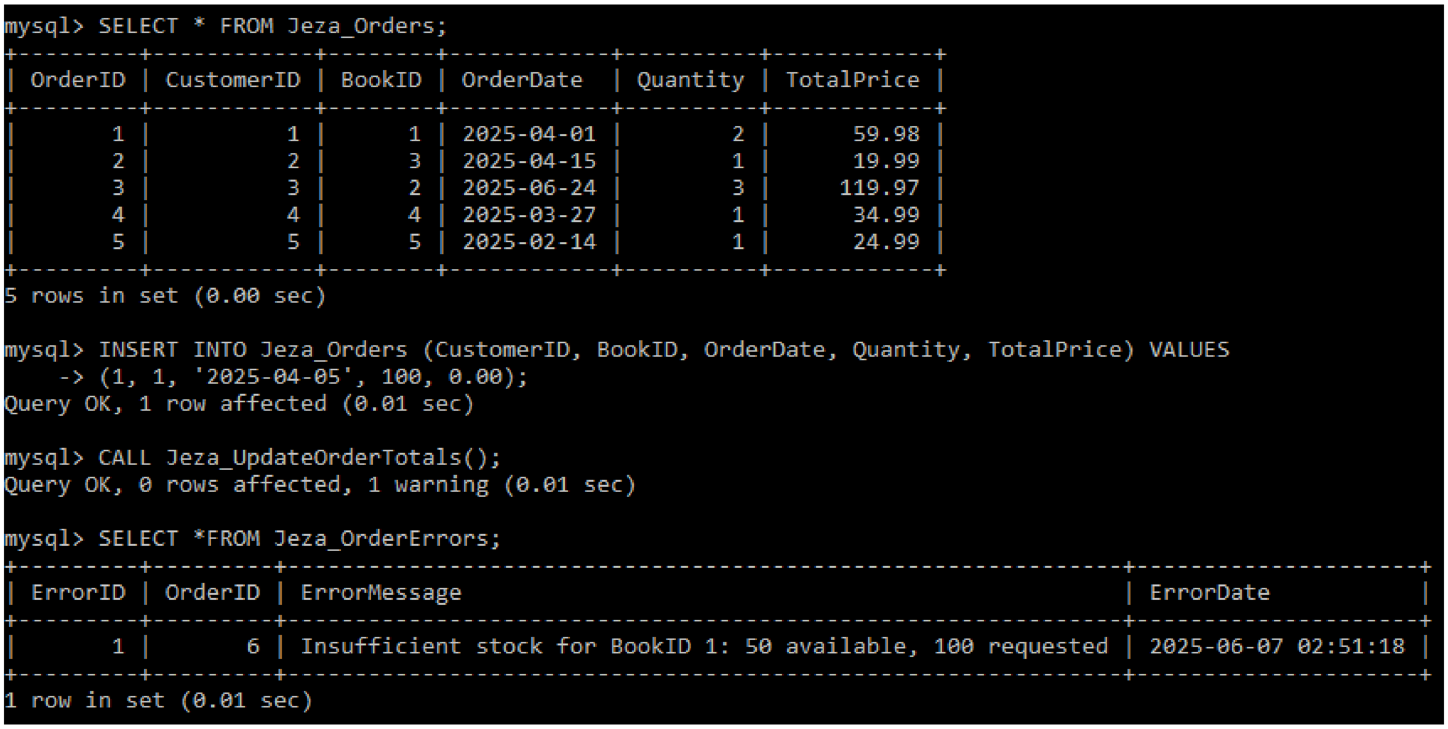
2.3. Exploration and Opportunities

Orders are processed row by row using a cursor, transactions are used to guarantee data consistency, and error handling is used to record problems. Opportunities provided by stored procedures include integrating tax computations, automating pricing adjustments following discounts, and sending out alerts for low stock. They can be scheduled using Events and improve security by limiting direct table access. According to my research, Stored Procedures are perfect for batch processing and may be expanded with parameters to manage particular date ranges or orders.

2.4. Curiosity and Learning

I explored Stored Procedures to understand how they centralize business logic, learning to use cursors and transactions in MySQL. Implementing error handling was challenging but demonstrated the importance of data validation. This procedure applies to real world Bookstore scenarios, such as ensuring accurate billing and inventory management.

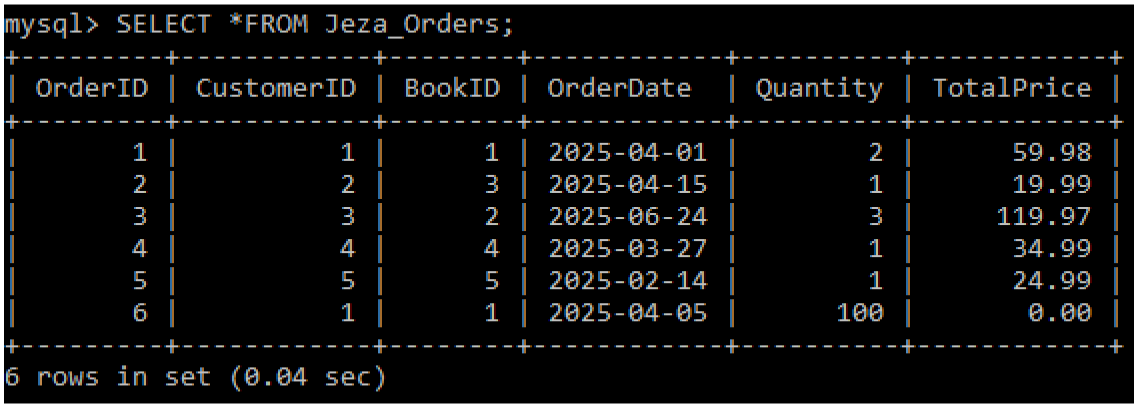
2.5. Output and Validation



```
mysql> SELECT * FROM Jeza_Orders;
+---------+------------+--------+------------+----------+------------+
| OrderID | CustomerID | BookID | OrderDate  | Quantity | TotalPrice |
+---------+------------+--------+------------+----------+------------+
|       1 |          1 |      1 | 2025-04-01 |        2 |      59.98 |
|       2 |          2 |      3 | 2025-04-15 |        1 |      19.99 |
|       3 |          3 |      2 | 2025-06-24 |        3 |     119.97 |
|       4 |          4 |      4 | 2025-03-27 |        1 |      34.99 |
|       5 |          5 |      5 | 2025-02-14 |        1 |      24.99 |
+---------+------------+--------+------------+----------+------------+
5 rows in set (0.00 sec)

mysql> INSERT INTO Jeza_Orders (CustomerID, BookID, OrderDate, Quantity, TotalPrice) VALUES
    -> (1, 1, '2025-04-05', 100, 0.00);
Query OK, 1 row affected (0.01 sec)

mysql> CALL Jeza_UpdateOrderTotals();
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> SELECT *FROM Jeza_OrderErrors;
+---------+---------+-------------------------------------------------------------+---------------------+
| ErrorID | OrderID | ErrorMessage                                                | ErrorDate           |
+---------+---------+-------------------------------------------------------------+---------------------+
|       1 |       6 | Insufficient stock for BookID 1: 50 available, 100 requested | 2025-06-07 02:51:18 |
+---------+---------+-------------------------------------------------------------+---------------------+
1 row in set (0.01 sec)
```

The Procedure was executed in MySQL 5.5, updating **Jeza_Orders** correctly for sample data (e.g. John Doe's order at R59.98). A test order with excessive quantity of 100 for **BookID 1** logged an error in **Jeza_OrderErrors** as show on the screenshot above. The procedure ensured data integrity and performed efficiently for the small dataset.

Updated **Jeza_Orders** table:



```
mysql> SELECT *FROM Jeza_Orders;
+---------+------------+--------+------------+----------+------------+
| OrderID | CustomerID | BookID | OrderDate  | Quantity | TotalPrice |
+---------+------------+--------+------------+----------+------------+
|       1 |          1 |      1 | 2025-04-01 |        2 |      59.98 |
|       2 |          2 |      3 | 2025-04-15 |        1 |      19.99 |
|       3 |          3 |      2 | 2025-06-24 |        3 |     119.97 |
|       4 |          4 |      4 | 2025-03-27 |        1 |      34.99 |
|       5 |          5 |      5 | 2025-02-14 |        1 |      24.99 |
|       6 |          1 |      1 | 2025-04-05 |      100 |       0.00 |
+---------+------------+--------+------------+----------+------------+
6 rows in set (0.04 sec)
```

2.6. Conclusion

Stored Procedure are powerful for automating complex tasks in MySQL. This demonstration highlights my ability to implement business logic and explore features like error handling and transactions. Future enhancements could include parameterized inputs or integration with other database operations.

3. Trigger – Update Inventory after Order Insert

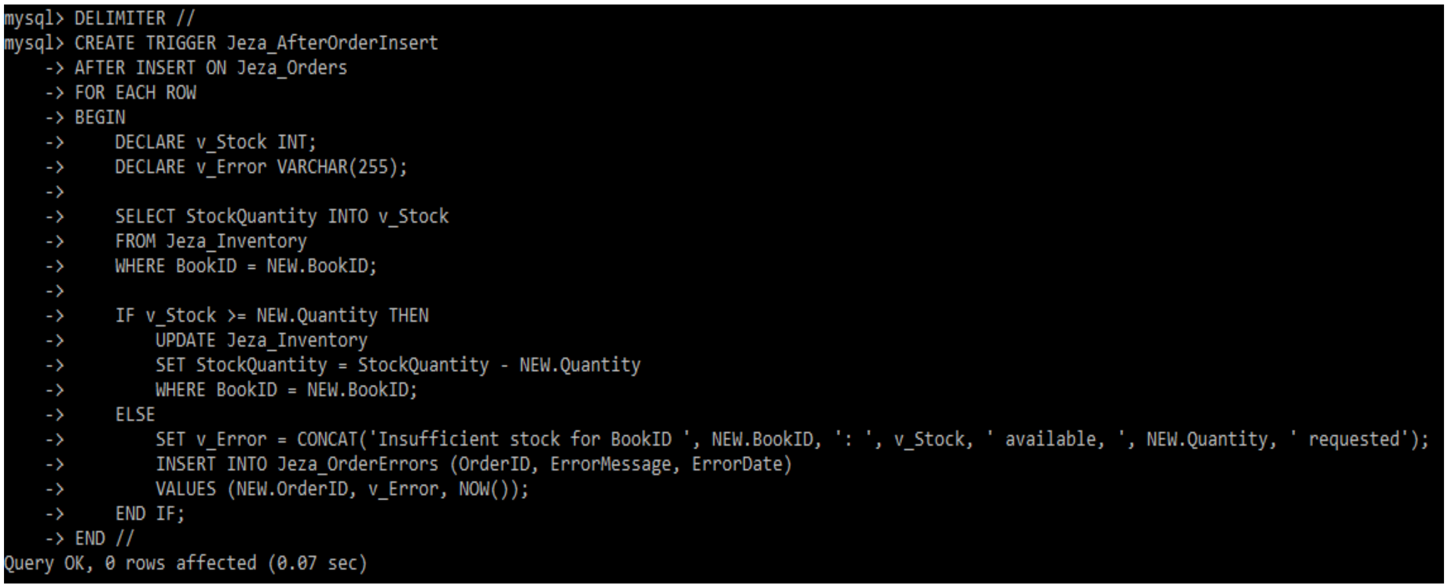
3.1. Introduction

In order to automatically update the **Jeza_Inventory** database whenever a new order is entered into **Jeza_Orders**, I created an **AFTER INSERT** Trigger in My SQL called **Jeza_AfterOrderInsert**. If there is not enough stock, the Trigger records errors in **Jeza_OrderErrors** after verifying stock availability. This illustrates my extensive MySQL skills and my ability to modify **INSERT** and **UPDATE** operations to automate inventory management in the bookstore database.

3.2. Trigger Description

The Trigger fires after each INSERT on **Jeza_Orders**, reducing **StockQuantity** in **Jeza_Inventory** for the ordered **BookID** if sufficient stock exists. Otherwise, it logs an error in **Jeza_OrderErrors**.

The SQL code below:



```
mysql> DELIMITER //
mysql> CREATE TRIGGER Jeza_AfterOrderInsert
    -> AFTER INSERT ON Jeza_Orders
    -> FOR EACH ROW
    -> BEGIN
    ->     DECLARE v_Stock INT;
    ->     DECLARE v_Error VARCHAR(255);
    ->
    ->     SELECT StockQuantity INTO v_Stock
    ->     FROM Jeza_Inventory
    ->     WHERE BookID = NEW.BookID;
    ->
    ->     IF v_Stock >= NEW.Quantity THEN
    ->         UPDATE Jeza_Inventory
    ->         SET StockQuantity = StockQuantity - NEW.Quantity
    ->         WHERE BookID = NEW.BookID;
    ->     ELSE
    ->         SET v_Error = CONCAT('Insufficient stock for BookID ', NEW.BookID, ': ', v_Stock, ' available, ', NEW.Quantity, ' requested');
    ->         INSERT INTO Jeza_OrderErrors (OrderID, ErrorMessage, ErrorDate)
    ->         VALUES (NEW.OrderID, v_Error, NOW());
    ->     END IF;
    -> END //
Query OK, 0 rows affected (0.07 sec)
```

3.3. Skills and Adaptations

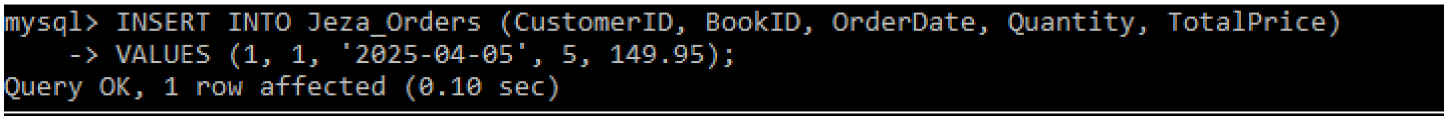
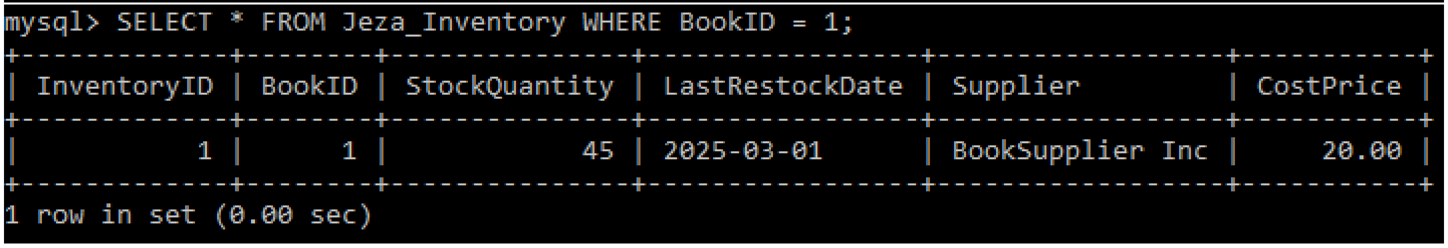
My understanding of **INSERT** and **UPDATE** is applied to a new setting real-time stock management-by the Trigger. It makes advantage of MySQL advanced capabilities, such as variable declarations and conditional logic (IF...ELSE). In addition to integrating with Store Procedure error logging to demonstrate a coherent database design, the Trigger guarantees data consistency, which is essential for Bookstore operations.

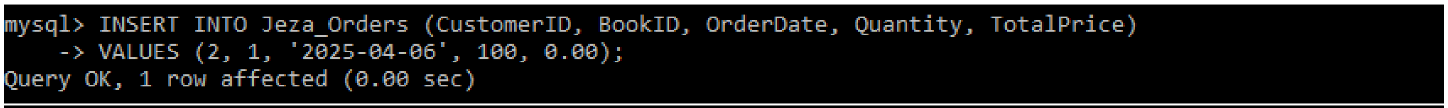
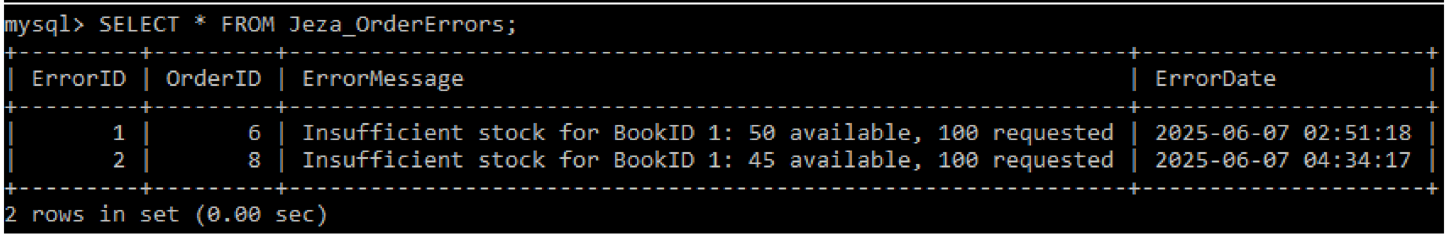
3.4. Exploration and Opportunities

Triggers reduce the need for human updates by automating data integrity procedures. It would be possible to expand this trigger to handle order cancellations (restocking inventory) or alert suppliers to low stock levels. Triggers are helpful for audit trails and enforcing business standards, according to my research; nevertheless, they need to be carefully tested to prevent unexpected modifications. For thorough automation, they can be used in conjunction with Stored Procedure.

3.5. Output and Validation

The Trigger was tested in MySQL 5.5 with two INSERT statements:
One valid order for **BookID** 1 of 5 quantity orders reduce stock from 50 to 45 in **Jeza_Inventory**.


```
mysql> INSERT INTO Jeza_Orders (CustomerID, BookID, OrderDate, Quantity, TotalPrice)
    -> VALUES (1, 1, '2025-04-05', 5, 149.95);
Query OK, 1 row affected (0.10 sec)
```

```
mysql> SELECT * FROM Jeza_Inventory WHERE BookID = 1;
+-------------+--------+---------------+-----------------+------------------+-----------+
| InventoryID | BookID | StockQuantity | LastRestockDate | Supplier         | CostPrice |
+-------------+--------+---------------+-----------------+------------------+-----------+
|           1 |      1 |            45 | 2025-03-01      | BookSupplier Inc |     20.00 |
+-------------+--------+---------------+-----------------+------------------+-----------+
1 row in set (0.00 sec)
```
And one invalid order of 100 quantity orders logged an error in **Jeza_OrderErrors**.


```
mysql> INSERT INTO Jeza_Orders (CustomerID, BookID, OrderDate, Quantity, TotalPrice)
    -> VALUES (2, 1, '2025-04-06', 100, 0.00);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM Jeza_OrderErrors;
+---------+---------+---------------------------------------------------------------+---------------------+
| ErrorID | OrderID | ErrorMessage                                                  | ErrorDate           |
+---------+---------+---------------------------------------------------------------+---------------------+
|       1 |       6 | Insufficient stock for BookID 1: 50 available, 100 requested | 2025-06-07 02:51:18 |
|       2 |       8 | Insufficient stock for BookID 1: 45 available, 100 requested | 2025-06-07 04:34:17 |
+---------+---------+---------------------------------------------------------------+---------------------+
2 rows in set (0.00 sec)
```
The results confirmed correct functionality, with foreign keys ensuring valid BookID values.

3.6. Conclusion

Triggers are powerful for automating database operations in MySQL. This demonstration highlights my ability to adapt DML operations to new scenarios, ensuring inventory consistency. Future enhancements include additional Triggers for other events or improved error handling.

4. Cursor, Function, and Event

4.1. Introduction

In order to show advanced database management in the **Jeza_230878369** Bookstore database, I developed a Function, Cursor, and Event in MySQL. The Event plans weekly inventory inspections, the Cursor records low-stock notifications, and the Function determines discounted book prices. These elements showcase contemporary database system concepts like automation, proactive monitoring, and modularity.

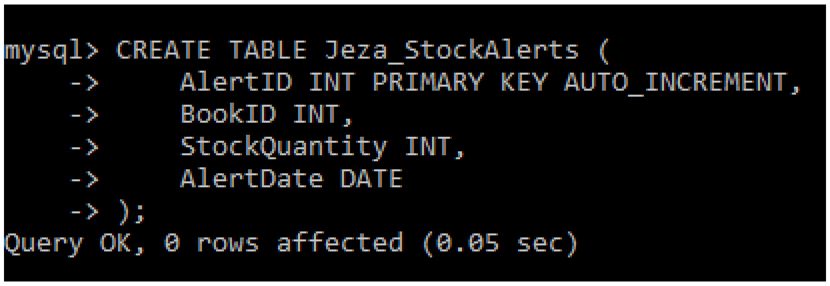
4.2. Components Description

4.2.1. Function (**Jeza_GetDiscountedPrice**)

Takes a **BookID** and **DiscountPercent**, returning the discounted price from **Jeza_Books**. It uses **DETERMINISTIC** for performance and handles invalid **BookID**.

4.2.2. Cursor (**Jeza_CheckLowStock**)

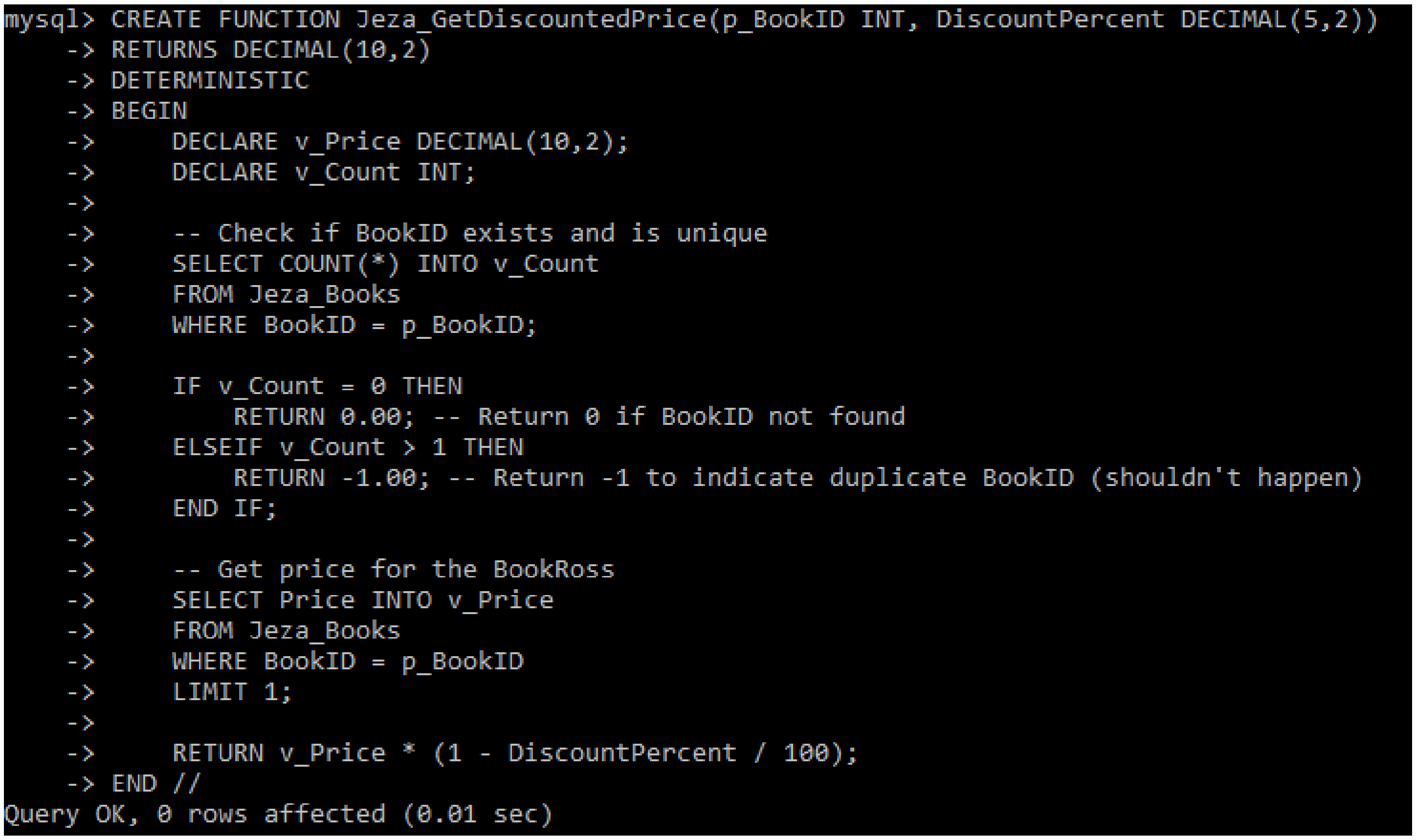
Iterates through **Jeza_Inventory** to find books with **StockQuantity < 25**, logging alert in **Jeza_StockAlerts**.



```
mysql> CREATE TABLE Jeza_StockAlerts (
    ->     AlertID INT PRIMARY KEY AUTO_INCREMENT,
    ->     BookID INT,
    ->     StockQuantity INT,
    ->     AlertDate DATE
    -> );
Query OK, 0 rows affected (0.05 sec)
```

4.2.3. Event (**Jeza_WeeklyInventoryCheck**)

Schedules Jeza_CheckLowStock to run weekly, automating inventory monitoring.
The SQL code below:



```
mysql> CREATE FUNCTION Jeza_GetDiscountedPrice(p_BookID INT, DiscountPercent DECIMAL(5,2))
    -> RETURNS DECIMAL(10,2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE v_Price DECIMAL(10,2);
    ->     DECLARE v_Count INT;
    ->
    ->     -- Check if BookID exists and is unique
    ->     SELECT COUNT(*) INTO v_Count
    ->     FROM Jeza_Books
    ->     WHERE BookID = p_BookID;
    ->
    ->     IF v_Count = 0 THEN
    ->         RETURN 0.00; -- Return 0 if BookID not found
    ->     ELSEIF v_Count > 1 THEN
    ->         RETURN -1.00; -- Return -1 to indicate duplicate BookID (shouldn't happen)
    ->     END IF;
    ->
    ->     -- Get price for the BookRoss
    ->     SELECT Price INTO v_Price
    ->     FROM Jeza_Books
    ->     WHERE BookID = p_BookID
    ->     LIMIT 1;
    ->
    ->     RETURN v_Price * (1 - DiscountPercent / 100);
    -> END //
Query OK, 0 rows affected (0.01 sec)
```

4.3. Current Trends

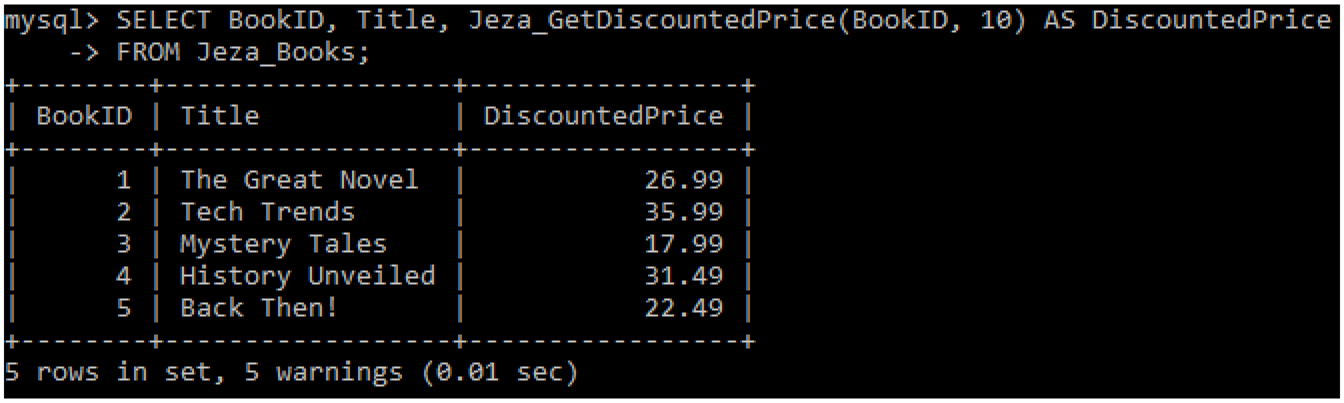
Functions encourage modularity, which is a trend in contemporary databases for reusable logic (e.g. e-commerce price calculations). Although set based alternatives have made them less popular, cursors are still utilized in older systems like MySQL 5.5 for row-by-row processing, including alert generation. Events show trends in automation, making it possible to perform pre-emptive operations like inventory checks-which are frequently performed in cloud databases-without the need for human intervention.

4.4. Exploration and Learning

I explored these components to understand their roles in database management. Functions were straightforward, enhancing code reuse. Cursors required learning loop handling, revealing their limitations in performance. Events showcased automation, aligning with trends in scheduled maintenance. This applied to Bookstore inventory management, ensuring timely restocking.

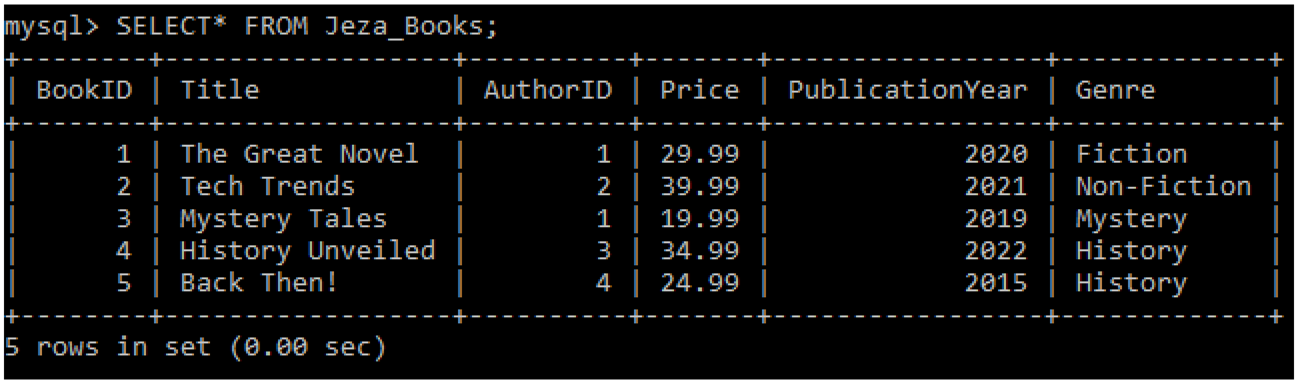
4.5. Output and Validation

The Function returned correct discounted prices (e.g. R26.99 for **BookID 1** at 10% discount from R29.99 original price):



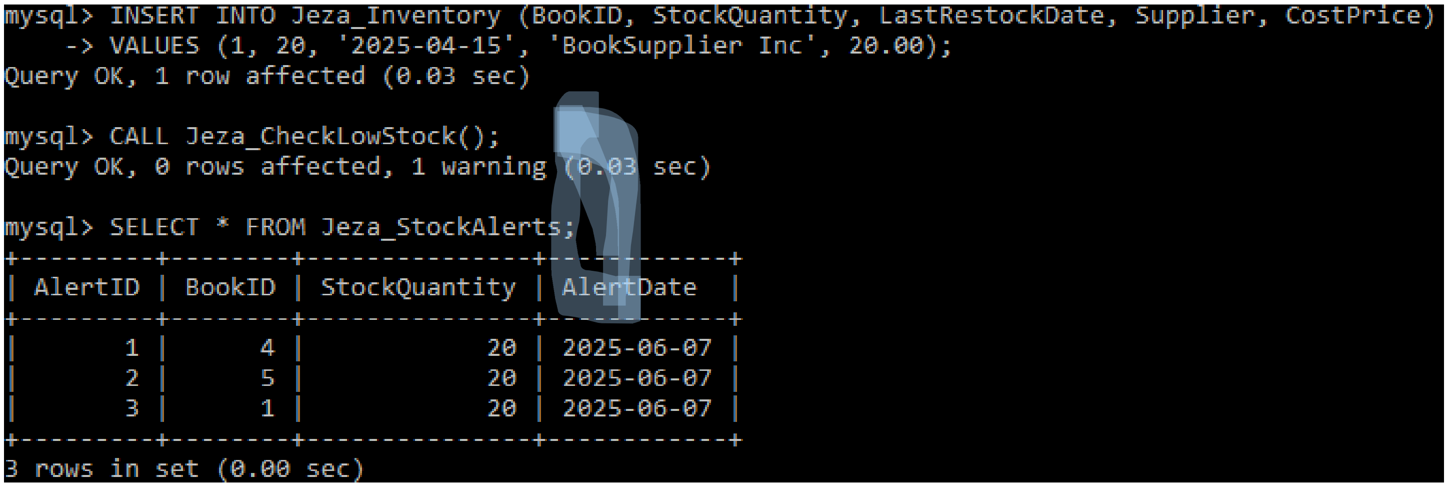
```
mysql> SELECT BookID, Title, Jeza_GetDiscountedPrice(BookID, 10) AS DiscountedPrice
    -> FROM Jeza_Books;
+--------+------------------+-----------------+
| BookID | Title            | DiscountedPrice |
+--------+------------------+-----------------+
|      1 | The Great Novel  |           26.99 |
|      2 | Tech Trends      |           35.99 |
|      3 | Mystery Tales    |           17.99 |
|      4 | History Unveiled |           31.49 |
|      5 | Back Then!       |           22.49 |
+--------+------------------+-----------------+
5 rows in set, 5 warnings (0.01 sec)
```

Prices before discount:



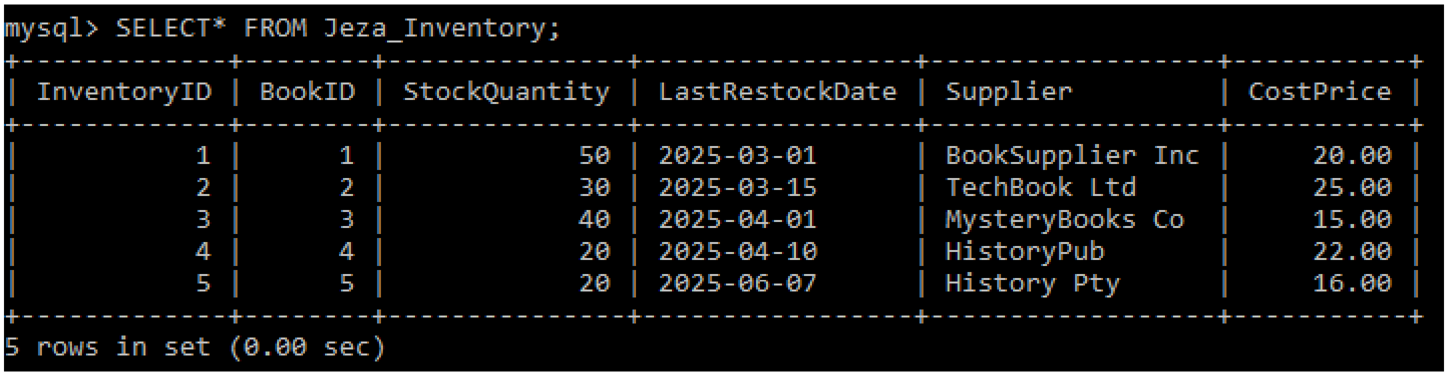
```
mysql> SELECT* FROM Jeza_Books;
+--------+------------------+----------+-------+-----------------+-------------+
| BookID | Title            | AuthorID | Price | PublicationYear | Genre       |
+--------+------------------+----------+-------+-----------------+-------------+
|      1 | The Great Novel  |        1 | 29.99 |            2020 | Fiction     |
|      2 | Tech Trends      |        2 | 39.99 |            2021 | Non-Fiction |
|      3 | Mystery Tales    |        1 | 19.99 |            2019 | Mystery     |
|      4 | History Unveiled |        3 | 34.99 |            2022 | History     |
|      5 | Back Then!       |        4 | 24.99 |            2015 | History     |
+--------+------------------+----------+-------+-----------------+-------------+
5 rows in set (0.00 sec)
```

The Cursor logged low-stock alerts for **BookID 1** (stock 20) in **Jeza_StockAlerts**:



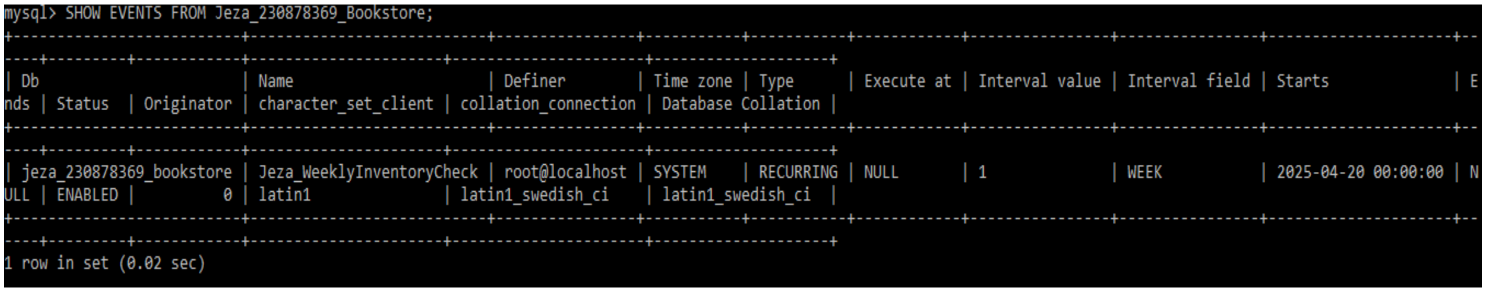
```
mysql> INSERT INTO Jeza_Inventory (BookID, StockQuantity, LastRestockDate, Supplier, CostPrice)
    -> VALUES (1, 20, '2025-04-15', 'BookSupplier Inc', 20.00);
Query OK, 1 row affected (0.03 sec)

mysql> CALL Jeza_CheckLowStock();
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> SELECT * FROM Jeza_StockAlerts;
+---------+--------+---------------+------------+
| AlertID | BookID | StockQuantity | AlertDate  |
+---------+--------+---------------+------------+
|       1 |      4 |            20 | 2025-06-07 |
|       2 |      5 |            20 | 2025-06-07 |
|       3 |      1 |            20 | 2025-06-07 |
+---------+--------+---------------+------------+
3 rows in set (0.00 sec)
```

The stock before any orders:



```
mysql> SELECT* FROM Jeza_Inventory;
+-------------+--------+---------------+-----------------+------------------+-----------+
| InventoryID | BookID | StockQuantity | LastRestockDate | Supplier         | CostPrice |
+-------------+--------+---------------+-----------------+------------------+-----------+
|           1 |      1 |            50 | 2025-03-01      | BookSupplier Inc |     20.00 |
|           2 |      2 |            30 | 2025-03-15      | TechBook Ltd     |     25.00 |
|           3 |      3 |            40 | 2025-04-01      | MysteryBooks Co  |     15.00 |
|           4 |      4 |            20 | 2025-04-10      | HistoryPub       |     22.00 |
|           5 |      5 |            20 | 2025-06-07      | History Pty      |     16.00 |
+-------------+--------+---------------+-----------------+------------------+-----------+
5 rows in set (0.00 sec)
```

The Event was created successfully, verified via **SHOW EVENTS**.



```
mysql> SHOW EVENTS FROM Jeza_230878369_Bookstore;
+--------------------------+---------------------------+----------------+-----------+-----------+------------+----------------+----------------+---------------------+--
----+---------+-----------+----------------------+----------------------+--------------------+
| Db                       | Name                      | Definer        | Time zone | Type      | Execute at | Interval value | Interval field | Starts              | E
nds | Status  | Originator | character_set_client | collation_connection | Database Collation |
+--------------------------+---------------------------+----------------+-----------+-----------+------------+----------------+----------------+---------------------+--
----+---------+-----------+----------------------+----------------------+--------------------+
| jeza_230878369_bookstore | Jeza_WeeklyInventoryCheck | root@localhost | SYSTEM    | RECURRING | NULL       | 1              | WEEK           | 2025-04-20 00:00:00 | N
ULL | ENABLED |         0 | latin1               | latin1_swedish_ci    | latin1_swedish_ci  |
+--------------------------+---------------------------+----------------+-----------+-----------+------------+----------------+----------------+---------------------+--
----+---------+-----------+----------------------+----------------------+--------------------+
1 row in set (0.02 sec)
```

The tests in MySQL confirmed functionality, with foreign keys ensuring data integrity.

4.6. Conclusion

Functions, cursors, and Events enhance database management in MySQL. This demonstration highlights my understanding of modular and automated solutions, reflecting current trends. Future improvements could include parameterized Cursors or Event-driven notifications.

5. Reflection on Advanced Queries and Features

By applying sophisticated MySQL capabilities to the **Jeza_230878369** Bookstore database in Tasks 1-4, important insights into database administration were gained, highlighting the advantages and disadvantages of various methods. This reflection discusses the difficulties, lessons learned, and possible uses of the derived table with variable-based ranking (Task 1), stored procedure (Task 2), trigger (Task 3), and cursor, function, event (Task 4).

Task 1: Derived Table with variable-based Ranking

The derived table query, which ranked customers by total spending, effectively generated a sales report, leveraging MySQL's subquery and variable capabilities. It was intuitive to structure but challenging due to MySQL 5.5's lack of window functions, requiring a workaround with user-defined variables (**@rank**). This approach was less robust in multi-user environments, teaching me the importance of understanding version-specific constrains. The query's reporting capability could be applied to other systems, such as ranking products by sales in e-commerce platforms.

Task 2: Stored Procedure

The **Jeza_UpdateOrderTotals** Stored Procedure automated price calculations with stock checks, enhancing efficiency and data integrity. Implementing cursors and transactions was challenging, as MySQL 5.5 limited error handling required careful variable management. I learned to encapsulate complex logic, a trend in modular database design. This procedure could be adapted for batch processing in financial systems, ensuring accurate transaction updates.

Task 3: Trigger

The **Jeza_AfterOrderInsert** Trigger maintained inventory consistency by updating stock after order insertions, demonstrating automation. Debugging the Trigger's conditional logic was difficult, as MySQL 5.5 lacks advanced debugging tools. This taught me the importance of testing data integrity constraints. Triggers could be used in warehouse systems to automate stock adjustments or audit trails.

Task 4: Cursor, Function and Event

The **Jeza_GetDiscountedPrice** Function, **Jeza_CheckLowStock** Cursor, and **Jeza_WeeklyInventoryCheck** Event showcased modularity and automation. The Function's initial error ("Result consisted of more than one row") highlighted the critical role of **PRIMARY KEY** constraints, resolved by adding checks and **LIMIT 1**. Cursors were less efficient than set-based queries, revealing their niche use in row-by-row tasks. The Event aligned with automation trends, scheduling inventory checks. These components could support e-commerce systems for pricing, stock alerts, and scheduled maintenance.

Challenges and Learning

MySQL 5.5's lack of CTEs and window functions forced creative solutions, such as derived tables and variables, deepening my understanding of legacy systems. Debugging the Function error taught me to verify schema constraints early. These challenges emphasized the need for robust testing and documentation.

Applications and Trends

These features align with trends like automation (Events, Triggers), modularity (Functions), and reporting (Queries). They could be applied to retail databases for real-time inventory updates, customer analytics, or automated pricing, enhancing efficiency and scalability.

Conclusion

Tasks 1-4 demonstrated MySQL 5.5’s capabilities and limitations, equipping me with skills to adapt advanced features to practical scenarios. Future improvements include exploring set-based alternatives for Cursors and upgrading to newer MySQL versions for enhanced functionality.

6. References

[1] MySQL, "MySQL 5.5 Reference Manual," Oracle Corporation, 2015. [Online]. Available: https://dev.mysql.com/doc/refman/5.5/en/

[2] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 6th ed. New York, NY, USA: McGraw-Hill, 2011.

[3] P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone, "Database Systems: Concepts, Languages and Architectures," ACM Computing Surveys, vol. 28, no. 4, pp. 105–107, Dec. 1996.

[3] T. M. Connolly and C. E. Begg, Database Systems: A Practical Approach to Design, Implementation, and Management, 5th ed. Boston, MA, USA: Addison-Wesley, 2010.

[4] J. A. Hoffer, R. Venkataraman, and H. Topi, Modern Database Management, 12th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2010.

[5] G. Harrison and S. Feuerstein, MySQL Stored Procedure Programming. Sebastopol, CA, USA: O'Reilly Media, 2006.

[6] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, 6th ed. Boston, MA, USA: Addison-Wesley, 2011.

[7] C. J. Date, An Introduction to Database Systems, 8th ed. Boston, MA, USA: Addison-Wesley, 2003.

[8] MySQL, "MySQL 5.5: Triggers," Oracle Corporation, 2015. [Online]. Available: https://dev.mysql.com/doc/refman/5.5/en/trigger-syntax.html

[9] P. DuBois, MySQL, 5th ed. Upper Saddle River, NJ, USA: Addison-Wesley Professional, 2013.

[10] A. K. Majumdar and P. Bhattacharya, Database Management Systems. New Delhi, India: Tata McGraw-Hill, 2009.

[11] B. Schwartz, P. Zaitsev, and V. Tkachenko, High Performance MySQL: Optimization, Backups, Replication, and More, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2012.

[12] S. K. Singh, "Database Management Systems: An Overview of Query Optimization Techniques," International Journal of Computer Applications, vol. 45, no. 10, pp. 12–18, May 2012.

[13] A. K. Jain and R. Gupta, "Automation in Database Management Systems: Trends and Challenges," Journal of Database Management, vol. 24, no. 3, pp. 45–60, Jul.–Sep. 2013.

[14] MySQLTutorial.org, "MySQL GROUP_CONCAT Function," 2014. [Online]. Available: https://www.mysqltutorial.org/mysql-group_concat/

[15] Stack Overflow, "How to Simulate Ranking in MySQL Without Window Functions," 2013. [Online]. Available: https://stackoverflow.com/questions/1895110/row-number-in-mysql