



PROGRAMME: BSC/HND COMPUTING PORTFOLIO

Assignment Title: Database Design &
Implementation

Module: Databases and Application Development

Module code: ACCA5026

Lecturer: Suresh Shanmugasundaram

Submission deadline: 1st May 2026 14:00 (2PM)

Student name: Andy Onyedikachukwu Ibeh

STUDENT ID: 2335550

Table of Contents

Introduction	4
Problem Statement	5
Objectives	6
System Analysis & Requirements	6
Business Requirements	6
Functional Requirements	6
Non-Functional Requirements	7
Database Design.....	8
Entity Identification.....	8
Attributes of Each Entity	8
Relationships	8
ER Diagram (ERD).....	10
Normalisation to Third Normal Form.....	10
Implementation Documentation	15
Table Creation and Constraints.....	15
supplier and customer tables	15
product and product_category tables.....	16
Orders and Order_item tables	17
Review table.....	17
Constraints	17
Sample Data Insertion	18
Testing the Database	19
Stored Procedure and Transactional Scenario	22
CRUD Stored Procedures	22
Transaction Handling and Scenario.....	23
PlaceOrder.....	23
Commit and Rollback Justification	24
Triggers and Functions	25
Triggers	25
reduce_stock.....	25
Functions	26

CalculateTotal	26
Auditing Mechanism.....	26
product_update_audit” trigger”	27
update statement	27
Performance Tuning Enhancement.....	28
Indexes of Product(name), Orders(customer_id), and Review(product_id).....	28
Query Optimisation	28
Product-Supplier Relationship	29
Customer Order History Query.....	29
Product Rating	29
Literature Review	30
Triggers and Database Automation	30
Indexing and Query Optimisation.....	30
Conclusion & Reflection	31
Conclusion	31
Reflection.....	31
Part B: Introduction	32
Website Design and Functionality	32
Website Overview.....	32
CRUD Operations	32
add option	32
Search Functionality.....	33
User Interface and Navigation.....	34
Login Page	35
Login and Registration.....	35
Book view and order functionality	36
Functionality of the Order Management.....	37
Add Book and Edit	38
Database Design and Structure.....	39
SQL Injection Overview	39
Security Measures and Testing	40
Conclusion.....	40

References	41
------------------	----

GreenLeaf Organics Database Design & Implementation Report

Introduction

GreenLeaf Organics is a UK-based sustainable company specialising in certified organic food, natural skincare, and biodegradable household goods. To accommodate a growing customer base and increasing demand for online convenience, the organisation is developing an e-commerce platform to streamline product management, supplier relationships, and customer engagement.

As an independent database consultant, this report outlines the design and implementation of a relational database prototype. The system manages supplier registration, product lifecycle, order processing, inventory control, and customer reviews.

MySQL was selected as the relational database management system due to its robust support for ACID transactions, stored procedures, triggers, and indexing, making it ideal for e-commerce applications.

[1]. Its widespread industry adoption ensures long-term support, a rich ecosystem of tools, and strong community resources.

Problem Statement

GreenLeaf Organics is confronted with an issue of controlling the growing business activities through fragmented and manual data management strategies. Tracking inventory also becomes a difficult process when the number of products available increases and the supplier is concerned, and there may be inconsistencies and stock management issues [2]. Supplier data and certification history are to be systematically organised to obtain optimum coordination. In addition, interactions with customers, monitoring orders, and managing feedback also necessitate the centralisation of the relational database system that will demonstrate reliability, efficiency, and improvement of functional performance.

Objectives

The purposes of this study are:

- To develop an Entity Relationship Diagram (ERD) (system entities and relationships).
- To deploy a relational database structure to GreenLeaf Organics.
- To design a stored procedure in CRUD.
- To introduce triggers to be used in automation and auditing.
- To optimise query and indexing of databases to improve the performance of the database.

System Analysis & Requirements

Business Requirements

GreenLeaf Organics has business needs that aim to facilitate effective management of suppliers, products, customers and orders in an online platform. This system facilitates supplier registration and certification management, which guarantees the authenticity of the products [3]. Product listing and category management enable the structured organisation of organic goods. Customer registration facilitates order tracking and purchase history management. Order processing ensures accurate transaction handling and stock control. A feedback and review system supports customer engagement and service improvement. Inventory management maintains adequate stock availability, while centralised data management enhances operational efficiency, scalability, and data uniformity throughout the organisation. Furthermore the system should be capable of supporting effective processing of orders, and this should be complemented with accurate transactions, prompt updates, and appropriate stock controls. The importance of inventory management is that it ensures the real-time stock level and prevents overstocking or stockouts [4].

Functional Requirements

The operational features of the GreenLeaf Organics database system are described by the functional requirements. The system facilitates supplier management since it allows the addition and maintenance of supplier information, that is, certification and contacts. Product management capability enables one to insert, update and delete product records, which have the right product information and availability. Category management allows categorising products based on the various categories in order to facilitate a systematic data retrieval process.

Non-Functional Requirements

The quality attributes of the GreenLeaf Organics database system are determined by the non-functional requirements. These requirements guarantee that the system is efficient, secure and reliable in its operation and supports future business growth.

Non-Functional Requirement	Description
Performance	The database system must provide fast query execution and efficient data retrieval to support product-browsing, order processing, and reporting operations.
Security	Access control mechanisms must be implemented to protect sensitive customer, supplier, and transaction data from unauthorized access.
Scalability	The system must support increasing numbers of products, suppliers, and customers without affecting database performance or reliability.
Data Integrity	Constraints, keys, and validation rules must ensure accurate and consistent data across all database tables.
Reliability	The system must maintain stable operations and ensure data availability during transactions and system usage.
Maintainability	The database structure must support easy updates, modifications, and future enhancements.
Availability	The database must remain accessible for users to perform operations such as product browsing and order processing.
Consistency	Transaction management using commit and rollback must maintain consistent database states.

Table 1: Functional Requirement

Database Design

Entity Identification

The key elements of the GreenLeaf Organics database system are determined by entity identification. Each entity represents a real-world object that is part of business processes and helps organise and manage data effectively.

Attributes of Each Entity

Entity	Attributes
Supplier	supplier_id (PK), name, email, phone, certification, address
Customer	customer_id (PK), name, email, phone, address, registration_date
Category	category_id (PK), category_name
Product	product_id (PK), supplier_id (FK), name, description, price, stock_quantity, expiry_date,
Product_Category	product_id (PK, FK), category_id (PK, FK)
Orders	order_id (PK), customer_id (FK), order_date, total_amount, status
Order_Item	order_item_id (PK), order_id (FK), product_id (FK), quantity, price
Review	review_id (PK), customer_id (FK), product_id (FK), rating, comment, review_date

Table 2: Entity and Attributes

Relationships

The relationships establish the communication of entities in GreenLeaf Organics database system. These relations ensure consistency of the data and also prove redundancy and speed of database operations [5]. The database design incorporates one-to-many and many-to-many relationships to accurately represent real-world business processes.

The Supplier and Product entities share a one-to-many (1:M) relationship, where a single supplier can provide multiple products, but each product is associated with only one supplier. This is implemented through a foreign key referencing the Supplier table

within the Product table, ensuring referential integrity and enabling products to be managed collectively under their respective suppliers.

The Customer and Orders entities also maintain a one-to-many (1:M) relationship. A single customer can place multiple orders; however, each order belongs to one customer. This is established through a customer_id foreign key in the Orders table, facilitating efficient order tracking, purchase history management, and transaction monitoring.

The Orders and Order_Item entities share a one-to-many (1:M) relationship, where a single order can contain multiple order items, but each item belongs to one order. The order_id foreign key within the Order_Item table enforces this relationship, allowing multiple products to be associated with a single order seamlessly.

The Product and Order_Item entities also maintain a one-to-many (1:M) relationship. Each order item references a single product, while a product can appear across multiple order items. The product_id foreign key in the Order_Item table supports this structure, enabling effective monitoring of product sales and detailed order information.

Finally, the Product and Category entities share a many-to-many (M:M) relationship, where a product can belong to multiple categories and a category can contain multiple products. This is implemented through a dedicated Product_Category junction table, effectively managing the complex associations between products and their classifications.

relationship, which has “product_id” and “category_id” as composite primary keys [6].

Customer and Review entities are related to one another with the 1: M (one-to-many) linkage, as they can have more than one review. On the same note, the Product and Review entities also have a one-to-many (1: M) relationship, whereby more than one review can be conducted on a specific product. These connections promote customer reviews and assessing products.

ER Diagram (ERD)

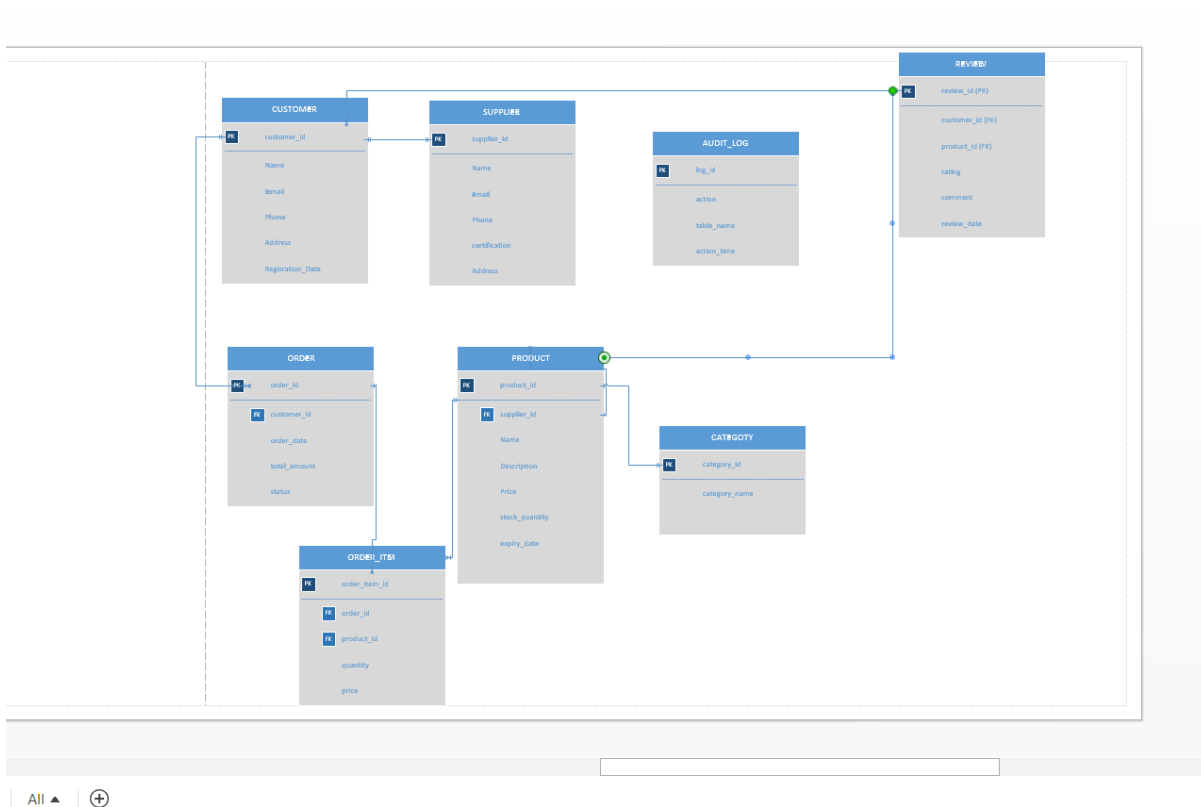


Figure 1: ER Diagram

Normalisation to Third Normal Form

Normalisation is the process of structuring a relational database to reduce redundancy and improve data integrity. The following table demonstrates that the GreenLeaf_Organic satisfies the requirements of the Third Normal Form.

Attribute	Constraint	Description
supplier_id	PK, AUTO_INCREMENT	Unique supplier identifier
name	NOT NULL	Supplier name
email	UNIQUE	Supplier email
phone	—	Contact number
certification	—	Organic certification details
address	—	Supplier address

Table 3: Supplier Entity

Attribute	Constraint	Description
customer_id	PK, AUTO_INCREMENT	Unique customer ID
name	NOT NULL	Customer name
email	UNIQUE	Customer email
phone	—	Customer phone
address	—	Customer address
registration_date	DEFAULT CURRENT_TIMESTAMP	Registration date

Table 4. Customer Entity

Attribute	Constraint	Description
-----------	------------	-------------

category_id	PK, AUTO_INCREMENT	Unique category ID
category_name	UNIQUE, NOT NULL	Category name

Table 5. Category Entity

Attribute	Constraint	Description
product_id	PK, AUTO_INCREMENT	Unique product ID
name	NOT NULL	Product name
description	—	Product description
price	CHECK(price > 0)	Product price
stock_quantity	CHECK(stock_quantity >= 0)	Stock quantity
expiry_date	—	Product expiry
supplier_id	FK	Supplier reference

Table 6. Product Entity

Attribute	Constraint	Description
product_id	PK, FK	Product ID
category_id	PK, FK	Category ID

Table 7. Product_Category Entity

Attribute	Constraint	Description
order_id	PK, AUTO_INCREMENT	Order ID
customer_id	FK	Customer reference
order_date	DEFAULT CURRENT_TIMESTAMP	Order date
total_amount	CHECK(total_amount >= 0)	Total order amount
status	CHECK(Pending, Completed, Cancelled)	Order status

Table 8. Orders Entity

Attribute	Constraint	Description
order_item_id	PK, AUTO_INCREMENT	Order item ID
order_id	FK	Order reference
product_id	FK	Product reference
quantity	CHECK(quantity > 0)	Quantity ordered
price	—	Product price

Table 9. Order_Item Entity

Attribute	Constraint	Description
review_id	PK, AUTO_INCREMENT	Review ID
customer_id	FK	Customer reference
product_id	FK	Product reference
rating	CHECK (1-5)	Product rating
comment	—	Review comment
review_date	DEFAULT CURRENT_TIMESTAMP	Review date

Table 10. Review Entity

Attribute	Constraint	Description
log_id	PK, AUTO_INCREMENT	Log ID
action	—	Action performed
table_name	—	Table name
action_time	DEFAULT CURRENT_TIMESTAMP	Action time

Table 11: 3NF Normalisation

Implementation Documentation

Table Creation and Constraints

The development of database tables outlines the structural core of the GreenLeaf_Organics system. Supplier, customer and category, product, orders, order, item product, category and review tables are all organising data into “logical entities”. “Primary keys” provide uniqueness of identification, as is the case with “foreign keys” which provide relationships between tables. “Data integrity” and “Assurance” are ensured by constraints like “NOT NULL”, “CHECK”, and “UNIQUE” constraints. This structure is facilitating product “normalisation”, elimination of “redundancy” and efficient data storage and retrieval, as well as managing to handle, retrieve and process customer “transactions” and feedback in the database system.

supplier and customer tables

This figure demonstrates how the GreenLeaf _Organic database structure is taking shape. Here, supplier and customer tables are being defined, so that they ensure order in data storage. It is aiding the development of the “database”, enhancing data management, aiding system implementation, and testing in carrying out the study.

```
✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0130 seconds.)

CREATE DATABASE GreenLeaf_Organic;

Edit inline ][ Edit ][ Create PHP code ]

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

USE GreenLeaf_Organic;

Edit inline ][ Edit ][ Create PHP code ]

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.1563 seconds.)

-- Creating Supplier Table CREATE TABLE Supplier ( supplier_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100) NOT NULL, email VARCHAR(100) UNIQUE, phone VARCHAR(20),
certification VARCHAR(100), address TEXT );

Edit inline ][ Edit ][ Create PHP code ]

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0318 seconds.)

-- Creating Customer Table CREATE TABLE Customer ( customer_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100) NOT NULL, email VARCHAR(100) UNIQUE, phone VARCHAR(20),
address TEXT, registration_date DATETIME DEFAULT CURRENT_TIMESTAMP );

Edit inline ][ Edit ][ Create PHP code ]
```

Figure 2: Creating the database, supplier and customer tables

product and product_category tables

The display of category tables, product tables, and product category tables is reflective of developing the database structure. The section is conceptualising product-category relationships, which uphold “many-to-many” mapping. It is enhancing data structuring, normalisation, and aiding effective product management and query processing in the implementation of the system.

```
✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0011 seconds.)

-- Creating Category Table CREATE TABLE Category ( category_id INT AUTO_INCREMENT PRIMARY KEY, category_name VARCHAR(100) NOT NULL UNIQUE );

Edit inline ][ Edit ][ Create PHP code ]

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0012 seconds.)

-- Creating Product Table CREATE TABLE Product ( product_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100) NOT NULL, description TEXT, price DECIMAL(10,2) NOT NULL CHECK
(price > 0), stock_quantity INT DEFAULT 0 CHECK (stock_quantity >= 0), expiry_date DATE, supplier_id INT NOT NULL, FOREIGN KEY (supplier_id) REFERENCES Supplier(supplier_id)
);

Edit inline ][ Edit ][ Create PHP code ]

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

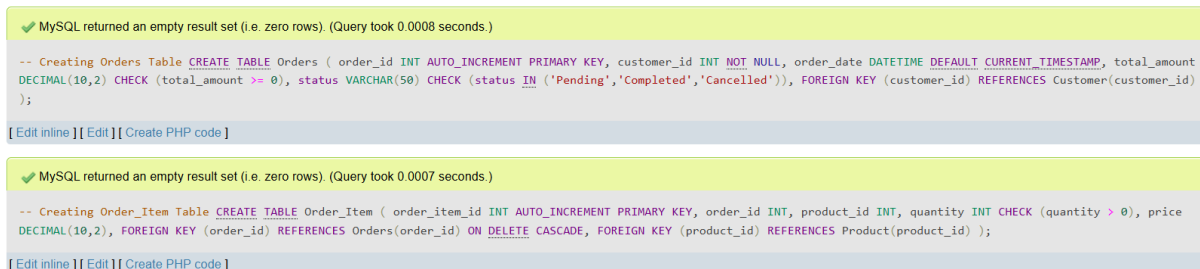
-- Creating Product_Category Table CREATE TABLE Product_Category ( product_id INT, category_id INT, PRIMARY KEY (product_id, category_id), FOREIGN KEY (product_id)
REFERENCES Product(product_id) ON DELETE CASCADE, FOREIGN KEY (category_id) REFERENCES Category(category_id) ON DELETE CASCADE );

Edit inline ][ Edit ][ Create PHP code ]
```

Figure 3: Creating category, product and product_category tables

Orders and Order_item tables

The display of category tables, product tables, and product category tables is reflective of developing the database structure. The section is conceptualising product-category relationships, which uphold “many-to-many” mapping. It is enhancing data structuring, “normalisation,” and aiding effective product management and query processing in the implementation of the system.



```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0008 seconds.)

-- Creating Orders Table CREATE TABLE Orders ( order_id INT AUTO_INCREMENT PRIMARY KEY, customer_id INT NOT NULL, order_date DATETIME DEFAULT CURRENT_TIMESTAMP, total_amount
DECIMAL(10,2) CHECK (total_amount >= 0), status VARCHAR(50) CHECK (status IN ('Pending','Completed','Cancelled')), FOREIGN KEY (customer_id) REFERENCES Customer(customer_id)
);

[ Edit inline ] [ Edit ] [ Create PHP code ]

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0007 seconds.)

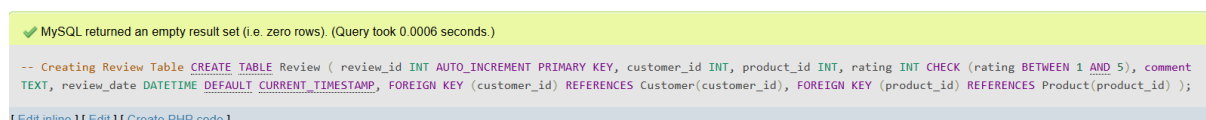
-- Creating Order_Item Table CREATE TABLE Order_Item ( order_item_id INT AUTO_INCREMENT PRIMARY KEY, order_id INT, product_id INT, quantity INT CHECK (quantity > 0), price
DECIMAL(10,2), FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE, FOREIGN KEY (product_id) REFERENCES Product(product_id) );

[ Edit inline ] [ Edit ] [ Create PHP code ]
```

Figure 4: Creating orders and order_item tables

Review table

The transaction management structure is being presented by the display of orders and order item tables. This part is connecting the customers, orders and purchase tracking of products. It assists in the processing of orders, safeguards the information, and assists in proper inventory and transaction management in the development of the database systems. Customer feedback management is being supported by defining the review table. Store ratings and comments are being stored in this section, which are related to customers and products. It enhances consumer interaction, “data integrity”, and assists in the evaluation of product quality and supplier performance when using it.



```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0006 seconds.)

-- Creating Review Table CREATE TABLE Review ( review_id INT AUTO_INCREMENT PRIMARY KEY, customer_id INT, product_id INT, rating INT CHECK (rating BETWEEN 1 AND 5), comment
TEXT, review_date DATETIME DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (customer_id) REFERENCES Customer(customer_id), FOREIGN KEY (product_id) REFERENCES Product(product_id) );

[ Edit inline ] [ Edit ] [ Create PHP code ]
```

Figure 5: Creating a review table

Constraints

Constraints also guarantee the accuracy and “integrity” of the data in all the “database” tables. In supplier, customer, product, orders and review tables, there are uniquely identifying records that are the “primary key” constraints. The foreign key constraints are enforcing relationships between the related tables, like the product and supplier, orders and customer and the order item with the product. UNIQUE types are stopping the same values as email and category name [7]. “CHECK” constraints are verifying price, quantity in stock, rating range and order status values. Timestamps on registration and order dates are being automatically allocated by default constraints, and this provides a stable and reliable database functionality.

Sample Data Insertion

The basic data creation involves the insertion of sample records into supplier tables, customer tables, category tables, product tables, order tables, review order table and order item table. These records depict real-life situations and maintain database testing. During database implementation and testing, sample data assists in verifying relationships, “validating constraints”, and proving system functionality. The addition of supplier and customer records is populating the “database” with mock sample data. This section is helping in the testing and validation of systems. It is enhancing data availability, making “relationship” checking easier, and assisting in the expression of database functionality, when it comes to both the implementation and evaluation phases.

```
✓ 10 rows inserted.
Inserted row id: 10 (Query took 0.0010 seconds.)

-- Inserting data into Supplier table INSERT INTO Supplier (name, email, phone, certification, address) VALUES ('Organic Farm Ltd', 'farm1@email.com', '1111111111', 'UK Organic Certified', 'London'), ('Eco Goods Ltd', 'eco1@email.com', '2222222222', 'Eco Certified', 'Manchester'), ('Green Harvest', 'green@email.com', '3333333333', 'Organic Certified', 'Birmingham'), ('Nature Fresh', 'nature@email.com', '4444444444', 'Eco Certified', 'Leeds'), ('Pure Organics', 'pure@email.com', '5555555555', 'Organic Certified', 'Glasgow'), ('Eco Life Ltd', 'eco2@email.com', '6666666666', 'Eco Certified', 'Liverpool'), ('Green Earth', 'earth@email.com', '7777777777', 'Organic Certified', 'Bristol'), ('Healthy Roots', 'roots@email.com', '8888888888', 'Organic Certified', 'Oxford'), ('Bio Suppliers', 'bio@email.com', '9999999999', 'Eco Certified', 'Cambridge'), ('Natural Source', 'source@email.com', '1010101010', 'Organic Certified', 'Sheffield');

[ Edit inline ] [ Edit ] [ Create PHP code ]
```

```
✓ 10 rows inserted.
Inserted row id: 10 (Query took 0.0008 seconds.)

-- Inserting data into Customer table INSERT INTO Customer (name, email, phone, address) VALUES ('John Doe', 'john@email.com', '9000000001', 'London'), ('Jane Smith', 'jane@email.com', '9000000002', 'Manchester'), ('Michael Brown', 'michael@email.com', '9000000003', 'Leeds'), ('Emily Davis', 'emily@email.com', '9000000004', 'Birmingham'), ('Chris Wilson', 'chris@email.com', '9000000005', 'Liverpool'), ('Sarah Taylor', 'sarah@email.com', '9000000006', 'Bristol'), ('David Anderson', 'david@email.com', '9000000007', 'Oxford'), ('Laura Thomas', 'laura@email.com', '9000000008', 'Cambridge'), ('James White', 'james@email.com', '9000000009', 'Sheffield'), ('Sophia Martin', 'sophia@email.com', '9000000010', 'Glasgow');

[ Edit inline ] [ Edit ] [ Create PHP code ]
```

Figure 6: Inserting data into supplier and customer tables

The addition of category and product records is filling the catalogue information. This area identifies products, suppliers, and categories to aid in the control of inventory. It is enhancing the determination of data, allowing the relationships to be tested, and assisting in the efficient retention of products when utilising the database system.

```
✓ 10 rows inserted.
Inserted row id: 10 (Query took 0.0010 seconds.)

-- Inserting data into Category table INSERT INTO Category (category_name) VALUES ('Organic Food'), ('Skincare'), ('Household'), ('Fruits'), ('Vegetables'), ('Dairy'), ('Personal Care'), ('Cleaning Products'), ('Health Supplements'), ('Eco Lifestyle');

[ Edit inline ] [ Edit ] [ Create PHP code ]
```

```
✓ 12 rows inserted.
Inserted row id: 12 (Query took 0.0010 seconds.)

-- Inserting data into Product table INSERT INTO Product (name, description, price, stock_quantity, expiry_date, supplier_id) VALUES ('Organic Apple', 'Fresh organic apples', 2.50, 100, '2026-05-01', 1), ('Organic Banana', 'Fresh bananas', 1.80, 120, '2026-05-02', 1), ('Natural Soap', 'Herbal soap', 5.00, 50, '2027-01-01', 2), ('Organic Milk', 'Dairy milk', 3.20, 80, '2026-04-15', 3), ('Eco Cleaner', 'Natural cleaning liquid', 6.50, 60, '2027-06-01', 4), ('Organic Carrot', 'Fresh carrots', 2.00, 150, '2026-04-20', 5), ('Face Cream', 'Natural skincare cream', 8.00, 40, '2027-02-01', 6), ('Shampoo', 'Herbal shampoo', 7.50, 45, '2027-03-01', 7), ('Organic Eggs', 'Farm eggs', 4.00, 90, '2026-04-25', 8), ('Dish Wash Liquid', 'Eco-friendly liquid', 3.50, 70, '2027-05-01', 9), ('Vitamin C', 'Health supplement', 10.00, 30, '2027-07-01', 10), ('Organic Spinach', 'Leafy vegetable', 1.50, 110, '2026-04-18', 5);

[ Edit inline ] [ Edit ] [ Create PHP code ]
```

Figure 7: Inserting data into category and product tables

Transaction management is being supported through establishing category-product relationships and order records. This section provides the linkage of products to more than one category and to the purchase order customers. This architecture is enhancing integrity in data, relationship testing and facilitating the verification of ordering and mapping capability in the database implementation.

```

✔ 24 rows inserted. (Query took 0.0010 seconds.)

-- Inserting data into Product_Category table INSERT INTO Product_Category VALUES (1,1),(1,4), (2,1),(2,4), (3,2),(3,7), (4,6),(4,1), (5,3),(5,8), (6,5),(6,1), (7,2),(7,7),
(8,2),(8,7), (9,6),(9,1), (10,3),(10,8), (11,9),(11,10), (12,5),(12,1);

[ Edit inline ] [ Edit ] [ Create PHP code ]

✔ 10 rows inserted.
Inserted row id: 10 (Query took 0.0010 seconds.)

-- Inserting data into Orders table INSERT INTO Orders (customer_id, total_amount, status) VALUES (1, 10.00, 'Completed'), (2, 15.00, 'Pending'), (3, 20.00, 'Completed'),
(4, 12.50, 'Cancelled'), (5, 8.00, 'Completed'), (6, 25.00, 'Pending'), (7, 30.00, 'Completed'), (8, 18.00, 'Completed'), (9, 22.00, 'Pending'), (10, 16.00, 'Completed');

[ Edit inline ] [ Edit ] [ Create PHP code ]

```

Figure 8: Inserting data into product_category and orders tables

Information, supporting transaction and feedback management, is being done by inserting order items and review records. In this section, orders are being linked with products and customer ratings, and comments are taken. Such a structure enhances the resolution and rigorousness of data, facilitating testing on associations and aiding testing of ordering and review properties.

```

✔ 15 rows inserted.
Inserted row id: 15 (Query took 0.0011 seconds.)

-- Inserting data into Order_Item table INSERT INTO Order_Item (order_id, product_id, quantity, price) VALUES (1,1,2,5.00), (1,3,1,5.00), (2,2,3,5.40), (3,4,2,6.40),
(4,5,1,6.50), (5,6,4,8.00), (6,7,2,16.00), (7,8,3,22.50), (8,9,2,8.00), (9,10,5,17.50), (10,11,1,10.00), (3,12,3,4.50), (2,6,2,4.00), (5,1,1,2.50), (7,4,2,6.40);

[ Edit inline ] [ Edit ] [ Create PHP code ]

✔ 10 rows inserted.
Inserted row id: 10 (Query took 0.0007 seconds.)

-- Inserting data into Review table INSERT INTO Review (customer_id, product_id, rating, comment) VALUES (1,1,5,'Excellent quality'), (2,2,4,'Very fresh'), (3,3,5,'Loved
it'), (4,4,3,'Good'), (5,5,4,'Works well'), (6,6,5,'Very fresh vegetables'), (7,7,4,'Nice product'), (8,8,5,'Highly recommended'), (9,9,3,'Average'), (10,10,4,'Good
value');

[ Edit inline ] [ Edit ] [ Create PHP code ]

```

Figure 9: Inserting data into order_item and review tables

Testing the Database

Database testing involves checking the relationships between tables, constraints, and data accuracy using sample queries. “Join queries” are those that access related data in two or more tables. Invalid data handling is being tested through constraint testing. Checking of “rollback” and “commit” is being done through transaction testing. This is a process that verifies the database reliability, integrity, and efficient system functionality.

It is the process of retrieving combined information by using the join “query” between the product and supplier tables. This section presents the names of products in line with suppliers. This query is enhancing data access, assisting in business reporting, and aiding in the “validation” of relational connections whenever testing and validating databases.

Showing rows 0 - 11 (12 total, Query took 0.0027 seconds.)

```
SELECT p.product_id, p.name AS product_name, s.name AS supplier_name FROM product p JOIN supplier s ON p.supplier_id = s.supplier_id;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

product_id	product_name	supplier_name
1	Organic Apple	Organic Farm Ltd
2	Organic Banana	Organic Farm Ltd
3	Natural Soap	Eco Goods Ltd
4	Organic Milk	Green Harvest
5	Eco Cleaner	Nature Fresh
6	Organic Carrot	Pure Organics
12	Organic Spinach	Pure Organics
7	Face Cream	Eco Life Ltd
8	Shampoo	Green Earth
9	Organic Eggs	Healthy Roots
10	Dish Wash Liquid	Bio Suppliers
11	Vitamin C	Natural Source

Show all | Number of rows: 25 | Filter rows: Search this table

Figure 10: Product with Supplier

Performing a join query between the product, product category and category tables retrieves information about products that have been categorised. Products together with their respective categories are being exhibited in this section. This query aids “many-to-many” relationship validation, optimising data search, and aiding verification of database relationships in testing.

Showing rows 0 - 23 (24 total, Query took 0.0027 seconds.)

```
SELECT p.name AS product, c.category_name FROM product p JOIN product_category pc ON p.product_id = pc.product_id JOIN category c ON pc.category_id = c.category_id;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table

Extra options

product	category_name
Eco Cleaner	Cleaning Products
Dish Wash Liquid	Cleaning Products
Organic Milk	Dairy
Organic Eggs	Dairy
Vitamin C	Eco Lifestyle
Organic Apple	Fruits
Organic Banana	Fruits
Vitamin C	Health Supplements
Eco Cleaner	Household
Dish Wash Liquid	Household
Organic Apple	Organic Food
Organic Banana	Organic Food
Organic Milk	Organic Food
Organic Carrot	Organic Food
Organic Eggs	Organic Food
Organic Spinach	Organic Food
Natural Soap	Personal Care
Face Cream	Personal Care
Shampoo	Personal Care

Console

Figure 11: Product with Categories

Running a “join” query among the customer, order, order item, and product tables is getting purchase details. This section shows customers and their ordered products and quantities. This query facilitates transaction examination, enhances data display, and facilitates verification of the functionality of the “relational database” when being tested

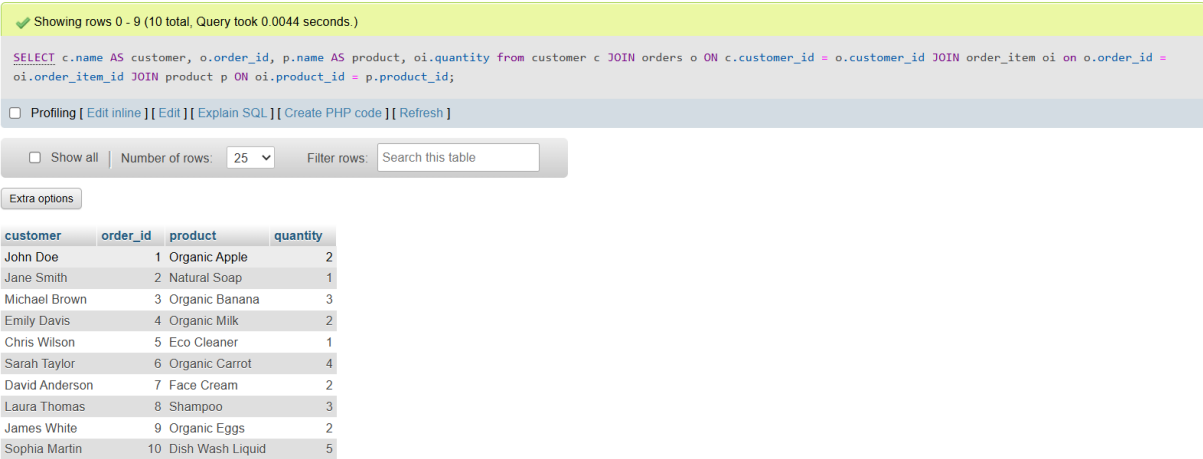


Figure 12: Customer Orders with Products

The average product rating calculation is the analysis of information on customer feedback. In this section, review data are being clumped together, and an average rating of the products is presented. The information is used to support performance assessment, enhance the decision-making process, and assist in evaluating the quality of the product in testing the database

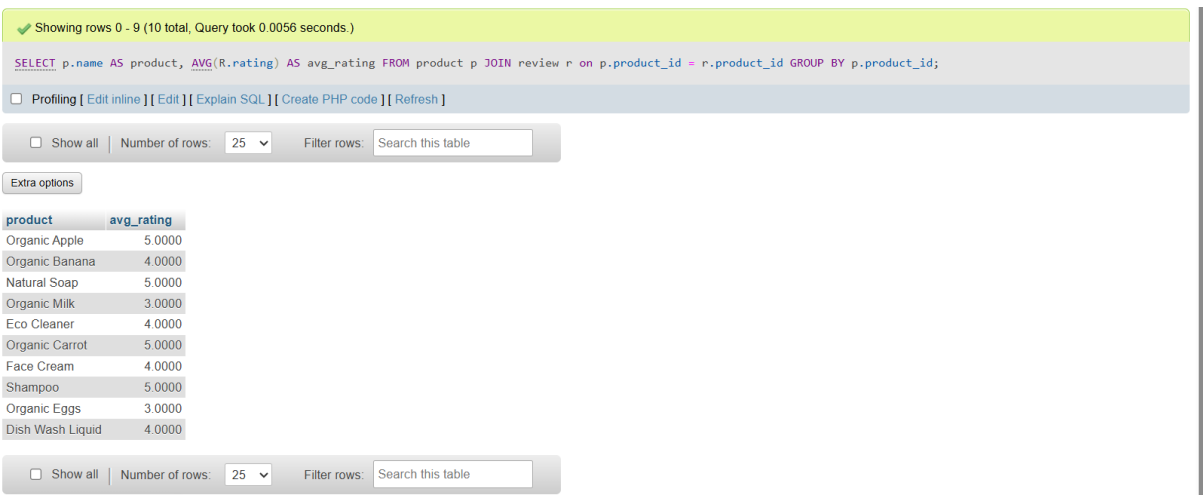


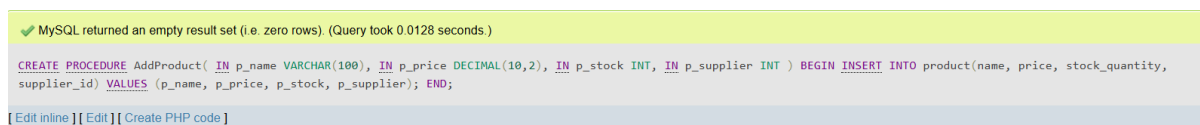
Figure 13: Average Product Rating

Stored Procedure and Transactional Scenario

CRUD Stored Procedures

CRUD stored procedures contribute to the successful control of product data within the GreenLeaf_Organic database system. The insert process ensures that it is easy to add new products and ensure that the products and inventory growth are listed correctly. The process of updating also allows changing the information about products, such as prices, although it maintains the accuracy of the data. The old information about products is deleted, and the databases are clean. All these processes streamline data integrity and efficiency and allow the regulation of database transactions in the system in a controlled fashion.

The stored procedure, consisting of AddProduct, inserts new entries of the product in the Product table and also includes inputs such as name, price, stock and supplier. The query sets parameters and has an insert statement in place. Ensuring successful procedure creation should be a confirmation that the production line 0 row(s) affected was successful. The process helps in the controlled mechanism of product insertion, data consistency, and the CRUD operations that are within the scope of transactional database management.



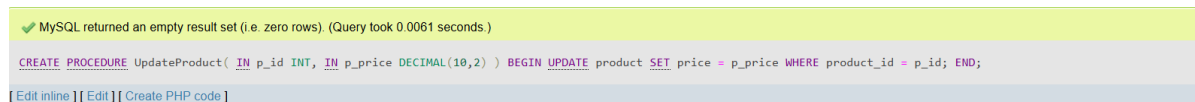
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0128 seconds.)

```
CREATE PROCEDURE AddProduct( IN p_name VARCHAR(100), IN p_price DECIMAL(10,2), IN p_stock INT, IN p_supplier INT ) BEGIN INSERT INTO product(name, price, stock_quantity, supplier_id) VALUES (p_name, p_price, p_stock, p_supplier); END;
```

[Edit inline] [Edit] [Create PHP code]

Figure 14: Creating a procedure for adding a product

Product updates are applied using the UpdateProduct stored procedure that updates the price of the products regarding the product ID and price parameters. The query finds the input variables and alters the product price in the Product table with an UPDATE statement. Ensuring successful procedure creation should confirm that the production line 0 row(s) affected were successful. This process enables a controlled update of data, improves data quality and strengthens CRUD in the transactional database management in GreenLeaf Organics.



MySQL returned an empty result set (i.e. zero rows). (Query took 0.0061 seconds.)

```
CREATE PROCEDURE UpdateProduct( IN p_id INT, IN p_price DECIMAL(10,2) ) BEGIN UPDATE product SET price = p_price WHERE product_id = p_id; END;
```

[Edit inline] [Edit] [Create PHP code]

Figure 15: Creating a procedure for updating a product

The DeleteProduct stored procedure is used to delete a product in the Product table according to the product ID parameter. The query sets one input parameter as “p_id” and executes a statement of such type as the Delete statement that has such a format as Delete From Product Where “product_id=p_id” to delete a record having a specified identification number of a product. Ensuring successful procedure creation should confirm that the production line 0 row(s) affected were successful. Such a process assists in ensuring controlled deletion, referential integrity and augmentation of CRUD transactional database functionality.

Show query box

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0054 seconds.)

CREATE PROCEDURE DeleteProduct(IN p_id INT) BEGIN DELETE FROM product WHERE product_id = p_id; END;

[Edit inline] [Edit] [Create PHP code]

Figure 16: Creating a procedure for deleting a product

Transaction Handling and Scenario

The “GreenLeaf_ Organics system relies on transaction processing as a means of providing stable and secure database processing. The “BEGIN” statement is used to start a transaction. Various operations can be performed within one transaction [8]. Successful changes are permanently committed with a statement of “COMMIT”, and undo operations are executed by a statement of “ROLLBACK” in the event that something goes wrong, such as a shortage of stock. This mechanism facilitates the “ACID” properties such as atomicity, consistency, isolation, and durability, proper order processing, data integrity, and incomplete or inconsistent database transactions.

PlaceOrder

The PlaceOrder stored procedure handles the customer orders based on transaction control. The query is used to retrieve stock and price, verify availability and insert order. When there is not enough stock, even the “ROLLBACK” cancel operation, otherwise order and order items can be added afterwards using “COMMIT”. The transaction provides integrity of the data, avoids inconsistent updates, and facilitates credible updates in order processing.

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0097 seconds.)

CREATE PROCEDURE PlaceOrder(IN p_customer_id INT, IN p_product_id INT, IN p_quantity INT) BEGIN DECLARE v_stock INT; -- Get current stock SELECT stock_quantity INTO v_stock FROM product WHERE product_id = p_product_id; -- Check if enough stock is available IF v_stock >= p_quantity THEN -- Reduce stock UPDATE product SET stock_quantity = stock_quantity - p_quantity WHERE product_id = p_product_id; -- Insert order (assuming orders table exists) INSERT INTO orders (customer_id, product_id, quantity) VALUES (p_customer_id, p_product_id, p_quantity); ELSE -- Not enough stock SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Not enough stock available'; END IF; END;

[Edit inline] [Edit] [Create PHP code]

Figure 17: Creating Transaction

The figure shows that the execution of the “placeorder(1, 1, 1000)” can be performed at the point where the requested quantity exceeds the stock that is available. It causes an error message to be sent out due to insufficient stock by the stored procedure, which causes a rollback. Failure code 1644 is used to check whether the transactions have failed. This prevents order insertion because of this rollback, the data remains consistent to avoid those incomplete transactions being logged, and the data is not lost.



Figure 18: Failed Place Order Execution with Rollback Due to Insufficient Stock

Commit and Rollback Justification

The essence of the transaction is that the systems of commit and rollback mechanisms are taken to ensure the system is in place to handle the transactions in the GreenLeaf_ Organics database. The mechanisms provide data uniformity and prevent incompleteness or invalidity in processing the data. There is a commit operation to be adopted in case a transaction has all operation steps carried out successfully. Order placement: the system calls in the inventory of the product and price information, checks its availability, and stores the order information in the Orders table. The corresponding records are then inserted into the table. OrderItem. Immediately such operations are properly put into place, the transaction is permanently encoded in the commit statement, representing appropriate processes of the orders and a high degree of database integrity.

Rollback is an error whereby an error is detected in the process of effecting a transaction. Looking at the above example, when a customer puts an order of an amount that surpasses the stock that is present in the stock, the system will identify a dearth of inventory. The stored procedure hides an error message, rollback begins, and this clause cancels all those operations that were executed in the transaction. It prevents the input of invalid order records and keeps the stock at the same level. Rollback also guarantees referential integrity since it can never be possible to add half-baked data to the associated tables.

Triggers and Functions

Triggers

Triggers are procedures on the automated database invoked following events like “INSERT”, “UPDATE”, or “DELETE”. They can impose business regulations and ensure consistency of data automatically. The reduce stock trigger present in the GreenLeaf Organics database is an “AFTER INSERT” trigger on the OrderItem table that would automatically update stock quantity in the Product table by subtracting the ordered quantity. In addition, the product_update_audit trigger performs “AFTER UPDATE” on the Product table to record the values of “OLD” and “NEW prices” and insert them into the “Audit_Log” table. These triggers resolve real-time update or depiction, accuracy and consistent data integrity in all other related tables.

reduce_stock

The (reduce_stock) trigger is depicted to be created, in which the “AFTER INSERT” event on the OrderItem table automatically updates stock on the product. The SQL statement reduces the ordered quantity in the inventory. A successful execution output assures proper automation, real-time management of stock and data integrity.

```
1 DELIMITER //
2
3 CREATE TRIGGER reduce_stock
4 AFTER INSERT ON Order_Item
5 FOR EACH ROW
6 BEGIN
7     UPDATE Product
8     SET stock_quantity = stock_quantity - NEW.quantity
9     WHERE product_id = NEW.product_id;
10 END //
11
12 DELIMITER ;
```

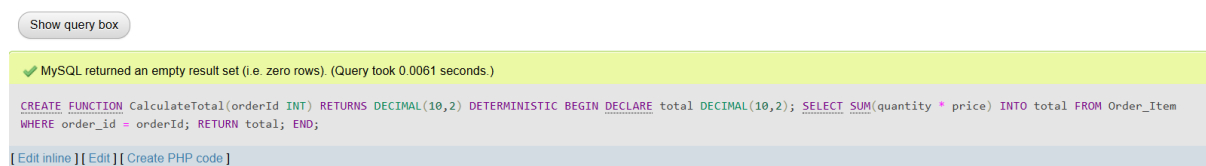
Figure 19: Creating Trigger reduce_stock

Functions

Functions are standardised database macros that create a one-value calculation and then can be used in multiple successive queries to enhance the efficiency and consistency of performing queries. The “calculateTotal” function in this system can be used to calculate the total value of an order based on the Order item table. It takes an “order_id” and computes the result based on the “SUM” (quantity-product of price) compilation. The function assigns a value in the form of a "DECIMAL", which anticipates the accuracy of financial addition [9]. The application of such logic into a system results in increased non-repetition, enhancement of query comprehension, as well as the preservation of regularity in the calculation of totals of orders. Functions aid in the effective handling of data and the general behaviour of the database system.

CalculateTotal

The “CalculateTotal” method is created to retrieve the sum of price and quantity in the Order_Item table is added to retrieve the complete sum of an order. It accepts an orderId as an input and returns a character number. The result of successful execution is the appropriate generation of functions that are correct and consistent calculation of order totals, optimisation of queries and minimised redundancy of the database system.



The screenshot shows a MySQL query editor interface. At the top, there is a button labeled "Show query box". Below it, a green status bar indicates "MySQL returned an empty result set (i.e. zero rows). (Query took 0.0061 seconds)". The main area displays the following SQL code:

```
CREATE FUNCTION CalculateTotal(orderId INT) RETURNS DECIMAL(10,2) DETERMINISTIC BEGIN DECLARE total DECIMAL(10,2); SELECT SUM(quantity * price) INTO total FROM Order_Item WHERE order_id = orderId; RETURN total; END;
```

 At the bottom, there are links for "[Edit inline]", "[Edit]", and "[Create PHP code]".

Figure 20: Creating a function for Calculate Total

Auditing Mechanism

The auditing system is created to monitor and record the database changes, which are traceable and accountable. Audit_Log table of this system holds records regarding the table name, description of actions, and date. The values of the OLD and NEW price values are captured in the automation of the product_update_audit trigger in response of the update operations carried out on the product table. When the price of a product is modified, a descriptive record is added to the Audit_Log table. This system is applied to ensure that automatic recording of all the critical changes is done. It enhances data integrity, helps to trace the mistakes, and gives a consistent history of database changes that can be tracked and explored.

Audit_Log table is generated and contains columns like the log_id, action description, the name of the table and the action time. The implementation output was effective in creating a table and making it easier to have changes to the database recorded in written form, which then leads to easy auditing, traceability, and data integrity of the system.

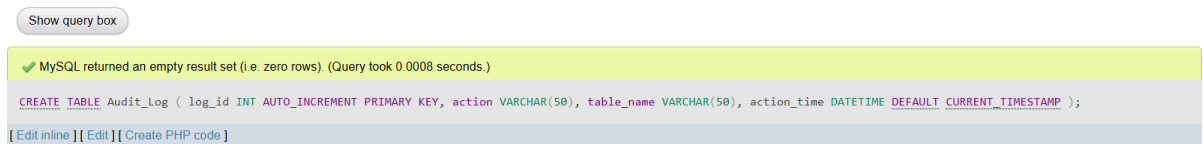


Figure 21: Creating Audit_log table

product_update_audit” trigger”

The “product_update_audit” trigger is constructed to be executed after any changes have taken place on the “Product” table. It sends the records to the “Audit_Log” table by getting the old price and the new price. The successful output validates the creation of the trigger, permits monitoring the changes to data automatically, and contributes to accountability.

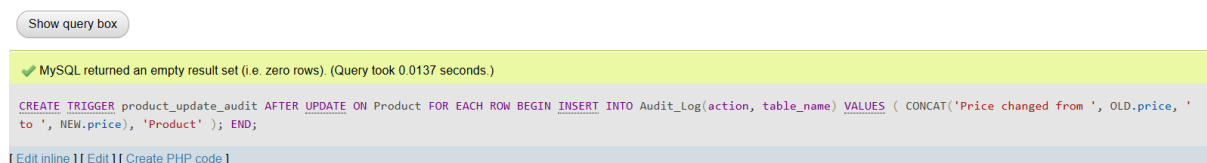


Figure 22: Creating Trigger product_update_audit

update statement

An update statement is issued on the “Product” table, followed by the retrieval of “Audit_Log” records. There are results of modifications on logs, past and revised prices, and time stamps. This demonstrates that changes are rightly recorded under the auditing mechanism that guarantees transparency, traceability, and proper monitoring.

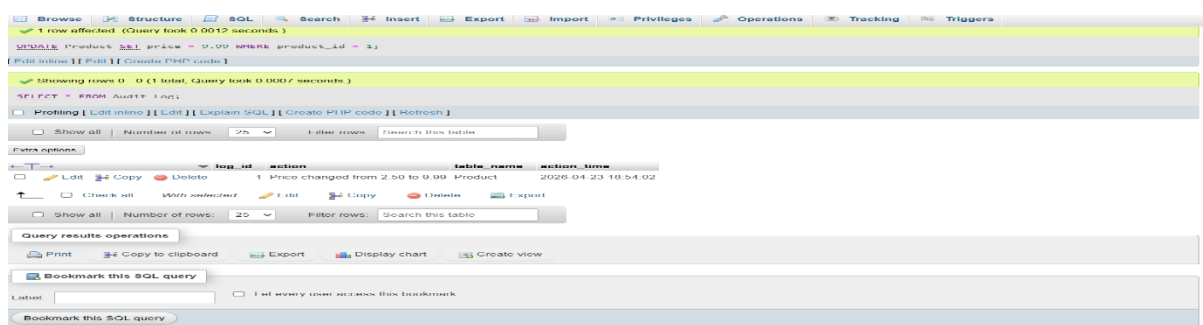


Figure 23: Showing Audit_log data

Performance Tuning Enhancement

Indexes of Product(name), Orders(customer_id), and Review(product_id)

Indexes of Product(name), Orders(customer_id), and Review(product_id) are created. The product of an effective implementation guarantees that the implementation process is carried out well. These indexes maximise the speed of retrieval, query execution and optimise the search, join and aggregation operations.

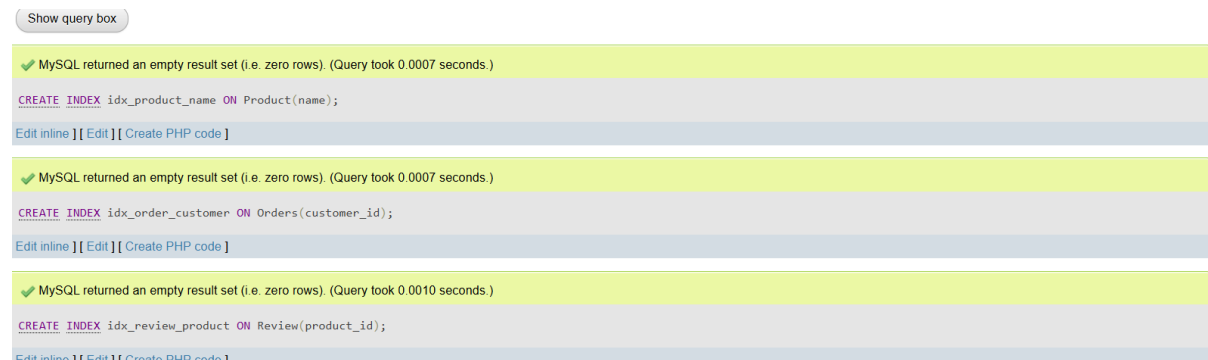


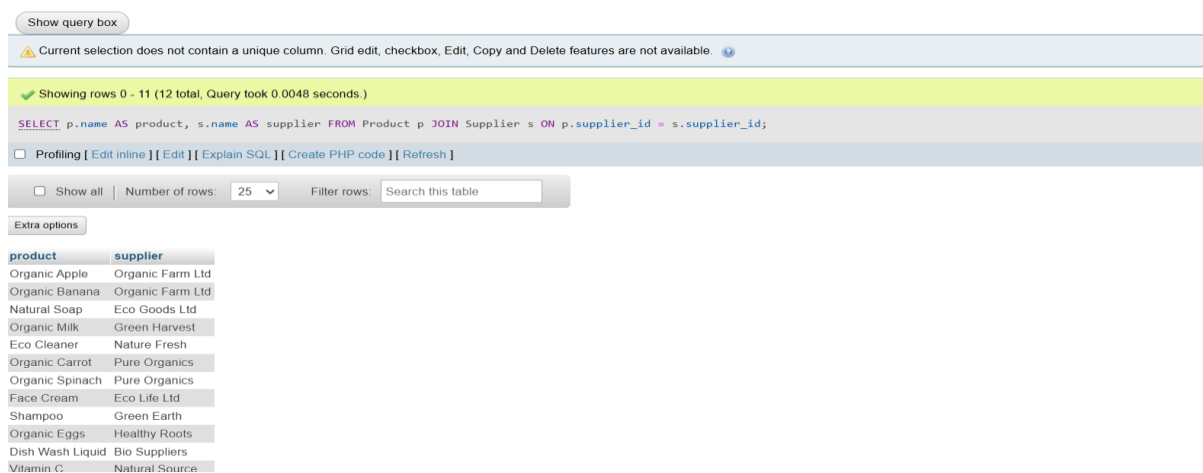
Figure 24: Creating indexes

Query Optimisation

Optimisation of the queries occurs under the combination of indexing and the efficient design of SQL queries. Indexed columns are used to optimise queries that are regularly used, such as product-supplier relationships, records of orders made by customers and reviews of product ratings. The indexing of four operations is beneficial, including Product and Supplier, Customer and Orders, since they decrease the search space and the speed of search. The optimal aggregation query that is to be indexed is also of type group by where the key is product_id. And in most cases, the table in which Review is the key. This reduces the burden on the computer memory and ensures faster execution. Optimised queries save on resources; improve system throughput and user experience by delivering results in a more efficient manner when conducting real-time database operations.

Product-Supplier Relationship

Product-Supplier Relationship Query provides a list of a product and a supplier. This query helps in confirming that the foreign key between the Supplier and Product tables is true and that data is successfully retrieved by using split operations without any problem.



Showing rows 0 - 11 (12 total, Query took 0.0048 seconds)

```
SELECT p.name AS product, s.name AS supplier FROM Product p JOIN Supplier s ON p.supplier_id = s.supplier_id;
```

Extra options

product	supplier
Organic Apple	Organic Farm Ltd
Organic Banana	Organic Farm Ltd
Natural Soap	Eco Goods Ltd
Organic Milk	Green Harvest
Eco Cleaner	Nature Fresh
Organic Carrot	Pure Organics
Organic Spinach	Pure Organics
Face Cream	Eco Life Ltd
Shampoo	Green Earth
Organic Eggs	Healthy Roots
Dish Wash Liquid	Bio Suppliers
Vitamin C	Natural Source

Figure 25: Product–Supplier Relationship Quer

Customer Order History Query

Customer Order History Query gets customer information and their orders. This query illustrates one to much relationship between the Customer and Orders tables and is applicable in tracking customer purchase history and analysing transaction data.



Showing rows 0 - 9 (10 total, Query took 0.0011 seconds)

```
SELECT c.name, o.order_id, o.total_amount FROM Customer c JOIN Orders o ON c.customer_id = o.customer_id;
```

Extra options

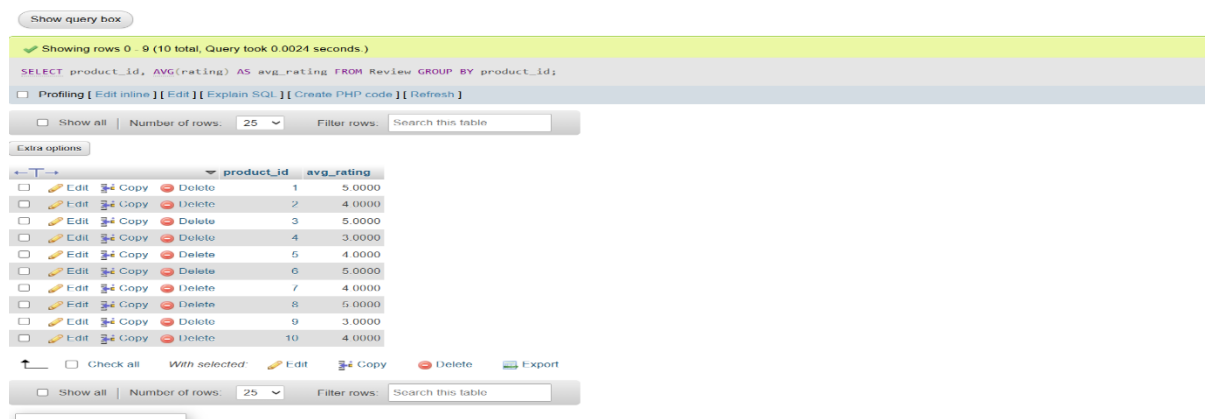
name	order_id	total_amount
John Doe	1	10.00
Jane Smith	2	15.00
Michael Brown	3	20.00
Emily Davis	4	12.50
Chris Wilson	5	8.00
Sarah Taylor	6	25.00
David Anderson	7	30.00
Laura Thomas	8	18.00
James White	9	22.00
Sophia Martin	10	16.00

Figure 26: Customer Order History Query

Product Rating

Product Rating Analysis Query determines the average rating of each product according to customer feedback. This query shows the application of aggregation operations and grouping

to process customer rate, assisting the organisation in identifying effective and ineffective items.



The screenshot displays a database query interface. At the top, a status bar indicates 'Showing rows 0 - 9 (10 total, Query took 0.0024 seconds.)'. Below this, the SQL query is shown: `SELECT product_id, AVG(rating) AS avg_rating FROM Review GROUP BY product_id;`. The interface includes buttons for 'Profiling', 'Edit inline', 'Edit', 'Explain SQL', 'Create PHP code', and 'Refresh'. A table view is shown with columns 'product_id' and 'avg_rating'. The table contains 10 rows of data. Each row has interactive icons for 'Edit', 'Copy', and 'Delete'. At the bottom, there are controls for 'Show all', 'Number of rows' (set to 25), and 'Filter rows'.

product_id	avg_rating
1	5.0000
2	4.0000
3	5.0000
4	3.0000
5	4.0000
6	5.0000
7	4.0000
8	5.0000
9	3.0000
10	4.0000

Figure 27: Product Rating Analysis Query

Literature Review

Database systems are essential for managing structured information in modern applications, enabling organisations to efficiently store, retrieve, and process large data volumes [10]. Relational databases organise data into structured tables, supporting transaction processing, query optimisation, and data integrity enforcement critical to business operations.

Literature highlights key database design mechanisms, including transactions, stored procedures, triggers, indexing, and constraints [11]. ACID properties guarantee reliable data processing, while triggers and stored procedures automate management tasks. Indexing enhances query performance, and constraints preserve data validity. The GreenLeaf Organics database applies these principles to achieve reliability, scalability, and effective data management.

Triggers and Database Automation

Database triggers are automated mechanisms that execute in response to specific events such as insert, update, or delete operations. They enforce business rules, automate critical processes, and maintain consistency throughout the database while reducing human error. Literature confirms that triggers play a vital role in preserving data integrity by automatically applying predefined rules whenever database changes occur.

Indexing and Query Optimisation

Indexing improves the speed and efficiency of database queries by enabling faster record retrieval without scanning entire tables. This is particularly beneficial for large datasets. By creating indexes on frequently queried columns, the system significantly reduces search time and enhances overall query execution performance.

Conclusion & Reflection

Conclusion

The GreenLeaf Organics database project demonstrates the successful implementation of a relational database system supporting sustainable business operations. The schema effectively models key entities — suppliers, products, customers, orders, and reviews — ensuring data consistency, integrity, and scalability. Entity-relationship modelling and normalisation minimise redundancy while improving storage and retrieval efficiency. Table structures, constraints, and sample data confirm accurate real-world business representation.

Advanced database features enhance system performance and reliability. Controlled CRUD operations and transaction management utilising COMMIT and ROLLBACK statements ensure safe and accurate order processing. Automated triggers and stored procedures streamline stock replenishment and calculations, reducing manual effort and improving accuracy. Indexing and query optimisation techniques further strengthen scalability and overall system performance.

Collectively, these implementations deliver a secure, efficient, and scalable database solution that supports core business operations, accelerates decision-making, and provides robust data management capabilities for GreenLeaf Organics

Reflection

This project provided valuable hands-on experience applying theoretical database concepts in a realistic business setting. Developing tables, defining relationships, and implementing constraints strengthened my understanding of data consistency and integrity. Working with stored procedures, triggers, and transactions deepened my knowledge of database automation and complex order processing operations.

Debugging SQL errors and managing dependencies challenged my problem-solving abilities and technical skills. I also recognised the importance of indexing and query optimisation for system performance. Moving forward, I aim to further develop my expertise in database security and access control. Overall, this project significantly enhanced my practical knowledge, analytical thinking, and confidence in designing reliable database systems.

Part B: Introduction

This study introduces the creation of a “database-based Book” Catalogue website in “PHP” and “MySQL”. The system is showing dynamic content, application of CRUD operations, searching, and authentication of the users. This site is enhancing the ease of use, good record management, and safe database communications.

Website Design and Functionality

Website Overview

The website is offering an active book catalogue that has books, authors, and categories. Users are searching records, examining details, and searching for information. The system is accessing information in a database, it is enhancing usability, increasing interaction, and assisting in the effective management of the dynamic content within the websites.

CRUD Operations

Administrators are being allowed to add, view, update, and delete records of books with the help of CRUD operations. The system dynamically controls the content that is contained in databases, increasing the accuracy of the data, and even improving the functionality of the system, while assisting administrators in keeping the information organised effectively.

add option

The add option is used to add invalidated records to the books, and the edit option is used to update the existing records. It is enhancing data precision, inhibiting SQL injection, and assisting administrators in controlling the content of databases effectively in a dynamic site.

```
ms-server > www - book catalogue - admin > W add-book.php
<?php
require_once '../config/constants.php';
require_once '../includes/admin_check.php';
require_once '../config/db.php';
$error = '';
$success = '';
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $title = trim($_POST['title'] ?? '');
    $author_id = (int)($_POST['author_id'] ?? 0);
    $category_id = (int)($_POST['category_id'] ?? 0);
    $year = (int)($_POST['published_year'] ?? 0);
    $price = trim($_POST['price'] ?? '');
    if (empty($title) || $author_id <= 0 || $category_id <= 0 || $year <= 0 || $price == '') {
        $error = "All fields are required.";
    } elseif (!is_numeric($price) || $price < 0) {
        $error = "Price must be a positive number.";
    } elseif ($year < 1000 || $year > 2099) {
        $error = "Please enter a valid year.";
    } else {
        $stmt = $conn->prepare("
            INSERT INTO books (title, author_id, category_id, published_year, price)
            VALUES (?, ?, ?, ?, ?)
        ");
        $stmt->bind_param("siiiid", $title, $author_id, $category_id, $year, $price);
        if ($stmt->execute()) {
            header("Location: " . BASE_URL . "public/index.php?msg=book_added");
            exit();
        } else {
            $error = "Failed to add book. Please try again.";
        }
        $stmt->close();
    }
}
$authors = $conn->query("SELECT author_id, name FROM authors ORDER BY name");
$categories = $conn->query("SELECT category_id, category_name FROM categories ORDER BY category_name");
require_once '../includes/header.php';
?>
```

Figure 1: Function for the add book

```
$error = '';
$stmt = $conn->prepare("SELECT * FROM books WHERE book_id = ?");
$stmt->bind_param("i", $id);
$stmt->execute();
$book = $stmt->get_result()->fetch_assoc();
$stmt->close();
if (!$book) {
    header('Location: " . BASE_URL . "public/index.php");
    exit();
}
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $title = trim($_POST['title'] ?? '');
    $author_id = (int)($_POST['author_id'] ?? 0);
    $category_id = (int)($_POST['category_id'] ?? 0);
    $year = (int)($_POST['published_year'] ?? 0);
    $price = trim($_POST['price'] ?? '');
    if (empty($title) || $author_id <= 0 || $category_id <= 0 || $year <= 0 || $price === '') {
        $error = "All fields are required.";
    } elseif (!is_numeric($price) || $price < 0) {
        $error = "Price must be a positive number.";
    } elseif ($year < 1000 || $year > 2099) {
        $error = "Please enter a valid year.";
    } else {
        $stmt = $conn->prepare("
            UPDATE books
            SET title = ?, author_id = ?, category_id = ?, published_year = ?, price = ?
            WHERE book_id = ?
        ");
        $stmt->bind_param("siiiid", $title, $author_id, $category_id, $year, $price, $id);
        if ($stmt->execute()) {
            header('Location: " . BASE_URL . "public/view.php?id={$id}&msg=updated");
            exit();
        } else {
            $error = "Update failed. Please try again.";
        }
    }
    $stmt->close();
}
```

Figure 2: Function for the edit book

Search Functionality

Search capability enables users to filter the books using keywords, author, category, and year. Dynamically retrieving matching records by the system enhances search accuracy, improves user experience, and assists users in finding appropriate information in a more efficient way [12].

```
if (session_status() === PHP_SESSION_NONE) {
    session_start();
}

$query = trim($_GET['query'] ?? '');
$author_id = isset($_GET['author_id']) ? (int)$_GET['author_id'] : 0;
$cat_id = isset($_GET['category_id']) ? (int)$_GET['category_id'] : 0;
$year_from = isset($_GET['year_from']) ? (int)$_GET['year_from'] : 0;
$year_to = isset($_GET['year_to']) ? (int)$_GET['year_to'] : 0;

$authors = $conn->query("SELECT author_id, name FROM authors ORDER BY name");
$categories = $conn->query("SELECT category_id, category_name FROM categories ORDER BY category_name");

$conditions = [];
$params = [];
$types = '';

$sql = "
    SELECT b.book_id, b.title, b.published_year, b.price,
           a.name AS author_name, c.category_name
    FROM books b
    JOIN authors a ON b.author_id = a.author_id
    JOIN categories c ON b.category_id = c.category_id
";

if (empty($query)) {
    $conditions[] = "(b.title LIKE ? OR a.name LIKE ? OR c.category_name LIKE ?)";
    $like = "%{$query}%";
    $params[] = $like;
    $params[] = $like;
    $params[] = $like;
    $types .= 'sss';
}

if ($author_id > 0) {
    $conditions[] = "b.author_id = ?";
    $params[] = $author_id;
    $types .= 'i';
}
```

Figure 3: Search Functionality

User Interface and Navigation

The layouts and ease of access to pages are being offered through the user interface and navigation. Navigation menus will help in navigating the users through functionalities, enhance usability, increase interaction, and allow users to use the functions of the website to do so effectively.

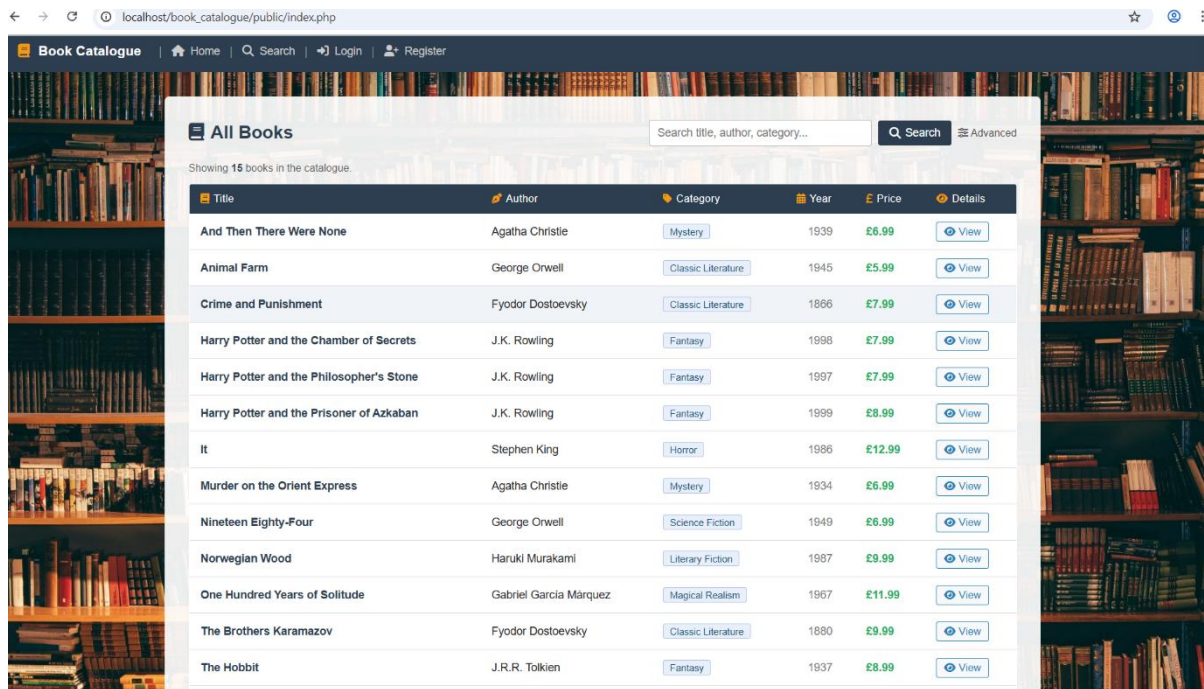


Figure 4: Index page

The interface is showing book records and offering search functionalities for filtering data. It is browsing books on the index page and refining the results on the basis of search options. This is enhancing navigability, user experience, and dynamic content retrieval.

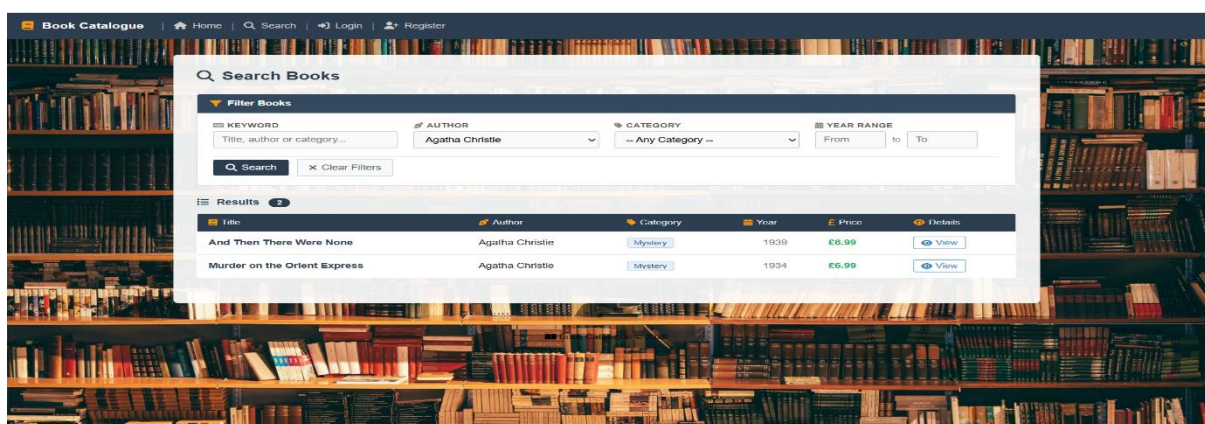


Figure 5: Search Page

Login Page

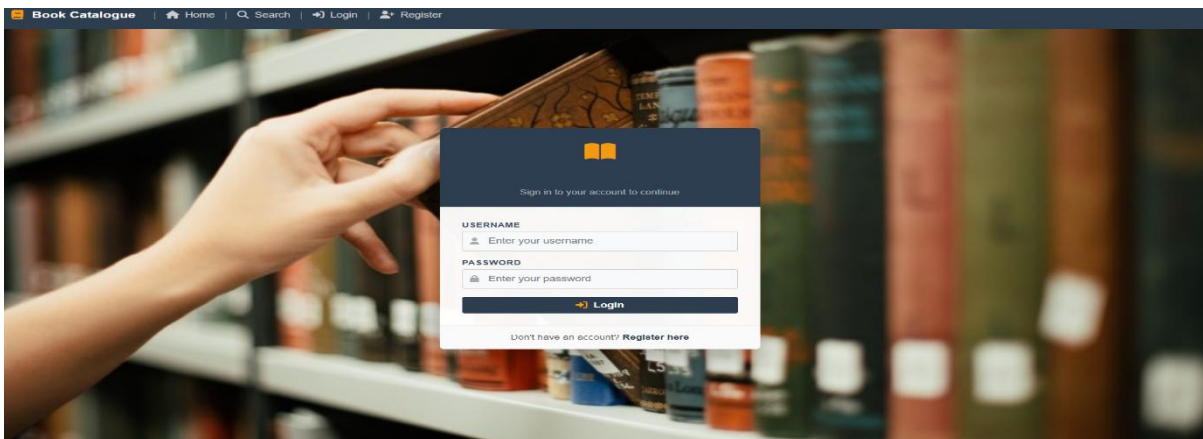


Figure 6: Login page

Login and Registration

Log-in and registration functionality facilitates secure authentication and account creation for users. It is enhancing the security and management of system entr

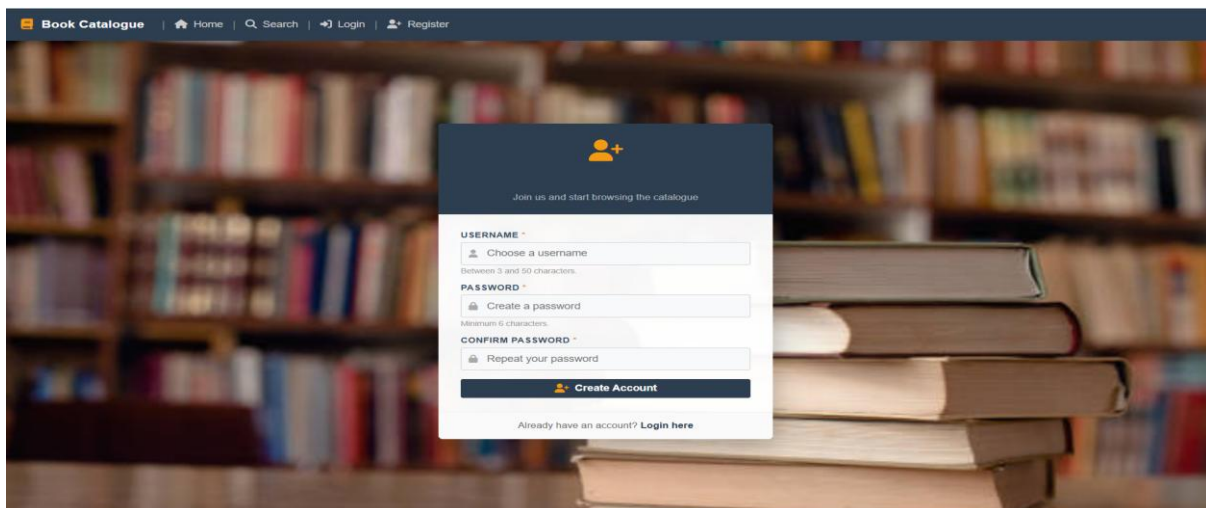


Figure 7: Register page

Book view and order functionality

Book view and order functionality are showing the detailed book information and enabling users to make an order. The view page is showing author, category, year, and price information.

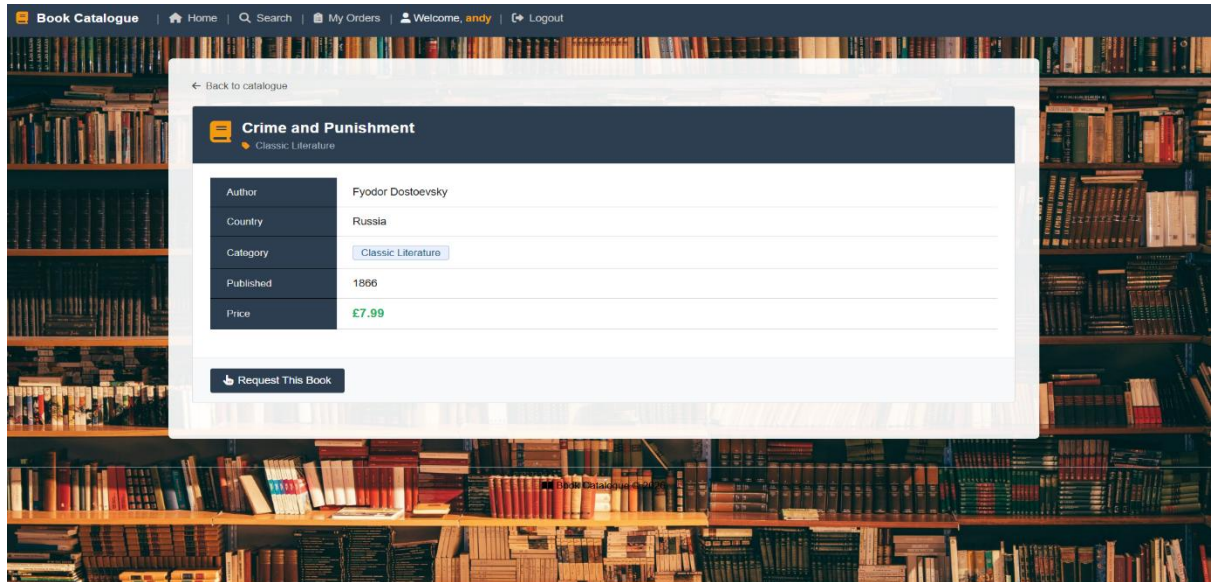


Figure 8: View page

A screenshot of the 'Place Order' form. The form is titled 'Place Order' and shows the book 'Crime and Punishment' by Fyodor Dostoevsky at £7.99 per copy. Below this is the 'Order Details' section, which includes a 'QUANTITY' field set to '1', a 'DELIVERY ADDRESS' section with fields for 'FULL NAME' (Recipient's full name), 'ADDRESS LINE' (Street address), 'CITY' (Town or city), and 'POSTCODE' (e.g. SW1A 1AA). At the bottom are 'Place Order' and 'Cancel' buttons. The footer of the page reads 'Book Catalogue © 2026'.

Figure 9: Order page

Functionality of the Order Management

The functionality of the order management and admin dashboard is showing user orders and system statistics. The ordered page displays order details, the delivery address, and the status. The administrator interface is displaying an overview of books, authors, categories, and cumulative orders.

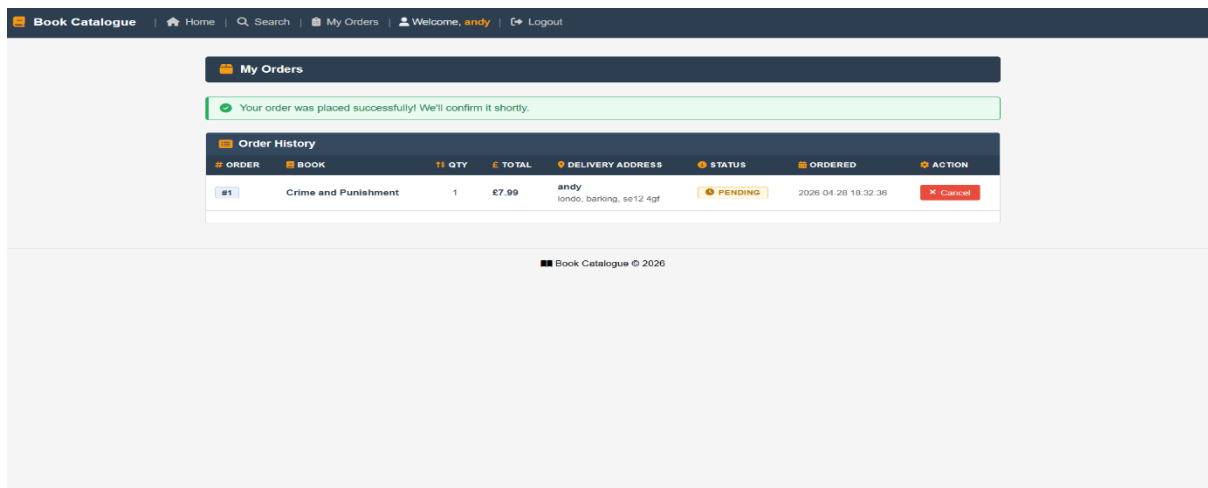


Figure 10: View ordered page

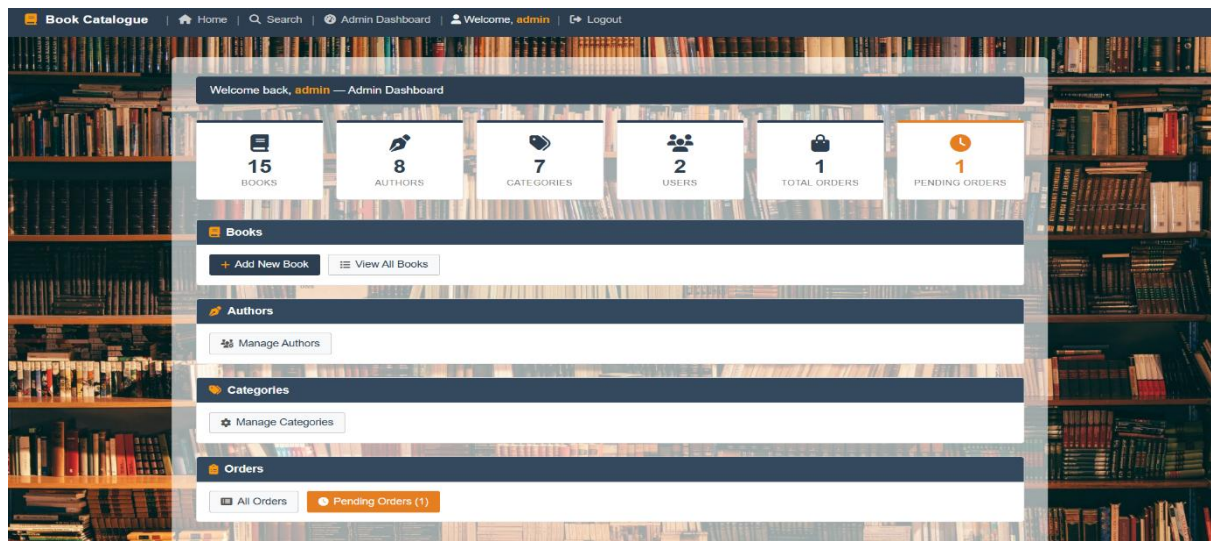


Figure 11: Admin dashboard page

Add Book and Edit

Book Add and edit are facilitating administration in book record-keeping. New book information is being added on the add book page, whereas existing records are being updated on

The screenshot shows the 'Add New Book' page in the Book Catalogue admin interface. The page has a dark blue header with navigation links: Home, Search, Admin Dashboard, Welcome, admin, and Logout. The main content area is light gray. A dark blue button labeled 'Add New Book' is at the top. Below it is a 'Book Details' form with the following fields: TITLE (text input with placeholder 'Enter book title'), AUTHOR (dropdown menu with 'Select Author' and a link 'Can't find the author? Add a new author'), CATEGORY (dropdown menu with 'Select Category' and a link 'Missing a category? Add a new category'), YEAR PUBLISHED (text input with '0'), and PRICE (£) (text input with '0.00'). At the bottom of the form are two buttons: '+ Add Book' and 'X Cancel'. The footer of the page says 'Book Catalogue © 2026'.

Figure 12: Add book page

The screenshot shows the 'Edit Book' page in the Book Catalogue admin interface. The page has a dark blue header with navigation links: Home, Search, Admin Dashboard, Welcome, admin, and Logout. The main content area is light gray. A dark blue button labeled 'Edit Book' is at the top, with a 'ID #5' label to its right. Below it is a 'Book Details' form with the following fields: TITLE (text input with 'Animal Farm'), AUTHOR (dropdown menu with 'George Orwell'), CATEGORY (dropdown menu with 'Classic Literature'), YEAR PUBLISHED (text input with '1945'), and PRICE (£) (text input with '5.99'). At the bottom of the form are two buttons: 'Save Changes' and 'X Cancel'. The footer of the page says 'Book Catalogue © 2026'.

Figure 13: Edit book page

Database Design and Structure

The database is storing various parallel tables such as user, book, category, and order. The connecting tables and the normalisation are supported by foreign keys [12]. The system is recalling dynamism records and enhancing information consistency, and aiding administrators in controlling information efficiently in the database-based site.

SQL Injection Overview

SQL injecting is enabling attackers to modify the database queries with malicious input. This is leading to intrusion of access control, leakage of data, and damage to databases [13]. Preventive is enhancing the security and safeguard the database-driven websites.

The secure login query is one that is based on prepared statements and binding of parameters to retrieve user information. User credentials are secured through hash password generation once they are registered.

```
$stmt = $conn->prepare("SELECT user_id, username, password, role FROM users WHERE username = ?");  
$stmt->bind_param("s", $username);  
$stmt->execute();
```

Figure 14: Secure Login Query Implementation

```
$hashed = password_hash($password, PASSWORD_DEFAULT);  
$stmt = $conn->prepare("INSERT INTO users (username, password, role) VALUES (?, ?, 'user')");  
$stmt->bind_param("ss", $username, $hashed);
```

Figure 15: Password Hashing Registration Function

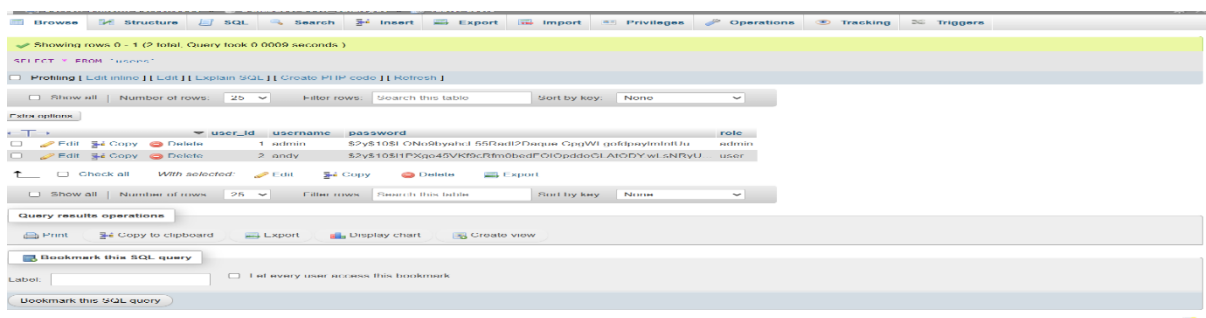


Figure 16: Password Hashing on MySQL

Security Measures and Testing

Security procedures include installing prepared statements, input validation, and hashing of passwords. This is a test of SQL injection vulnerability and verification of login. This is enhancing security, averting attacks, and securing the data of database-based websites.

Conclusion

The website is adopting a database-based book catalogue, which is powered by PHP and MySQL. The usability is being enhanced with CRUD operations, search facilities, and user authentication. Data and SQL injections are being blocked by security measures. The system is enabling the dynamic management of the content, enhancing efficiency and assisting the administrators and users of the site to deal with the site safely and efficiently.

References

- [1] A. a. I. I. Nishat, "Designing E-Commerce Data Models with Relational SQL Databases," *Baltic Journal of Multidisciplinary Research*, vol. 2, no. 2, pp. 112-120, 2025.
- [2] O. D. O. A. K. a. A. O. Madamidola, "A Review of existing inventory management systems," *International Journal of Research in Engineering and Science*, vol. 12, no. 9, pp. 40-50, 2024.
- [3] S. a. K. S. Mishra, "Automated Data Mapping and Schema Matching For Improving Data Quality in Master Data Management," *International Journal of Emerging Trends in Computer Science and Information Technology*, vol. 4, no. 3, pp. 80-90, 2023.
- [4] S. Duggineni, "Impact of controls on data integrity and information systems," *Science and technology*, vol. 13, no. 2, pp. 29-35, 2023.
- [5] R. Nie, "Database Design and Minimum Coverage Algorithm for Urban Environmental Design Information System," *Procedia Computer Science*, vol. 26, no. 2, pp. 457-464, 2025.
- [6] I. Y. M. N. F. a. G. A. Saidu, "Indexing techniques and structured queries for relational databases management systems," *Journal of the Nigerian Society of Physical Sciences*, pp. 2155-2155, 2024.
- [7] N. a. J. L. Preda, "Teaching SQL with Simplicity: A Shift from Relational Algebra to Canonical Databases," *In Proceedings of the 4th International Workshop on Data Systems Education: Bridging Education Practice with Education Research*, pp. 6-10, 2025.
- [8] A. Lingala, "Challenges and Handling Mechanisms of Database Transactions," *INTERNATIONAL SCIENTIFIC JOURNAL OF ENGINEERING AND MANAGEMENT and Indospace Publications*, vol. 2, no. 02, pp. 1-7, 2023.
- [9] M. B. M. V. P. S. J. a. M. P. Abbasi, "Revisiting database indexing for parallel and accelerated computing," *A comprehensive study and novel approaches. Information*, vol. 15, no. 8, pp. 395-429, 2024.
- [10] R. E. a. S. Navathe, "Fundamentals of Database Systems," *ACM Computing Surveys*, vol. 28, no. 1, p. 1-18, 1996.

- [11] P. B. a. E. Newcomer, "Principles of transaction processing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 4, p. 563–580, 1997.
- [12] A. Singhal, "Modern information retrieval: A brief overview," *IEEE Data Engineering Bulletin*, vol. 24, no. 4, p. 35–43, 2001.
- [13] R. R. a. J. Gehrke, "Database management systems," *ACM Computing Surveys*, vol. 31, no. 3, p. 110–145, 1999.
- [14] J. V. a. A. O. W. Halfond, "A classification of SQL injection attacks and countermeasures," *IEEE Transactions on Software Engineering*, vol. 12, no. 4, p. 13–23, 2006.
- [15] C. J. Date, "An Introduction to Database Systems, 8th ed. Boston, MA," *Addison-Wesley*, vol. 1, p. 105–130, 2004.
- [16] R. E. a. S. Navathe, "Fundamentals of Database Systems, 7th ed. Hoboken," *NJ Pearson*, vol. 1, p. 105–130, 2004.

