Project Report: Small Business Inventory and Sales System

Filbert Ethan Hermawan	(24/532773/PA/22535)
Satria Bramantio Arbian	(24/533017/PA/22559)
Keita Kawabata	(24/549127/PA/23221)

Department of Computer Science and Electronics, Faculty of Mathematics and Natural Sciences,
Universitas Gadjah Mada
Yogyakarta, Indonesia

The following table shows which member worked on each part of the project and report.

Section / Feature	**Member in Charge**
I. Introduction	Keita
II. Database Design	Satria
III. Database Implementation	Satria
IV. Application Implementation	Satria
4.1 System Architecture	Keita

4.2 Authentication	Keita
4.3 Product Management	Filbert
4.4 Category Management	Keita
4.5 Customer Management	Filbert
4.6 Sales Management	Keita
4.7 Dashboard & Reporting	Keita
V. Testing and Results	Keita
VI. Conclusion	Filbert

I. Introduction

1.1 Problem Background

Small businesses often struggle to manage their inventory and sales efficiently. Many still rely on manual record-keeping methods such as paper notebooks or spreadsheets, which can lead to several problems:

- **Inaccurate stock levels** - Products may run out without warning, or excess stock may go unnoticed
- **Lost transaction records** - Sales data can be misplaced or recorded incorrectly
- **Slow reporting** - Generating sales reports and performance summaries takes too much time
- **Difficulty tracking customers** - Customer purchase history is hard to maintain and retrieve

These problems make it difficult for business owners to make informed decisions and can result in lost revenue and customer dissatisfaction.

To solve these challenges, we developed a system that automates inventory management, records all sales transactions, and provides real-time reports. This system ensures data accuracy, improves efficiency, and helps business owners make better decisions.

1.2 Objectives

The main objectives of this project are:

1. **Design and implement a relational database** that manages product inventory, categories, sales transactions, customers, and system users
2. **Develop a user-friendly application** that allows staff to easily record and track daily operations including product updates, customer records, and sales
3. **Provide real-time stock monitoring** with automatic alerts when inventory levels fall below a specified threshold
4. **Generate sales and performance reports** that help business owners understand their revenue, popular products, and inventory status
5. **Ensure data security and integrity** through proper user authentication, role-based access control, and database constraints

1.3 Target Users

1. **Administrator (Business Owner/Manager)**
    - Full access to all system features
    - Can add, edit, and delete products, categories, and customers
    - Can manage user accounts and assign roles
    - Can view all sales reports and analytics
2. **Cashier/Staff**
    - Limited access focused on daily operations
    - Can create new sales transactions
    - Can view product and customer information
    - Cannot modify products or access sensitive settings

1.4 Use Cases

UC1: Product & Inventory Management

- Administrator adds new products with details (name, SKU, price, stock quantity, category)
- Administrator updates product information or stock levels
- Administrator deletes products that are no longer sold
- System filters out inactive products from product lists

UC2: Sales Transaction

- Cashier selects customer (optional) and products to sell
- Cashier enters quantity for each product
- System validates stock availability
- System calculates total amount and updates inventory automatically
- System records the sale with timestamp and operator information

UC3: Customer Management

- Staff adds new customer details (name, phone number)
- System tracks customer purchase history through sales records
- Administrator can update or remove customer information

UC4: Inventory Monitoring

- System displays current stock levels for all products
- System alerts when product quantity falls below threshold
- Administrator receives low-stock notifications on dashboard

UC5: Sales Reporting

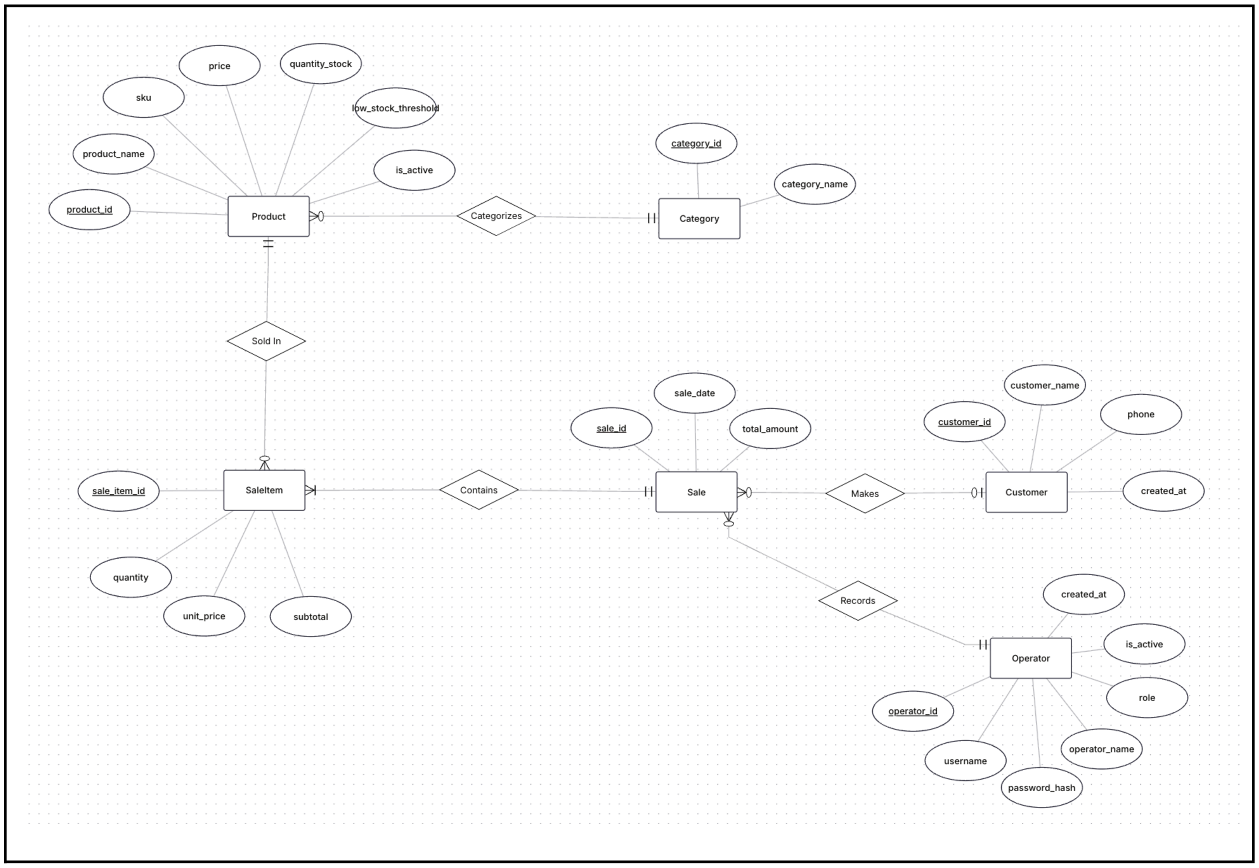
- Administrator views total revenue over time
- System displays recent sales transactions
- System shows top-selling products
- Administrator generates reports for business analysis

UC6: User Authentication

- Users log in with username and password
- System verifies credentials and grants access based on role
- System restricts features according to user permissions (admin vs. cashier

II. Database Design

2.1 Entity-Relationship Diagram (ERD)



2.2 Entity Explanations

The database design consists of several core entities that represent the main components of the inventory and sales system.

- **User**
  The User entity stores system user accounts, including administrators and cashiers. Each user has a unique username, a hashed password, and a role attribute that determines access privileges. This entity supports authentication and role-based authorization.
- **Category**
  The Category entity represents product classifications such as Electronics or Stationery. It is used

to group products logically and simplify inventory organization. Each category has a unique identifier and a category name.

- **Product**
  The Product entity stores information about items sold by the business. Attributes include SKU, product name, price, current stock quantity, low-stock threshold, and a foreign key referencing Category. This entity supports inventory tracking and stock monitoring.
- **Customer**
  The Customer entity stores customer information such as name, phone number, and creation date. Customers are optionally associated with sales transactions, allowing the system to track purchase history while still supporting walk-in customers.
- **Sale**
  The Sale entity represents a completed sales transaction. It records the transaction date and time, total amount, operator (user) who processed the sale, and the associated customer if applicable. This entity acts as the parent record for detailed sale items.
- **SaleItem**
  The SaleItem entity represents individual products within a sale. It stores the quantity sold, unit price at the time of sale, and subtotal. This separation ensures historical pricing accuracy and supports multi-item transactions.

2.3 Normalization Steps

The database was normalized to **Third Normal Form (3NF)** to eliminate redundancy and ensure data integrity.

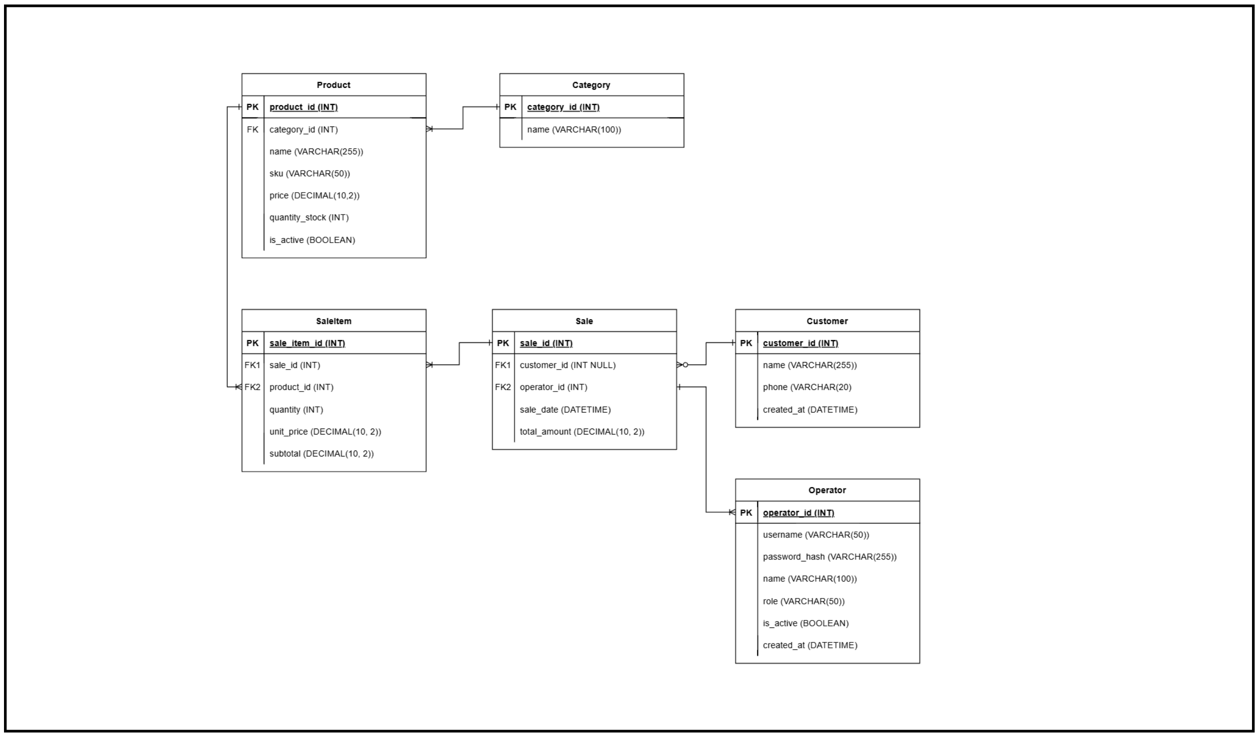
- **First Normal Form (1NF)**
  All tables contain atomic values with no repeating groups. Each table has a primary key, and attributes store single values only (e.g., quantity, price).
- **Second Normal Form (2NF)**
  All non-key attributes are fully dependent on the entire primary key. Composite keys are avoided by introducing surrogate keys where necessary, such as in the SaleItem table.
- **Third Normal Form (3NF)**
  There are no transitive dependencies. Each non-key attribute depends only on the primary key.

For example, category names are stored only in the Category table and referenced via foreign keys from Product.

This normalization ensures consistency, minimizes redundancy, and simplifies maintenance.

III. Database Implementation

3.1 Relational Schema



3.2 Key Constraints

The following constraints are applied to ensure data integrity:

- **Primary Keys (PK)**
  Each table has a primary key to uniquely identify records.

- **Foreign Keys (FK)**

- products.category_id → categories.category_id
- sales.user_id → users.user_id
- sales.customer_id → customers.customer_id
- sale_items.sale_id → sales.sale_id
- sale_items.product_id → products.product_id

- **Unique Constraints**

- usernames must be unique
- SKUs must be unique
- category names must be unique

- **NOT NULL Constraints**
- Applied to essential attributes such as usernames, product names, prices, and quantities.

These constraints prevent invalid data insertion and preserve relational consistency.

3.3 Example SQL Statements

1. **Insert a New Category**

```
INSERT INTO categories (category_name)
VALUES ('Electronics');
```

What this does:
- This statement adds a new product category into the categories table.
- The category_name column stores the name of the category.
- The category_id is automatically generated because it is the primary key (usually using SERIAL or AUTO INCREMENT).

Why it is needed in the system:

- Categories help organize products.
- Products must reference an existing category through a foreign key.
- This ensures data consistency and prevents products from belonging to non-existent categories.

2. Insert a New Product

```
INSERT INTO products (sku, product_name, price, stock_quantity, low_stock_threshold,
category_id)
VALUES ('ELEC-001', 'Wireless Mouse', 15.00, 20, 10, 1);
```

What this does:

- Inserts a new product into the inventory.
- Each column represents a key attribute:
    - sku: Unique identifier for the product.
    - product_name: Name shown in the system.
    - price: Selling price.
    - stock_quantity: Current inventory count.
    - low_stock_threshold: Used for low-stock alerts.
    - category_id: Links the product to the Electronics category.

Important behavior:

- The category_id must already exist in the categories table.
- If it does not exist, the database rejects the insert due to the foreign key constraint.

Why it is needed in the system:

- This supports inventory management and stock monitoring.
- The low-stock threshold enables automatic alerts on the dashboard.

3. Create a Sale Record

```
INSERT INTO sales (sale_date, total_amount, user_id, customer_id)
VALUES (NOW(), 30.00, 1, 2);
```

What this does:

- Creates a new sales transaction record.
- Stores high-level transaction information:
    - sale_date: Timestamp of the transaction.
    - total_amount: Total cost of all items in the sale.
    - user_id: Identifies the cashier or admin who processed the sale.
    - customer_id: Links the sale to a customer (can be NULL for walk-in customers).

Important behavior:

- user_id must exist in the users table.
- customer_id must exist in the customers table or be null.
- This table acts as the parent record for detailed sale items.

Why it is needed in the system:

- Enables reporting such as total revenue and sales history.
- Separates transaction metadata from item details.

IV. Application Implementation

4.1 System Architecture

The following table shows the technology stack used in this system:

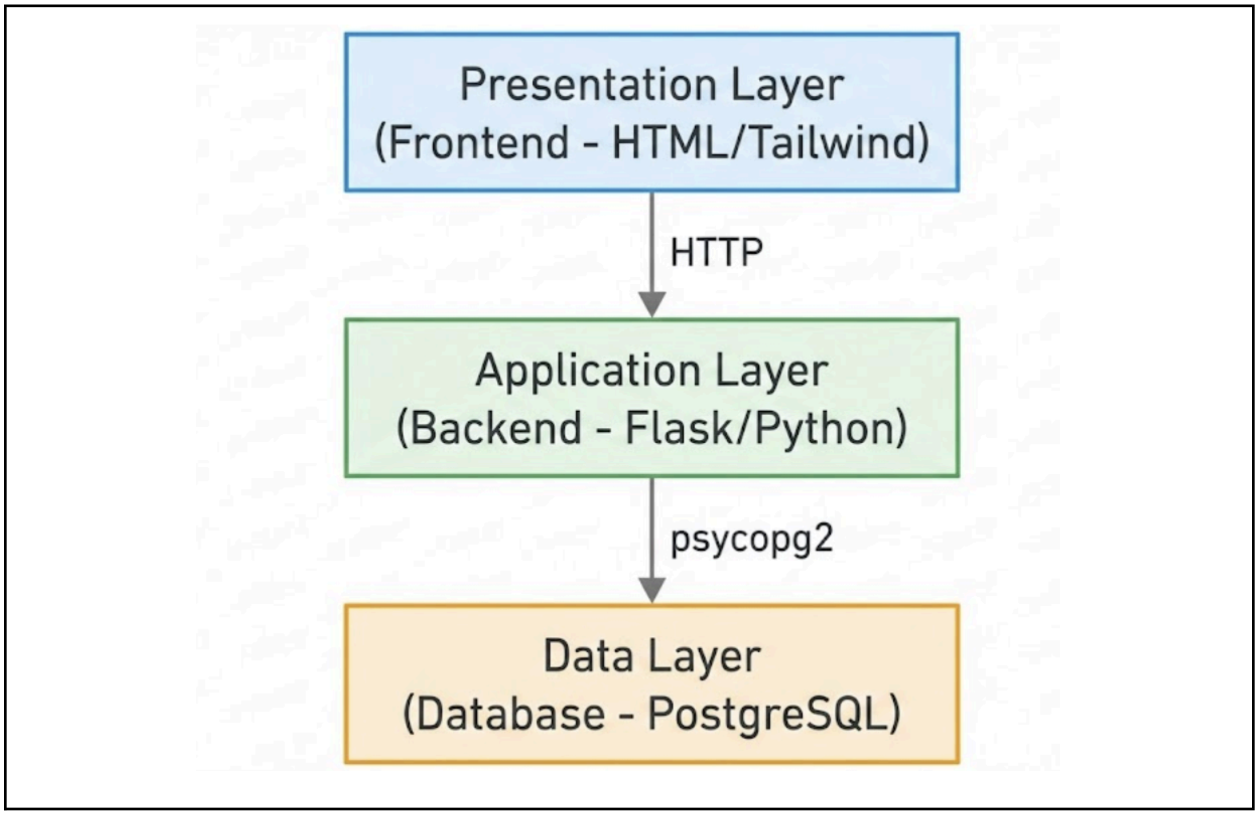
Component	Technology	Purpose
Database	PostgreSQL (Neon DB)	Cloud-hosted relational database for data storage
Backend	Python Flask	Web application framework for routing and business logic
Database Driver	psycopg2	PostgreSQL adapter for Python database connection

Authentication	Flask-Login + bcrypt	User session management and password hashing
Frontend	HTML + Tailwind CSS	User interface with responsive design

This system follows a three-tier architecture pattern:

1. **Presentation Layer (Frontend)**
    - User interface built with HTML and Tailwind CSS
    - Displays data and receives user input through web browser
2. **Application Layer (Backend)**
    - Flask handles HTTP requests and business logic
    - Processes user actions (CRUD operations, authentication)
    - Communicates between frontend and database
3. **Data Layer (Database)**
    - PostgreSQL stores all persistent data
    - Manages products, sales, customers, and users
    - Ensures data integrity with constraints and relationships

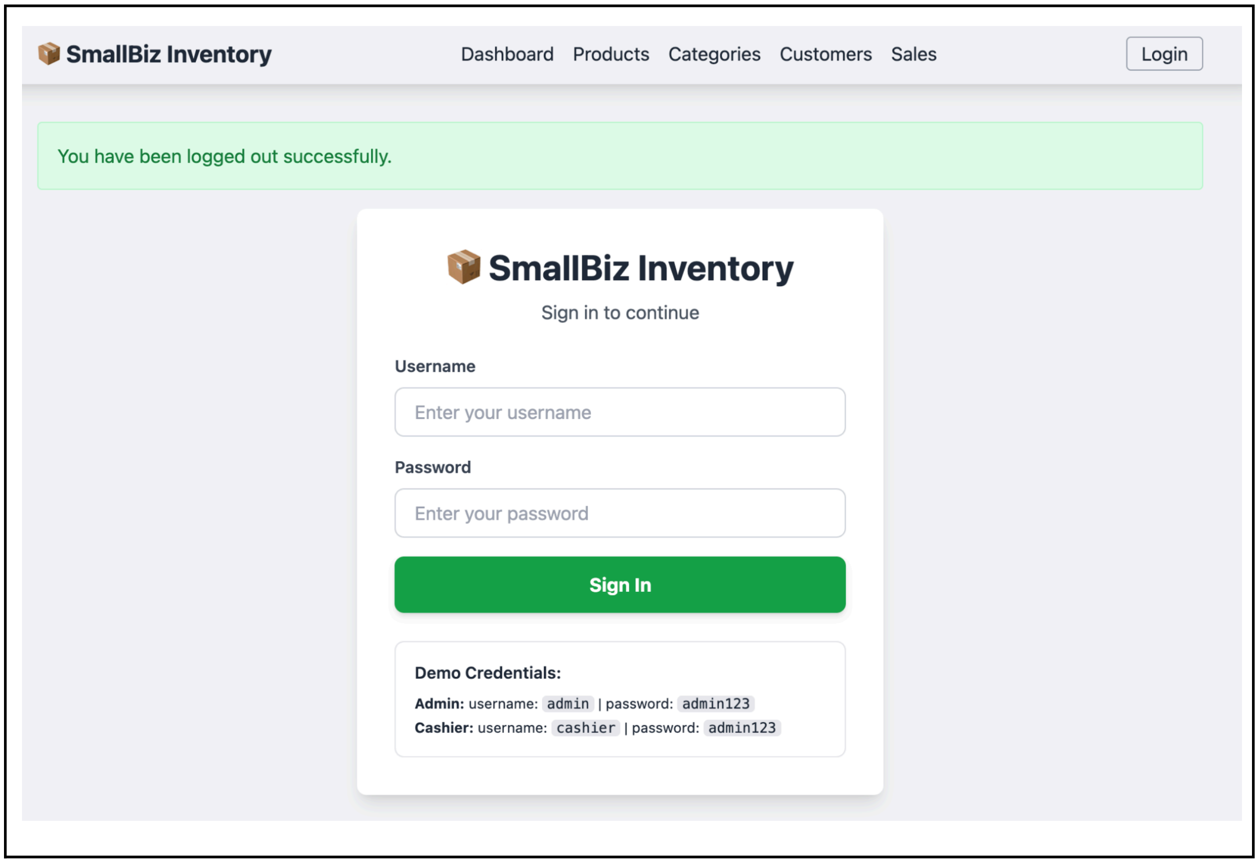
This architecture ensures that the frontend remains responsive while the backend handles complex database operations.



4.2 Authentication & Authorization

This module handles user login, session management, and role-based access control.

Login Page:



You have been logged out successfully.

SmallBiz Inventory

Sign in to continue

Username

Enter your username

Password

Enter your password

Sign In

Demo Credentials:

Admin: username: `admin` | password: `admin123`

Cashier: username: `cashier` | password: `admin123`

The login page allows users to enter their credentials. The system uses bcrypt for secure password hashing, and Flask-Login for session management

Role-Based Access Control:

The system implements two user roles:

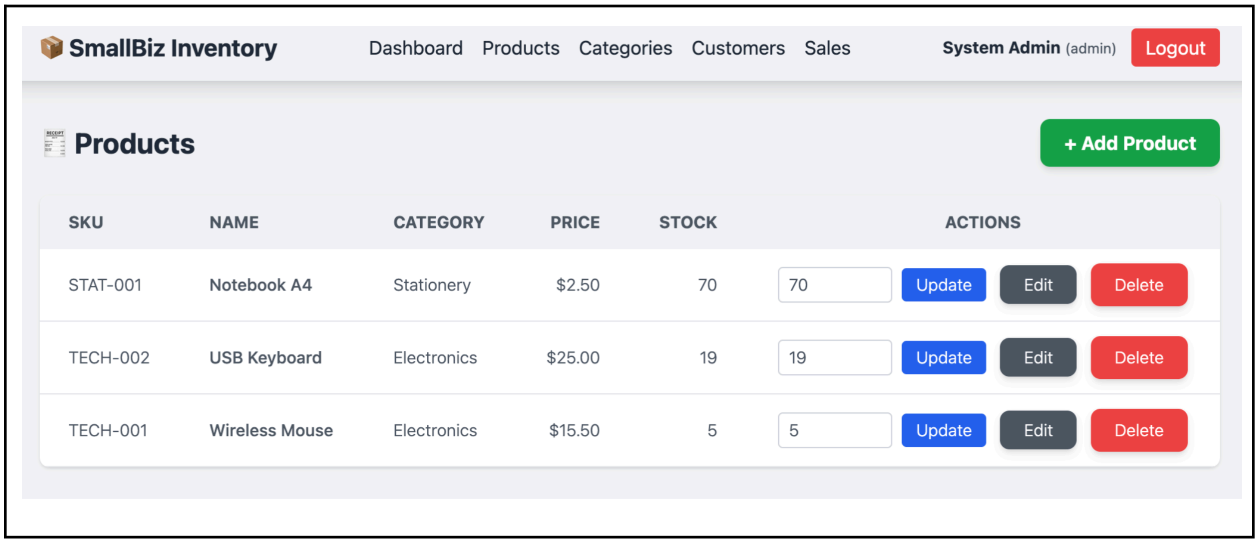
1. **Admin:**
    a. Full access to all features
    b. Can add, edit, delete products, categories, and customers
2. **Cashier:**
    a. Can create sales transactions
    b. Can view products and customers
    c. Cannot modify data (view-only access)

Routes are protected using:

- `@login_required` - Requires user login
- `@role_required('admin')` - Restricts to admin users only

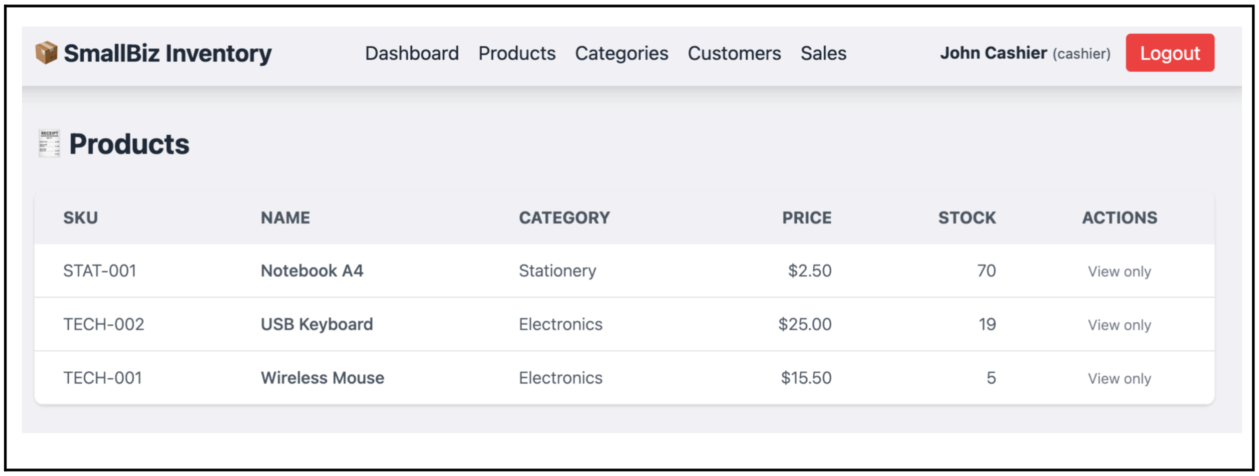
Access Control Example (Products Page):

Admin:



SKU	NAME	CATEGORY	PRICE	STOCK	ACTIONS
STAT-001	Notebook A4	Stationery	$2.50	70	70 Update Edit Delete
TECH-002	USB Keyboard	Electronics	$25.00	19	19 Update Edit Delete
TECH-001	Wireless Mouse	Electronics	$15.50	5	5 Update Edit Delete

Cashier:

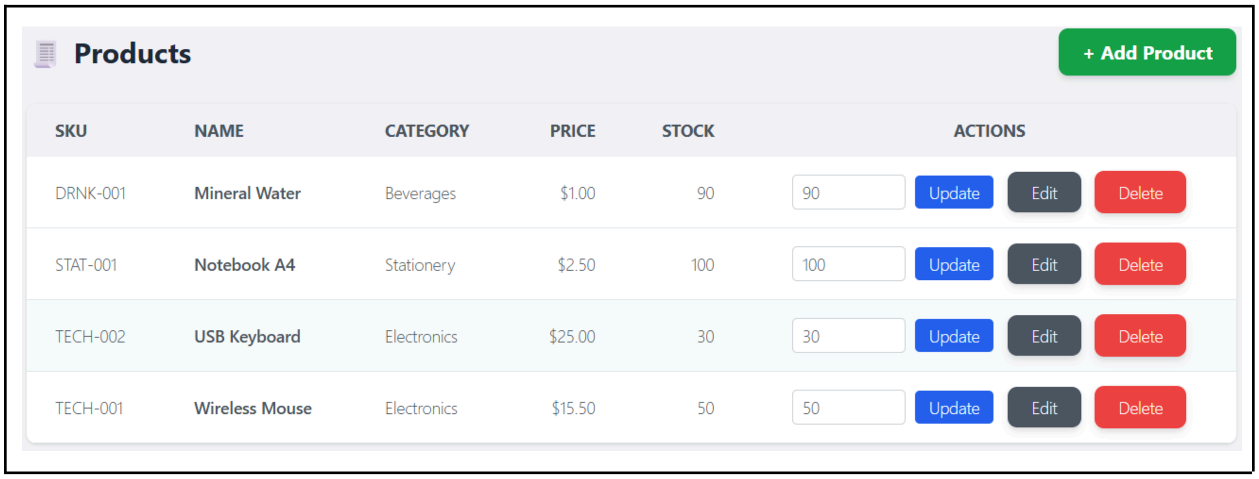


SKU	NAME	CATEGORY	PRICE	STOCK	ACTIONS
STAT-001	Notebook A4	Stationery	$2.50	70	View only
TECH-002	USB Keyboard	Electronics	$25.00	19	View only
TECH-001	Wireless Mouse	Electronics	$15.50	5	View only

4.3 Product Management Module

This module allows administrators to manage products.

Product List Page



Products

+ Add Product

SKU	NAME	CATEGORY	PRICE	STOCK	ACTIONS
DRNK-001	Mineral Water	Beverages	$1.00	90	90 Update Edit Delete
STAT-001	Notebook A4	Stationery	$2.50	100	100 Update Edit Delete
TECH-002	USB Keyboard	Electronics	$25.00	30	30 Update Edit Delete
TECH-001	Wireless Mouse	Electronics	$15.50	50	50 Update Edit Delete

The product list page displays all product categories with the following features:

- Product SKU, name, category, price, and stock
- Add new product button (admin only)
- Update stock, edit, and delete buttons for each product (admin only)

Add Category

Administrators can add new products by:

1. Clicking the "+ Add Product" button
2. Input the appropriate information into each slot.
   This includes the product name, a new SKU, selecting from pre-existing category list, price, and current stock.
3. Submit the form.

Update Category

Administrators can easily update the current stock of a product by:

1. Typing in the update stock count to the associated product.
2. Using the arrows to increment or decrement the stock count by 1.

Edit Category

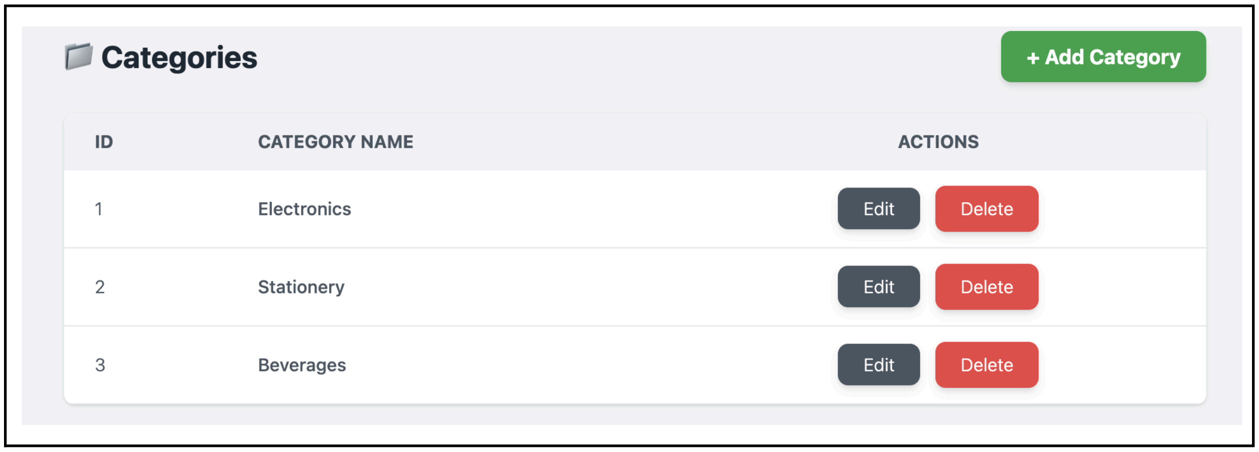
Administrators can update existing products by:

1. Clicking on the edit button on the product list.
2. Modifying the information.
3. Save changes.

4.4 Category Management Module

This module allows administrators to manage product categories for better organization and classification.

Category List Page



Categories

+ Add Category

ID	CATEGORY NAME	ACTIONS
1	Electronics	Edit Delete
2	Stationery	Edit Delete
3	Beverages	Edit Delete

The category list page displays all product categories with the following features:

- Category ID and name
- Add new category button (admin only)
- Edit and delete buttons for each category (admin only)

Add Category

Administrators can create new categories by:

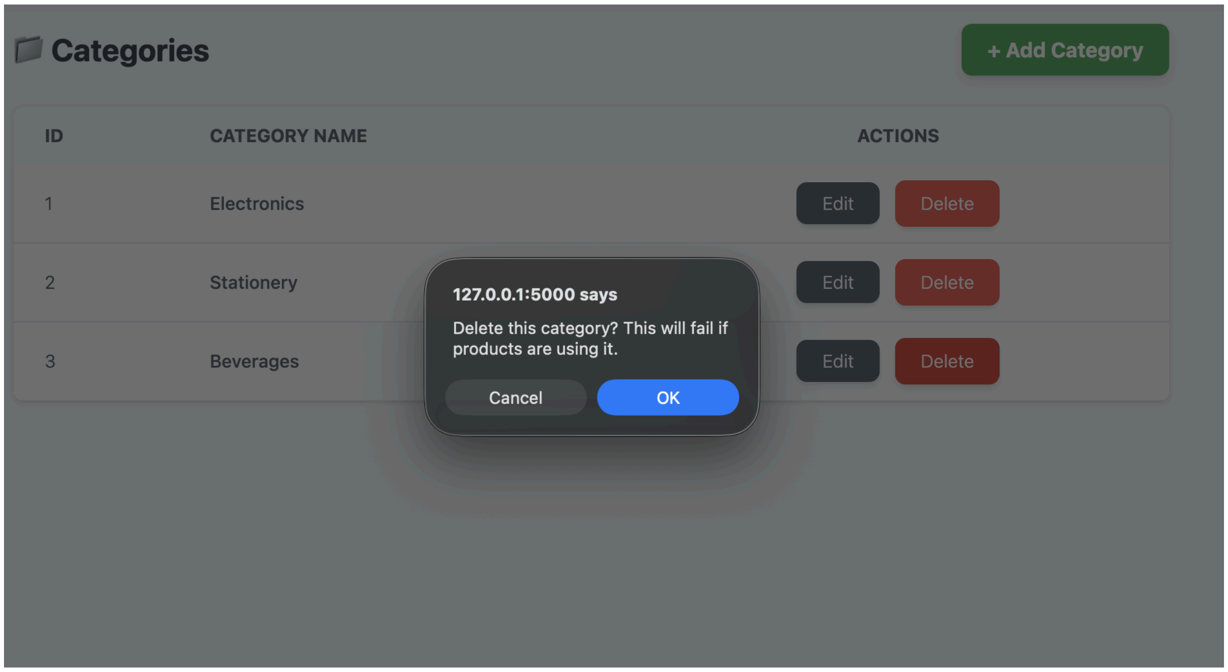
1. Click the "+ Add Category" button
2. Input the category name
3. Submit the form

Edit Category

Administrators can update existing categories:

1. Click "Edit" button on the category list
2. Modify the category name
3. Save changes

Delete Category



Categories

+ Add Category

ID	CATEGORY NAME	ACTIONS
1	Electronics	Edit Delete
2	Stationery	Edit Delete
3	Beverages	Edit Delete

127.0.0.1:5000 says

Delete this category? This will fail if products are using it.

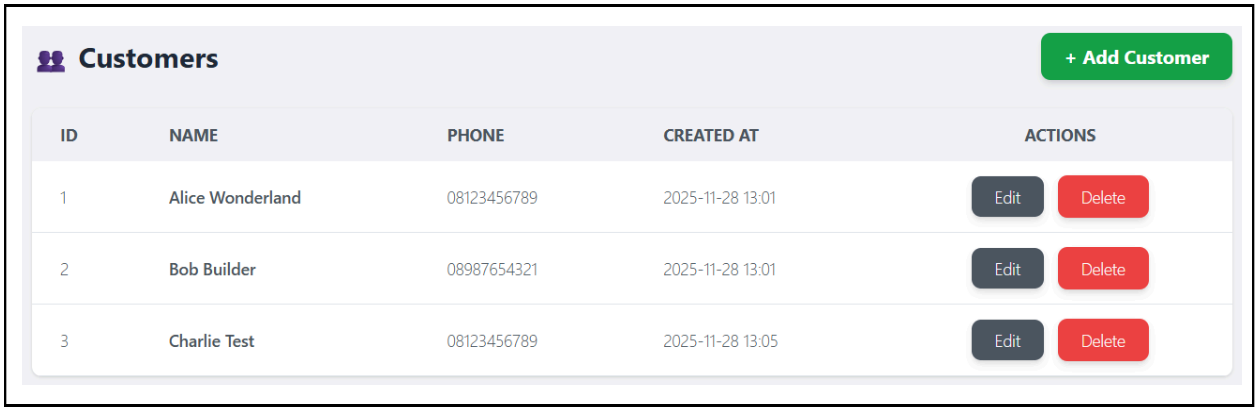
Administrators can remove categories, but with a safety check:

1. Click the "Delete" button on the category list.
2. The system checks if any products are currently assigned to this category.
3. If products exist, the system prevents deletion and displays an error to ensure data integrity. If empty, the category is deleted.

4.5 Customer Management Module

This module allows administrators to manage customers.

Customer List Page



ID	NAME	PHONE	CREATED AT	ACTIONS
1	Alice Wonderland	08123456789	2025-11-28 13:01	Edit Delete
2	Bob Builder	08987654321	2025-11-28 13:01	Edit Delete
3	Charlie Test	08123456789	2025-11-28 13:05	Edit Delete

The product list page displays all product categories with the following features:

- Customer ID, name, phone, date of creation
- Add new customer button (admin only)
- Edit and delete buttons for each customer (admin only)

Add Customer

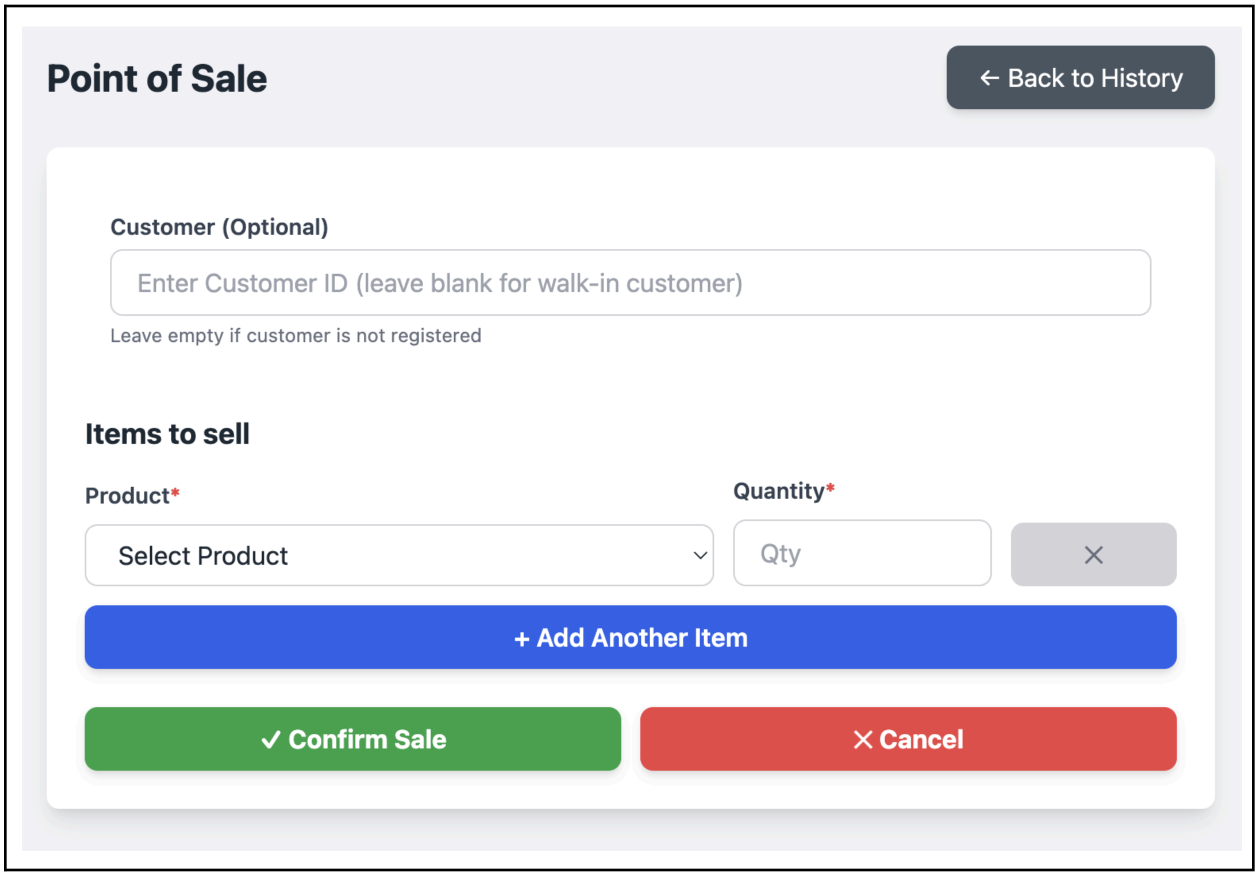
Administrators can create new categories by:

1. Click the "+ Add Customer" button
2. Input the customer name & phone number
3. Submit the form

4.6 Sales Management Module

This module handles sales transactions, inventory updates, and sales history tracking for both administrators and cashiers.

Create New Sale



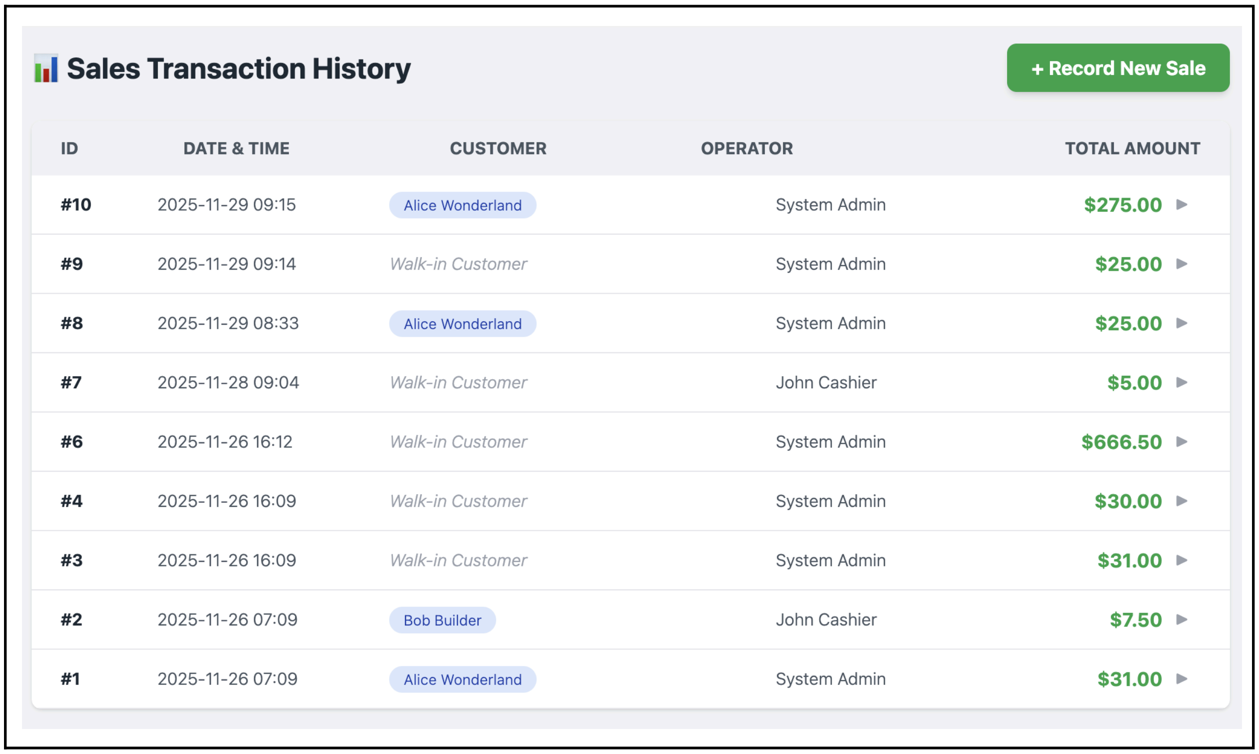
1. Select customer (optional) from dropdown
2. Adding products to cart with quantity
3. System calculates subtotal for each item
4. System shows total amount
5. Click "Complete Sale" to finalize transaction

The system performs the following operations automatically:

- **Stock validation**: Checks if sufficient quantity is available
- **Inventory update**: Decreases product stock by sold quantity
- **Total calculation**: Computes sale total from all items
- **Transaction recording**: Saves sale with timestamp and operator information

- **Error Validation:** Cannot proceed if stock is insufficient
    - Not enough stock for 'Product Name'. Available: 10, Requested: 15 - Prevents overselling
    - Transaction rollback on any failure ensures data consistency

Sales History



ID	DATE & TIME	CUSTOMER	OPERATOR	TOTAL AMOUNT
#10	2025-11-29 09:15	Alice Wonderland	System Admin	$275.00 ▸
#9	2025-11-29 09:14	*Walk-in Customer*	System Admin	$25.00 ▸
#8	2025-11-29 08:33	Alice Wonderland	System Admin	$25.00 ▸
#7	2025-11-28 09:04	*Walk-in Customer*	John Cashier	$5.00 ▸
#6	2025-11-26 16:12	*Walk-in Customer*	System Admin	$666.50 ▸
#4	2025-11-26 16:09	*Walk-in Customer*	System Admin	$30.00 ▸
#3	2025-11-26 16:09	*Walk-in Customer*	System Admin	$31.00 ▸
#2	2025-11-26 07:09	Bob Builder	John Cashier	$7.50 ▸
#1	2025-11-26 07:09	Alice Wonderland	System Admin	$31.00 ▸

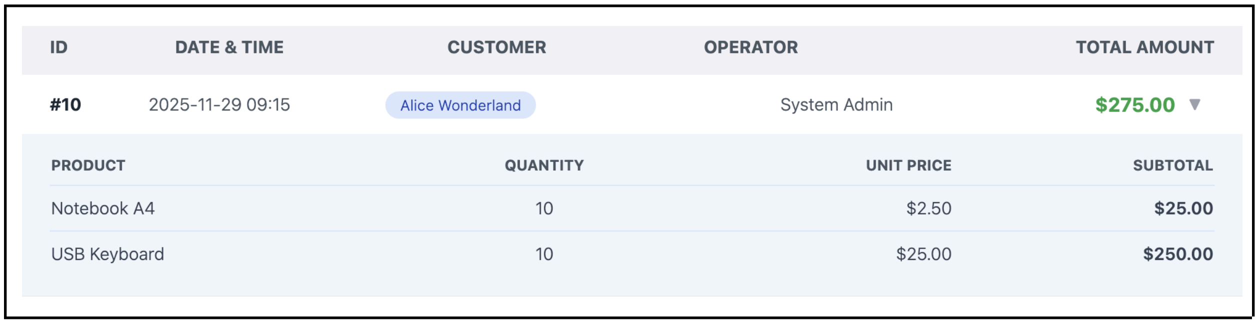
Each transaction is shown as a single row in the table, acting as a summary view. The sales history page displays all completed transactions with:

- Sale ID and transaction date/time
- Operator name (who processed the sale)
- Customer name (if provided, otherwise shows “Walk-in Customer”)
- Total amount

Clicking a row expands it inline to reveal the detailed breakdown of the sale (items, quantities, unit prices, subtotals), instead of navigating via a separate “View Details” page.

Also, both administrators and cashiers can view the complete sales history.

Sale Details



ID	DATE & TIME	CUSTOMER	OPERATOR	TOTAL AMOUNT
#10	2025-11-29 09:15	Alice Wonderland	System Admin	$275.00 ▼

PRODUCT	QUANTITY	UNIT PRICE	SUBTOTAL
Notebook A4	10	$2.50	$25.00
USB Keyboard	10	$25.00	$250.00

Clicking on a sale expands the row to reveal the full transaction details. The expanded view displays an items table showing:

- Product name
- Quantity
- Unit price
- Subtotal

This detailed view helps several important use cases:

- Makes it easy to see which products were sold together
- Preserves historical pricing as it was at the time of the transaction

Transaction Details

Transaction Process:

1. The system validates all product quantities against current stock levels.
2. A Sale record is created with operator and customer information.
3. Corresponding SaleItem records are created for each product in the cart.
4. Product stock levels are decreased automatically based on the sale.
5. All operations are executed within a single database transaction and are either fully committed or rolled back if any error occurs.

ACID Compliance:

The sale process is executed within a single database transaction, ensuring that all related operations (sale creation, sale items insertion, and stock updates) are committed atomically or rolled back entirely in case of an error. This guarantees data consistency and prevents partial or conflicting sales records.

Access Control:

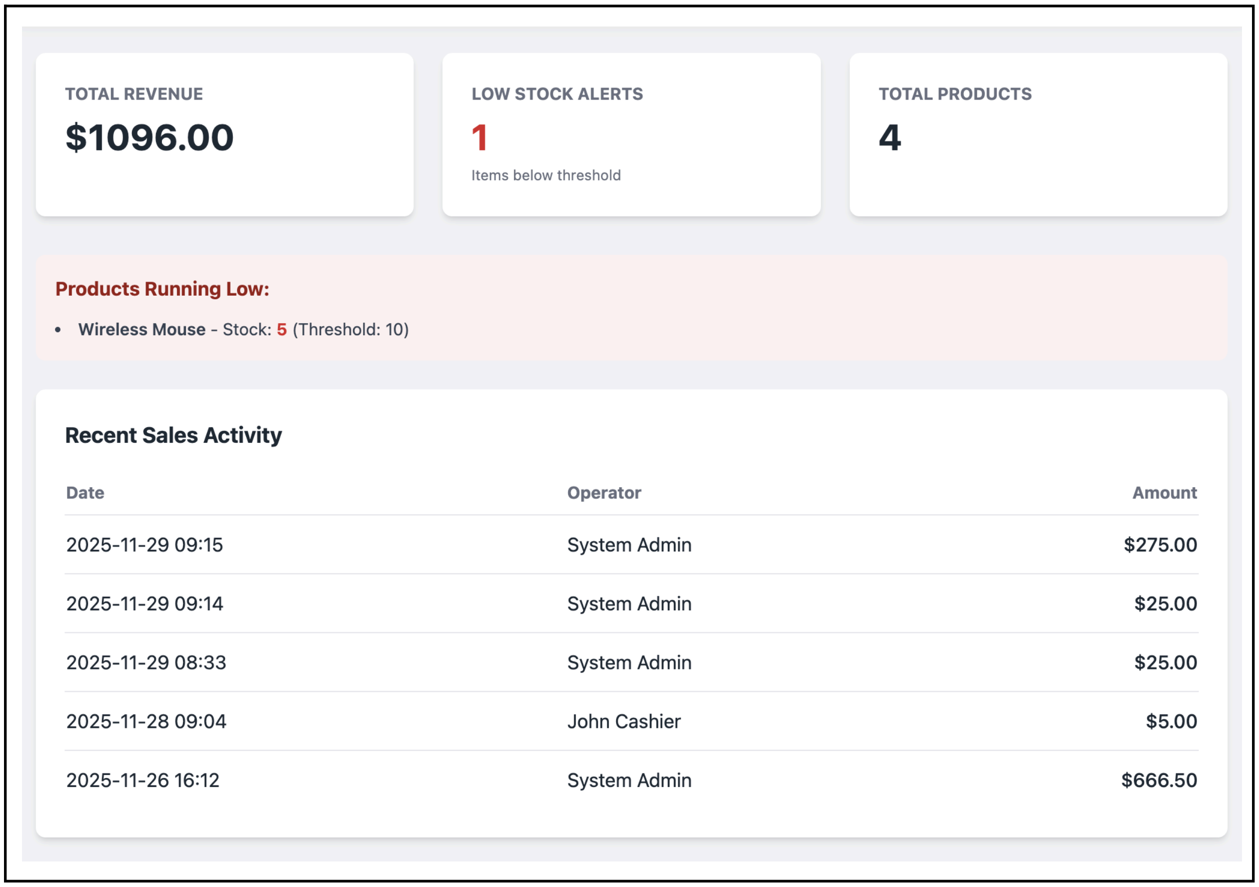
- **Administrator:** Can view all sales across all operators.

- **Cashier:** Can create new sales and view sales history.

4.7 Dashboard & Reporting Module

This module provides real-time business insights and performance monitoring for administrators.

Dashboard Overview



TOTAL REVENUE
$1096.00

LOW STOCK ALERTS
1
Items below threshold

TOTAL PRODUCTS
4

Products Running Low:

- **Wireless Mouse** - Stock: 5 (Threshold: 10)

Recent Sales Activity

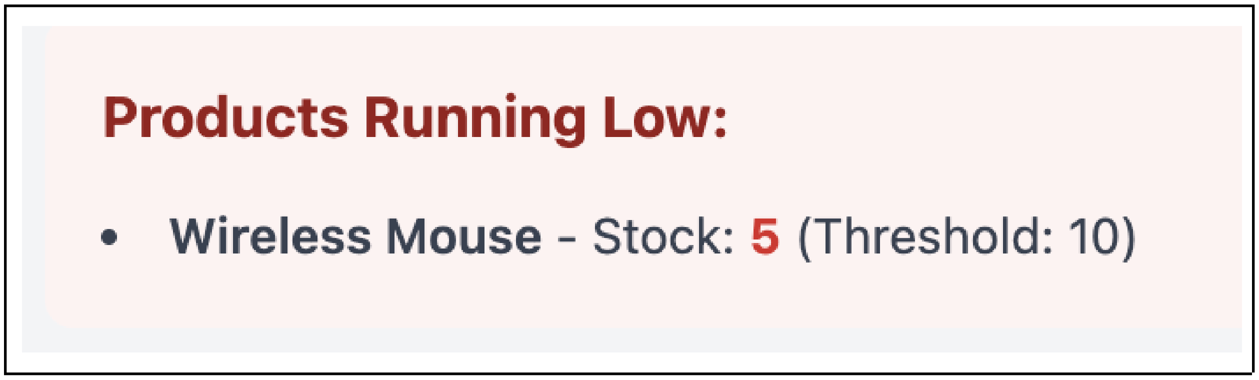
Date	Operator	Amount
2025-11-29 09:15	System Admin	$275.00
2025-11-29 09:14	System Admin	$25.00
2025-11-29 08:33	System Admin	$25.00
2025-11-28 09:04	John Cashier	$5.00
2025-11-26 16:12	System Admin	$666.50

The dashboard displays key business metrics at a glance:

- **Total Revenue:** Sum of all sales transactions
- **Low Stock Alerts:** Number of products below threshold
- **Total Products:** Count of active products in inventory
- **Recent Sales:** List of the 5 most recent transactions

This centralized view helps administrators quickly assess business performance and identify issues requiring attention.

Low Stock Monitoring



Products Running Low:

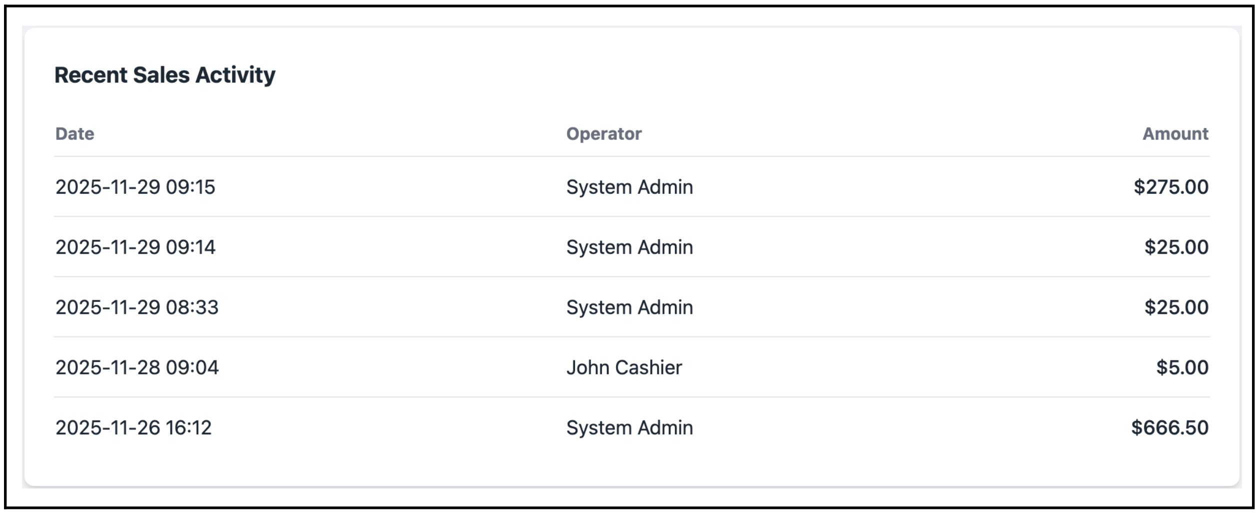
- Wireless Mouse - Stock: 5 (Threshold: 10)

The dashboard displays products requiring restocking:

- Product Name
- Current stock quantity
- Low stock threshold.

The 'Low Stock' widget automatically filters the inventory to show only items where the current stock is at or below the defined threshold (e.g., 10 units).

Recent Sales Activity



Recent Sales Activity

Date	Operator	Amount
2025-11-29 09:15	System Admin	$275.00
2025-11-29 09:14	System Admin	$25.00
2025-11-29 08:33	System Admin	$25.00
2025-11-28 09:04	John Cashier	$5.00
2025-11-26 16:12	System Admin	$666.50

The recent sales widget shows the latest 5 transactions with:

- Transaction date and time
- Operator name
- Total amount

Implementation Details

The dashboard uses optimized SQL aggregation queries to compute statistics directly in the database, such as total revenue, low-stock counts, and recent sales. Appropriate indexes are applied to ensure efficient query execution, allowing the dashboard to load within acceptable response times under typical data volumes.

Access Control

Available to all logged-in users, both admin and cashier, which means all users see the same dashboard information.

V. Testing and Results

5.1 Transaction Logic Test

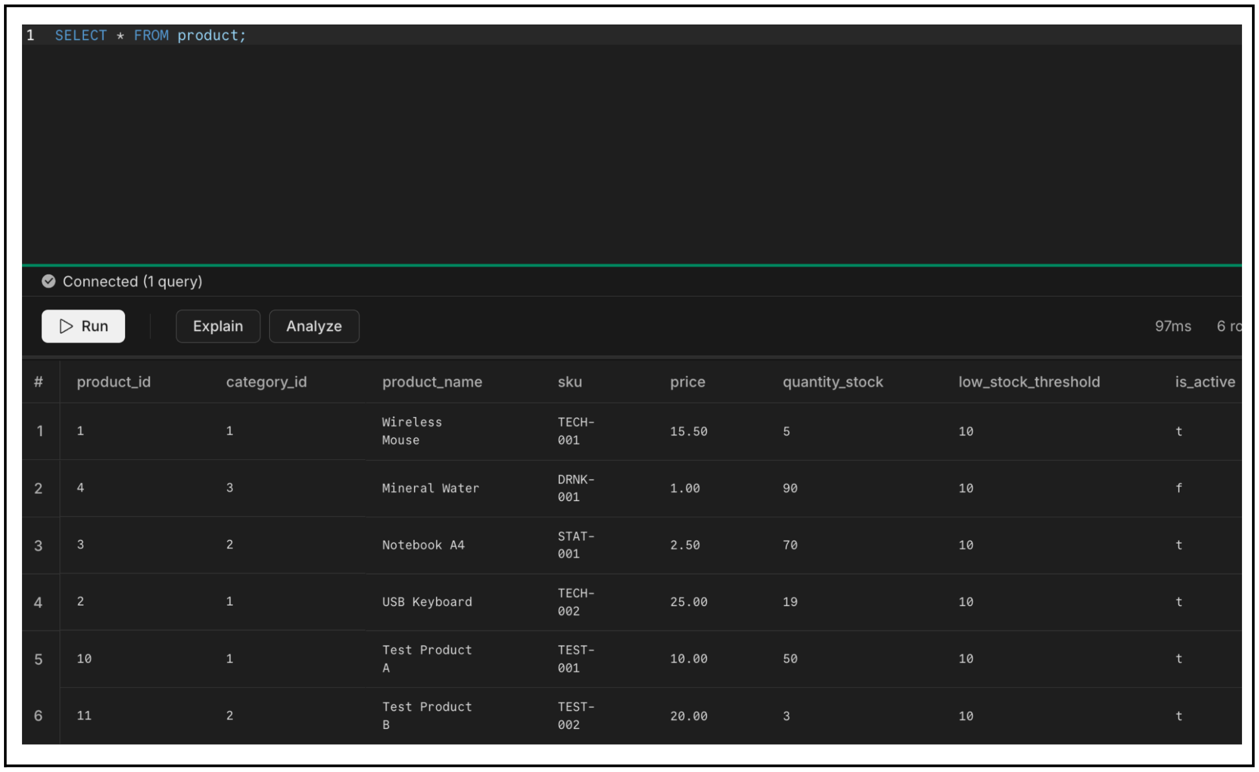
This section checks whether the Sales Module works correctly and whether it handles errors properly when there is abnormal input, such as insufficient stock.

Test Data Setup:

To make the verification easier to understand, the following test products are registered and used.

Product Name	SKU	Price	Initial Stock
Test Product A	TEST-001	$10.00	50 (enough stock)
Test Product B	TEST-002	$20.00	3 (low stock)

Product Table:



```
SELECT * FROM product;
```

#	product_id	category_id	product_name	sku	price	quantity_stock	low_stock_threshold	is_active
1	1	1	Wireless Mouse	TECH-001	15.50	5	10	t
2	4	3	Mineral Water	DRNK-001	1.00	90	10	f
3	3	2	Notebook A4	STAT-001	2.50	70	10	t
4	2	1	USB Keyboard	TECH-002	25.00	19	10	t
5	10	1	Test Product A	TEST-001	10.00	50	10	t
6	11	2	Test Product B	TEST-002	20.00	3	10	t

Scenario A: Multi-Item Sale

Purpose:

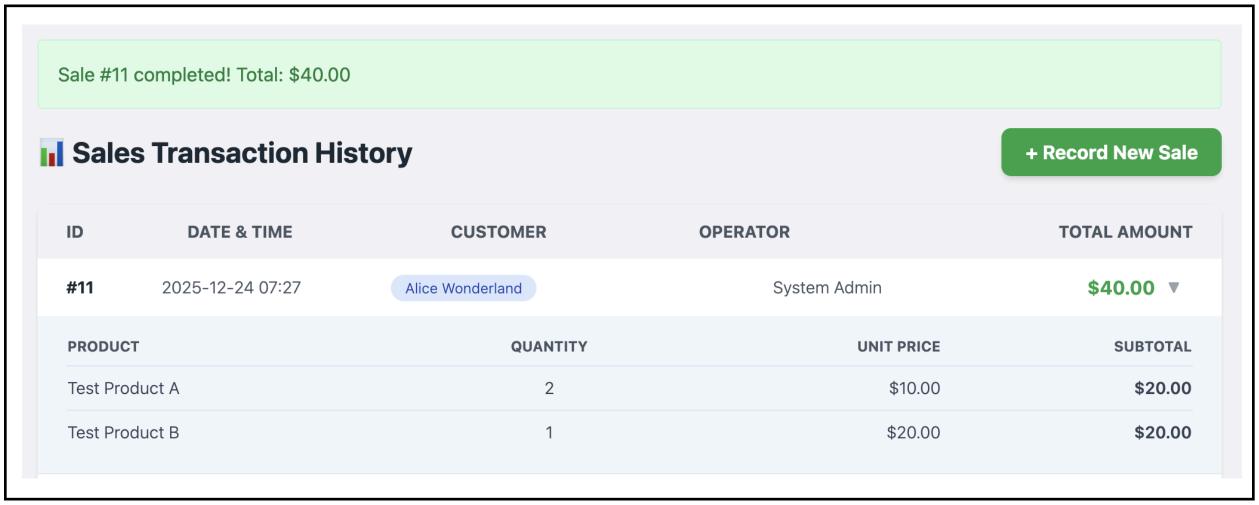
To confirm that a transaction is completed successfully when purchasing a product with sufficient stock.

Procedure:

1. Enter Customer ID “1”. (= Alice Wonderland)
2. Add to Cart:
    a. Test Product A × 2
    b. Test Product B × 1
3. Click “Confirm Sale” button.

Expected Result:

Application Side:

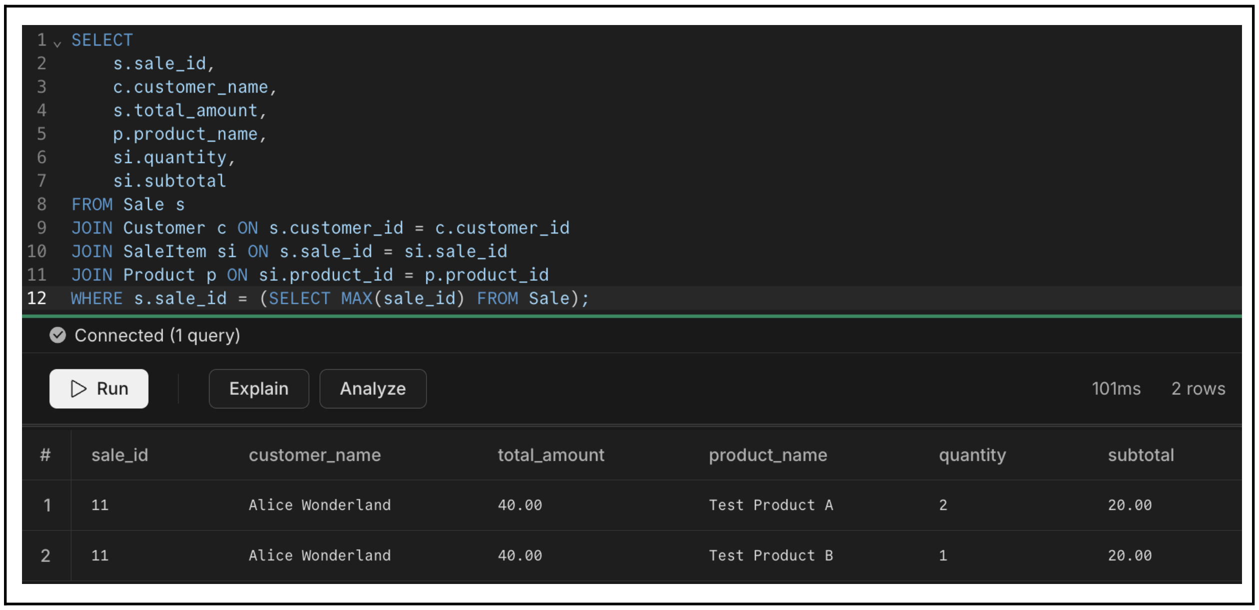

Sale #11 completed! Total: $40.00

📊 Sales Transaction History

ID	DATE & TIME	CUSTOMER	OPERATOR	TOTAL AMOUNT
#11	2025-12-24 07:27	Alice Wonderland	System Admin	$40.00 ▼

PRODUCT	QUANTITY	UNIT PRICE	SUBTOTAL
Test Product A	2	$10.00	$20.00
Test Product B	1	$20.00	$20.00

Database Side:


```
SELECT
    s.sale_id,
    c.customer_name,
    s.total_amount,
    p.product_name,
    si.quantity,
    si.subtotal
FROM Sale s
JOIN Customer c ON s.customer_id = c.customer_id
JOIN SaleItem si ON s.sale_id = si.sale_id
JOIN Product p ON si.product_id = p.product_id
WHERE s.sale_id = (SELECT MAX(sale_id) FROM Sale);
```

Connected (1 query)

101ms 2 rows

#	sale_id	customer_name	total_amount	product_name	quantity	subtotal
1	11	Alice Wonderland	40.00	Test Product A	2	20.00
2	11	Alice Wonderland	40.00	Test Product B	1	20.00

Observation:

The database query results exactly match the application interface. This confirms that:

- Foreign Keys correctly linked the customer (Alice) and products to the sale.
- Data Integrity is maintained across the Sale and SaleItem tables.
- Calculations for subtotals and total amounts are accurate in both the frontend and backend.

Scenario B: Insufficient Stock Handling

Purpose:

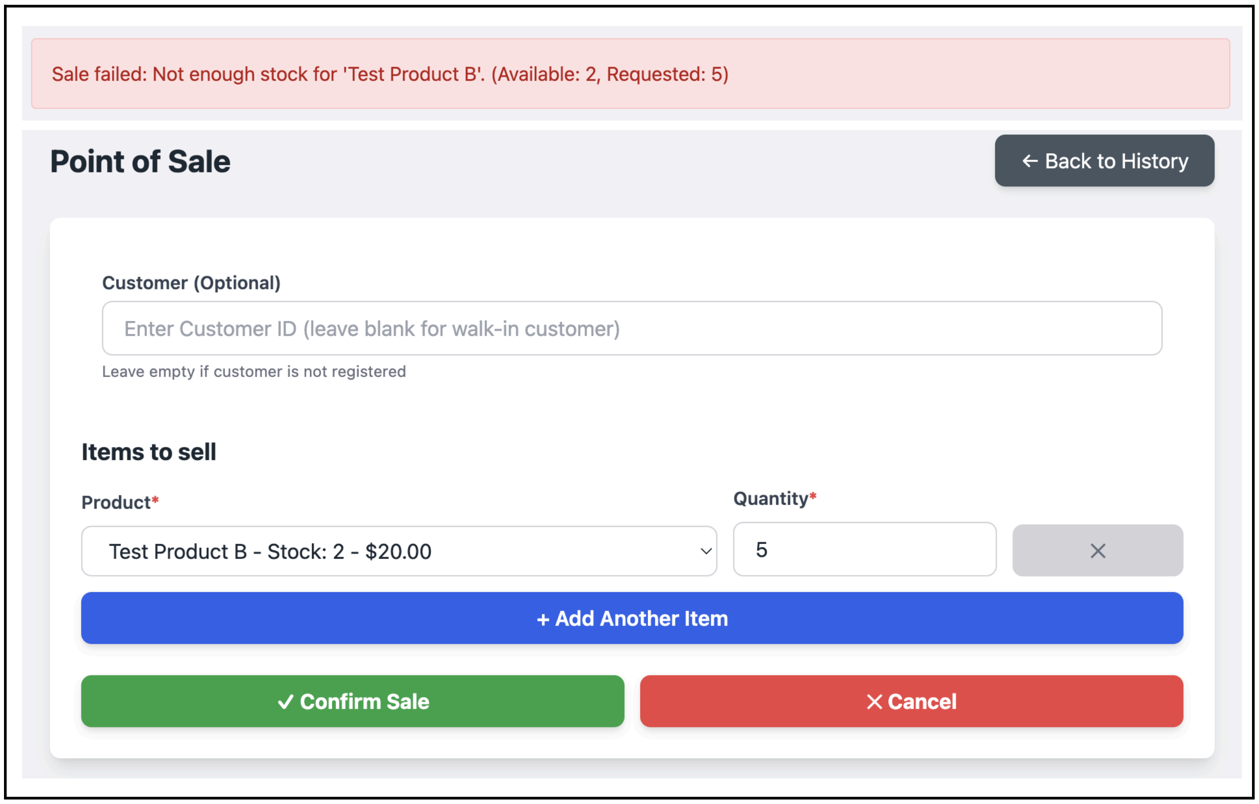
To confirm that the system returns an error and prevents the sale when the order quantity exceeds available stock, ensuring data consistency.

Procedure:

4. Select Test Product B (Stock: 2).
5. Add to Cart: Test Product B × 5
6. Click "Confirm Sale" button.

Result:

Application Side:



Sale failed: Not enough stock for 'Test Product B'. (Available: 2, Requested: 5)

Point of Sale

Customer (Optional)

Enter Customer ID (leave blank for walk-in customer)

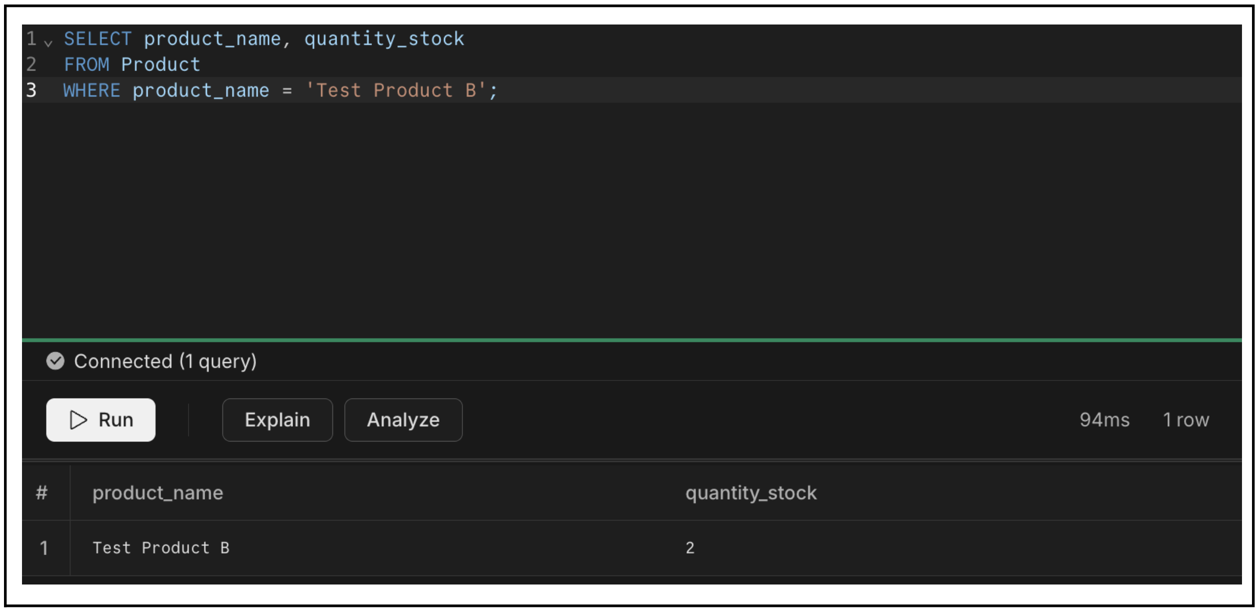
Leave empty if customer is not registered

Items to sell

Product*: Test Product B - Stock: 2 - $20.00

Quantity*: 5

Database Side:



```sql
SELECT product_name, quantity_stock
FROM Product
WHERE product_name = 'Test Product B';
```

```
#   product_name      quantity_stock
1   Test Product B    2
```

Observation:

The system correctly blocked the transaction because the requested quantity “5” exceeded the available stock “2”. The database query confirms that the stock level remained unchanged at 2, proving that the transaction was fully rolled back and data integrity was preserved.

5.2 Inventory Update Verification

Purpose:

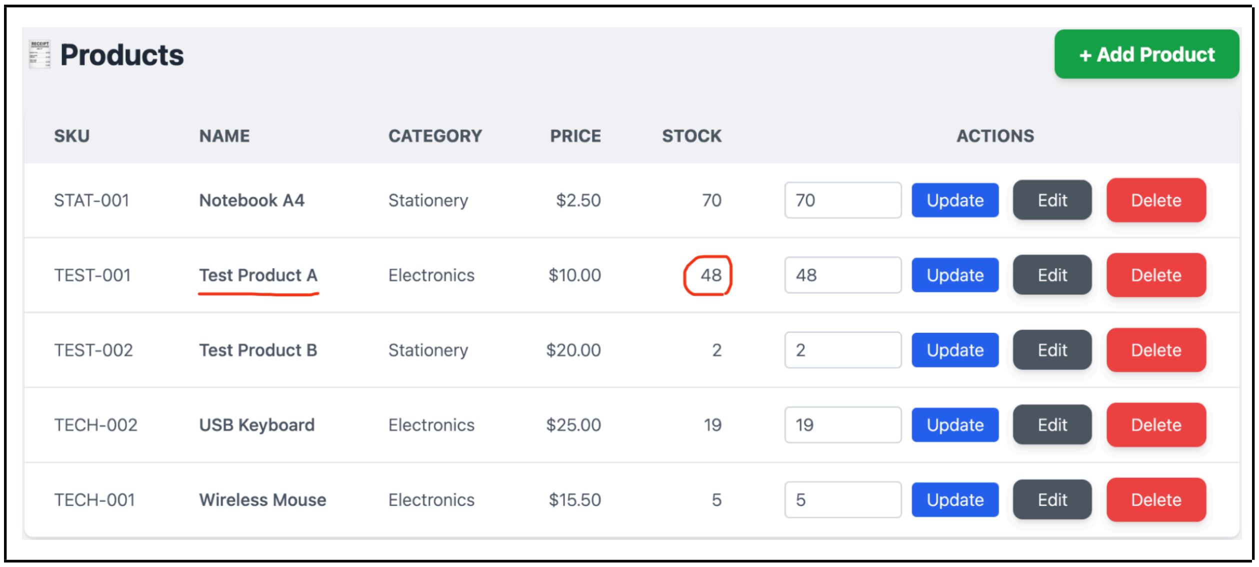
To verify that the product inventory in the database is automatically and accurately deducted based on the sales transaction recorded in Section 5.1.

Data Consistency Check:

Product	Quantity Sold (in Scenario A)	Initial Stock	Expected Current Stock
Test Product A	2	50 (enough stock)	48

Result:

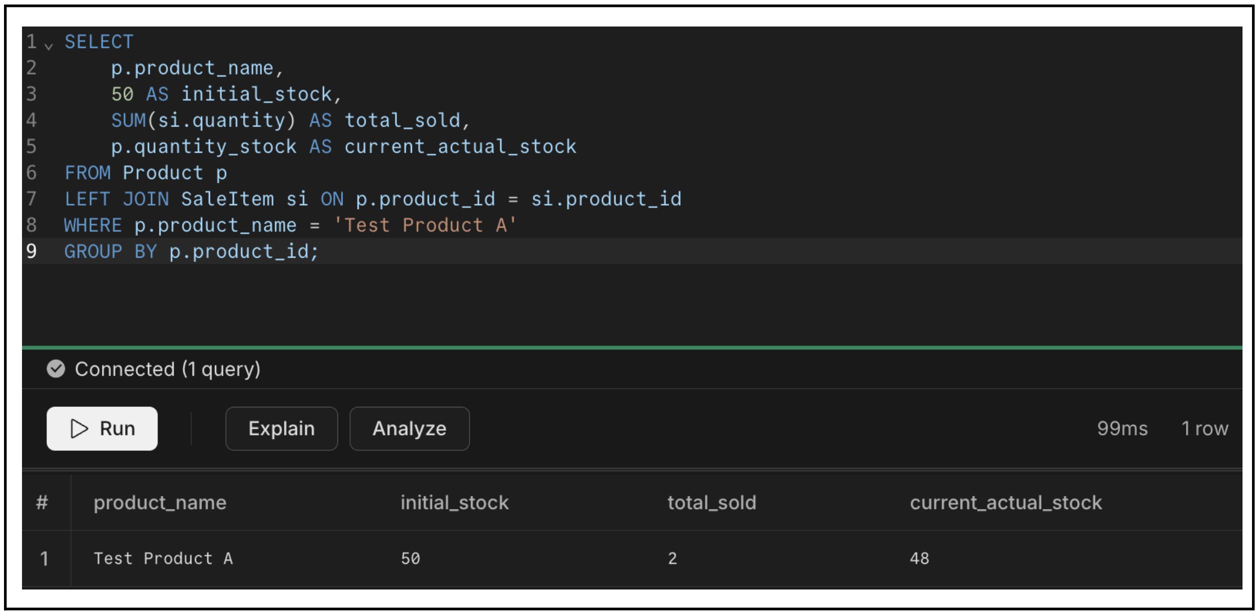
Application Side:



Products

SKU	NAME	CATEGORY	PRICE	STOCK	ACTIONS
STAT-001	Notebook A4	Stationery	$2.50	70	70 Update Edit Delete
TEST-001	Test Product A	Electronics	$10.00	48	48 Update Edit Delete
TEST-002	Test Product B	Stationery	$20.00	2	2 Update Edit Delete
TECH-002	USB Keyboard	Electronics	$25.00	19	19 Update Edit Delete
TECH-001	Wireless Mouse	Electronics	$15.50	5	5 Update Edit Delete

Database Side:



```sql
SELECT
    p.product_name,
    50 AS initial_stock,
    SUM(si.quantity) AS total_sold,
    p.quantity_stock AS current_actual_stock
FROM Product p
LEFT JOIN SaleItem si ON p.product_id = si.product_id
WHERE p.product_name = 'Test Product A'
GROUP BY p.product_id;
```

Connected (1 query)

99ms 1 row

#	product_name	initial_stock	total_sold	current_actual_stock
1	Test Product A	50	2	48

Observation:

The query confirms that the current_actual_stock (48) matches exactly with the calculated expected_stock (50 - 2 = 48). This confirms that the trigger for inventory deduction is functioning correctly.

5.3 Reporting Accuracy

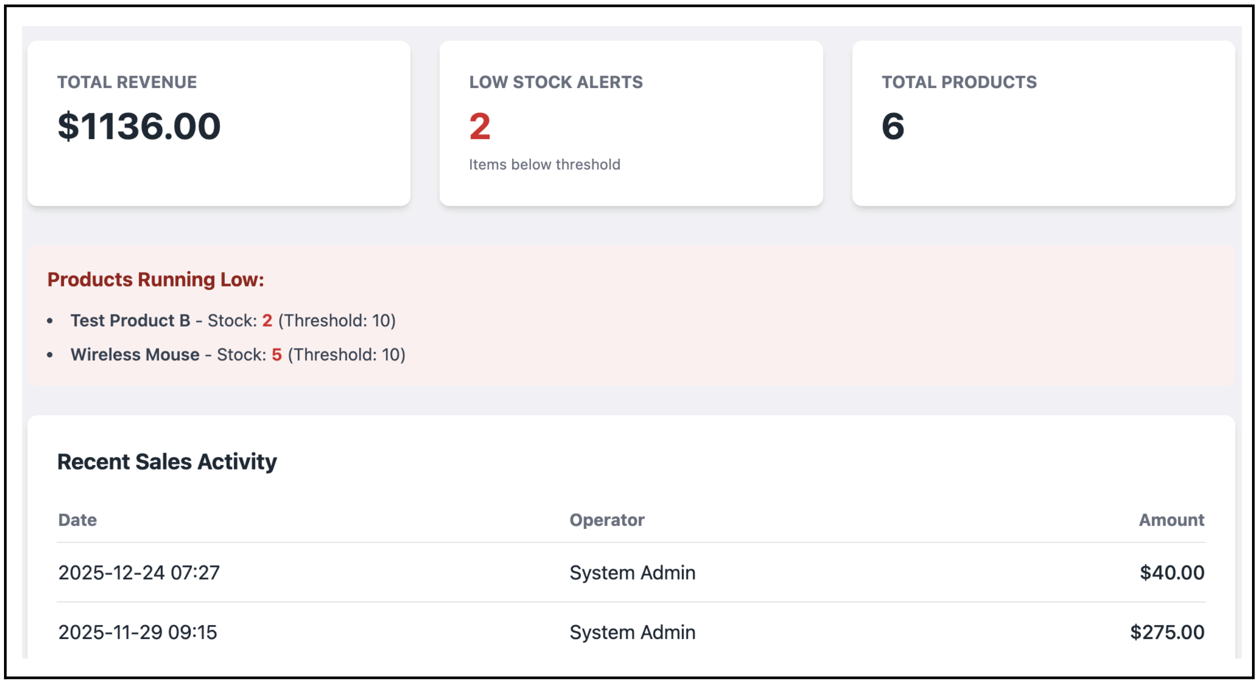
Purpose:
To verify that the Admin Dashboard correctly aggregates data in real-time. We compare the metrics displayed on the UI against the direct SQL query results.

Verification 1: Low Stock Alerts

Logic: The system should count products where quantity_stock <= low_stock_threshold.
Based on previous tests, Test Product B (Stock: 2, Threshold: 10) should trigger this alert.

Result:

Application Side:



TOTAL REVENUE
$1136.00

LOW STOCK ALERTS
2
Items below threshold

TOTAL PRODUCTS
6

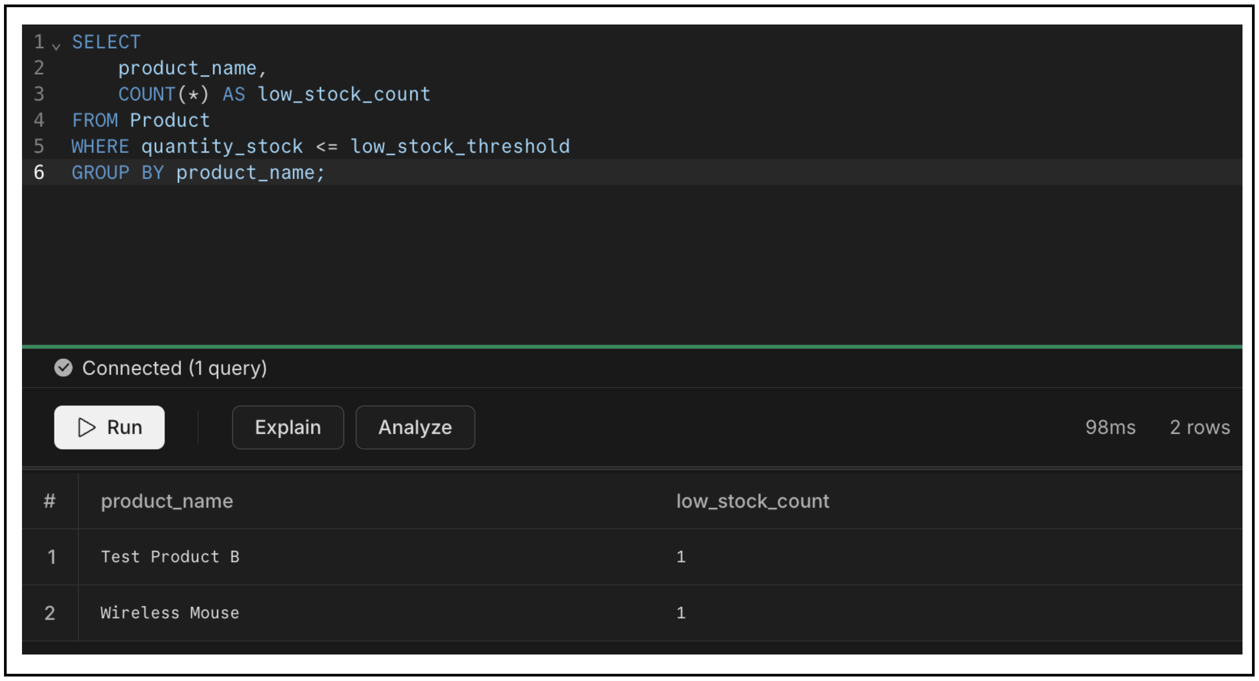
Products Running Low:

- **Test Product B** - Stock: **2** (Threshold: 10)
- **Wireless Mouse** - Stock: **5** (Threshold: 10)

Recent Sales Activity

Date	Operator	Amount
2025-12-24 07:27	System Admin	$40.00
2025-11-29 09:15	System Admin	$275.00

Database Side:



```sql
SELECT
    product_name,
    COUNT(*) AS low_stock_count
FROM Product
WHERE quantity_stock <= low_stock_threshold
GROUP BY product_name;
```

#	product_name	low_stock_count
1	Test Product B	1
2	Wireless Mouse	1

Observation:

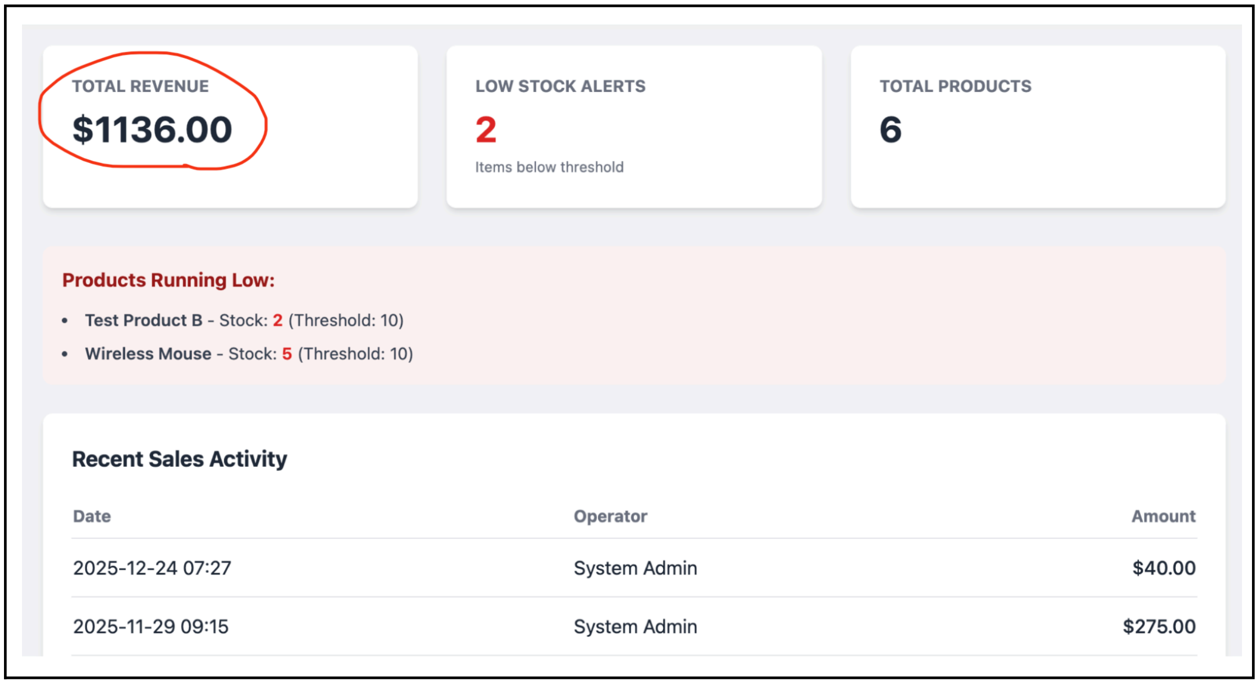
The dashboard correctly displays "2" for Low Stock Alerts, which matches the SQL query result (2 records returned. “Test Product B” and “Wireless Mouse”). This confirms the alerting logic is accurate.

Verification 2: Total Revenue

Logic: The sum of total_amount from all recorded sales.

Result:

Application Side:


TOTAL REVENUE
$1136.00

LOW STOCK ALERTS
2
Items below threshold

TOTAL PRODUCTS
6

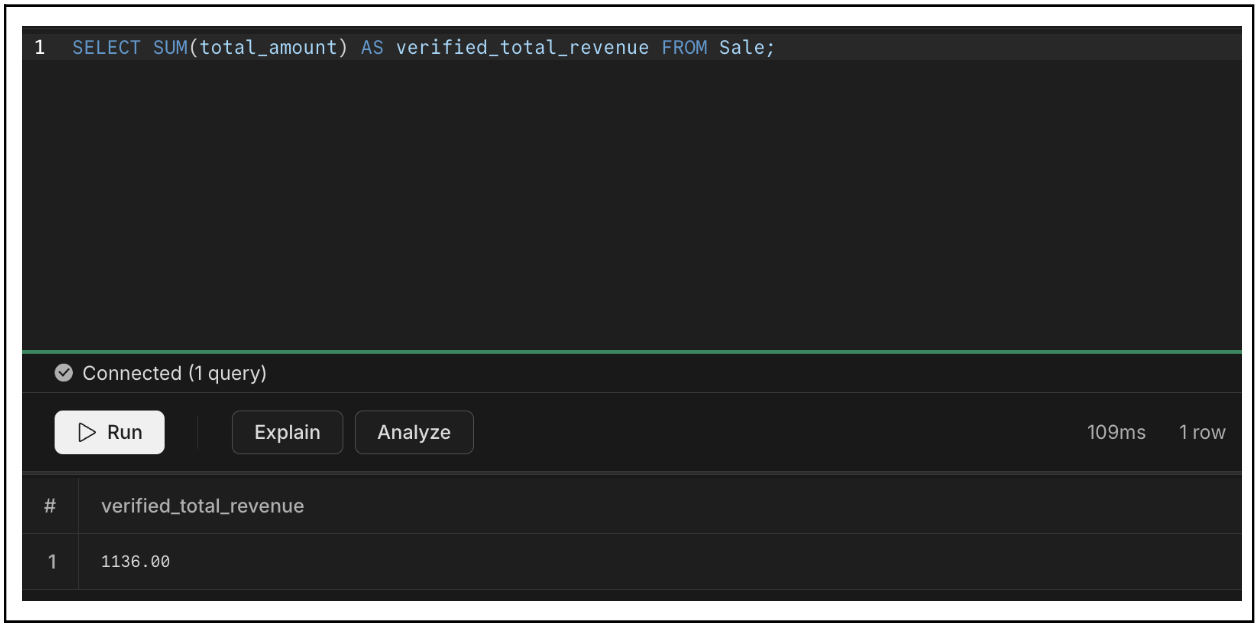
Products Running Low:

- **Test Product B** - Stock: **2** (Threshold: 10)
- **Wireless Mouse** - Stock: **5** (Threshold: 10)

Recent Sales Activity

Date	Operator	Amount
2025-12-24 07:27	System Admin	$40.00
2025-11-29 09:15	System Admin	$275.00

Database Side:



```
SELECT SUM(total_amount) AS verified_total_revenue FROM Sale;
```

Connected (1 query)

```
#  verified_total_revenue
1  1136.00
```

Observation:

The Total Revenue displayed on the dashboard exactly matches the SQL aggregation result. This confirms that all sales transactions are correctly summed up without discrepancy.

VI. Conclusion and Reflection

6.1 Lessons Learned

Through the development of the Small Business Inventory and Sales System, the team gained practical insight into how relational database concepts are applied in real-world applications. The project highlighted the importance of a well-structured relational schema, where entities such as Product, Category, Sale, and SaleItem are clearly separated and linked through primary and foreign keys to ensure data integrity. The team also learned how CRUD operations must adhere closely to database constraints, particularly when handling updates and deletions involving relational dependencies. Integrating the database with the application further reinforced the relationship between database logic and application-level functionality, ensuring user actions were accurately reflected in the system.

6.2 Challenges Faced

One of the main challenges encountered was maintaining consistency between the database schema and the application code as the system evolved. Changes to table structures and constraints required adjustments in application logic to prevent conflicts and errors. The team also faced difficulties handling database restrictions such as foreign key and unique constraints, especially when deleting records linked to existing transactions. Additionally, configuration issues related to database connections and environment variables posed challenges during testing, emphasizing the importance of proper environment setup and validation.

6.3 Future Improvements

Future development of the system may focus on expanding reporting features, including more detailed sales analysis and inventory summaries to support decision-making. Improvements to user management, such as enhanced authentication and role-based authorization, could also strengthen system security. Furthermore, optimizing the application interface and database interactions would improve performance and scalability as data volume increases, making the system more suitable for long-term use.