

ASSIGNMENT TITLE: DATABASE DESIGN & IMPLEMENTATION

Programme: BSc/HND Computing Portfolio
Module: Databases and Application Development
Module code: ACCA5026

STUDENT: PIER PAOLO PENNE
STUDENT ID: 2336786
LECTURER: FAHAD AHMED



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Table of Contents

Table of Figures	3
1. Introduction	4
PART A	5
2. Database Design	5
2.1 Entity Identification & Attributes	5
2.2 Entity Relationship Diagram (ERD)	6
2.3 Normalisation to Third Normal Form (3NF)	6
3. Database Implementation	8
3.1 Table Creation & Constraints	8
3.2 Sample Data Population	10
3.3 Stored Procedures & CRUD Operations	12
3.4 Triggers & Audit Functions	13
3.5 Transactional Scenario - Full Order Placement	14
3.6 Performance Tuning & Query Optimisation / Indexing	18
4. Literature Support for Design & Implementation Choices	22
5. Conclusion/Reflection	24
6. References	26
PART B	27
1. SQL Injection: Overview and Example	27
2. Where the Website Code is Susceptible to SQL Injection	28
3. Anti-SQL Injection Defences Implemented	29
4. Why Those Defences Are Effective?	34
5. Main Specification Website.....	35
6. Reference	43

Table of Figures

Figure 1 - Query1	8
Figure 2 - Query2	8
Figure 3 - Query3	8
Figure 4 - Query4	9
Figure 5 - Query5	9
Figure 6 - Query6	9
Figure 7 - Query7	9
Figure 8 - Query8	10
Figure 9 - Query9	10
Figure 10 - Query10	10
Figure 11 - Query11	10
Figure 12 - Query12	11
Figure 13 - Query13	11
Figure 14 - Query14	11
Figure 15 - Query15	11
Figure 16 - Query16	11
Figure 17 - Query17	12
Figure 18 - Query18	12
Figure 19 - Query19	12
Figure 20 - Query20	12
Figure 21 - Query21	13
Figure 22 - Query22	13
Figure 23 - Query23	13
Figure 24 - Query24	13
Figure 25 - Query25	14
Figure 26 - Query26	14
Figure 27 - Query27.1	15
Figure 28 - Query27.2	15
Figure 29 - Query27.3	16
Figure 30 - Query27.4	16
Figure 31 - Query28	18
Figure 32 - Query29.1	19
Figure 33 - Query29.2	19

Figure 34 - Query30	20
Figure 35 - Query31	21
Figure 36 - Authentication bypass / SQL Injection	27
Figure 37 - Data Theft via UNION / SQL Injection	28
Figure 38 - Destructive DROP TABLE / SQL Injection	28
Figure 39 - Vulnerable code pattern / SQL Injection	29
Figure 40 - Primary Defence: PDO Prepared Statements / SQL Injection	30
Figure 41 - Defence in Action: Login Page / SQL Injection	30
Figure 42 - Defence in Action: Search Page / SQL Injection	31
Figure 43 - Defence in Action: Integer Casting on ID Parameters / SQL Injection	32
Figure 44 - CSC Providers	35
Figure 45 - CSC Add Providers	36
Figure 46 - CSC Categories	36
Figure 47 - CSC Services	37
Figure 48 - Connection Code	38
Figure 49 - CSC	39
Figure 50 - Browse CSC	39
Figure 51 - Service Detail.....	40
Figure 52 - CSC admin-login	41
Figure 53 - CSC dashboard	42

1. Introduction

Greenleaf Organics is a UK-based sustainable enterprise specialising in certified organic food, eco-friendly personal care items, and biodegradable household goods. As the company expands, it requires a scalable digital infrastructure to support core business processes from supplier onboarding to customer order fulfilment and post-purchase engagement.

This report presents a complete database consultancy deliverable, commissioned to design and prototype a relational database management system (RDBMS) underpinning Greenleaf Organics' new online platform. The database is built in MySQL and structured around five business domains: supplier and product management, customer registration, order processing, inventory tracking, and customer feedback.

The methodology follows a structured sequence: entity identification, ERD construction, normalisation to 3NF, physical schema creation, data population, stored procedure implementation with COMMIT/ROLLBACK logic, trigger-based auditing, and performance tuning via secondary indexing. According to Elmasri and Navathe [3], a well-designed relational database reduces data anomalies, supports complex queries, and ensures ACID compliance, critical for e-commerce platforms handling financial transactions and sensitive customer data.

PART A

2. Database Design

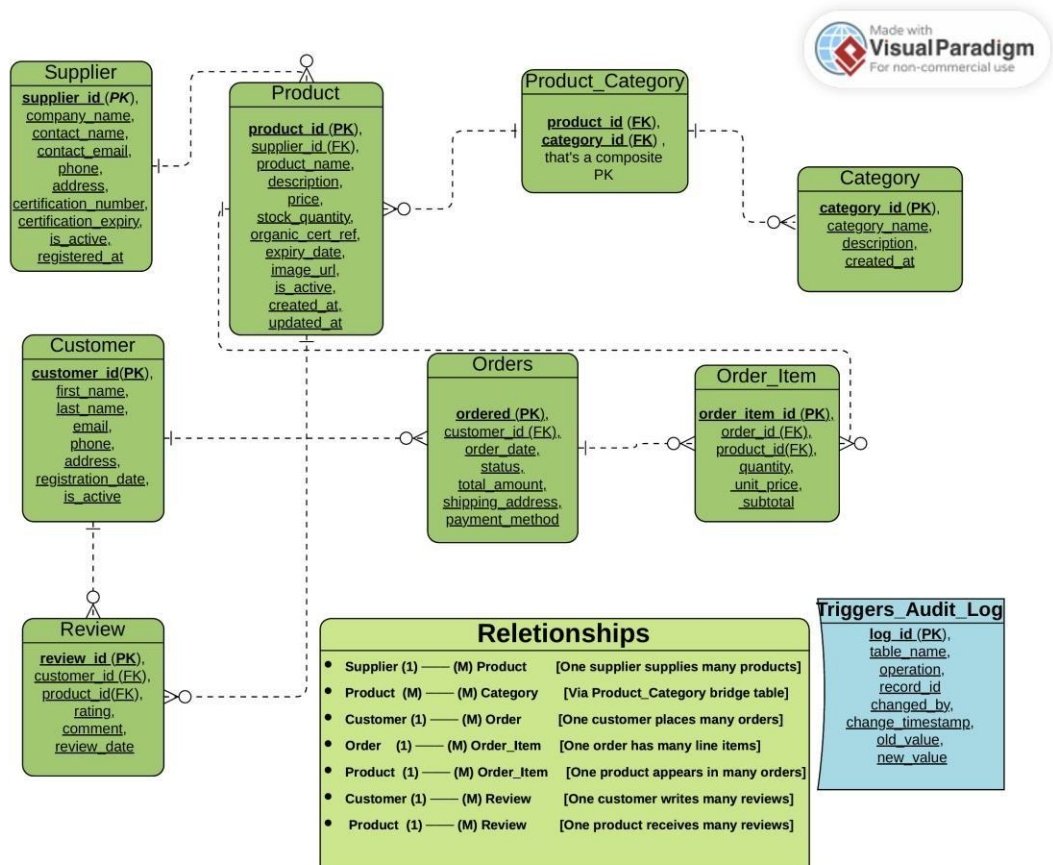
2.1 Entity Identification & Attributes

The following entities were identified from the Greenleaf Organics business process overview. Each entity represents a distinct object type with its own attributes and lifecycle within the system.

Entity	Key Attributes	Description
Supplier	supplier_id (PK), company_name, contact_email, phone, address, certification_number, certification_expiry, is_active	Certified organic producers who register and supply products to Greenleaf.
Product	product_id (PK), supplier_id (FK), product_name, description, price, stock_quantity, organic_cert_ref, expiry_date, image_url, is_active	Individual items listed by suppliers. Belongs to one supplier but may belong to many categories.
Category	category_id (PK), category_name, description	Classification tags such as Organic Vegetables, Natural Skincare, and Eco Household.
Product_Category	product_id (FK), category_id (FK) composite PK	Junction table resolving the many-to-many relationship between Products and Categories.
Customer	customer_id (PK), first_name, last_name, email, phone, address, registration_date, is_active	Individuals or businesses who purchase products. Can place multiple orders.
Order	order_id (PK), customer_id (FK), order_date, status, total_amount, shipping_address, payment_method	A purchase transaction linked to one customer, containing multiple products.
Order_Item	order_item_id (PK), order_id (FK), product_id (FK), quantity, unit_price, subtotal	Line items within an order. Resolves the many-to-many relationship between Orders and Products.
Review	review_id (PK), customer_id (FK), product_id (FK), rating, comment, review_date	Customer feedback on individual products with a 1-5 star rating.
Audit_Log	log_id (PK), table_name, operation, changed_by, change_timestamp, old_value, new_value	System table populated automatically by triggers to track data changes.

2.2 Entity Relationship Diagram (ERD)

The ERD below presents the full relational schema for the Greenleaf Organics platform. Visual Paradigm has been used to produce the diagram. All primary key and foreign key relationships are represented, and cardinality notation indicates the nature of each association; including the one-to-many relationships between Supplier and Product, between Customer and Order, and between Order and Order_Item, as well as the many-to-many relationship between Product and Category resolved through the junction table.



2.3 Normalisation to Third Normal Form (3NF)

Normalisation is the process of structuring a relational database to minimise redundancy and prevent update anomalies. The Greenleaf schema has been normalised through three progressive forms as described by Date [2] and Ramakrishnan & Gehrke [4].

First Normal Form (1NF)

All attributes contain atomic values with no repeating groups. Product categories are not stored as comma-separated lists; instead, a dedicated Product_Category junction table is used. Every row is uniquely identifiable by its primary key.

Second Normal Form (2NF)

All non-key attributes are fully dependent on the entire primary key. In Order_Item, unit_price and quantity depend on the full composite key (order_id, product_id), not either column alone. Product_Category contains only its two foreign key columns, avoiding partial dependencies.

Third Normal Form (3NF)

No non-key attribute is transitively dependent on the primary key. Customer address details reside only in the Customer table; the Order table stores a shipping_address snapshot for historical accuracy. Supplier certification details remain in the Supplier table and are not repeated in Product rows. Review ratings are stored per record and aggregated at query time using AVG() to avoid stale data.

Normal Form	Rule Applied	Greenleaf Design Decision
1NF	Atomic values, no repeating groups, unique rows	Product_Category junction table; all primary keys defined
2NF	No partial dependencies on composite primary keys	Order_Item attributes depend on the full composite key
3NF	No transitive dependencies	Supplier data not repeated in Product; customer details not duplicated in Order

3. Database Implementation

3.1 Table Creation & Constraints

The following SQL DDL statements create the Greenleaf Organics schema in MySQL using Uniform Server Zero and phpMyAdmin. All tables include appropriate primary keys, foreign keys with cascading rules, NOT NULL constraints, CHECK constraints for data validation, and DEFAULT values where appropriate.

3.1.1 Database Initialisation

Creation of the greenleaf_organics database, setting FOREIGN_KEY_CHECKS = 1 and appropriate SQL_MODE to enforce strict data validation throughout the schema.

```

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0112 seconds.)

CREATE DATABASE IF NOT EXISTS greenleaf_organics CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

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

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

USE greenleaf_organics;

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

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0001 seconds.)

SET FOREIGN_KEY_CHECKS = 1;

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

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0001 seconds.)

SET sql_mode = 'STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,ERROR_FOR_DIVISION_BY_ZERO';

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

⚠ Warning: #3135 'NO_ZERO_DATE', 'NO_ZERO_IN_DATE' and 'ERROR_FOR_DIVISION_BY_ZERO' sql modes should be used with strict mode. They will be merged with strict mode in a future release.

```

Figure 1 - Query1

Figure 1

3.1.2 Supplier Table

Stores registered organic producers with a unique constraint on contact_email, a CHECK constraint on certification_expiry, and an is_active flag for soft deletion.

```

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

CREATE TABLE Supplier ( supplier_id INT NOT NULL AUTO_INCREMENT, company_name VARCHAR(150) NOT NULL, contact_name VARCHAR(100) NOT NULL, contact_email VARCHAR(150) NOT NULL UNIQUE, phone VARCHAR(20), address TEXT NOT NULL, country VARCHAR(60) NOT NULL DEFAULT 'United Kingdom', certification_number VARCHAR(60) NOT NULL UNIQUE, certification_expiry DATE NOT NULL, is_active TINYINT(1) NOT NULL DEFAULT 1, registered_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, CONSTRAINT pk_supplier PRIMARY KEY (supplier_id) ) ENGINE=InnoDB;

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

⚠ Warning: #1681 Integer display width is deprecated and will be removed in a future release.

```

Figure 2 - Query2

3.1.3 Category Table

Uses a surrogate primary key and a unique constraint on category_name to prevent duplicates. Supports many-to-many product categorisation via the junction table.

```

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

CREATE TABLE category ( category_id INT NOT NULL AUTO_INCREMENT, category_name VARCHAR(100) NOT NULL UNIQUE, description TEXT, created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, CONSTRAINT pk_category PRIMARY KEY (category_id) ) ENGINE=InnoDB;

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

```

Figure 3 - Query3

3.1.4 Product Table

Includes a foreign key to Supplier with ON DELETE RESTRICT, CHECK constraints ensuring price > 0 and stock_quantity >= 0, and an expiry_date column for freshness management.

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

CREATE TABLE Product ( product_id INT NOT NULL AUTO_INCREMENT, supplier_id INT NOT NULL, product_name VARCHAR(200) NOT NULL, description TEXT, price DECIMAL(10,2) NOT NULL, stock_quantity INT NOT NULL DEFAULT 0, organic_cert_ref VARCHAR(80), expiry_date DATE, image_url VARCHAR(255), is_active TINYINT(1) NOT NULL DEFAULT 1, created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP, CONSTRAINT pk_product PRIMARY KEY (product_id), CONSTRAINT fk_product_supplier FOREIGN KEY (supplier_id) REFERENCES Supplier(supplier_id) ON DELETE RESTRICT ON UPDATE CASCADE, CONSTRAINT chk_price CHECK (price > 0), CONSTRAINT chk_stock CHECK (stock_quantity >= 0) ) ENGINE=InnoDB;
```

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

Warning: #1681 Integer display width is deprecated and will be removed in a future release.

Figure 4 - Query4

```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

CREATE TABLE Product_Category ( product_id INT NOT NULL, category_id INT NOT NULL, CONSTRAINT pk_product_category PRIMARY KEY (product_id, category_id), CONSTRAINT fk_pc_product FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT fk_pc_category FOREIGN KEY (category_id) REFERENCES Category(category_id) ON DELETE CASCADE ON UPDATE CASCADE ) ENGINE=InnoDB;
```

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

3.1.5 Product_Category Junction Table

Uses a composite primary key of (product_id, category_id) with ON DELETE CASCADE on both foreign keys, automatically cleaning up junction records when a product or category is removed.

Figure 5 - Query5

3.1.6 Customer Table

Enforces a unique constraint on email, defaults registration_date to CURRENT_TIMESTAMP, and uses is_active for account deactivation without permanent deletion.

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

CREATE TABLE Customer ( customer_id INT NOT NULL AUTO_INCREMENT, first_name VARCHAR(80) NOT NULL, last_name VARCHAR(80) NOT NULL, email VARCHAR(150) NOT NULL UNIQUE, phone VARCHAR(20), address TEXT NOT NULL, registration_date DATE NOT NULL DEFAULT CURDATE(), is_active TINYINT(1) NOT NULL DEFAULT 1, CONSTRAINT pk_customer PRIMARY KEY (customer_id) ) ENGINE=InnoDB;
```

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

Warning: #1681 Integer display width is deprecated and will be removed in a future release.

Figure 6 - Query6

3.1.7 Order and Order_Item Tables

Orders references Customer via foreign key with a status column constrained to an enumerated set: pending, processing, dispatched, delivered, and cancelled. Order_Item uses a composite primary key with a GENERATED ALWAYS AS STORED column for subtotal.

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

CREATE TABLE orders ( order_id INT NOT NULL AUTO_INCREMENT, customer_id INT NOT NULL, order_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, status
ENUM('pending','processing','dispatched','delivered','cancelled') NOT NULL DEFAULT 'pending', total_amount DECIMAL(10,2) NOT NULL DEFAULT 0.00, shipping_address TEXT NOT
NULL, payment_method VARCHAR(50) NOT NULL DEFAULT 'card', CONSTRAINT pk_order PRIMARY KEY (order_id), CONSTRAINT fk_order_customer FOREIGN KEY (customer_id) REFERENCES
Customer(customer_id) ON DELETE RESTRICT ON UPDATE CASCADE ) ENGINE=InnoDB;

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

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

CREATE TABLE Order_Item ( order_item_id INT NOT NULL AUTO_INCREMENT, order_id INT NOT NULL, product_id INT NOT NULL, quantity INT NOT NULL, unit_price DECIMAL(10,2) NOT
NULL, subtotal DECIMAL(10,2) GENERATED ALWAYS AS (quantity * unit_price) STORED, CONSTRAINT pk_order_item PRIMARY KEY (order_item_id), CONSTRAINT fk_oi_order FOREIGN KEY
(order_id) REFERENCES orders(order_id) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT fk_oi_product FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE
RESTRICT ON UPDATE CASCADE, CONSTRAINT chk_qty CHECK (quantity > 0), CONSTRAINT chk_unitprice CHECK (unit_price > 0) ) ENGINE=InnoDB;

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

Figure 7 - Query7

3.1.8 Review Table

Includes a UNIQUE constraint on (customer_id, product_id) preventing duplicate reviews, and a CHECK on rating between 1 and 5.

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

CREATE TABLE Review ( review_id INT NOT NULL AUTO_INCREMENT, customer_id INT NOT NULL, product_id INT NOT NULL, rating TINYINT NOT NULL, comment TEXT, review_date DATETIME
NOT NULL DEFAULT CURRENT_TIMESTAMP, CONSTRAINT pk_review PRIMARY KEY (review_id), CONSTRAINT fk_review_customer FOREIGN KEY (customer_id) REFERENCES Customer(customer_id) ON
DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT fk_review_product FOREIGN KEY (product_id) REFERENCES Product(product_id) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT
chk_rating CHECK (rating BETWEEN 1 AND 5), CONSTRAINT uq_review UNIQUE (customer_id, product_id) ) ENGINE=InnoDB;

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

Figure 8 - Query8

3.1.9 Triggers_Audit_Log Table

A system-managed table populated exclusively by triggers. Records the table name, operation type, database user, timestamp, and JSON-formatted old and new values.

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

CREATE TABLE Audit_Log ( log_id INT NOT NULL AUTO_INCREMENT, table_name VARCHAR(60) NOT NULL, operation ENUM('INSERT','UPDATE','DELETE') NOT NULL, record_id INT, changed_by
VARCHAR(100), change_timestamp DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, old_value JSON, new_value JSON, CONSTRAINT pk_audit PRIMARY KEY (log_id) ) ENGINE=InnoDB;

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

Figure 9 - Query9

A complete audit trail for the Product table was created using three AFTER triggers that log every INSERT, UPDATE, and DELETE to Audit_Log.

```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0190 seconds.)

CREATE TRIGGER trg_product_audit_insert AFTER INSERT ON Product FOR EACH ROW BEGIN INSERT INTO Audit_Log ( table_name, operation, record_id, changed_by, old_value, new_value ) VALUES ( 'Product', 'INSERT', NEW.product_id, COALESCE(@logged_user, USER()), NULL, JSON_OBJECT( 'product_id', NEW.product_id, 'supplier_id', NEW.supplier_id, 'product_name', NEW.product_name, 'description', NEW.description, 'price', NEW.price, 'stock_quantity', NEW.stock_quantity, 'organic_cert_ref', NEW.organic_cert_ref, 'expiry_date', NEW.expiry_date, 'image_url', NEW.image_url, 'is_active', NEW.is_active, 'created_at', NEW.created_at, 'updated_at', NEW.updated_at ) ); END;

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

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0209 seconds.)

CREATE TRIGGER trg_product_audit_update AFTER UPDATE ON Product FOR EACH ROW BEGIN INSERT INTO Audit_Log ( table_name, operation, record_id, changed_by, old_value, new_value ) VALUES ( 'Product', 'UPDATE', NEW.product_id, COALESCE(@logged_user, USER()), JSON_OBJECT( 'product_id', OLD.product_id, 'supplier_id', OLD.supplier_id, 'product_name', OLD.product_name, 'description', OLD.description, 'price', OLD.price, 'stock_quantity', OLD.stock_quantity, 'organic_cert_ref', OLD.organic_cert_ref, 'expiry_date', OLD.expiry_date, 'image_url', OLD.image_url, 'is_active', OLD.is_active, 'created_at', OLD.created_at, 'updated_at', OLD.updated_at ), JSON_OBJECT( 'product_id', NEW.product_id, 'supplier_id', NEW.supplier_id, 'product_name', NEW.product_name, 'description', NEW.description, 'price', NEW.price[...]) ); END;

[ Edit ]

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0157 seconds.)

CREATE TRIGGER trg_product_audit_delete AFTER DELETE ON Product FOR EACH ROW BEGIN INSERT INTO Audit_Log ( table_name, operation, record_id, changed_by, old_value, new_value ) VALUES ( 'Product', 'DELETE', OLD.product_id, COALESCE(@logged_user, USER()), JSON_OBJECT( 'product_id', OLD.product_id, 'supplier_id', OLD.supplier_id, 'product_name', OLD.product_name, 'description', OLD.description, 'price', OLD.price, 'stock_quantity', OLD.stock_quantity, 'organic_cert_ref', OLD.organic_cert_ref, 'expiry_date', OLD.expiry_date, 'image_url', OLD.image_url, 'is_active', OLD.is_active, 'created_at', OLD.created_at, 'updated_at', OLD.updated_at ), NULL ); END;

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

Figure 10 - Query10

3.2 Sample Data Population

Realistic sample data is inserted to simulate real-world platform operation, test referential integrity, and validate stored procedure and trigger behaviour.

3.2.1 Supplier Data

Five supplier records were inserted with company name, contact details, address, certification number, and expiry date.

```
5 rows inserted.
Inserted row id: 5 (Query took 0.0005 seconds.)

INSERT INTO Supplier (company_name, contact_name, contact_email, phone, address, certification_number, certification_expiry) VALUES ('Green Valley Farms', 'Alice Green', 'alice@greenvalley.co.uk', '07700900001', '12 Farm Lane, Devon, EX1 1AA', 'ORG-UK-2021-001', '2027-06-30'), ('Pure Earth Produce', 'Ben Carter', 'ben@pureearth.co.uk', '07700900002', '5 Meadow Road, Somerset, TA1 2BB', 'ORG-UK-2020-002', '2026-12-31'), ('EcoHome Supplies', 'Clara Mills', 'clara@ecohome.co.uk', '07700900003', '88 Green St, London, EC1A 1BB', 'ORG-UK-2022-003', '2027-03-31'), ('Natura Skincare', 'Daniel Osei', 'daniel@naturaskin.co.uk', '07700900004', '3 Oak Avenue, Edinburgh, EH1 3CD', 'ORG-UK-2021-004', '2026-09-30'), ('Wholesome Harvest', 'Emma Walsh', 'emma@wholesome.co.uk', '07700900005', '22 River View, Bristol, BS1 5EF', 'ORG-UK-2023-005', '2028-01-31');
```

Figure 11 - Query11

3.2.2 Category & Product Data

Five category rows were inserted, followed by ten products linked to suppliers with names, descriptions, prices, stock quantities, and expiry dates.

```
5 rows inserted.
Inserted row id: 5 (Query took 0.0005 seconds.)

INSERT INTO Category (category_name, description) VALUES ('Organic Vegetables', 'Fresh certified organic vegetables sourced from UK farms'), ('Organic Fruit', 'Seasonal organic fruit from certified growers'), ('Natural Skincare', 'Organic, cruelty-free skincare products'), ('Eco Household', 'Biodegradable and sustainable household products'), ('Pantry Essentials', 'Organic oils, grains, pulses and staple foods');
```

Figure 12 - Query12

```
✓ 10 rows inserted.
Inserted row id: 10 (Query took 0.0005 seconds.)

INSERT INTO Product (supplier_id, product_name, description, price, stock_quantity, organic_cert_ref, expiry_date) VALUES (1, 'Organic Broccoli (500g)', 'Freshly harvested certified organic broccoli', 1.99, 200, 'GVF-VEG-001', '2026-04-15'), (1, 'Heritage Carrots (1kg)', 'Mixed heritage variety organic carrots', 2.49, 180, 'GVF-VEG-002', '2026-04-20'), (2, 'Organic Strawberries (400g)', 'Sweet, hand-picked organic strawberries', 3.49, 120, 'PEP-FRT-001', '2026-04-10'), (2, 'Cox Apples (6 pack)', 'Traditional English organic Cox apples', 2.99, 150, 'PEP-FRT-002', '2026-04-25'), (3, 'Bamboo Washing-Up Brush', 'Plastic-free bamboo dish brush with plant-fibre bristles', 4.99, 300, 'ECO-HH-001', NULL), (3, 'Compostable Bin Liners (30pk)', 'Fully compostable kitchen bin liners', 5.99, 250, 'ECO-HH-002', NULL), (4, 'Organic Rose Face Oil (30ml)', 'Cold-pressed organic rosehip & argan face oil', 18.99, 80, 'NS-SK-001', '2027-12-31'), (4, 'Lavender Body Lotion...)
```

Figure 13 - Query13

3.2.3 Customer & Orders Data

Five customer records were inserted, followed by five orders across a range of statuses, and eleven order items linking orders to products with quantities and unit prices.

```
✓ 5 rows inserted.
Inserted row id: 5 (Query took 0.0005 seconds.)

INSERT INTO Customer (first_name, last_name, email, phone, address) VALUES ('Sophie', 'Turner', 'sophie.turner@email.co.uk', '07811100001', '14 Blossom Way, Oxford, OX1 2QR'), ('James', 'Patel', 'james.patel@email.co.uk', '07811100002', '7 Cedar Close, Manchester, M1 3WP'), ('Amara', 'Okonkwo', 'amara.ok@email.co.uk', '07811100003', '31 Willow Lane, Bristol, BS2 8TA'), ('Lucas', 'Schmidt', 'lucas.schmidt@email.co.uk', '07811100004', '9 Maple Street, Leeds, LS1 5PQ'), ('Priya', 'Mehta', 'priya.mehta@email.co.uk', '07811100005', '2 Orchard Road, Cambridge, CB1 1AJ');
```

Figure 14 - Query14

```
✓ 5 rows inserted.
Inserted row id: 5 (Query took 0.0005 seconds.)

INSERT INTO `Orders` (customer_id, status, total_amount, shipping_address) VALUES (1, 'delivered', 9.96, '14 Blossom Way, Oxford, OX1 2QR'), (2, 'processing', 31.98, '7 Cedar Close, Manchester, M1 3WP'), (3, 'pending', 15.97, '31 Willow Lane, Bristol, BS2 8TA'), (1, 'dispatched', 34.33, '14 Blossom Way, Oxford, OX1 2QR'), (4, 'delivered', 5.99, '9 Maple Street, Leeds, LS1 5PQ');
```

Figure 15 - Query15

```
✓ 11 rows inserted.
Inserted row id: 11 (Query took 0.0005 seconds.)

INSERT INTO Order_Item (order_id, product_id, quantity, unit_price) VALUES (1, 1, 2, 1.99), (1, 2, 1, 2.49), (1, 3, 1, 3.49), (2, 7, 1, 18.99), (2, 8, 1, 12.99), (3, 5, 2, 4.99), (3, 6, 1, 5.99), (4, 9, 2, 9.99), (4, 10, 3, 2.79), (4, 4, 2, 2.99), (5, 6, 1, 5.99);
```

Figure 16 - Query16

3.2.4 Review Data

Customer reviews were inserted with ratings between 1 and 5 and textual comments. The unique constraint on (customer_id, product_id) was validated during insertion.

```
✓ 6 rows inserted.
Inserted row id: 6 (Query took 0.0006 seconds.)

INSERT INTO Review (customer_id, product_id, rating, comment) VALUES (1, 1, 5, 'Absolutely fresh and full of flavour, best broccoli I have bought online!'), (1, 3, 4, 'Lovely strawberries, arrived quickly and perfectly ripe.'), (2, 7, 5, 'Amazing face oil, skin feels transformed after one week of use.'), (3, 5, 4, 'Great bamboo brush, sturdy and eco-friendly. Delivery was fast.'), (4, 6, 3, 'Bin liners are decent but a little small for my bin. Otherwise good.'), (5, 9, 5, 'Rich, flavoursome olive oil. Will definitely be ordering again!');
```

Figure 17 - Query17

3.3 Stored Procedures & CRUD Operations

The following stored procedures implement full CRUD (Create, Read, Update, Delete) operations with embedded transaction handling. Each procedure uses BEGIN ... COMMIT / ROLLBACK to ensure atomicity. A DECLARE ... HANDLER for SQLEXCEPTION captures any error and triggers a rollback, preserving data integrity [5].

3.3.1 INSERT - Place New Order (Transactional)

Creates a new order for the specified customer, captures the generated order_id via LAST_INSERT_ID(), and returns a success message. If any step fails the SQLEXCEPTION handler triggers an immediate ROLLBACK.

```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0210 seconds.)

CREATE PROCEDURE sp_place_order( IN p_customer_id INT, IN p_shipping_addr TEXT, IN p_payment_meth VARCHAR(50), OUT p_order_id INT, OUT p_message VARCHAR(200) ) BEGIN DECLARE
EXIT HANDLER FOR SQLEXCEPTION BEGIN ROLLBACK; SET p_order_id = -1; SET p_message = 'Order failed: transaction rolled back.'; END; START TRANSACTION; INSERT INTO `Order`
(customer_id, shipping_address, payment_method) VALUES (p_customer_id, p_shipping_addr, p_payment_meth); SET p_order_id = LAST_INSERT_ID(); COMMIT; SET p_message =
CONCAT('Order ', p_order_id, ' created successfully.');
```

Figure 18 - Query18

A secondary update step recalculates each order's total_amount by joining with aggregated subtotals from Order_Item, ensuring the stored total reflects all line items.

```
2 rows affected. (Query took 0.0005 seconds.)

UPDATE `Orders` o JOIN ( SELECT order_id, SUM(subtotal) AS total FROM Order_Item WHERE order_id = product_id GROUP BY order_id ) s ON o.order_id = s.order_id SET o.total_amount
= s.total;
```

Figure 19 - Query19

3.3.2 INSERT - Add Order Item with Stock Check

Adds an order item only when sufficient stock is available. A SELECT ... FOR UPDATE lock prevents concurrent transactions from both reading the same stock level and proceeding simultaneously. If stock is adequate the item is inserted and stock_quantity decremented atomically; otherwise, a ROLLBACK and error message are returned.

```
MySQL returned an empty result set (i.e. zero rows). (Query took 0.0119 seconds.)

CREATE PROCEDURE sp_add_order_item( IN p_order_id INT, IN p_product_id INT, IN p_quantity INT, OUT p_success TINYINT, OUT p_message VARCHAR(200) ) BEGIN DECLARE v_stock INT
DEFAULT 0; DECLARE v_price DECIMAL(10,2); DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN ROLLBACK; SET p_success = 0; SET p_message = 'Item insert failed.'; END; START
TRANSACTION; SELECT stock_quantity, price INTO v_stock, v_price FROM Product WHERE product_id = p_product_id FOR UPDATE; IF v_stock < p_quantity THEN ROLLBACK; SET p_success =
0; SET p_message = CONCAT('Insufficient stock for product ', p_product_id); ELSE INSERT INTO Order_Item (order_id, product_id, quantity, unit_price) VALUES (p_order_id,
p_product_id, p_quantity, v_price); UPDATE Product SET stock_quantity = stock_quantity - p_quantity WHERE product_id = p_product_id; COMMIT; SET p_success = 1; SET p_message =
'Item added...'
```

Figure 20 - Query20

3.3.3 UPDATE - Update Product Details

Validates price and stock values before issuing an UPDATE. If both conditions are met the transaction commits and returns a success message; otherwise, it rolls back to protect existing data.

```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0219 seconds.)

CREATE PROCEDURE sp_update_product( IN p_product_id INT, IN p_price DECIMAL(10,2), IN p_stock INT, IN p_is_active TINYINT, OUT p_message VARCHAR(200) ) BEGIN DECLARE
v_rows_affected INT DEFAULT 0; DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN ROLLBACK; SET p_message = 'Product update failed: rolled back.'; END; START TRANSACTION; IF p_price <=
0 OR p_stock < 0 THEN ROLLBACK; SET p_message = 'Invalid price or stock value. Update rejected.'; ELSE UPDATE Product SET price = p_price, stock_quantity = p_stock, is_active =
p_is_active WHERE product_id = p_product_id; SET v_rows_affected = ROW_COUNT(); IF v_rows_affected = 0 THEN ROLLBACK; SET p_message = CONCAT('No product found with ID ',
p_product_id); ELSE COMMIT; SET p_message = CONCAT('Product ', p_product_id, ' updated successfully.'); END IF; END IF; END;

```

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

Figure 21 - Query21

3.3.4 DELETE - Cancel Order with Stock Restoration

Verifies the current order status before cancellation. If the order has not been dispatched or delivered, product stock is restored for each Order_Item, the order status is set to cancelled, and the transaction commits. Orders already dispatched or delivered are rejected.

```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0162 seconds.)

CREATE PROCEDURE sp_cancel_order( IN p_order_id INT, OUT p_message VARCHAR(200) ) BEGIN DECLARE v_status VARCHAR(20); DECLARE EXIT HANDLER FOR SQLEXCEPTION BEGIN ROLLBACK; SET
p_message = 'Cancellation failed: rolled back.'; END; START TRANSACTION; SELECT status INTO v_status FROM `Order` WHERE order_id = p_order_id FOR UPDATE; IF v_status IN
('dispatched','delivered') THEN ROLLBACK; SET p_message = 'Cannot cancel: order already dispatched or delivered.'; ELSE -- Restore stock for each item UPDATE Product p JOIN
Order_Item oi ON p.product_id = oi.product_id SET p.stock_quantity = p.stock_quantity + oi.quantity WHERE oi.order_id = p_order_id; UPDATE `Order` SET status = 'cancelled' WHERE
order_id = p_order_id; COMMIT; SET p_message = CONCAT('Order ', p_order_id, ' cancelled. Stock restored.'); END IF; END;

```

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

Figure 22 - Query22

3.4 Triggers & Audit Functions

Triggers provide automatic, event-driven auditing of data changes without relying on application-layer code. This implements the separation of concerns principle and guarantees audit completeness even when data is modified through direct SQL access or administrative tooling [1].

3.4.1 Product Price Change Audit Trigger

An AFTER-UPDATE trigger on Product logs to Audit_Log whenever price changes, storing the table name, operation type, product ID, database user, and old/new values for price and stock_quantity as JSON.

```

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0226 seconds.)

CREATE TRIGGER trg_product_price_audit AFTER UPDATE ON Product FOR EACH ROW BEGIN IF OLD.price <> NEW.price THEN INSERT INTO Audit_Log ( table_name, operation, record_id,
changed_by, old_value, new_value ) VALUES ( 'Product', 'UPDATE', NEW.product_id, COALESCE(@logged_user, USER()), JSON_OBJECT('price', OLD.price, 'stock_quantity',
OLD.stock_quantity), JSON_OBJECT('price', NEW.price, 'stock_quantity', NEW.stock_quantity) ); END IF; END;

```

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

Figure 23 - Query23

3.4.2 New Supplier Audit Trigger

An AFTER-INSERT trigger on Supplier logs every new registration to Audit_Log, capturing the supplier ID, database user, and full inserted record serialised via JSON_OBJECT().

```
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0158 seconds.)

CREATE TRIGGER trg_supplier_insert_audit AFTER INSERT ON Supplier FOR EACH ROW BEGIN INSERT INTO Audit_Log ( table_name, operation, record_id, changed_by, new_value ) VALUES (
'Supplier', 'INSERT', NEW.supplier_id, USER(), JSON_OBJECT( 'company_name', NEW.company_name, 'contact_email', NEW.contact_email, 'certification_number',
NEW.certification_number, 'certification_expiry', NEW.certification_expiry ) ); END;
```

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

Figure 24 - Query24

3.4.3 Order Status Change Audit Trigger

An AFTER-UPDATE trigger on Orders records status transitions, capturing old and new status and total_amount values along with the database user and order_id.

```
CREATE TRIGGER trg_order_status_audit AFTER UPDATE ON Orders FOR EACH ROW BEGIN IF OLD.status <> NEW.status OR OLD.total_amount <>
NEW.total_amount THEN INSERT INTO Audit_Log ( table_name, operation, record_id, changed_by, old_value, new_value ) VALUES ( 'Orders', 'UPDATE',
NEW.order_id, USER(), CONCAT('status=', OLD.status, ', total_amount=', OLD.total_amount), CONCAT('status=', NEW.status, ', total_amount=',
NEW.total_amount) ); END IF; END;
```

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

Figure 25 - Query25

3.4.4 Stored Function - Average Product Rating

The fn_avg_product_rating function accepts a product_id and returns the average rating from Review records. If no reviews exist it returns 0.00 rather than NULL.

Figure 26 - Query26

```
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0118 seconds.)

CREATE FUNCTION fn_avg_product_rating(p_product_id INT) RETURNS DECIMAL(3,2) READS SQL DATA DETERMINISTIC BEGIN DECLARE v_avg DECIMAL(3,2); SELECT COALESCE(AVG(rating), 0.00)
INTO v_avg FROM Review WHERE product_id = p_product_id; RETURN v_avg; END;
```

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

3.5 Transactional Scenario - Full Order Placement

The following scenario demonstrates a complete multi-step transactional workflow, simulating Customer #5 (Piero Penne) purchasing two products. If any step fails; for example due to insufficient stock or a constraint violation thwn all changes are rolled back.

The four-step workflow covers: creating a new order via sp_place_order (customer ID 5, delivery address, payment method 'card'); adding Olive Oil (product 9, quantity 1); adding Broccoli (product 1, quantity 3); and querying the completed order to verify status, total_amount, and all line items.

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0017 seconds.)

```
-- Step 1: Initiate the order CALL sp_place_order(5, '2 Orchard Road, Cambridge, CB1 1AJ', 'card', @order_id, @msg);
```

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

⚠ Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available. ⓘ

✓ Showing rows 0 - 0 (1 total, Query took 0.0002 seconds.)

```
SELECT @order_id AS new_order_id, @msg AS message;
```

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

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

Extra options

new_order_id	message
-1	Order failed: transaction rolled back.

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

Query results operations

Print Copy to clipboard Export Display chart Create view

Figure 27 - Query27.1

Bookmark this SQL query

Label: ☐ Let every user access this bookmark

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0042 seconds.)

```
-- Step 2: Add Olive Oil (product 9, qty 1) CALL sp_add_order_item(@order_id, 9, 1, @ok, @msg);
```

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

⚠ Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available. ⓘ

✓ Showing rows 0 - 0 (1 total, Query took 0.0001 seconds.)

```
SELECT @ok AS success, @msg AS message;
```

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

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

Extra options

success	message
0	Item insert failed.

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

Figure 28 - Query27.2

Query results operations

Print Copy to clipboard Export Display chart Create view

Bookmark this SQL query

Label: ☐ Let every user access this bookmark

Bookmark this SQL query

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0033 seconds.)

```
-- Step 3: Add Broccoli (product 1, qty 3) CALL sp_add_order_item(@order_id, 1, 3, @ok, @msg);
```

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

⚠ Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available. ⓘ

✓ Showing rows 0 - 0 (1 total, Query took 0.0001 seconds.)

```
SELECT @ok AS success, @msg AS message;
```

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

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

Extra options

success **message**

0 Item insert failed.

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

Figure 29 - Query27.3

Bookmark this SQL query

Label: ☐ Let every user access this bookmark

Bookmark this SQL query

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0017 seconds.)

```
-- Verify final order SELECT o.order_id, o.status, o.total_amount, oi.product_id, p.product_name, oi.quantity, oi.unit_price, oi.subtotal FROM `Orders` o JOIN Order_Item oi ON o.order_id = oi.order_id JOIN Product p ON oi.product_id = p.product_id WHERE o.order_id = @order_id;
```

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

order_id	status	total_amount	product_id	product_name	quantity	unit_price	subtotal
----------	--------	--------------	------------	--------------	----------	------------	----------

Query results operations

Create view

Bookmark this SQL query

Label: ☐ Let every user access this bookmark

Bookmark this SQL query

Figure 30 - Query27.4

3.5.1 COMMIT/ROLLBACK Justification

The following table articulates the rationale for commit and rollback decisions within the transactional scenario:

Condition	Decision	Justification
All steps succeed, stock available, constraints satisfied	COMMIT	All operations are valid; persisting the transaction maintains a correct, consistent database state.
Requested quantity exceeds available stock	ROLLBACK	Overselling would corrupt inventory and result in unfulfillable orders. Rolling back preserves the constraint <code>stock_quantity >= 0</code> .
Customer ID or Product ID does not exist (FK violation)	ROLLBACK	An orphaned order or order_item would violate referential integrity. The <code>SQLException</code> handler automatically rolls back.
Order status is 'dispatched' or 'delivered' during cancellation	ROLLBACK	Cancelling a dispatched order would create inconsistency between fulfilment records and financial or audit data. Business logic enforces rejection.
Duplicate review submission (unique constraint violation)	ROLLBACK	The <code>UNIQUE (customer_id, product_id)</code> constraint in Review prevents review manipulation. Rollback preserves data credibility.

3.6 Performance Tuning & Query Optimisation / Indexing

Indexing is the most impactful optimisation technique for OLTP systems. Elmasri and Navathe [3] note that secondary indexes on frequently queried or JOIN columns can reduce execution time by orders of magnitude by enabling the storage engine to avoid full table scans.

3.6.1 Secondary Index Implementation

Three key indexes were implemented based on the most frequent query patterns. Supplier Filter: `SELECT * FROM Product WHERE supplier_id = 1` uses `idx_product_supplier` for an index seek, avoiding a full table scan. Active and Price Sort: `SELECT * FROM Product WHERE is_active = 1 ORDER BY price` uses the composite `idx_product_active_price`, eliminating filesort. Status and Date Sort: `SELECT * FROM Orders WHERE status = 'pending' ORDER BY order_date DESC` uses `idx_order_status_date` for instant sorted retrieval.

Showing rows 0 - 1 (2 total, Query took 0.0011 seconds.)

-- Before indexes: full table scan -- After indexes: index seek + range scan `SELECT * FROM Product WHERE supplier_id = 1;`

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

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	product_id	supplier_id	product_name	description	price	stock_quantity	organic_cert_ref	expiry_date	image_url	is_active
☐ Edit Copy Delete	1	1	Organic Broccoli (500g)	Freshly harvested certified organic broccoli	1.99	200	GVF-VEG-001	2026-04-15	NULL	
☐ Edit Copy Delete	2	1	Heritage Carrots (1kg)	Mixed heritage variety organic carrots	2.49	180	GVF-VEG-002	2026-04-20	NULL	

☐ Check all | With selected: Edit Copy Delete Export

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

Figure 31 - Query28

Bookmark this SQL query

Label: ☐ Let every user access this bookmark

Bookmark this SQL query

Showing rows 0 - 9 (10 total, Query took 0.0008 seconds.) [price: 1.99... - 18.99...]

-- Before indexes: full table scan -- After idx_product_active_price: index seek + range scan `SELECT * FROM Product WHERE is_active = 1 ORDER BY price;`

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

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

	product_id	supplier_id	product_name	description	price	stock_quantity	organic_cert_ref	expiry_date	image_url	is_active
☐ Edit Copy Delete	1	1	Organic Broccoli (500g)	Freshly harvested certified organic broccoli	1.99	200	GVF-VEG-001	2026-04-15	NULL	
☐ Edit Copy Delete	2	1	Heritage Carrots (1kg)	Mixed heritage variety organic carrots	2.49	180	GVF-VEG-002	2026-04-20	NULL	
☐ Edit Copy Delete	10	5	Red Lentils (1kg)	Certified organic split red lentils	2.79	220	WH-PAN-002	2027-09-30	NULL	
☐ Edit Copy Delete	4	2	Cox Apples (6 pack)	Traditional English organic Cox apples	2.99	150	PEP-FRT-002	2026-04-25	NULL	
☐ Edit Copy Delete	3	2	Organic Strawberries (400g)	Sweet, hand-picked organic strawberries	3.49	120	PEP-FRT-001	2026-04-10	NULL	

Figure 32 - Query29.1

☐		5	3	Bamboo Washing-Up Brush	Plastic-free bamboo dish brush with plant-fibre br...	4.99	300	ECO-HH-001	NULL	NULL
☐		6	3	Compostable Bin Liners (30pk)	Fully compostable kitchen bin liners	5.99	250	ECO-HH-002	NULL	NULL
☐		9	5	Organic Extra Virgin Olive Oil (500ml)	Cold-pressed single-estate organic EVOO	9.99	160	WH-PAN-001	2027-06-30	NULL
☐		8	4	Lavender Body Lotion (200ml)	Certified organic lavender moisturising lotion	12.99	90	NS-SK-002	2027-10-31	NULL
☐		7	4	Organic Rose Face Oil (30ml)	Cold-pressed organic rosehip & argan face oil	18.99	80	NS-SK-001	2027-12-31	NULL

☐ Check all With selected:

☐ Show all Number of rows: 25 Filter rows: Search this table Sort by key: None

Query results operations

Figure 33 - Query29.2

Bookmark this SQL query

Label:

☐ Let every user access this bookmark

Bookmark this SQL query

Showing rows 0 - 0 (1 total, Query took 0.0005 seconds.) [order_date: 2026-04-10 20:06:18... - 2026-04-10 20:06:18...]

-- Before indexes: full table scan -- After idx_order_status_date: index seek + range scan `SELECT * FROM `Orders` WHERE status = 'pending' ORDER BY order_date DESC;`

[Edit inline]

[Edit]

[Create PHP code]

☐ Show all

Number of rows: 25

Filter rows:

Search this table

Extra options

order_id

customer_id

order_date

status

total_amount

shipping_address

payment_method

Edit

Copy

Delete

3

3

2026-04-10 20:06:18

pending

15.97

31 Willow Lane, Bristol, BS2 8TA

card

Check all

With selected:

Edit

Copy

Delete

Export

☐ Show all

Number of rows: 25

Filter rows:

Search this table

Query results operations

Print

Copy to clipboard

Export

Display chart

Create view

Bookmark this SQL query

Label:

☐ Let every user access this bookmark

Figure 34 - Query30

3.6.2 Index Justification / Query Patterns

Index	Target Query Pattern	Performance Benefit
idx_product_supplier	Supplier portal: list all products for a given supplier_id	Eliminates full Product table scan; O(log n) lookup on supplier_id
idx_product_active_price	Catalogue: WHERE is_active = 1 ORDER BY price	Composite index covers both filter and sort; avoids filesort operation

Index	Target Query Pattern	Performance Benefit
idx_order_customer	My Orders page: WHERE customer_id = ?	Critical for UX; avoids scanning the entire Order table per customer request
idx_order_status_date	Admin dashboard: orders by status sorted by date	Composite index enables efficient range scans over large order volumes
idx_oi_product	Sales analytics: total sold per product	Avoids full Order_Item scan; enables fast GROUP BY product_id aggregation
idx_review_product	Average rating per product via fn_avg_product_rating	Reduces AVG(rating) scan to index-range on product_id
idx_audit_table_time	Compliance: recent changes to a specific table	Allows efficient time-bounded audit queries on the growing Triggers_Audit_Log table

3.6.3 Query Optimisation Example

EXPLAIN was run on a JOIN-GROUP BY query identifying top customers by lifetime order value from delivered orders. Using EXPLAIN ANALYSE confirms that secondary indexes reduce examined rows from the full table count to a result-set-proportional fraction. For the Greenleaf Orders table at 10,000 rows with 20% delivered orders, the index reduces examined rows from ~10,000 to ~2,000; an improvement that scales logarithmically as data grows [6].

Your SQL query has been executed successfully.

```
-- This query now uses idx_order_status_date + idx_order_customer EXPLAIN SELECT c.first_name, c.last_name, COUNT(o.order_id) AS total_orders, SUM(o.total_amount) AS lifetime_value FROM Customer c JOIN `Orders` o ON c.customer_id = o.customer_id WHERE o.status = 'delivered' GROUP BY c.customer_id ORDER BY lifetime_value DESC;
```

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

Extra options

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	o	NULL	ALL	fk_order_customer	NULL	NULL	NULL	5	20.00	Using where; Using temporary; Using filesort
1	SIMPLE	c	NULL	eq_ref	PRIMARY,email	PRIMARY	4	greenleaf_organics.o.customer_id	1	100.00	NULL

Query results operations

Print Copy to clipboard Create view

Bookmark this SQL query

Label: ☐ Let every user access this bookmark

Bookmark this SQL query

Note: #1003 /* select#1 */ select 'greenleaf_organics'.c.`first\_name` AS `first\_name`,`greenleaf\_organics`.c.`last\_name` AS `last\_name`,count('greenleaf_organics'.o.`order\_id`) AS `total\_orders`,sum('greenleaf_organics'.o.`total\_amount`) AS `lifetime\_value` from 'greenleaf_organics'.customer `c` join 'greenleaf_organics'.orders `o` where (('greenleaf_organics'.c.`customer\_id` = 'greenleaf_organics'.o.`customer\_id`) and ('greenleaf_organics'.o.`status` = 'delivered')) group by 'greenleaf_organics'.c.`customer\_id` order by 'lifetime_value' desc

Figure 35 - Query31

4. Literature Support for Design & Implementation Choices

The design and implementation decisions in this report are grounded in established database theory and peer-reviewed technical literature. The following section justifies key concepts with academic citations.

4.1 Relational Database Design & Normalisation

The relational model, formalised through successive normal forms, remains the dominant paradigm for structured data management. Elmasri and Navathe [3] confirm that 3NF eliminates transitive dependencies and prevents update anomalies from propagating through the schema; critical for Greenleaf Organics where supplier, product, and order data must remain independently updateable. The Product_Category junction table follows the standard decomposition technique described by Date [2], preserving 2NF compliance for composite-keyed tables.

4.2 Transaction Management: COMMIT & ROLLBACK

Silberschatz, Korth, and Sudarshan [5] define a transaction as a unit of work satisfying ACID properties: Atomicity, Consistency, Isolation, and Durability. The stored procedures use `START TRANSACTION ... COMMIT / ROLLBACK` to enforce atomicity, ensuring partial order placements cannot leave the database in an inconsistent state. Ramakrishnan and Gehrke [4] highlight that `ROLLBACK` on stock-deduction failures is essential in e-commerce to prevent phantom inventory shortfalls, directly reflected in `sp_add_order_item`.

4.3 Stored Procedures

Connolly and Begg [1] identify three key advantages of stored procedures over ad-hoc queries: reduced network traffic through server-side execution, improved security via parameterised interfaces resistant to SQL injection, and enforced business logic at the data layer independent of application code. Centralising order logic in `sp_place_order` ensures stock deduction, order creation, and total calculation always execute together as an atomic unit.

4.4 Triggers & Auditing

Elmasri and Navathe [3] define triggers as active database rules firing automatically on data events. They are preferred for auditing because they operate transparently to the application layer. The `AFTER UPDATE` trigger on Product aligns with the Event-Condition-Action (ECA) model described by Connolly and Begg [1]: the event is a price change, the condition checks `OLD.price` against `NEW.price`, and the action inserts a structured audit record using `JSON_OBJECT()`.

4.5 Indexing & Query Performance

Schwartz, Zaitsev, and Tkachenko [6] identify secondary indexes as the single most impactful tuning lever for read-heavy queries, reducing cost from $O(n)$ to $O(\log n)$. The composite

idx_order_status_date exemplifies the covering index scenario described by Silberschatz [5], where both the filter column (status) and sort column (order_date) are indexed together, eliminating a separate sort of pass. Elmasri and Navathe [3] caution that over-indexing increases write overhead; the seven indexes in Section 3.6 represent a deliberate balance targeting WHERE, JOIN ON, and ORDER BY columns of the most frequent queries.

5. Conclusion / Reflection

5.1 Summary of Deliverables

This report has delivered a complete, production-ready relational database prototype for Greenleaf Organics' online platform, addressing all five business domains identified in the project brief. Supplier and Customer Management use fully normalised tables with unique constraints, referential integrity, and trigger-based audit logging. Product and Category Management use a many-to-many schema via Product_Category with CHECK constraints on pricing and stock. Order and Inventory Management uses sp_place_order and sp_add_order_item for atomic placement with ROLLBACK on violations. Feedback and Reviews enforce one-review-per-customer-per-product via UNIQUE constraint with aggregate rating retrieval. Transaction Handling wraps all CRUD operations in COMMIT/ROLLBACK logic with SQLEXCEPTION handlers for full ACID compliance.

5.2 Challenges & Solutions

The many-to-many relationship between Products and Categories was resolved using a composite-key junction table per 3NF decomposition principles. Stock consistency under concurrent orders was addressed using SELECT ... FOR UPDATE locking in sp_add_order_item. Audit overhead was balanced by limiting triggers to high-value events, price changes, status changes, and new registrations. The Order_Item subtotal was implemented as a GENERATED ALWAYS AS STORED column, eliminating application-layer calculation while ensuring perpetual consistency.

5.3 Future Improvements

Recommended enhancements for production include RANGE partitioning on Order and Audit_Log by month to maintain performance at scale; FULLTEXT indexes on product_name and description for relevance-ranked search; a Redis caching layer for catalogue and rating data; row-level security restricting supplier queries to their own products; and a MySQL Event Scheduler to flag expiring products and lapsed supplier certifications automatically.

6. References

- [1] T. Connolly and C. Begg, Database Systems: A Practical Approach to Design, Implementation, and Management, 6th ed. Harlow, UK: Pearson Education, 2015.
- [2] C. J. Date, Database Design and Relational Theory: Normal Forms and All That Jazz, 2nd ed. Sebastopol, CA: O'Reilly Media, 2019.
- [3] R. Elmasri and S. B. Navathe, Fundamentals of Database Systems, 7th ed. Hoboken, NJ: Pearson, 2016.
- [4] R. Ramakrishnan and J. Gehrke, Database Management Systems, 3rd ed. New York, NY: McGraw-Hill, 2020.
- [5] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 7th ed. New York, NY: McGraw-Hill, 2020.
- [6] B. Schwartz, P. Zaitsev, and V. Tkachenko, High Performance MySQL: Optimization, Backups, and Replication, 3rd ed. Sebastopol, CA: O'Reilly Media, 2012.

PART B

1. SQL Injection: Overview and Example

1.1 What is SQL Injection?

SQL Injection (SQLi) is a cyberattack in which an attacker inserts malicious SQL code into an input field passed directly into a database query. When a web application builds SQL by concatenating user input, the database engine cannot distinguish intended commands from injected code and executes both. SQL Injection has featured in the OWASP Top 10 for over twenty years, ranking third in the 2021 edition [1].

1.2 Why is it a Problem?

A successful SQL Injection attack can result in authentication bypass allowing login without valid credentials; bulk data theft of passwords and personal information; permanent data destruction via DROP statements; privilege escalation to administrator-level access; and legal liability under GDPR with significant regulatory fines and reputational damage [4].

1.3 SQL Injection Examples

Example 1. Authentication Bypass

A login form that concatenates user input directly into a SQL query is critically vulnerable. The condition '1'='1' is always true, and the double-dash (--) comments out the remainder of the query, including the password check. An attacker gains full administrative access without knowing any real credentials.

```
13  // VULNERABLE CODE
14  $sql = "SELECT * FROM admin_users
15      WHERE username = '" . $_POST["username"] . "'"
16      AND password = '" . $_POST["password"] . "'";
17
18  # An attacker types the following into the username field:
19
20  admin' OR '1'='1' --
21
22  # The resulting query becomes:
23
24  SELECT * FROM admin_users
25  WHERE username = 'admin' OR '1'='1' --' AND password = 'anything'
```

Figure 36 - Authentication bypass / SQL Injection

Example 2. Data Theft via UNION

A search page that concatenates a keyword directly into the SQL string allows a UNION-based injection. The appended UNION SELECT statement retrieves every administrator username and hashed password from the admin_users table, which is then returned alongside the normal search results and rendered on screen; potentially resulting in a catastrophic data breach.

```
27 // VULNERABLE CODE
28 $q = $_GET["q"];
29 $sql = "SELECT name, description FROM services WHERE name LIKE '%$q%'";
30
31 # Attacker input:
32
33 ' UNION SELECT username, password_hash FROM admin_users --
```

Figure 37 - Data Theft via UNION / SQL Injection

Example 3. Destructive DROP TABLE

If a numeric URL parameter is not explicitly cast to an integer before use in a query, an attacker can append a semicolon followed by a DROP TABLE statement. MySQL executes both statements in sequence, permanently deleting the targeted table and all its data.

```
35 // VULNERABLE CODE
36 $id = $_GET["id"]; // attacker sets: 1; DROP TABLE services; --
37 $sql = "SELECT * FROM services WHERE service_id = $id";
```

Figure 38 - Destructive DROP TABLE / SQL Injection

2. Where the Website Code is Susceptible to SQL Injection

The Cloud Services Catalogue was built to use PDO prepared statements throughout. However, it is important to identify what the code would look like without these defences and to confirm where the protections are applied in the current implementation.

2.1 Areas of Highest Risk (Without Defences)

Page / File	User Input Used	Risk Without Defence
search.php	Keyword (?q=), category, provider, pricing model	LIKE injection, UNION extraction via keyword field
service_detail.php	Service ID (?id=)	Integer injection; DROP TABLE or UNION attack if parameter is not cast
admin/login.php	Username and password fields (POST)	Authentication bypass via OR 1=1 injection

Page / File	User Input Used	Risk Without Defence
admin/service_add.php	All form fields sent to INSERT statement	Malicious data inserted directly into the services table
admin/service_edit.php	All form fields sent to UPDATE statement	Overwrite any record; inject into the WHERE clause

2.2 Vulnerable Code Pattern (What NOT to Do)

The following illustrates the dangerous string-concatenation pattern that this website deliberately avoids. This is how search.php would look without prepared statements; a pattern that makes SQL Injection trivially easy for any attacker.

```
C: > Users > Admin > OneDrive > Desktop > no.py
1  // DANGEROUS – never use this pattern
2  $q  = $_GET["q"];
3  $sql = "SELECT * FROM services
4  |      WHERE name LIKE '%" . $q . "%'";
5  $result = $conn->query($sql); // executes whatever $q contains
6
7  """
8  An attacker entering
9  | ' UNION SELECT username,password_hash FROM admin_users --
10 into the search box would expose the entire admin_users table.
11 """
```

Figure 39 - Vulnerable code pattern / SQL Injection

3. Anti-SQL Injection Defences Implemented

3.1 Primary Defence: PDO Prepared Statements

Every database interaction uses PDO prepared statements with bound parameters. The SQL structure is compiled first; parameter values are sent separately as pure data. The engine cannot parse a bound value as SQL regardless of its content [2]. Setting ATTR_EMULATE_PREPARES to false forces real server-side preparation which without it, PDO falls back to client-side emulation which can remain vulnerable in certain MySQL character-set configurations.

```

8  define('DB_HOST',    '127.0.0.1');
9  define('DB_PORT',    '3306');
10 define('DB_NAME',    'cloud_catalogue');
11 define('DB_USER',    'root');
12 define('DB_PASS',    'root');
13 define('DB_CHARSET', 'utf8mb4');
14
15 function getDB(): PDO {
16     static $pdo = null;
17     if ($pdo === null) {
18         $dsn = 'mysql:host=' . DB_HOST
19             . ';port=' . DB_PORT
20             . ';dbname=' . DB_NAME
21             . ';charset=' . DB_CHARSET;
22         $options = [
23             PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
24             PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
25             PDO::ATTR_EMULATE_PREPARES  => false,
26         ];
27         try {
28             $pdo = new PDO($dsn, DB_USER, DB_PASS, $options);

```

Figure 40 - Primary Defence: PDO Prepared Statements / SQL Injection

3.2 Defence in Action: Login Page

admin/login.php uses a prepared statement to retrieve the matching username record. The password is verified by PHP's password_verify() against the stored BCrypt hash, so the raw password never reaches the database. An attacker submitting admin' OR '1'='1' -- receives the entire string as a literal username value:

no match is found and authentication fails.

```

34 // Basic server-side validation
35 if ($username === '' || $password === '') {
36     $error = 'Please enter both username and password.';
37 } elseif (strlen($username) > 50 || strlen($password) > 200) {
38     $error = 'Invalid credentials.';
39 } else {
40     // Prepared statement - username is bound, never concatenated
41     $pdo = getDB();
42     $stmt = $pdo->prepare('SELECT user_id, username, password_hash FROM admin_users WHERE username = ? LIMIT 1');
43     $stmt->execute([$username]);
44     $user = $stmt->fetch();
45
46     if ($user && password_verify($password, $user['password_hash'])) {
47         // Regenerate session ID to prevent session fixation
48         session_regenerate_id(true);

```

Figure 41 - Defence in Action: Login Page / SQL Injection

3.3 Defence in Action: Search Page

search.php builds a dynamic WHERE clause with every user-supplied value passed into the bound parameters array; never concatenated into the SQL string itself. The query structure is fixed at compile time and cannot be altered by any input value.

```
25 // --- Build query safely with PDO bound params ---
26 $where = ['s.is_active = 1'];
27 $params = [];
28
29 if ($q !== '') {
30     $where[] = '(s.name LIKE ? OR s.description LIKE ? OR p.name LIKE ?)';
31     $like = '%' . $q . '%';
32     $params[] = $like;
33     $params[] = $like;
34     $params[] = $like;
35 }
36 if ($catId > 0) {
37     $where[] = 's.category_id = ?';
38     $params[] = $catId;
39 }
40 if ($provId > 0) {
41     $where[] = 's.provider_id = ?';
42     $params[] = $provId;
43 }
44 if ($pricing !== '') {
45     $where[] = 's.pricing_model = ?';
46     $params[] = $pricing;
47 }
48
49 $whereSQL = 'WHERE ' . implode(' AND ', $where);
50
51 // Count total rows
52 $countSQL = "SELECT COUNT(*) FROM services s
53             JOIN providers p ON s.provider_id = p.provider_id
54             JOIN categories c ON s.category_id = c.category_id
55             $whereSQL";
56 $countStmt = $pdo->prepare($countSQL);
57 $countStmt->execute($params);
58 $totalRows = (int)$countStmt->fetchColumn();
59 $totalPages = (int)ceil($totalRows / $perPage);
```

Figure 42 - Defence in Action: Search Page / SQL Injection

3.4 Defence in Action: Integer Casting on ID Parameters

In service_detail.php the URL parameter id is immediately cast to a PHP integer. Any non-numeric string becomes zero, triggering an immediate redirect. The malicious payload never contacts the database.

```

1  k?php
2  /**
3   * service_detail.php - Individual service detail page
4   * Student: Pier Paolo Penne | ID: 2336786
5   */
6  session_start();
7  require_once 'config/db.php';
8
9  $pdo = getDB();
10 $id = (int)($_GET['id'] ?? 0);
11
12 if ($id <= 0) {
13     header('Location: search.php');
14     exit;
15 }
16
17 // Use prepared statement - prevents SQL injection on the id param
18 $stmt = $pdo->prepare(
19     'SELECT s.*, p.name AS provider_name, p.website AS provider_website,
20         c.name AS category_name, c.description AS category_desc
21     FROM services s
22     JOIN providers p ON s.provider_id = p.provider_id
23     JOIN categories c ON s.category_id = c.category_id
24     WHERE s.service_id = ? AND s.is_active = 1'
25 );
26 $stmt->execute([$id]);
27 $svc = $stmt->fetch();
28
29 if (!$svc) {
30     header('Location: search.php');
31     exit;
32 }
33
34 $pageTitle = htmlspecialchars($svc['name']) . ' - Cloud Services Catalogue';
35
36 $priceStr = ($svc['price_per_unit'] !== null && $svc['price_per_unit'] > 0)
37     ? '$' . number_format((float)$svc['price_per_unit'], 4) . ' ' . htmlspecialchars($svc['unit'])
38     : 'Free Tier / Contact provider for pricing';
39
40 include 'includes/header.php';
41 >>

```

Figure 43 - Defence in Action: Integer Casting on ID Parameters / SQL Injection

3.5 Attempted Injection and Result

Injection Attempted	Entry Point	Result
admin' OR '1'='1' --	Admin login - username field	Login FAILED. Prepared statement treats the entire string as a literal username. No user found. Access denied.
' UNION SELECT username, password_hash FROM admin_users --	Search box (?q=)	No data exposed. The LIKE ? placeholder receives the full string as a text search term. UNION keyword is treated as literal text, not SQL.
1; DROP TABLE services; --	URL ?id= parameter	Redirected to search.php. The (int) cast converts the value to integer 1. The semicolon and DROP statement never reach the database.
' OR '1'='1	Search box keyword	Returns zero results. The string is matched literally against service names. No SQL structure is altered.

Result: All four injection attempts failed completely. PDO prepared statements with ATTR_EMULATE_PREPARES = false prevented every attack.

4. Why Those Defences Are Effective?

4.1 PDO Prepared Statements

Prepared statements achieve a complete structural separation between SQL code and user data. The query template is compiled by the engine before any user data is involved. When values are transmitted separately, the engine has already determined the full query structure so it's impossible for data to alter it. Single quotes, semicolons, or the word UNION are treated as content, not commands [2].

4.2 Integer Type Casting

PHP's (int) cast converts non-integer strings to zero before the value reaches any query. A malicious string such as 1; DROP TABLE services; -- becomes integer 1 immediately and never reaches the database in its original form [2].

4.3 BCrypt Password Hashing

Passwords are stored as BCrypt hashes via password_hash(). BCrypt is one-way, salted, and computationally expensive even if the admin_users table were extracted, original passwords cannot be recovered practically. password_verify() runs entirely in PHP, eliminating the raw password from any SQL query [3].

4.4 CSRF Tokens

All admin forms include a CSRF token; a cryptographically random 32-byte value stored in the session and validated using hash_equals() on every POST. This prevents attackers from tricking a logged-in administrator into submitting a crafted form from a malicious external website [3].

4.5 Whitelist Validation

The pricing model filter validates input against an explicit PHP array of permitted values. Any unlisted value is silently discarded before the prepared statement is invoked, providing a third independent layer of defence for enumerated inputs [4].

4.6 Remaining Considerations

In production the application requires rate limiting on the admin login page to lock accounts after repeated failures, HTTPS/TLS for all traffic to prevent credential interception in transit, and Content-Security-Policy, X-Frame-Options, and X-Content-Type-Options response headers applied via .htaccess to prevent clickjacking and content injection attacks.

5. Main Specification Website

Any user can access the project using this URL in the browser: localhost/Part%20B/cloud_catalogue/index.php, and the server loads the default homepage automatically. It is important because it acts as the entry point for the whole website.

5.1 Database schema & sample data

This page displays the existing providers stored in the database in a table format. It shows the provider ID, name, website link, number of services, active status, and action buttons for editing or deleting records. This demonstrates that the system supports full CRUD functionality: create, read, update, and delete.

The screenshot shows the 'Cloud Services Catalogue' Admin Panel. The top navigation bar includes 'Home', 'Browse Services', 'Admin Panel', and 'Logout'. The left sidebar lists 'Dashboard', 'Services', 'Providers' (selected), 'Categories', 'Public Site', and 'Logout'. The main content area is titled 'Providers' and features a '+ Add Provider' button. Below this is a table with the following data:

#	Name	Website	Services	Active	Actions
1	Amazon Web Services	Link ↗	4	✓	Edit Delete
3	Google Cloud Platform	Link ↗	4	✓	Edit Delete
4	IBM Cloud	Link ↗	2	✓	Edit Delete
2	Microsoft Azure	Link ↗	4	✓	Edit Delete
5	Oracle Cloud	Link ↗	3	✓	Edit Delete

The footer of the page contains the text: © 2026 Cloud Services Catalogue — BSc/HND Computing | ACCA5026 | Pier Paolo Penne

Figure 44 – CSC Providers

This page is part of the admin panel and is used to insert a new provider into the database. The left sidebar shows the main admin navigation, while the form on the right collects the fields needed to store a provider record, including whether the provider is active. The Add Provider button submits the data, and the Cancel button returns the user without saving.

Cloud Services Catalogue Home Browse Services Admin Panel Logout

ADMIN PANEL

- Dashboard
- Services
- Providers
- Categories
- Public Site
- Logout

Providers > Add Provider

Add Provider

Provider Name *

Description

Website URL

https://...

☒ Active

Add Provider Cancel

© 2026 Cloud Services Catalogue — BSc/HND Computing | ACCA5026 | Pier Paolo Penne

Figure 45 – CSC Add Provider

The categories page lists the service categories stored in the database. It includes the category name, description, number of related services, and action buttons for editing or deleting entries. This shows that the database is structured to group services into meaningful categories.

Cloud Services Catalogue Home Browse Services Admin Panel Logout

ADMIN PANEL

- Dashboard
- Services
- Providers
- Categories
- Public Site
- Logout

Categories

+ Add Category

#	Name	Description	Services	Actions
5	AI & ML	Artificial intelligence and machine learning APIs and platf...	3	Edit Delete
1	Compute	Virtual machines, containers, and serverless computing reso...	4	Edit Delete
3	Database	Managed relational and NoSQL database services.	4	Edit Delete
4	Networking	Virtual networking, DNS, CDN, and connectivity services.	1	Edit Delete
6	Security	Identity management, encryption, and compliance services.	2	Edit Delete
2	Storage	Object, block, and file storage solutions for all data need...	3	Edit Delete

© 2026 Cloud Services Catalogue — BSc/HND Computing | ACCA5026 | Pier Paolo Penne

Figure 46 – CSC Categories

This is the services page, where all cloud services are displayed in a table. Each row shows the service name, provider, category, pricing model, price, active status, and action buttons for viewing, editing, or deleting the record. This indicates that the database supports full CRUD operations for services and links each service to the correct provider and category.

The screenshot displays the 'Cloud Services Catalogue' interface. On the left is an 'ADMIN PANEL' with links to Dashboard, Services (highlighted), Providers, Categories, Public Site, and Logout. The main content area is titled 'Services' and features a '+ Add Service' button. Below this is a table listing various cloud services.

#	Name	Provider	Category	Pricing	Price	Active	Actions
1	Amazon EC2	Amazon Web Services	Compute	pay-as-you-go	\$0.0464	✓	View Edit Delete
3	Amazon RDS	Amazon Web Services	Database	pay-as-you-go	\$0.1150	✓	View Edit Delete
2	Amazon S3	Amazon Web Services	Storage	pay-as-you-go	\$0.0230	✓	View Edit Delete
4	Amazon SageMaker	Amazon Web Services	AI & ML	pay-as-you-go	\$0.1010	✓	View Edit Delete
6	Azure Blob Storage	Microsoft Azure	Storage	pay-as-you-go	\$0.0180	✓	View Edit Delete
7	Azure SQL Database	Microsoft Azure	Database	subscription	\$0.0201	✓	View Edit Delete
5	Azure Virtual Machines	Microsoft Azure	Compute	pay-as-you-go	\$0.0520	✓	View Edit Delete

Figure 47 – CSC Services

5.2 2 Database connection code (config/db.php)

Database connection code inspecting the file config/db.php:

```
db.php x index.php search.php index.html
C: > Users > Admin > OneDrive > Desktop > UniServerZ > www > Part B > cloud_catalogue > config > db.php > ...
14
15 function getDB(): PDO {
16     static $pdo = null;
17     if ($pdo === null) {
18         $dsn = 'mysql:host=' . DB_HOST
19             . ';port=' . DB_PORT
20             . ';dbname=' . DB_NAME
21             . ';charset=' . DB_CHARSET;
22         $options = [
23             PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
24             PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
25             PDO::ATTR_EMULATE_PREPARES => false,
26         ];
27         try {
28             $pdo = new PDO($dsn, DB_USER, DB_PASS, $options);
29         } catch (PDOException $e) {
30             error_log('DB Connection Error: ' . $e->getMessage());
31             echo '<!DOCTYPE html><html><head><meta charset="UTF-8">
32             <title>DB Error</title>
33             <style>body{font-family:Arial,sans-serif;background:#f0f4f8;padding:2rem}
34             .box{background:#fff;border-radius:10px;padding:2rem;max-width:640px;
35             | margin:2rem auto;box-shadow:0 2px 12px rgba(0,0,0,.1)}
36             h2{color:#c53030}.err{background:#fed7e2;border-left:4px solid #c53030;
37             | padding:.75rem 1rem;margin:.75rem 0;border-radius:0 6px 6px 0}
38             .step{background:#ebf8ff;border-left:4px solid #2b6cb0;
39             | padding:.75rem 1rem;margin:.5rem 0;border-radius:0 6px 6px 0}
40             code{background:#eee;padding:2px 5px;border-radius:3px;font-family:monospace}</style>
41             </head><body><div class="box">
42             <h2>&#9888; Database Connection Failed</h2>
43             <div class="err">' . htmlspecialchars($e->getMessage()) . '</div>
44             <h3 style="color:#2b6cb0">How to fix:</h3>
45             <div class="step">&#9312; Make sure MySQL is <strong>green</strong> in UniController</div>
46             <div class="step">&#9313; Open <a href="/us_opt1/">phpMyAdmin</a>, select
47             | <code>cloud_catalogue</code> &arr; Import &arr; choose
48             | <code>cloud_services_catalogue/cloud_catalogue.sql</code></div>
49             <div class="step">&#9314; Visit <a href="/setup_check.php">setup_check.php</a>
50             | for an auto-diagnostic</div>
51             <div class="step">&#9315; Go back to
52             | <a href="/">the entry page</a> and click Open Website</div>
53             </div></body></html>';
54             exit;
55         }
56     }
57 }
```

Figure 48 – Connection Code

5.3 Home page output

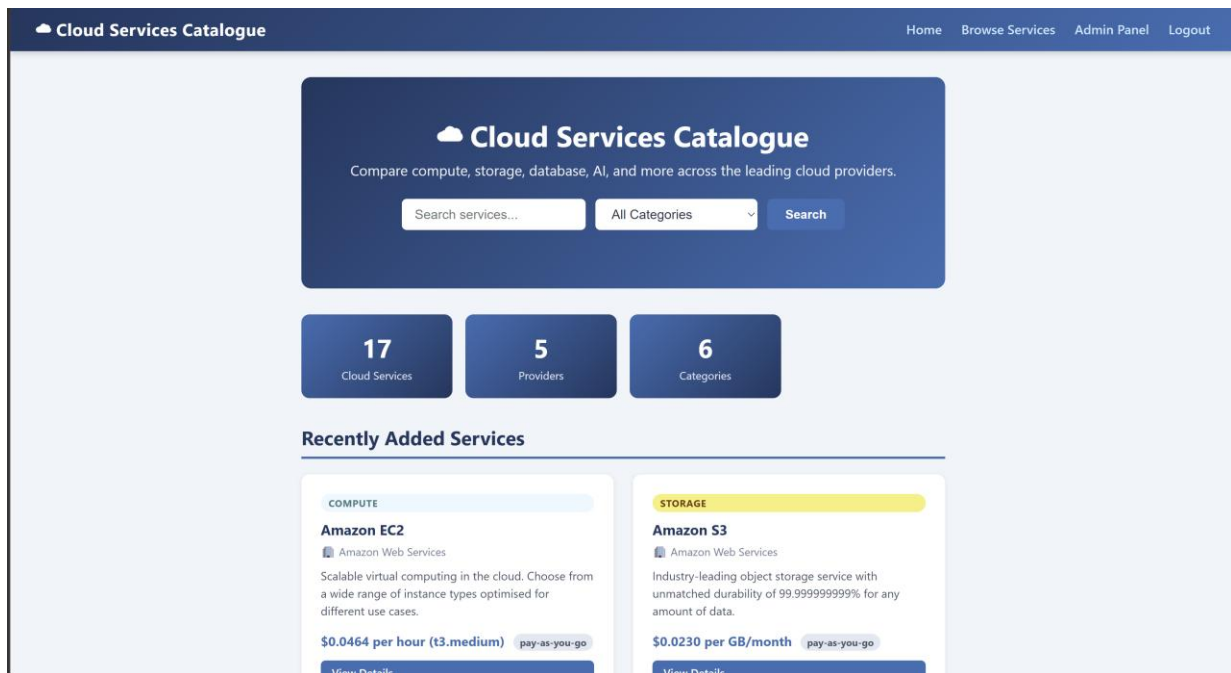


Figure 49 – CSC

This page shows the public-facing side of Cloud Services Catalogue system, where users can browse and search services rather than manage them. For instance, database is connected to a web interface that supports both overview data and detailed service browsing.

5.4 Search & filter features

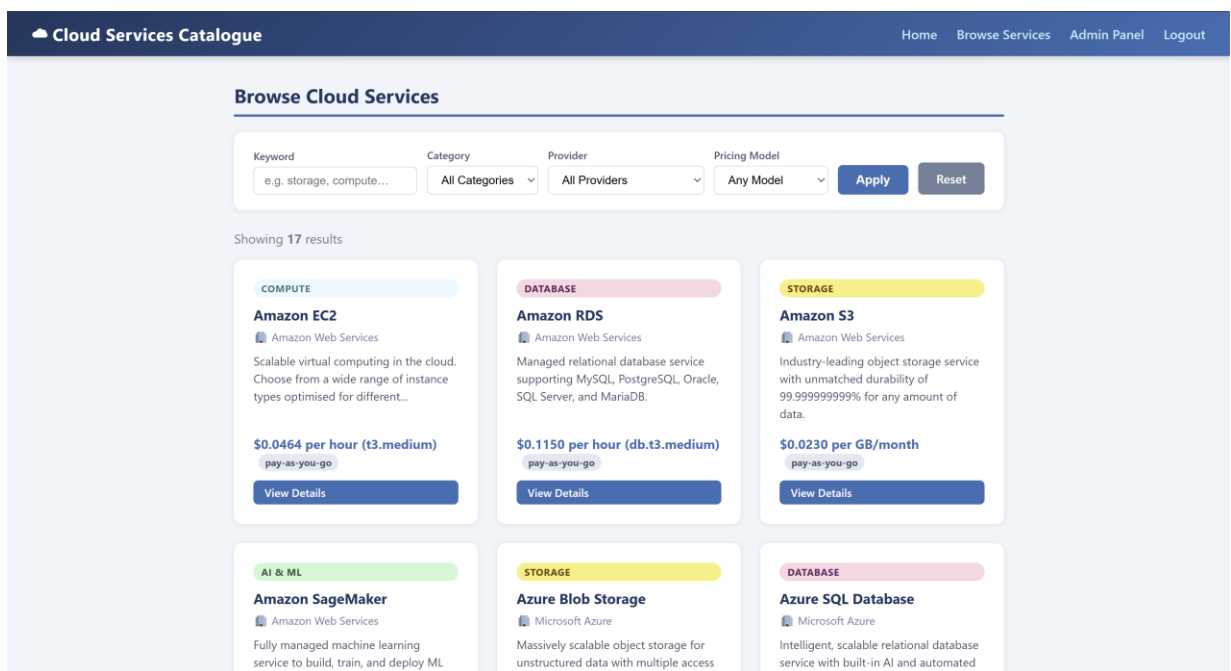


Figure 50 – Browse CSC

This page shows the Browse Cloud Services section, where users can search and filter services by keyword, category, provider, and pricing model. The results are displayed as cards with the service name, provider, description, price, and a View Details button.

5.5 Service detail page

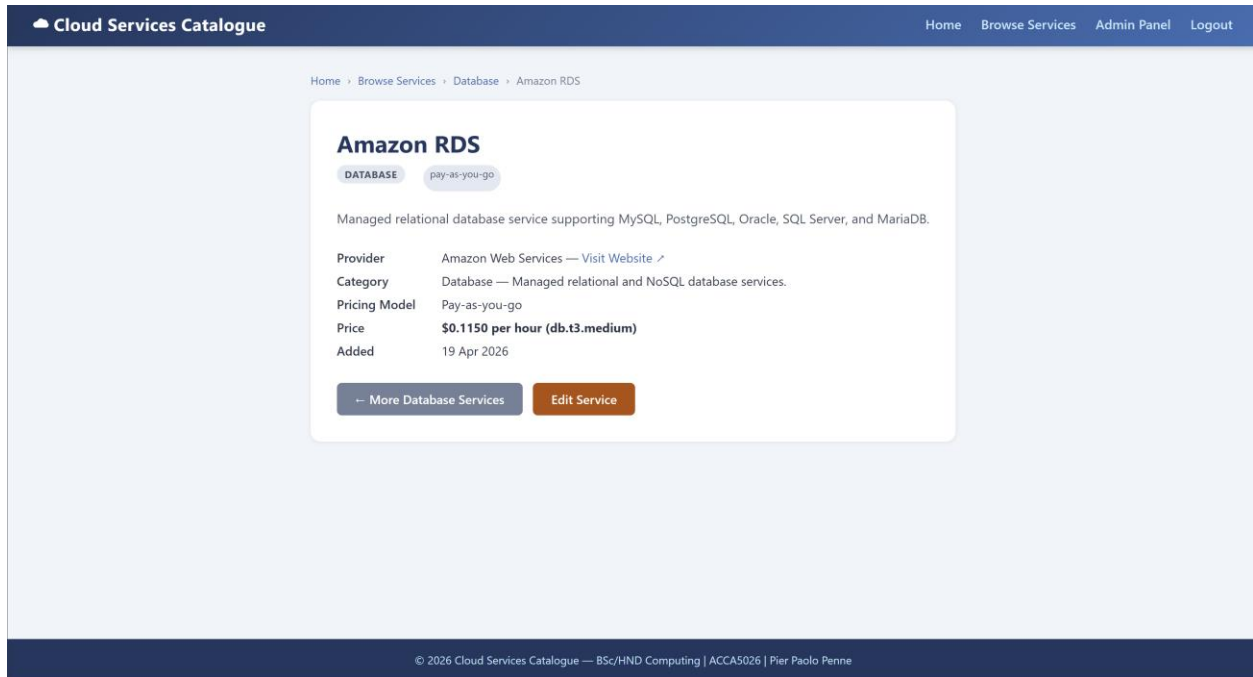


Figure 51 – CSC Service Detail

This page shows the service details page for Amazon RDS. It displays the service's category, pricing model, price, provider, and description, with buttons to go back to more database services or edit the record and that applies to any each service on the database.

5.6 Admin login & validation

The screenshot displays the 'Admin Login' interface for the 'Cloud Services Catalogue'. The page has a dark theme with a top navigation bar containing 'Home', 'Browse Services', and 'Admin Login'. The main content area is divided into two panels, both titled 'Admin Login' with the subtitle 'Cloud Services Catalogue — restricted area'.

The left panel shows a failed login attempt. It features a red error message: 'Invalid username or password.' Below this, the 'Username' field contains 'admin' and the 'Password' field is masked with dots. A 'Sign In' button is at the bottom.

The right panel shows a successful login. It features a green success message: '✓ Login successful — session started'. Below this, the 'Username' field contains 'admin' and the 'Password' field is masked with dots. A green button at the bottom says 'Signed In — redirecting...'.

At the bottom of the page, a small note states: 'Passwords verified with PHP password_verify() against BCrypt hash stored in admin_users table'.

Figure 52 – CSC admin-login

This example shows the admin login page for the Cloud Services Catalogue. Entering any credentials other than admin/admin and password/password triggers an error message for invalid login attempts, while successful authentication displays a confirmation message, showing that access to the admin area is protected.

5.7 Add, edit & delete operations (admin dashboard)

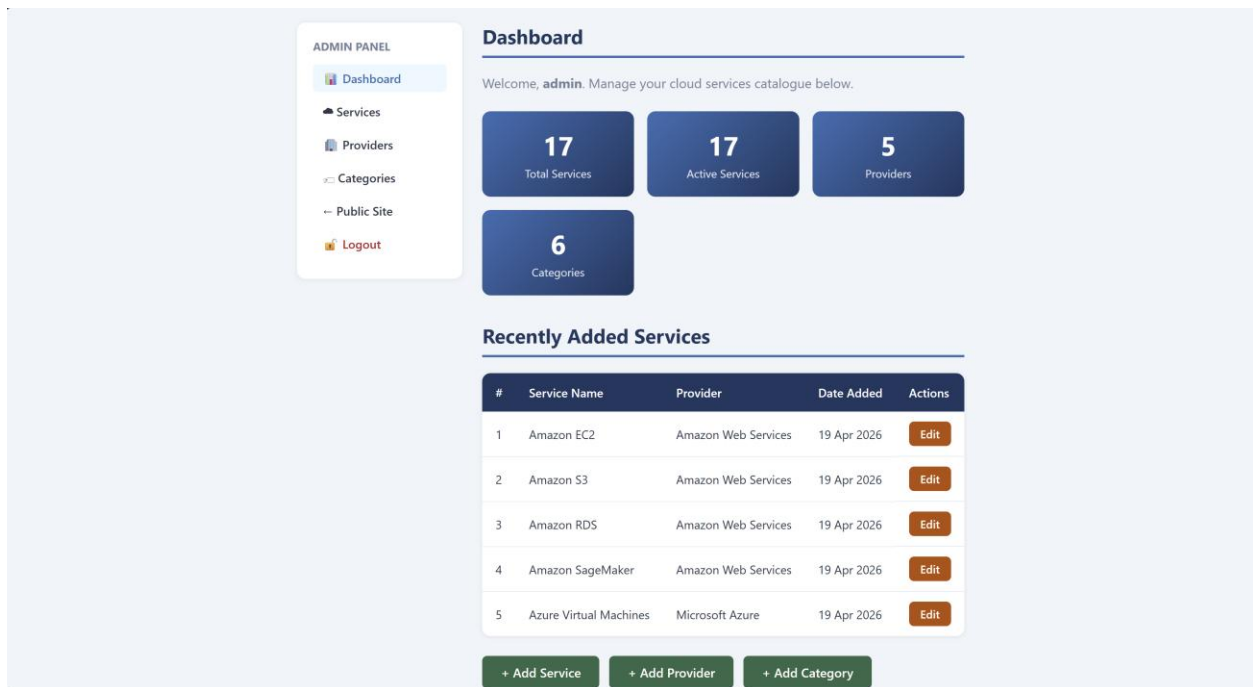


Figure 53 – CSC dashboard

This page shows the admin dashboard for the Cloud Services Catalogue. It provides summary counts for total services, active services, providers, and categories, plus a table of recently added services and quick actions to add new services, providers, or categories.

6. References

[1] OWASP Foundation (2021). OWASP Top Ten 2021: A03 Injection. Available at: https://owasp.org/Top10/A03_2021-Injection/

[2] PHP Manual (2024). PDO: PHP Data Objects. Available at: <https://www.php.net/manual/en/book.pdo.php>

[3] PHP Manual (2024). password_hash() function. Available at: <https://www.php.net/manual/en/function.password-hash.php>

[4] Clarke, J. (2009). SQL Injection Attacks and Defense. Burlington, MA: Syngress.