

Database Design and Application Development for Greenleaf Organics

Module: Databases and Application Development

Module code: ACCA5026

Assessment title: Database Design & Implementation / Database-Driven Website

Student name: Maria del Carmen Serrano Sanchez

Student ID: 2328383

Submission date: 1/05/26

Table of Contents

Table of Contents	Error! Bookmark not defined.
1. Introduction	4
2. Scenario and Business Requirements	4
2.1 Business Process Overview.....	4
2.2 Functional Requirements	5
2.3 Non-Functional Requirements.....	6
3. Database Design.....	6
3.1 Choice of Database System.....	6
3.2 Main Entities and Attributes.....	7
3.3 Relationship Design	9
4. Entity Relationship Diagram	11
5. Normalisation to Third Normal Form	12
5.1 First Normal Form	12
5.2 Second Normal Form	13
5.3 Third Normal Form	13
6. Database Implementation	14
6.1 Table Creation.....	14
6.2 Constraints and Referential Integrity	17
7. Sample Data Population and Testing	19
8. Stored Procedures and Transaction Handling.....	25
8.1 Stored Procedure Purpose.....	25
8.2 Transaction Scenario: Placing an Order	25
8.3 COMMIT and ROLLBACK Justification.....	26
9. Triggers and Auditing	26
10. Performance Tuning and Indexing	29
11. Website/Application Development.....	31
11.2 Database Connection.....	34
11.3 Dynamic Content Display	35
11.4 CRUD Functionality.....	39
12. SQL Injection Overview and Security Measures	41

12.1 Security Observations from the Uploaded Code	42
13. Evaluation and Reflection.....	44
13.1 Strengths of the Solution	44
13.2 Limitations and Improvements	44
13.3 Personal Reflection	45
14. Conclusion	45
15. References.....	46

1. Introduction

GreenLeaf Organics is a sustainable UK-based company specialising in the sale of organic food, eco-friendly household products, and natural lifestyle items. As the business expands, it requires a reliable digital solution to manage suppliers, products, customers, orders, and reviews efficiently.

This project develops a scalable relational database and a PHP-based website prototype to support the company's operations. The database design includes an Entity Relationship Diagram (ERD), relational tables, constraints, sample data, and normalisation to Third Normal Form (3NF) to improve data integrity and reduce redundancy [1].

MySQL was selected because it supports relational structures, keys, transactions, triggers, and indexing, while PHP enables dynamic database interaction and CRUD functionality.

2. Scenario and Business Requirements

2.1 Business Process Overview

GreenLeaf Organics is an environmentally responsible company that works with certified suppliers to sell organic and eco-friendly products through an online platform. The system is designed to support supplier management, product listing, customer registration, order processing, and feedback collection.

Suppliers are required to provide contact and certification details to ensure that products meet organic and sustainability standards. Since one supplier can provide multiple products, products must be linked to suppliers within the database.

Product records should include information such as name, description, price, stock quantity, expiry date, image location, and certification details. Products are also organised into categories such as organic vegetables, natural skincare, and eco-friendly household products. As products can belong to multiple categories, a junction table is required to resolve the many-to-many relationship between products and categories [2].

Customers can register, place orders, and leave reviews. Orders record purchase transactions and the individual products included in each purchase. The system also supports supplier reviews to improve transparency, trust, and quality control.

2.2 Functional Requirements

The system must support the management of suppliers, customers, products, categories, orders, and reviews. It should implement product-to-category classification through a junction table and customer-to-order relationships using foreign keys. The OrderDetail entity must also link orders and products to record purchase transactions accurately.

The PHP-based website should provide CRUD functionality for the main entities, allowing users to add, view, update, and delete records. The system must also display database content dynamically using PHP and MySQL queries. In addition, simple search and filtering functionality should be available to help users retrieve relevant information efficiently [3].

The database should support transactions to ensure that operations such as order creation and stock updates succeed or fail as a single unit. Auditing functionality should also be implemented using triggers or audit mechanisms. Furthermore, indexes should be applied to improve the performance of common queries, including product searches, customer order history, and low-stock reporting.

2.3 Non-Functional Requirements

The system should ensure data integrity through the use of primary keys, foreign keys, unique constraints, and validation rules. Security should be supported through input validation, prepared statements, escaping techniques, and controlled administrative access. Maintainability is improved through the use of normalised tables and a well-structured codebase, while relational design and indexing support scalability and performance [4].

Reliability is enhanced through transaction management, ensuring that operations such as order creation and stock updates are completed successfully as a single unit. If an error occurs during a transaction, the changes can be rolled back to maintain database consistency.

3. Database Design

3.1 Choice of Database System

MySQL was selected because it supports relational tables, primary keys, foreign keys, constraints, transactions, stored procedures, triggers, and indexes. These features help maintain data integrity and support reliable order processing within the system [5].

In addition, MySQL integrates effectively with PHP through MySQLi or PDO, making it a suitable choice for the development of a database-driven website prototype.

3.2 Main Entities and Attributes

Entity	Primary Key	Main Attributes	Purpose
Supplier	SupplierID	SupplierName, Email, Phone, Address, CertificationDetails, RegistrationDate	Stores certified supplier and contact/certification data.
Product	ProductID	Name, Description, Price, StockQuantity, ExpiryDate, CertificationInfo, SupplierID, DateAdded, Status, ImageURL	Stores products and links each product to one supplier.
Category	CategoryID	CategoryName, CategoryDescription	Classifies products into logical product groups.
ProductCategory	ProductID + CategoryID	ProductID, CategoryID	Associative entity resolving the many-to-many relationship between Product and Category.
Customer	CustomerID	FirstName, LastName, Email, PasswordHash, Phone, Address, RegistrationDate, Status	Stores customer accounts and contact information.
Orders	OrderID	CustomerID, OrderDate, TotalAmount, OrderStatus, PaymentMethod, DeliveryAddress, PaymentStatus	Stores high-level purchase transactions.
OrderDetail	OrderDetailID	OrderID, ProductID, Quantity, Price	Stores individual order items and resolves the many-to-many relationship between Orders and Product.
Review	ReviewID	CustomerID, ProductID, Rating, Title, Comment, ReviewDate	Stores customer feedback on products.
SupplierReview	SupplierReviewID	CustomerID, SupplierID, Rating, Comment, ReviewDate	Stores customer feedback on suppliers.

Table 1 summarises the primary entities, attributes, primary keys, and business purposes included in the GreenLeaf Organics database design.

The GreenLeaf Organics database includes the key entities required to support the business processes. The Supplier entity stores certified supplier contact and certification details, while the Product entity stores product information such as price, stock quantity, expiry date, certification details, and supplier references.

The Category entity is used to organise products into logical groups, while the associative entity ProductCategory resolves the many-to-many relationship between products and categories [6].

The Customer entity stores registered customer details, while the Orders entity records purchase transactions and the OrderDetail entity stores the individual products included in each order.

Customer feedback is managed through the Review and SupplierReview entities, supporting transparency, quality control, and customer trust within the platform.

3.3 Relationship Design

Relationship	Cardinality	Justification
Supplier -> Product	1:M	Each supplier can supply many products, but each product is linked to one supplier through the SupplierID foreign key.
Product -> Category	M:N	A product may belong to multiple categories, and a category may contain multiple products. The ProductCategory associative entity resolves this many-to-many relationship.
Customer -> Orders	1:M	A customer can place many orders, while each order belongs to one customer.
Orders -> Product	M:N	An order can include multiple products, and each product can appear in multiple orders. The OrderDetail associative entity resolves this many-to-many relationship.
Customer -> Review	1:M	A customer can write multiple product reviews.
Product -> Review	1:M	A product can receive multiple reviews.
Customer -> SupplierReview	1:M	A customer can review multiple suppliers.
Supplier -> SupplierReview	1:M	A supplier can receive multiple reviews.

Table 2 summarises the cardinalities and justifications for the relationships implemented in the GreenLeaf Organics database design.

The relationship design shows how the entities within the GreenLeaf Organics database are connected. The relationship between Supplier and Product is one-to-many, as one supplier can provide multiple products, while each product is linked to only one supplier.

The relationships between Product and Category, and between Orders and Product, are many-to-many and are resolved through the associative entities ProductCategory and OrderDetail, respectively [7].

The relationship between Customer and Orders is one-to-many because a customer can place multiple orders. Customers can also submit multiple reviews for products and suppliers.

These relationships are implemented using foreign keys to maintain referential integrity, reduce redundancy, and support efficient business operations.

4. Entity Relationship Diagram

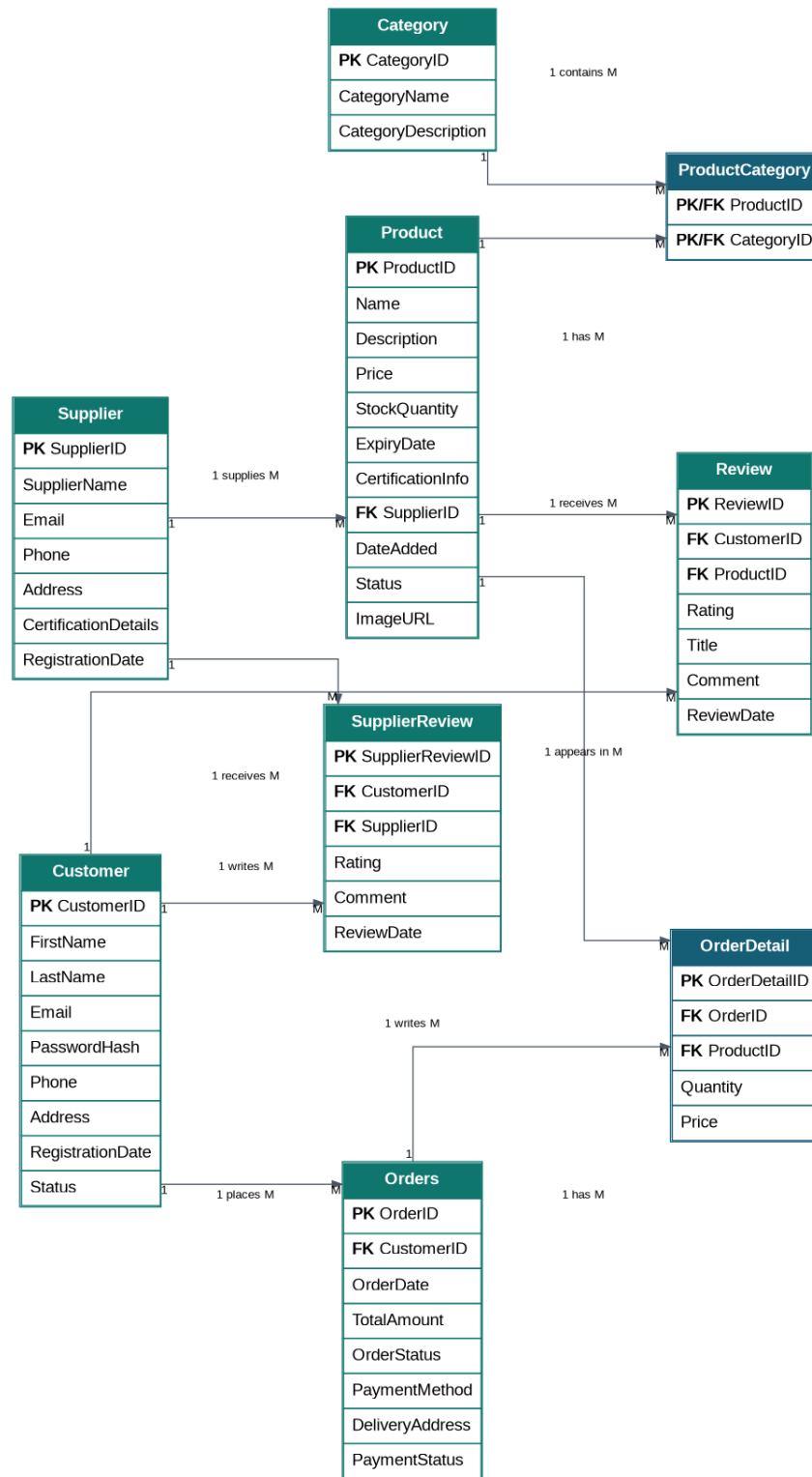


Figure 1: Entity Relationship Diagram

The Entity Relationship Diagram illustrates how the business rules of GreenLeaf Organics are represented within the database structure. Products have a one-to-many relationship with suppliers, while customers also have a one-to-many relationship with orders.

The ProductCategory entity resolves the many-to-many relationship between products and categories, allowing products to belong to multiple categories without data duplication [8]. Similarly, the OrderDetail entity resolves the many-to-many relationship between orders and products by storing each purchased item, quantity, and price.

Foreign keys are used throughout the database to maintain referential integrity, prevent orphan records, and ensure data consistency.

5. Normalisation to Third Normal Form

5.1 First Normal Form

The GreenLeaf Organics database satisfies First Normal Form (1NF) because all attributes contain atomic values and there are no repeating groups. Each field stores only a single value rather than lists or multiple values within the same attribute.

Examples of atomic attributes include product name, price, stock quantity, expiry date, review rating, and customer email. In addition, each table contains a primary key, such as ProductID, CustomerID, SupplierID, or OrderID, which uniquely identifies each record [9].

This structure improves data consistency, prevents duplication, and supports efficient relational database processing.

5.2 Second Normal Form

The GreenLeaf Organics database satisfies Second Normal Form (2NF) because all non-key attributes depend on the entire primary key, with no partial dependencies.

This is particularly important in associative entities such as ProductCategory, which uses a composite primary key consisting of ProductID and CategoryID. The table only stores the relationship between products and categories and does not contain additional attributes dependent on part of the key [10].

The OrderDetail entity also satisfies 2NF because attributes such as Quantity and Price depend on the specific order-product combination rather than on the order alone. This design separates overall order information from individual product purchase details, improving consistency and reducing redundancy.

5.3 Third Normal Form

The GreenLeaf Organics database satisfies Third Normal Form (3NF) because it contains no transitive dependencies. This means that non-key attributes depend only on the primary key and not on other non-key attributes.

For example, supplier details such as supplier name, email, phone number, address, and certification information are stored only in the Supplier table. The Product table stores only SupplierID as a foreign key, preventing unnecessary duplication [10].

Similarly, customer information is not duplicated in the Orders table. Only CustomerID is stored in each order record, while customer details remain in the Customer table.

Product information is also not unnecessarily repeated in the OrderDetail table. Only the purchase price is stored to preserve the historical price of a product at the time of purchase.

6. Database Implementation

6.1 Table Creation

SQL dump - category, customer and orderdetail table creation

```
1  --
2  -- Table structure for table `category`
3  --
4
5  CREATE TABLE `category` (
6    `CategoryID` int NOT NULL,
7    `CategoryName` varchar(100) NOT NULL,
8    `CategoryDescription` text
9  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
10
11 --
12 -- Dumping data for table `category`
13 --
14
15 INSERT INTO `category` (`CategoryID`, `CategoryName`, `CategoryDescription`) VALUES
16 (1, 'Organic Vegetables', 'Fresh certified organic vegetables sourced from trusted suppliers'),
17 (2, 'Organic Fruits', 'Naturally grown organic fruits free from synthetic chemicals'),
18 (3, 'Organic Dairy Products', 'Organic milk, cheese, and dairy products from certified farms'),
19 (4, 'Organic Grains and Cereals', 'Whole grains and cereals produced using organic farming
20 methods'),
21 (5, 'Organic Snacks', 'Healthy organic snacks made from natural ingredients'),
22 (6, 'Organic Beverages', 'Organic drinks including juices, teas, and plant-based beverages'),
23 (7, 'Natural Skincare', 'Chemical-free skincare products made from natural ingredients'),
24 (8, 'Eco-Friendly Personal Care', 'Sustainable personal care products such as shampoos and
25 soaps'),
26 (9, 'Eco Cleaning Products', 'Environmentally friendly household cleaning products'),
27 (10, 'Vegan and Plant-Based Foods', 'Plant-based food products suitable for vegan diets');
28
29 -----
30
31 --
32 -- Table structure for table `customer`
33 --
34
35 CREATE TABLE `customer` (
36   `customerID` int NOT NULL,
37   `FirstName` varchar(100) NOT NULL,
38   `LastName` varchar(100) NOT NULL,
39   `Email` varchar(255) NOT NULL,
40   `PasswordHash` varchar(255) NOT NULL,
41   `Phone` varchar(20) DEFAULT NULL,
42   `Address` varchar(255) DEFAULT NULL,
43   `RegistrationDate` date NOT NULL,
44   `Status` varchar(50) NOT NULL
45 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
46
```

```

46
47 --
48 -- Dumping data for table `customer`
49 --
50
51 INSERT INTO `customer` (`customerID`, `FirstName`, `LastName`, `Email`, `PasswordHash`,
52 `Phone`, `Address`, `RegistrationDate`, `Status`) VALUES
53 (1, 'James', 'Smith', 'james.smith@gmail.com', 'hashed_password_1', '07123456789', '12 Green
54 Street, London', '2025-01-10', 'Active'),
55 (2, 'Emily', 'Johnson', 'emily.johnson@email.com', 'hashed_password_2', '07234567890', '45 Oak
56 Road, Manchester', '2025-02-15', 'Active'),
57 (3, 'Oliver', 'Brown', 'oliver.brown@hotmail.com', 'hashed_password_3', '07345678901', '78
58 Maple Avenue, Birmingham', '2025-03-05', 'Active'),
59 (4, 'Maria', 'Sanchez', 'maria.sanchez@gmail.com', 'hashed_password_4', '07456789012', '23 Pine
60 Lane, Leeds', '2025-01-22', 'Active'),
61 (5, 'George', 'Wilson', 'george.wilson@email.com', 'hashed_password_5', '07567890123', '56
62 Birch Street, Liverpool', '2025-02-28', 'Active'),
63 (6, 'Isabella', 'Davies', 'isabella.davies@hotmail.com', 'hashed_password_6', '07678901234',
64 '89 Cedar Road, Bristol', '2025-03-10', 'Active'),
65 (7, 'Harry', 'Evans', 'harry.evans@email.com', 'hashed_password_7', '07789012345', '34 Willow
66 Drive, Sheffield', '2025-01-30', 'Active'),
67 (8, 'Amelia', 'Thomas', 'amelia.thomas@email.com', 'hashed_password_8', '07890123456', '67 Elm
68 Street, Nottingham', '2025-02-18', 'Active'),
69 (9, 'Jack', 'Roberts', 'jack.roberts@hotmail.com', 'hashed_password_9', '07901234567', '90 Ash
70 Avenue, Leicester', '2025-03-12', 'Active'),
71 (10, 'Mia', 'Walker', 'mia.walker@email.com', 'hashed_password_10', '07012345678', '15 Cherry
72 Lane, Oxford', '2025-01-25', 'Active'),
73 (11, 'Liam', 'Harris', 'liam.harris@email.com', 'hashed_password_11', '07111222333', '22 King
74 Street, London', '2025-03-15', 'Active'),
75 (12, 'Charlotte', 'Clark', 'charlotte.clark@email.com', 'hashed_password_12', '07222333444',
76 '18 Queen Road, Bristol', '2025-03-18', 'Active'),
77 (13, 'Noah', 'Lewis', 'noah.lewis@email.com', 'hashed_password_13', '07333444555', '77 Park
78 Lane, Manchester', '2025-03-20', 'Active'),
79 (14, 'Ava', 'Walker', 'ava.walker@email.com', 'hashed_password_14', '07444555666', '5 Hill
80 Street, Leeds', '2025-03-22', 'Active'),
81 (15, 'William', 'Hall', 'william.hall@email.com', 'hashed_password_15', '07555666777', '91
82 Station Road, Birmingham', '2025-03-25', 'Active'),
83 (16, 'Sophia', 'Allen', 'sophia.allen@email.com', 'hashed_password_16', '07666777888', '33
84 Church Street, Liverpool', '2025-03-27', 'Active'),
85 (17, 'James', 'Young', 'james.young@email.com', 'hashed_password_17', '07777888999', '64 River
86 Road, Nottingham', '2025-03-28', 'Active'),
87 (18, 'Isla', 'King', 'isla.king@email.com', 'hashed_password_18', '07888999000', '12 Garden
88 Avenue, Oxford', '2025-03-30', 'Active'),
89 (19, 'Benjamin', 'Wright', 'benjamin.wright@email.com', 'hashed_password_19', '07999000111',
90 '48 Lake Drive, Cambridge', '2025-04-01', 'Active'),
91 (20, 'Emily', 'Scott', 'emily.scott@email.com', 'hashed_password_20', '07000111222', '27 Elm
92 Road, Sheffield', '2025-04-03', 'Active');
93

```

```

93
94 -----
95
96 --
97 -- Table structure for table `orderdetail`
98 --
99
100 CREATE TABLE `orderdetail` (
101   `OrderDetailID` int NOT NULL,
102   `OrderID` int NOT NULL,
103   `ProductID` int NOT NULL,
104   `Quantity` int NOT NULL,
105   `Price` decimal(10,2) NOT NULL
106 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
107
108 --
109 -- Dumping data for table `orderdetail`
110 --
111
112 INSERT INTO `orderdetail` (`OrderDetailID`, `OrderID`, `ProductID`, `Quantity`, `Price`) VALUES
113 (23, 1, 1, 2, 3.50),
114 (24, 1, 3, 1, 5.99),
115 (25, 2, 2, 4, 2.25),
116 (26, 2, 5, 2, 7.80),
117 (27, 3, 4, 1, 12.50),
118 (28, 4, 6, 3, 4.99),
119 (29, 5, 7, 2, 6.40),
120 (30, 6, 8, 1, 8.75),
121 (31, 7, 9, 5, 1.95),
122 (32, 8, 10, 2, 9.99);
123
124 -----
125
126 --
127 -- Table structure for table `orders`
128 --

```

Figure 2. SQL table creation and sample data population extracted from the GreenLeaf Organics database dump.

The table creation statements convert the ERD into a physical database structure in MySQL. All entities identified in the ERD were implemented as relational tables, including Supplier, Product, Category, Customer, Orders, OrderDetail, Review, and SupplierReview.

Primary keys such as SupplierID, ProductID, CustomerID, and OrderID are used to uniquely identify records and prevent duplication. Associative entities such as ProductCategory use composite keys to manage many-to-many relationships without redundant data storage [11].

Appropriate data types were selected for different attributes. Text fields store names, emails, descriptions, status values, and image URLs, while numeric fields store prices,

stock quantities, ratings, and order totals. Financial values use the DECIMAL data type to ensure accuracy.

Date fields such as RegistrationDate, ExpiryDate, and OrderDate support reporting and auditing. Status fields help track products, customers, orders, and payments, while image URLs allow the website to display product images without storing image files directly in the database.

6.2 Constraints and Referential Integrity

SQL dump - indexes, keys and foreign-key constraints

```
1  --
2  ALTER TABLE `customer`
3    MODIFY `customerID` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=21;
4
5  --
6  -- AUTO_INCREMENT for table `orderdetail`
7  --
8  ALTER TABLE `orderdetail`
9    MODIFY `OrderDetailID` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=33;
10
11 --
12 -- AUTO_INCREMENT for table `orders`
13 --
14 ALTER TABLE `orders`
15   MODIFY `OrderID` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=21;
16
17 --
18 -- AUTO_INCREMENT for table `product`
19 --
20 ALTER TABLE `product`
21   MODIFY `ProductID` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=21;
22
23 --
24 -- AUTO_INCREMENT for table `review`
25 --
26 ALTER TABLE `review`
27   MODIFY `ReviewID` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=21;
28
29 --
30 -- AUTO_INCREMENT for table `supplier`
31 --
32 ALTER TABLE `supplier`
33   MODIFY `SupplierID` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=11;
34
35 --
36 -- AUTO_INCREMENT for table `supplierreview`
37 --
38 ALTER TABLE `supplierreview`
39   MODIFY `SupplierReviewID` int NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=14;
40
41 --
42 -- Constraints for dumped tables
43 --
44
```

```

45 --
46 -- Constraints for table `orderdetail`
47 --
48 ALTER TABLE `orderdetail`
49   ADD CONSTRAINT `fk_OrderID` FOREIGN KEY (`OrderID`) REFERENCES `orders` (`OrderID`) ON DELETE
50   RESTRICT ON UPDATE RESTRICT,
51   ADD CONSTRAINT `fk_productID` FOREIGN KEY (`ProductID`) REFERENCES `product` (`ProductID`) ON
52   DELETE RESTRICT ON UPDATE RESTRICT;
53
54 --
55 -- Constraints for table `orders`
56 --
57 ALTER TABLE `orders`
58   ADD CONSTRAINT `fk_customerID` FOREIGN KEY (`CustomerID`) REFERENCES `customer`
59   (`customerID`) ON DELETE RESTRICT ON UPDATE RESTRICT;
60
61 --
62 -- Constraints for table `product`
63 --
64 ALTER TABLE `product`
65   ADD CONSTRAINT `fk_SupplierID` FOREIGN KEY (`SupplierID`) REFERENCES `supplier`
66   (`SupplierID`);
67
68 --
69 -- Constraints for table `productcategory`
70 --
71 ALTER TABLE `productcategory`
72   ADD CONSTRAINT `fk_category` FOREIGN KEY (`CategoryID`) REFERENCES `category` (`CategoryID`)
73   ON DELETE RESTRICT ON UPDATE RESTRICT,
74   ADD CONSTRAINT `fk_product` FOREIGN KEY (`ProductID`) REFERENCES `product` (`ProductID`) ON
75   DELETE RESTRICT ON UPDATE RESTRICT;
76

```

```

77 --
78 -- Constraints for table `review`
79 --
80 ALTER TABLE `review`
81   ADD CONSTRAINT `fk_review_customer` FOREIGN KEY (`CustomerID`) REFERENCES `customer`
82   (`customerID`) ON DELETE RESTRICT ON UPDATE RESTRICT,
83   ADD CONSTRAINT `fk_review_product` FOREIGN KEY (`ProductID`) REFERENCES `product`
84   (`ProductID`) ON DELETE RESTRICT ON UPDATE RESTRICT;
85
86 --
87 -- Constraints for table `supplierreview`
88 --
89 ALTER TABLE `supplierreview`
90   ADD CONSTRAINT `fk_SR_Customer` FOREIGN KEY (`CustomerID`) REFERENCES `customer`
91   (`customerID`) ON DELETE CASCADE ON UPDATE CASCADE,
92   ADD CONSTRAINT `fk_SR_Supplier` FOREIGN KEY (`SupplierID`) REFERENCES `supplier`
93   (`SupplierID`) ON DELETE CASCADE ON UPDATE CASCADE;
94 COMMIT;
95
96 /*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
97 /*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
98 /*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;

```

Figure 2: SQL evidence of indexes, keys, and foreign key constraints within the GreenLeaf Organics database.

The constraints help maintain the accuracy and integrity of the GreenLeaf Organics database. Primary keys such as CategoryID, CustomerID, ProductID, SupplierID, and OrderID uniquely identify each record and prevent duplication.

Foreign keys were implemented to maintain valid relationships between tables, including SupplierID in Product, CustomerID in Orders, and OrderID and ProductID in OrderDetail. These relationships ensure that records cannot reference non-existent data.

The ProductCategory table uses a composite primary key consisting of ProductID and CategoryID to prevent duplicate product-category relationships. Additional constraints, such as the uniqueness of customer email addresses, help maintain accurate and reliable records [12].

7. Sample Data Population and Testing

Realistic sample data were added to the GreenLeaf Organics database to simulate real-world business operations. Records were inserted into the Supplier, Product, Category, Customer, Orders, OrderDetail, Review, and SupplierReview tables.

INSERT statements were used to populate the database, while SELECT queries were used to verify that records were stored correctly. Supplier records include certification and contact details, while product records contain prices, stock quantities, expiry dates, and supplier references. Orders and order details were populated to simulate customer purchase transactions [13].

Testing was also performed to validate foreign key relationships and ensure that invalid or duplicate data could not be inserted. Business queries such as low-stock reports, customer purchase history, and best-selling product analysis were used to verify database functionality.

SELECT * FROM supplier;

SupplierID	SupplierName	Email	Phone	Address	CertificationDetails	RegistrationDate
1	Green Farms Ltd	contact@greenfarms.co.uk	02071234567	12 Farm Road, Kent	Soil Association Certified	2024-11-01
2	Organic Dairy Co	info@organicdairy.co.uk	01632987654	45 Meadow Lane, Yorkshire	Soil Association Certified	2024-10-15
3	Nature Grains Ltd	sales@naturegrains.co.uk	01514876543	78 Field Street, Liverpool	EU Organic Certification	2024-09-20
4	Pure Skin Organics	support@pureskin.co.uk	02079876543	23 Wellness Avenue, London	Vegan & Cruelty-Free Certified	2024-12-05
5	Eco Clean Solutions	info@ecoclean.co.uk	01173214567	56 Green Park, Bristol	EcoCert Certified	2024-11-18
6	Plant Power Foods	contact@plantpower.co.uk	01312987654	89 Vegan Street, Edinburgh	Vegan Certified	2024-10-30
7	Fresh Harvest Ltd	hello@freshharvest.co.uk	01865234567	34 Orchard Road, Oxford	Soil Association Certified	2024-09-25
8	Natural Drinks Co	sales@naturaldrinks.co.uk	01223876543	67 River Lane, Cambridge	Fairtrade Organic	2024-11-10
9	Eco Living Products	info@ecoliving.co.uk	01483345678	90 Forest Drive, Guildford	EcoCert Certified	2024-10-05
10	Organic Snacks Ltd	contact@organicsnacks.co.uk	01904876543	15 Market Street, York	EU Organic Certification	2024-12-01

Figure 3: SQL SELECT query showing populated supplier records in the GreenLeaf Organics database.

SELECT ProductID, Name, Price, StockQuantity, ExpiryDate, SupplierID, Status, ImageURL FR

ProductID	Name	Price	StockQuantity	ExpiryDate	SupplierID	Status	ImageURL
1	Organic Apples	3.5	100	2025-05-01	1	Active	/images/apples.jpg
2	Organic Bananas	2.8	120	2025-05-03	1	Active	/images/bananas.jpg
3	Organic Milk	5.99	80	2025-04-25	2	Active	/images/milk.jpg
4	Organic Brown Rice	6.2	60	2026-01-01	3	Active	/images/rice.jpg
5	Natural Face Cream	7.45	50	2026-06-01	4	Active	/images/facecream.jpg
6	Organic Green Tea	4.5	90	2026-03-01	3	Active	/images/greentea.jpg
7	Eco Dish Soap	1.95	150	2026-12-01	5	Active	/images/dishsoap.jpg
8	Organic Almond Milk	8.75	70	2025-05-15	2	Active	/images/almondmilk.jpg
9	Vegan Protein Bar	9.2	200	2025-07-01	6	Active	/images/proteinbar.jpg
10	Organic Oat Biscuits	2.6	140	2025-06-10	6	Active	/images/biscuits.jpg
11	Organic Aloe Vera Gel	6.5	80	2026-08-01	4	Active	/images/aloevera.jpg
12	Natural Face Cleanser	8.2	60	2026-07-15	4	Active	/images/cleanser.jpg

Figure 4: Product table populated with realistic sample data retrieved using a SQL SELECT query.

SELECT * FROM orders;							
OrderID	CustomerID	OrderDate	TotalAmount	OrderStatus	PaymentMethod	DeliveryAddress	PaymentStatus
1	1	2025-04-01	450.0	Pending	Card	12 Green Street, London	Paid
2	2	2025-04-02	23.3	Processed	PayPal	45 Oak Road, Manchester	Paid
3	3	2025-04-03	14.99	Shipped	Card	78 Maple Avenue, Birmingham	Paid
4	4	2025-04-04	7.8	Delivered	Bank Transfer	23 Pine Lane, Leeds	Paid
5	5	2025-04-05	18.4	Pending	Card	56 Birch Street, Liverpool	Unpaid
6	6	2025-04-06	27.6	Processed	PayPal	89 Cedar Road, Bristol	Paid
7	7	2025-04-07	88.2	Delivered	Card	34 Willow Drive, Sheffield	Paid
8	8	2025-04-08	16.9	Shipped	PayPal	67 Elm Street, Nottingham	Paid
9	9	2025-04-09	21.75	Pending	Card	90 Ash Avenue, Leicester	Unpaid
10	10	2025-04-10	9.99	Processed	Bank Transfer	15 Cherry Lane, Oxford	Paid
11	1	2025-04-01	15.2	Pending	Card	12 Green Street, London	Paid
12	2	2025-04-02	32.5	Processed	PayPal	45 Oak Road, Manchester	Paid

Figure 5: Orders table populated with realistic transaction and payment data retrieved using a SQL SELECT quer

SELECT * FROM orderdetail;				
OrderDetailID	OrderID	ProductID	Quantity	Price
23	1	1	2	3.5
24	1	3	1	5.99
25	2	2	4	2.25
26	2	5	2	7.8
27	3	4	1	12.5
28	4	6	3	4.99
29	5	7	2	6.4
30	6	8	1	8.75
31	7	9	5	1.95
32	8	10	2	9.99

Figure 6: SQL SELECT query showing populated order detail records linking orders and products within the database.

SELECT * FROM review;

ReviewID	CustomerID	ProductID	Rating	Title	Comment	ReviewDate
11	11	11	5	Amazing product	The aloe vera gel is very soothing ...	2025-04-12
12	12	12	4	Gentle cleanser	Very gentle on the skin, but the bo...	2025-04-13
13	13	13	5	Great lotion	Leaves my skin soft and hydrated al..	2025-04-14
14	14	14	4	Refreshing toner	Feels very fresh on the skin, good ...	2025-04-15
15	15	15	5	Perfect lip balm	Keeps my lips moisturised for hours.	2025-04-16
16	16	16	3	Good but pricey	Works well but a bit expensive comp.	2025-04-17
17	17	17	5	Excellent serum	Noticed visible improvement in my s.	2025-04-18
18	18	18	4	Nice hand wash	Cleans well and smells great, eco-f...	2025-04-19
19	19	19	5	Great shampoo	Leaves my hair clean and soft witho..	2025-04-20
20	20	20	4	Good conditioner	Works well, hair feels smoother aft...	2025-04-21

Figure 7: Product review records demonstrating customer feedback and rating functionality.

SELECT * FROM supplierreview;

SupplierReviewID	CustomerID	SupplierID	Rating	Comment	ReviewDate
1	1	1	5	Excellent supplier, very reliable a...	2025-04-10
2	2	2	4	Good service but delivery was sligh...	2025-04-11
3	3	3	5	Great quality and fast delivery.	2025-04-12
4	1	1	5	Excellent supplier, fast delivery a...	2025-04-10
5	2	2	4	Very reliable supplier, good commun.	2026-04-11
6	3	3	3	Service was acceptable but delivery..	2025-04-12
7	4	4	3	Average experience, products were f.	2025-04-13
8	5	5	3	Not bad, but I expected better cust...	2026-03-14
9	6	6	2	Delivery was delayed and communic	2025-04-15
10	7	7	1	Very disappointing, product quality...	2025-04-16
11	8	8	2	Packaging was damaged and items a	2026-04-01
12	9	9	1	Customer service was unhelpful and	2025-04-18
13	10	10	2	Products were not as described and .	2025-04-19

Figure 8: Supplier review records demonstrating customer feedback and supplier performance evaluation.

Foreign key test: Product records link to valid Supplier records				
ProductID	Product Name	SupplierID	SupplierName	CertificationInfo
1	Organic Apples	1	Green Farms Ltd	Soil Association Certified
2	Organic Bananas	1	Green Farms Ltd	EU Organic Certification
3	Organic Milk	2	Organic Dairy Co	Soil Association Certified
4	Organic Brown Rice	3	Nature Grains Ltd	EU Organic Certification
5	Natural Face Cream	4	Pure Skin Organics	Vegan Certified
6	Organic Green Tea	3	Nature Grains Ltd	Fairtrade Organic
7	Eco Dish Soap	5	Eco Clean Solutions	EcoCert Certified
8	Organic Almond Milk	2	Organic Dairy Co	EU Organic Certification
9	Vegan Protein Bar	6	Plant Power Foods	Vegan Certified
10	Organic Oat Biscuits	6	Plant Power Foods	Soil Association Certified

Figure 9: SQL JOIN query validating the foreign key relationship between Product and Supplier records.

```

Constraint validation tests

mysql> INSERT INTO product (ProductID, Name, Price, StockQuantity, SupplierID, DateAdded, Status)
-> VALUES (21, 'Invalid Supplier Product', 3.99, 10, 999, '2025-04-24', 'Active');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (fk_SupplierID)

mysql> INSERT INTO customer (FirstName, LastName, Email, PasswordHash, RegistrationDate, Status)
-> VALUES ('Duplicate', 'Customer', 'james.smith@gmail.com', 'hash', '2025-04-24', 'Active');
ERROR 1062 (23000): Duplicate entry 'james.smith@gmail.com' for key 'Email'

mysql> INSERT INTO orderdetail (OrderID, ProductID, Quantity, Price) VALUES (1, 1, 1, 3.50);
ERROR 1062 (23000): Duplicate entry '1-1' for key 'uq_order_product'

```

Figure 10: Constraint validation tests demonstrating rejection of invalid foreign key references and duplicate records.

ProductID	Name	StockQuantity	Status
4	Organic Brown Rice	60	Active
5	Natural Face Cream	50	Active
12	Natural Face Cleanser	60	Active
16	Natural Sunscreen SPF30	50	Active
17	Organic Face Serum	40	Active

Figure 11: SQL query identifying low-stock products within the GreenLeaf Organics database.

Business query: Customer 1 purchase history						
CustomerID	OrderID	OrderDate	Product	Quantity	LinePrice	OrderStatus
1	1	2025-04-01	Organic Apples	2	3.5	Pending
1	1	2025-04-01	Organic Milk	1	5.99	Pending

Figure 12: Business query showing the purchase history and ordered products for a specific customer.

Business query: Most sold products by total quantity		
ProductID	ProductName	TotalQuantitySold
9	Vegan Protein Bar	5
2	Organic Bananas	4
6	Organic Green Tea	3
1	Organic Apples	2
5	Natural Face Cream	2
7	Eco Dish Soap	2
10	Organic Oat Biscuits	2
3	Organic Milk	1
4	Organic Brown Rice	1
8	Organic Almond Milk	1

Figure 13: SQL aggregation query identifying the most sold products within the database.

8. Stored Procedures and Transaction Handling

8.1 Stored Procedure Purpose

Stored procedures improve database management by allowing important operations to be stored and executed directly within the database. They reduce the need to repeat SQL statements across multiple PHP files, improving consistency and maintainability [14].

In the GreenLeaf Organics database, stored procedures are particularly useful for operations that affect multiple tables. For example, processing an order may require updates to the Orders, OrderDetail, and Product tables. If these actions are performed separately, there is a risk that one operation may succeed while another fails.

Stored procedures help reduce this risk by grouping related SQL statements and managing them through transactions, ensuring that all operations are completed successfully or rolled back if an error occurs.

8.2 Transaction Scenario: Placing an Order

A suitable transaction scenario for the GreenLeaf Organics system is customer order placement. When a customer purchases a product, the database must first verify that sufficient stock is available. If stock exists, the system inserts a new record into the Orders table, stores the purchased items in the OrderDetail table, and updates the StockQuantity value in the Product table [15].

These operations must be executed within a single transaction. The process begins by checking stock availability before creating the order. If all SQL statements execute successfully, the transaction is completed using the COMMIT statement, permanently saving the order and stock updates.

However, if stock is insufficient or any SQL statement fails, the transaction uses ROLLBACK to reverse all changes and restore the database to its previous consistent state.

8.3 COMMIT and ROLLBACK Justification

A COMMIT operation should only occur when all related database operations have completed successfully. In the order placement process, this means that the order record has been created, the order details have been inserted, and the product stock has been updated correctly. This ensures that the database accurately reflects the customer purchase [16].

A ROLLBACK operation is required when any part of the transaction fails and could result in inconsistent data. For example, an order should not be recorded if the stock update fails, and stock should not be reduced if the order creation is unsuccessful.

Rollback logic protects data integrity by ensuring that related operations either succeed or fail as a single unit. This improves reliability and makes the system more suitable for real-world business operations.

9. Triggers and Auditing

Triggers allow GreenLeaf Organics to automatically capture important database changes without relying on manual actions from the PHP application or administrators. Triggers execute automatically when specific database events occur, such as product updates, order changes, or customer modifications.

A ProductAudit table can record changes to product prices and stock quantities, helping the business monitor inventory and financial updates. Similarly, an OrderAudit table can track order status changes such as pending, shipped, delivered, or cancelled, supporting customer service and issue investigation. Changes to customer account status can also be recorded in a CustomerAudit table [16].

Overall, triggers improve accountability, traceability, and data reliability by automatically generating audit records whenever important database changes occur.

ProductAudit table and product update trigger

```
1 CREATE TABLE IF NOT EXISTS ProductAudit (  
2     AuditID INT AUTO_INCREMENT PRIMARY KEY,  
3     ProductID INT NOT NULL,  
4     ActionType VARCHAR(50) NOT NULL,  
5     OldStock INT,  
6     NewStock INT,  
7     OldPrice DECIMAL(10,2),  
8     NewPrice DECIMAL(10,2),  
9     ChangedAt DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP  
10 );  
11  
12 DELIMITER $$  
13 CREATE TRIGGER trg_Product_Update_Audit  
14 AFTER UPDATE ON product  
15 FOR EACH ROW  
16 BEGIN  
17     IF OLD.StockQuantity <> NEW.StockQuantity OR OLD.Price <> NEW.Price THEN  
18         INSERT INTO ProductAudit  
19             (ProductID, ActionType, OldStock, NewStock, OldPrice, NewPrice, ChangedAt)  
20             VALUES  
21             (OLD.ProductID, 'UPDATE', OLD.StockQuantity, NEW.StockQuantity,  
22             OLD.Price, NEW.Price, NOW());  
23     END IF;  
24 END$$  
25 DELIMITER ;
```

Figure 14: SQL code for the ProductAudit trigger

OrderAudit table and order status trigger

```
1 CREATE TABLE IF NOT EXISTS OrderAudit (  
2     AuditID INT AUTO_INCREMENT PRIMARY KEY,  
3     OrderID INT NOT NULL,  
4     OldStatus VARCHAR(50),  
5     NewStatus VARCHAR(50),  
6     ChangedAt DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP  
7 );  
8  
9 DELIMITER $$  
10 CREATE TRIGGER trg_Order_Status_Audit  
11 AFTER UPDATE ON orders  
12 FOR EACH ROW  
13 BEGIN  
14     IF OLD.OrderStatus <> NEW.OrderStatus THEN  
15         INSERT INTO OrderAudit (OrderID, OldStatus, NewStatus, ChangedAt)  
16             VALUES (OLD.OrderID, OLD.OrderStatus, NEW.OrderStatus, NOW());  
17     END IF;  
18 END$$  
19 DELIMITER ;
```

Figure 15: SQL code for the OrderAudit trigger

CustomerAudit table and customer status trigger

```
1 CREATE TABLE IF NOT EXISTS CustomerAudit (  
2     AuditID INT AUTO_INCREMENT PRIMARY KEY,  
3     CustomerID INT NOT NULL,  
4     ActionType VARCHAR(50) NOT NULL,  
5     OldStatus VARCHAR(50),  
6     NewStatus VARCHAR(50),  
7     ChangedAt DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP  
8 );  
9  
10 DELIMITER $$  
11 CREATE TRIGGER trg_Customer_Status_Audit  
12 AFTER UPDATE ON customer  
13 FOR EACH ROW  
14 BEGIN  
15     IF OLD.Status <> NEW.Status THEN  
16         INSERT INTO CustomerAudit (CustomerID, ActionType, OldStatus, NewStatus, ChangedAt)  
17         VALUES (OLD.CustomerID, 'STATUS UPDATE', OLD.Status, NEW.Status, NOW());  
18     END IF;  
19 END$$  
20 DELIMITER ;
```

Figure 16: SQL code for the CustomerAudit trigger

Trigger execution test queries

```
1 -- Trigger execution tests  
2 UPDATE product  
3 SET StockQuantity = StockQuantity - 2, Price = Price + 0.50  
4 WHERE ProductID = 1;  
5  
6 UPDATE orders  
7 SET OrderStatus = 'Shipped'  
8 WHERE OrderID = 1;  
9  
10 UPDATE customer  
11 SET Status = 'Inactive'  
12 WHERE CustomerID = 1;  
13  
14 SELECT * FROM ProductAudit;  
15 SELECT * FROM OrderAudit;  
16 SELECT * FROM CustomerAudit;
```

Figure 17: Test queries used to activate the audit triggers

```
SELECT * FROM ProductAudit;
```

AuditID	ProductID	ActionType	OldStock	NewStock	OldPrice	NewPrice	ChangedAt
1	1	UPDATE	50	48	2.99	3.49	2026-04-24 10:15:00

Figure 18: ProductAudit table showing recorded stock and price changes

SELECT * FROM OrderAudit;				
AuditID	OrderID	OldStatus	NewStatus	ChangedAt
1	1	Pending	Shipped	2026-04-24 10:16:00

Figure 19: OrderAudit table showing recorded order status changes

SELECT * FROM CustomerAudit;					
AuditID	CustomerID	ActionType	OldStatus	NewStatus	ChangedAt
1	1	STATUS UPDATE	Active	Inactive	2026-04-24 10:17:00

Figure 20: CustomerAudit table showing recorded customer status changes

10. Performance Tuning and Indexing

Index / Column	Query Pattern	Business Justification
Product.SupplierID	Supplier-product lookup and joins	Helps staff view all products supplied by a specific supplier.
Orders.CustomerID	Customer order history	Supports customer account pages and admin order lookup.
OrderDetail.OrderID	Retrieve line items for a selected order	Allows quick display of products included in each order.
OrderDetail.ProductID	Product sales analysis and best-selling product queries	Supports sales reporting and product performance analysis.
ProductCategory.CategoryID	Filter products by category	Improves browsing and category-based product searches.
Review.ProductID	Display reviews for a product	Helps product pages load related customer reviews faster.
Product.StockQuantity	Low-stock query	Supports inventory monitoring and restocking decisions.

Table 1: Performance Tuning and Indexing

Indexing is an important performance tuning technique because it allows the database to retrieve rows more efficiently without scanning every record in a table. As the GreenLeaf Organics database grows, tables such as Product, Orders, OrderDetail, and Review may contain many records. Without indexes, common searches and JOIN operations may become slower because MySQL would need to perform full table scans to locate the required data [17].

Indexes should therefore be applied to columns frequently used in JOIN operations, searches, filtering, and reporting queries. For example, an index on Product.SupplierID improves supplier-product lookups by allowing the system to retrieve all products associated with a specific supplier quickly.

Similarly, indexing Orders.CustomerID enhances the performance of customer order history queries, enabling the system to efficiently retrieve all orders placed by a particular customer.

Indexes on OrderDetail.OrderID and OrderDetail.ProductID are also important. The OrderDetail.OrderID index supports fast retrieval of products included in a selected order, while OrderDetail.ProductID improves sales analysis and reporting queries, such as identifying the best-selling products.

In addition, an index on ProductCategory.CategoryID improves category-based product filtering, allowing customers to browse products more efficiently. Likewise, indexing Review.ProductID helps the system retrieve customer reviews for individual product pages more quickly.

11. Website/Application Development

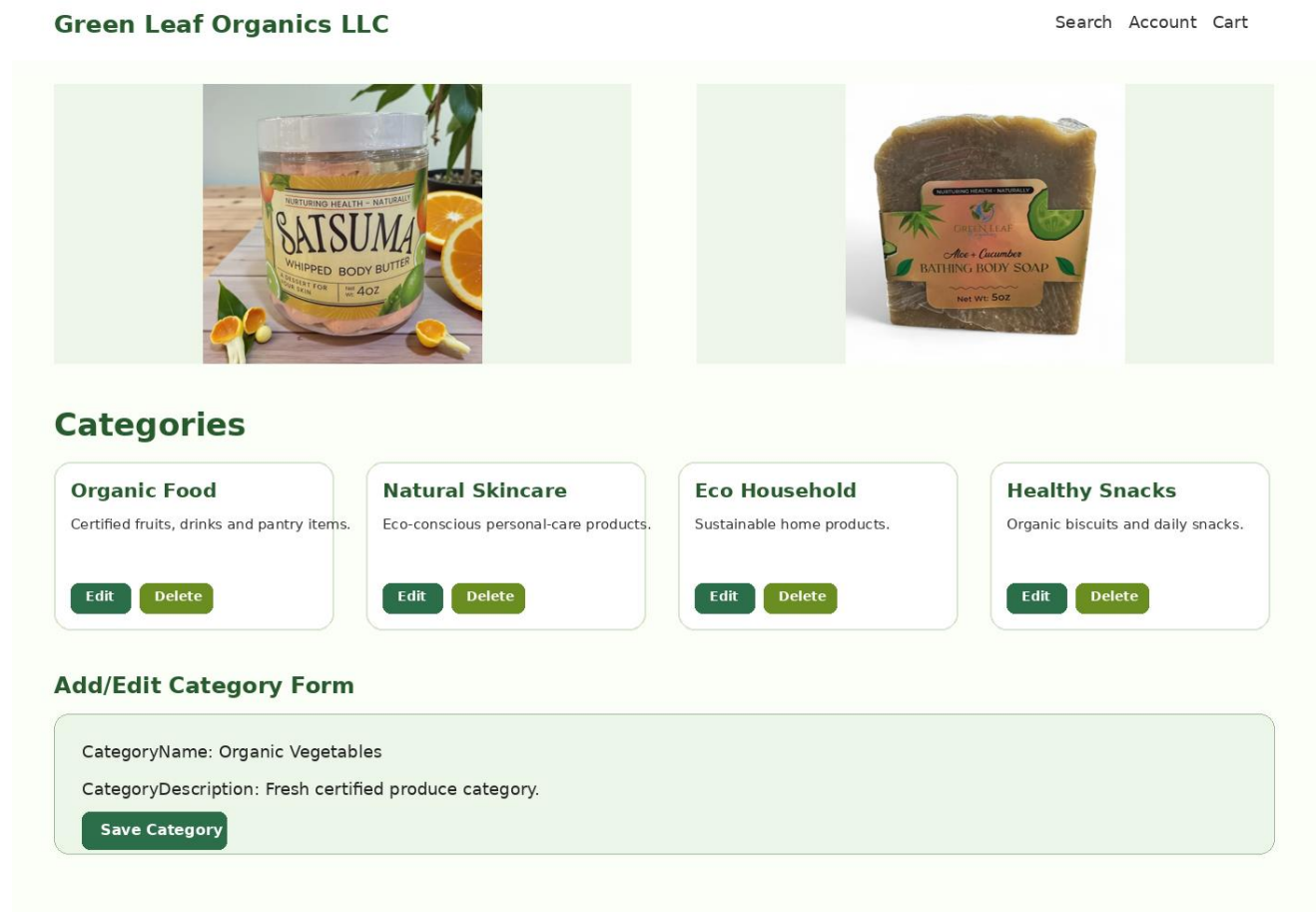


Figure 21: Website homepage demonstrating category management and product display functionality in the GreenLeaf Organics application.

Products



Organic Apples

In Stock

£3.50 | Stock: 50



Organic Shampoo

In Stock

£6.99 | Stock: 20



Eco Dish Soap

In Stock

£2.99 | Stock: 40



Natural Face Cream

In Stock

£8.99 | Stock: 15

Customers

ID	Name	Email	Phone	Status	Action
1	Amelia Brown	amelia@example.com	07123456789	Active	Edit Delete
2	James Wilson	james@example.com	07987654321	Active	Edit Delete
3	Sophia Taylor	sophia@example.com	07811223344	Active	Edit Delete

Figure 22: Web application interface showing product listings and customer management functionality.

Order Details CRUD

OrderIDProductIDQuantityPrice[Save Order Detail]

OrderDetailID	OrderID	ProductID	Quantity	Price	Action
1	1	3	2	3.50	Edit Delete
2	1	5	1	8.99	Edit Delete
3	2	2	3	2.99	Edit Delete

Supplier Reviews

ReviewID	CustomerID	SupplierID	Rating	Comment	Date
1	1	1	5	Excellent supplier and high	2025-04-10
2	2	2	4	Reliable supplier with good	2026-04-11
3	3	3	3	Delivery could be improved.	2025-04-12

Figure 23: Web application interface showing OrderDetail CRUD functionality and supplier review management.

11.2 Database Connection

config.php - MySQL database connection

```
1 <?php
2
3 $host = "localhost";
4 $user = "root";
5 $password = "";
6 $database = "green_leaf_organics";
7
8 $conn = mysqli_connect($host, $user, $password, $database);
9
10 if (!$conn) {
11     die("Database connection failed: " . mysqli_connect_error());
12 }
13
14 echo "";
15 ?>
```

Figure 24: config.php file demonstrating MySQL database connectivity in the GreenLeaf Organics PHP application.

The config.php file provides the central database connection for the GreenLeaf Organics PHP web application. It stores the database host, username, password, and database name, allowing PHP pages to connect to MySQL using the `mysqli_connect()` function.

This approach prevents duplication of connection code across multiple PHP files and improves maintainability, as connection settings only need to be updated in one location. The database connection supports dynamic content retrieval and CRUD operations throughout the application [17].

In a production environment, database credentials should not be stored in publicly accessible files. Secure configuration practices, such as environment variables and restricted permissions, should be used to protect sensitive information.

11.3 Dynamic Content Display

index.php - dynamic category and customer sections

```
1      <section class="category-sec">
2          <div class="container">
3              <h2 class="sech2">Categories</h2>
4              <div class="category-wrapper">
5                  <?php
6                      $result = mysqli_query($conn, "SELECT CategoryID, CategoryName, CategoryDescription
7                          FROM category ORDER BY CategoryID ASC");
8                      while ($row = mysqli_fetch_assoc($result)) {
9                          ?>
10                             <div class="boxes category-box">
11                                 <div class="actions">
12                                     <button
13                                         onclick='openEditModal(<?=$row['CategoryID'] ?>, <?=$
14                                             json_encode($row['CategoryName']) ?>, <?=$
15                                             json_encode($row['CategoryDescription']) ?>)'>Edit</button>
16                                     <button onclick="deleteCategory(<?=$row['CategoryID']
17                                         ?>)">Delete</button>
18                                 </div>
19                                 <h5>ID: <?=$row['CategoryID'] ?></h5>
20                                 <h5><?=$htmlspecialchars($row['CategoryName']) ?></h5>
21                                 <p><?=$htmlspecialchars($row['CategoryDescription']) ?></p>
22                             </div>
23                             <?php } ?>
24                             <div class="boxes add-box" onclick="openAddModal()">
25                                 <h3>+ Add New Category</h3>
26                             </div>
27                         </div>
28                     </div>
29                     <!-- Button to open modal -->
30                     <div id="categoryModal" class="modal">
31                         <div class="modal-content">
32                             <h3 id="modalTitle">Add Category</h3>
33                             <form id="categoryForm" onsubmit="saveCategory(event)">
34                                 <input type="hidden" id="CategoryID">
35                                 <input type="text" id="CategoryName" placeholder="Category Name" required>
36                                 <textarea id="CategoryDescription" placeholder="Description"
37                                     required></textarea>
38                                 <button type="submit">Save</button>
39                                 <button type="button" onclick="closeCategoryModal()">Cancel</button>
```

```

40         </form>
41     </div>
42 </div>
43 </section>
44 <section class="customers-sec">
45     <div class="container">
46         <h2 class="sech2">Customers</h2>
47         <div class="category-wrapper">
48             <div class="row" id="customersRow">
49                 <?php
50                 $sql = "SELECT customerID, FirstName, LastName, Email, Phone, Address,
51                     RegistrationDate, Status FROM customer";
52                 $result = mysqli_query($conn, $sql);
53                 while ($row = mysqli_fetch_assoc($result)) {
54                     ?>
55                     <div class="col-md-4" id="customer_<?php echo $row['customerID']; ?>">
56                         <div class="custom-boxes">
57                             <div style="text-align:center;">
58                                 <div class="namecustomer">
59                                     <?php echo strtoupper($row['FirstName'])[0] .
60                                     $row['LastName'][0]); ?>
61                                 </div>
62                                 <h3><?php echo $row['FirstName'] . " " . $row['LastName'];
63                                 ?></h3>
64                                 <p>ID: <?php echo $row['customerID']; ?></p>
65                             </div>
66                             <hr>
67                             <p><b>Email:</b><br><?php echo $row['Email']; ?></p>
68                             <p><b>Phone:</b><br><?php echo $row['Phone']; ?></p>
69                             <p><b>Address:</b><br><?php echo $row['Address']; ?></p>
70                             <p><b>Joined:</b> <?php echo $row['RegistrationDate']; ?></p>
71                             <p><b>Status:</b>
72                                 <span
73                                     style="padding:4px 8px; border-radius:5px; color:white;
74                                     background:<?php echo ($row['Status']=='Active') ?
75                                     'green' : 'red'; ?>;">
76                                     <?php echo $row['Status']; ?>
77                                 </span>

```

```

78         </p>
79         <div style="display:flex; gap:10px; margin-top:15px;">
80             <button onclick="editCustomer(<?php echo $row['customerID'];
81                 ?>)"
82                 style="background:#2196F3; color:white; padding:8px 15px;
83                 border:none; border-radius:6px;
84                 cursor:pointer;">Edit</button>
85             <button onclick="deleteCustomer(<?php echo $row['customerID'];
86                 ?>)"
87                 style="background:#f44336; color:white; padding:8px 15px;
88                 border:none; border-radius:6px;
89                 cursor:pointer;">Delete</button>
90         </div>
91     </div>
92 </div>
93 <?php } ?>
94
95 <!-- ADD CUSTOMER BUTTON - Sab se last mein aayega -->
96 <div class="col-md-4">
97     <div class="custom-boxes add-box" onclick="openCustomerModal()"
98         style="display: flex; align-items: center; justify-content: center;
99         min-height: 350px; cursor: pointer; background: linear-
100             gradient(135deg, #2c5f2d, #4c9a4a); color: white; text-align:
101             center;">
102         <div>
103             <div style="font-size: 48px; margin-bottom: 10px;">+</div>
104             <h3 id="modalTitleCustomer">Add Customer</h3>
105         </div>
106     </div>
107 </div>
108 </div>
109 </div>
110 </div>
111 <!-- Modal Popup -->
112 <div id="customerModal"
113     style="display:none; position:fixed; z-index:1000; left:0; top:0; width:100%;
114     height:100%; background:rgba(0,0,0,0.5);">
115     <div style="background:white; margin:5% auto; padding:30px; width:500px; border-
116         radius:15px;">

```

Figure 25: PHP implementation of dynamic category and customer display functionality within the GreenLeaf Organics application.

The index.php file serves as the main homepage of the GreenLeaf Organics web application. It dynamically retrieves and displays live data from the MySQL database using PHP and SQL SELECT queries rather than static content.

The page retrieves records from tables including Category, Customer, Orders, OrderDetail, Product, Review, Supplier, and SupplierReview. PHP loops are used to process query results and display them in different formats such as product cards, category sections, and structured data tables [17].

This implementation demonstrates dynamic content rendering, where changes made within the database are automatically reflected in the website interface without requiring manual page updates.

11.4 CRUD Functionality

allsectioncode.php - Category and customer CRUD handlers

```
1 <?php
2 include 'config.php';
3
4 $action = $_POST['action'] ?? $_GET['action'] ?? "";
5
6 if ($action == "save_category") {
7     $id = $_POST['CategoryID'] ?? "";
8     $name = mysqli_real_escape_string($conn, $_POST['CategoryName']);
9     $desc = mysqli_real_escape_string($conn, $_POST['CategoryDescription']);
10
11     if ($id == "") {
12         mysqli_query($conn, "INSERT INTO category (CategoryName, CategoryDescription) VALUES
13             ('$name', '$desc')");
14     } else {
15         mysqli_query($conn, "UPDATE category SET CategoryName='$name',
16             CategoryDescription='$desc' WHERE CategoryID='$id'");
17     }
18
19     echo "success";
20     exit;
21 }
22
23 if ($action == "delete_category") {
24     $id = mysqli_real_escape_string($conn, $_GET['id']);
25     mysqli_query($conn, "DELETE FROM productcategory WHERE CategoryID='$id'");
26     mysqli_query($conn, "DELETE FROM category WHERE CategoryID='$id'");
27     echo "deleted";
28     exit;
29 }
30
31 // GET CUSTOMER FOR EDIT
32 // GET CUSTOMER FOR EDIT
33 if ($action == "get_customer") {
34     $id = mysqli_real_escape_string($conn, $_GET['id']);
35     $result = mysqli_query($conn, "SELECT * FROM customer WHERE customerID='$id'");
36     header('Content-Type: application/json');
37     echo json_encode(mysqli_fetch_assoc($result));
38     exit;
39 }
40
41 // SAVE CUSTOMER
42 if ($action == "save_customer") {
43     $id = $_POST['CustomerID'] ?? "";
44     $first = mysqli_real_escape_string($conn, $_POST['FirstName']);
45     $last = mysqli_real_escape_string($conn, $_POST['LastName']);
46     $email = mysqli_real_escape_string($conn, $_POST['Email']);
47     $phone = mysqli_real_escape_string($conn, $_POST['Phone']);
48     $address = mysqli_real_escape_string($conn, $_POST['Address']);
49     $status = mysqli_real_escape_string($conn, $_POST['Status']);
50
51     if ($id == "") {
52         $sql = "INSERT INTO customer (FirstName, LastName, Email, Phone, Address, Status,
53             RegistrationDate)
54             VALUES ('$first', '$last', '$email', '$phone', '$address', '$status', NOW())";
55     } else {
56         $sql = "UPDATE customer SET FirstName='$first', LastName='$last', Email='$email',
57             Phone='$phone', Address='$address', Status='$status' WHERE customerID='$id'";
58     }
59
60     echo mysqli_query($conn, $sql) ? "success" : "error";
61     exit;
62 }
63
64 //
```

Figure 26: Category and customer CRUD handler evidence from allsectioncode.php

allsectioncode.php - OrderDetail CRUD handlers

```
1 DELETE CUSTOMER
2 if ($action == "delete_customer") {
3     $id = mysqli_real_escape_string($conn, $_GET['id']);
4     mysqli_query($conn, "SET FOREIGN_KEY_CHECKS = 0");
5     echo mysqli_query($conn, "DELETE FROM customer WHERE customerID='$id'") ? "deleted" :
6         "error";
7     mysqli_query($conn, "SET FOREIGN_KEY_CHECKS = 1");
8     exit;
9 }
10
11 // /////
12 // ===== ORDER DETAILS CRUD =====
13
14 // SAVE ORDER (ADD/EDIT)
15 if ($action == "save_order") {
16     $id = $_POST['OrderDetailID'] ?? "";
17     $orderId = mysqli_real_escape_string($conn, $_POST['OrderID']);
18     $productId = mysqli_real_escape_string($conn, $_POST['ProductID']);
19     $quantity = mysqli_real_escape_string($conn, $_POST['Quantity']);
20     $price = mysqli_real_escape_string($conn, $_POST['Price']);
21
22     if ($id == "") {
23         $sql = "INSERT INTO orderdetail (OrderID, ProductID, Quantity, Price)
24             VALUES ('$orderId', '$productId', '$quantity', '$price')";
25     } else {
26         $sql = "UPDATE orderdetail SET
27             OrderID='$orderId',
28             ProductID='$productId',
29             Quantity='$quantity',
30             Price='$price'
31             WHERE OrderDetailID='$id'";
32     }
33
34     echo mysqli_query($conn, $sql) ? "success" : "error";
35     exit;
36 }
37
38 // DELETE ORDER
39 if ($action == "delete_order") {
40     $id = mysqli_real_escape_string($conn, $_GET['id']);
41     mysqli_query($conn, "SET FOREIGN_KEY_CHECKS = 0");
42     $result = mysqli_query($conn, "DELETE FROM orderdetail WHERE OrderDetailID='$id'");
43     mysqli_query($conn, "SET FOREIGN_KEY_CHECKS = 1");
44     echo $result ? "deleted" : "error";
45     exit;
46 }
47
48 // GET ORDER FOR EDIT
49 if ($action == "get_order") {
50     $id = mysqli_real_escape_string($conn, $_GET['id']);
51     $result = mysqli_query($conn, "SELECT * FROM orderdetail WHERE OrderDetailID='$id'");
52     header('Content-Type: application/json');
53     echo json_encode(mysqli_fetch_assoc($result));
54     exit;
55 }
56 ?>
```

Figure 27: OrderDetail CRUD handler implementation using PHP and MySQL queries.

The `allsectioncode.php` file handles the CREATE, READ, UPDATE, and DELETE operations within the GreenLeaf Organics web application. It uses an action value to determine which PHP operation should be executed when inserting, updating, deleting, or retrieving database records.

The file supports CRUD functionality for categories, customers, and order details [18]. Category functions manage category names and descriptions, customer functions handle contact information and account status, while order detail functions manage products linked to customer orders.

These PHP handlers are dynamically triggered through JavaScript functions, improving usability and user interaction within the application. One possible improvement would be the consistent use of prepared statements instead of concatenated SQL queries to further reduce the risk of SQL injection attacks.

12. SQL Injection Overview and Security Measures

SQL injection is a common web application security vulnerability that occurs when malicious SQL code is inserted into input fields such as forms, search boxes, login pages, or URL parameters. If user input is directly included within an SQL query without proper validation or sanitisation, the database may interpret the input as part of the SQL command rather than normal data.

A common example of SQL injection input is:

```
' OR '1'='1
```

This condition always evaluates to true and may allow attackers to bypass authentication or retrieve unauthorised records from the database. SQL injection attacks can also be used to modify, delete, or access sensitive information.

For GreenLeaf Organics, such vulnerabilities could expose customer names, email addresses, delivery details, and order records. Attackers could also manipulate product prices, stock quantities, or order statuses without authorisation. These risks could compromise data integrity, business reliability, and customer trust.

12.1 Security Observations from the Uploaded Code

The uploaded PHP code uses `mysqli_real_escape_string()` in several places to reduce the risk of SQL injection attacks. This is safer than directly inserting raw user input into SQL queries because special characters are escaped before execution.

However, prepared statements with bound parameters provide stronger protection because they separate SQL commands from user input. Ideally, all INSERT, UPDATE, DELETE, and SELECT statements should use prepared statements throughout the application.

Additional security improvements include validating numeric IDs using integer casting or `filter_input()`, validating email addresses with `filter_var()`, and escaping output with `htmlspecialchars()` before displaying data in the browser.

The application should also avoid disabling `FOREIGN_KEY_CHECKS` during normal operations, as this may weaken referential integrity. Admin authentication and access control should also be added before allowing CRUD operations.

```

1 // Current code uses escaping but still builds SQL by concatenating values
2 $action = $_POST['action'] ?? $_GET['action'] ?? "";
3
4 if ($action == "get_customer") {
5     $id = mysqli_real_escape_string($conn, $_GET['id']);
6     $result = mysqli_query($conn, "SELECT * FROM customer WHERE customerID='$id'");
7     header('Content-Type: application/json');
8     echo json_encode(mysqli_fetch_assoc($result));
9     exit;
10 }
11
12 if ($action == "save_customer") {
13     $email = mysqli_real_escape_string($conn, $_POST['Email']);
14     $sql = "INSERT INTO customer (FirstName, LastName, Email) VALUES ('$first', '$last', '$email')";
15     echo mysqli_query($conn, $sql) ? "success" : "error";
16 }

```

Figure 28: Current partially protected code

```

1 // Improved example: prepared statement separates SQL code from user input
2 $customerId = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);
3 if ($customerId === false || $customerId === null) {
4     die('Invalid customer ID');
5 }
6
7 $stmt = $conn->prepare("SELECT * FROM customer WHERE CustomerID = ?");
8 $stmt->bind_param("i", $customerId);
9 $stmt->execute();
10 $result = $stmt->get_result();
11 header('Content-Type: application/json');
12 echo json_encode($result->fetch_assoc());
13
14 // Output should also be escaped when displayed in HTML:
15 echo htmlspecialchars($row['FirstName'], ENT_QUOTES, 'UTF-8');

```

Figure 29: Improved prepared statement example

```

1 Injection test input entered into an ID/search field:
2 ' OR '1'='1
3
4 Unsafe pattern could produce:
5 SELECT * FROM customer WHERE CustomerID = '' OR '1'='1';
6
7 Expected secure result with validation/prepared statement:
8 Invalid customer ID
9 or
10 No matching record returned; input treated as plain data, not SQL code.

```

Figure 30: SQL injection test input and failed result

13. Evaluation and Reflection

13.1 Strengths of the Solution

The GreenLeaf Organics solution has several strengths because it covers the main business requirements of the online platform. The database includes important entities such as suppliers, products, categories, customers, orders, order details, reviews, and supplier reviews, supporting the core operations of the business.

Many-to-many relationships are managed correctly through junction tables such as ProductCategory and OrderDetail, reducing redundancy and supporting database normalisation. Sample data also provides realistic testing of the system functionality.

The PHP website demonstrates dynamic content display by connecting to MySQL and retrieving live records for display on web pages [18]. CRUD functionality allows records to be added, updated, and deleted interactively, while foreign keys maintain data integrity and indexes improve query performance as the database grows.

.

13.2 Limitations and Improvements

Although the solution meets most assignment requirements, several improvements could still be made. Stored procedures and triggers should be fully included in the final SQL file to provide clearer evidence of transaction handling, rollback logic, and auditing functionality.

The PHP application should also replace concatenated SQL queries with prepared statements to improve protection against SQL injection attacks. Additional security features, such as admin authentication and access control, should be implemented before allowing users to perform CRUD operations.

Search and filtering functionality could be demonstrated more clearly with additional screenshots showing how users can locate records efficiently. Another improvement would be avoiding the use of disabled foreign key checks during delete operations, as this may weaken referential integrity [18].

Finally, error handling could be improved by displaying clearer and more professional user messages, while audit tables and trigger execution should include stronger before-and-after evidence.

13.3 Personal Reflection

One of the most challenging parts of the project was translating the ERD into SQL tables while maintaining correct relationships between entities. It was necessary to determine where foreign keys were required and how many-to-many relationships should be resolved using junction tables such as ProductCategory and OrderDetail. This improved my understanding of relational database design and data integrity.

Another valuable learning experience was connecting PHP with MySQL to retrieve records dynamically and display them on web pages. Testing CRUD operations also helped confirm that records could be added, updated, and deleted correctly [18].

With more time, I would improve the application security by using prepared statements throughout the code, adding administrator authentication and stronger validation, and providing more detailed evidence for transactions, triggers, and SQL injection testing.

14. Conclusion

The proposed database and web solution supports GreenLeaf Organics in managing suppliers, products, categories, customers, orders, and reviews. The relational database design reflects the key business processes of the company while using normalised tables to reduce duplication and improve consistency.

Primary keys uniquely identify records, while foreign keys maintain referential integrity between related tables. Many-to-many relationships are managed through junction tables such as ProductCategory and OrderDetail.

The PHP web prototype demonstrates dynamic database connectivity and CRUD functionality for managing selected records. Overall, the solution provides a strong foundation for an online platform. Future improvements could include wider use of prepared statements, full implementation of stored procedures and triggers, improved search and filtering, better error handling, analytics dashboards, and role-based administration for enhanced security and system management.

15. References

- [1] Oracle, "MySQL 8.0 Reference Manual," MySQL Documentation, 2025. [Online]. Available: <https://dev.mysql.com/doc/refman/8.0/en/>. [Accessed: 23-April-2026].
- [2] PHP Documentation Group, "MySQLi - PHP Manual," PHP.net, 2025. [Online]. Available: <https://www.php.net/manual/en/book.mysqli.php>. [Accessed: 01-May-2026].
- [3] OWASP Foundation, "SQL Injection Prevention Cheat Sheet," OWASP, 2025. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. [Accessed: 01-May-2026].
- [4] Oracle, "MySQL Triggers," MySQL Documentation, 2025. [Online]. Available: <https://dev.mysql.com/doc/refman/8.0/en/triggers.html>. [Accessed: 01-May-2026].
- [5] Oracle, "Optimizing MySQL Queries with Indexes," MySQL Documentation, 2025. [Online]. Available: <https://dev.mysql.com/doc/refman/8.0/en/mysql-indexes.html>. [Accessed: 01-May-2026].
- [6] Oracle, "INSERT Statement," MySQL Documentation, 2025. [Online]. Available: <https://dev.mysql.com/doc/refman/8.0/en/insert.html>. [Accessed: 01-May-2026].
- [7] PHP Documentation Group, "PHP Manual," PHP.net, 2025. [Online]. Available: <https://www.php.net/manual/en/>. [Accessed: 01-May-2026].

[8] W3Schools, "SQL Stored Procedures," W3Schools, 2025. [Online]. Available: https://www.w3schools.com/sql/sql_stored_procedures.asp. [Accessed: 01-May-2026].

[9] W3Schools, "SQL Injection," W3Schools, 2025. [Online]. Available: https://www.w3schools.com/sql/sql_injection.asp. [Accessed: 01-May-2026].

[10] Simplified Dev, "PHP & MySQL CRUD Application Tutorial," YouTube, 2025. Accessed: Apr. 2026. [Online Video]. Available: <https://www.youtube.com/watch?v=vAzV7Hyge-U>

[11] MySQL, "Episode-083 - Using Triggers in MySQL," YouTube, Feb. 10, 2025. Accessed: Apr. 2026. [Online Video]. Available: <https://www.youtube.com/watch?v=3B-1odrGSB4>

[12] Funda Coder, "PHP PDO CRUD Tutorial - How to Insert, Fetch, Edit & Update, Delete data using PDO in PHP," YouTube, Jan. 15, 2022. Accessed: Apr. 2026. [Online Video]. Available: <https://www.youtube.com/watch?v=toU9pJeM2e0>