Attention is All You Need for Inventory Loss!

SIPMS

Smart Inventory & Procurement Management System

COMP3030: Databases and Database Systems

Group 12 - The Fool: Tran Quang Khai, Thai Huu Tri, Nguyen Ngoc Han

RECAP VINUNI'S DORM LIFE

Before Tet

After Tet

After Tet a month

Guess how many food will be dumped...

SMART INVENTORY MANAGEMENT SYSTEM

A MySQL-backed information system with a web interface supporting inventory operations, procurement, sales, inter-location transfers, and reporting.

Time budget: two weeks

Target Users

Small retail stores struggle to maintain real-time inventory.

Issues

(i) **stockouts** by delayed replenishment decisions,
(ii) **inconsistent stock records** between locations,
(iii) **limited insight** into best-selling products,
(iv) **manual processes** -> difficult auditing & accountability.

REQUIREMENT ANALYSIS

Functional Requirements

- Entity & User Management
- Real-Time Inventory Tracking
- Procurement & Sales Workflows
- Inter-Location Transfers
- Reporting & Analytics

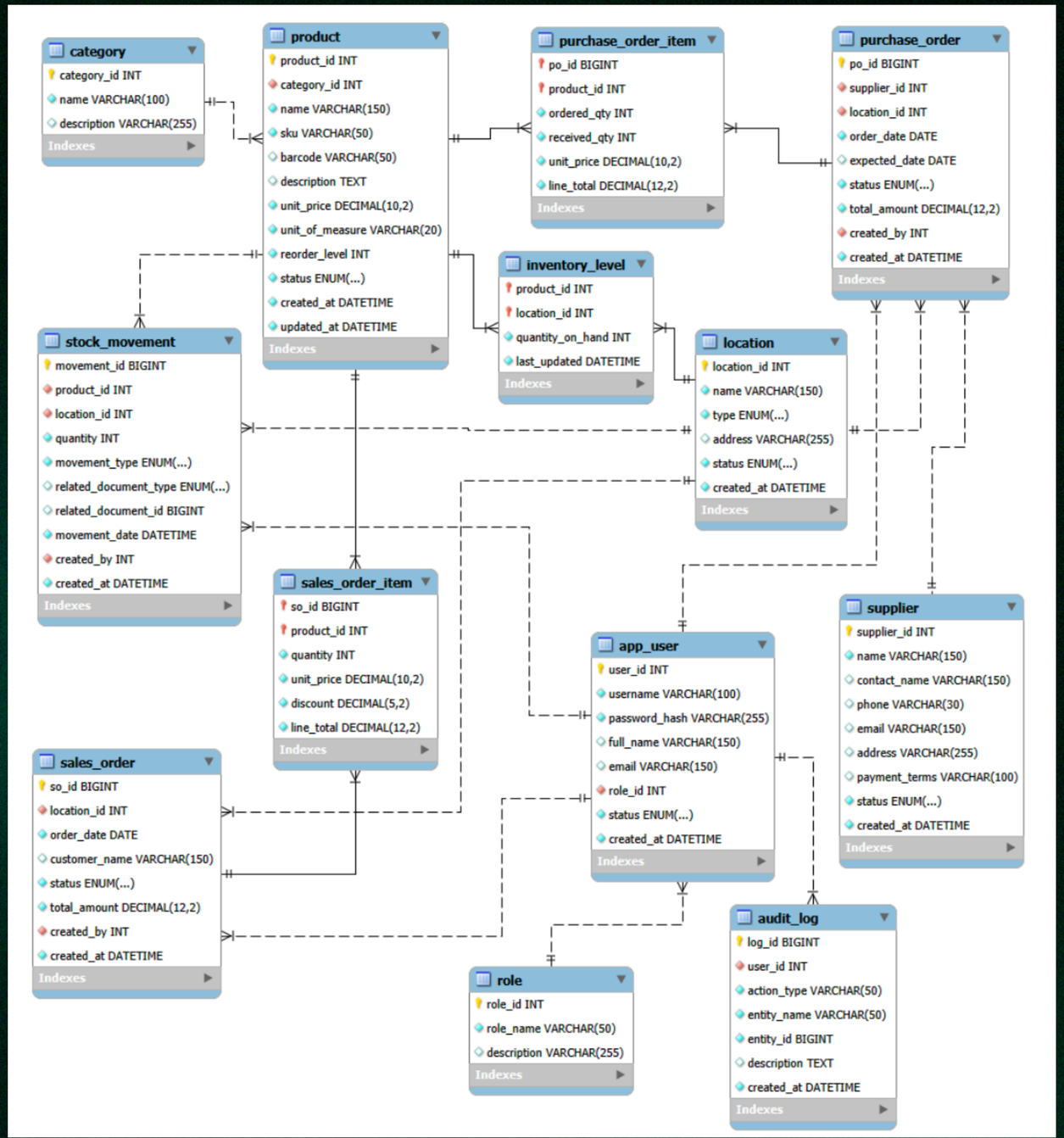
Non-Functional Requirements

- Performance & Concurrency
- Data Integrity & Reliability
- Security & Auditability
- Scalability & Maintainability
- Usability & Accessibility

CONCEPTUAL & PHYSICAL DB DESIGN

Feature	Admin	Manager	Clerk
CRUD Categories / Products	✓	✓	×
CRUD Suppliers / Locations	✓	✓	×
Create Purchase Order	✓	✓	×
Approve / Cancel Purchase Order	✓	✓	×
Receive Goods	✓	✓	×
Create Sales Order	✓	✓	✓
Create Stock Transfer	✓	✓	×
View Dashboard	✓	✓	✓
Manage Users / Roles	✓	×	×

Table 2.1: Role-Based Access Control Matrix



ER DIAGRAM & RELATIONSHIPS

Relational design includes:

(i) **master data** (products, suppliers, locations),
(ii) **transactional documents** (purchase orders, sales orders),
(iii) **inventory state & auditability** (inventory level, stock movement, audit log).

SQL Scripts

- indexes.sql
- procedures.sql
- schema.sql
- security.sql
- seed_data.sql
- triggers.sql
- views.sql

Core tables by domain

Identity and roles.

- `role`: role definitions (e.g., ADMIN, MANAGER, CLERK).
- `app_user`: application users, linked to `role` via foreign key.

Master data.

- `category`: product categories.
- `product`: product master (SKU, barcode, pricing, reorder level), linked to `category`.
- `supplier`: supplier master data used in procurement.
- `location`: physical locations (warehouse/store) where inventory is tracked.

Procurement and sales documents.

- `purchase_order` and `purchase_order_item`: purchase orders and their line items.
- `sales_order` and `sales_order_item`: sales orders and their line items.

Inventory and audit trail.

- `stock_movement`: immutable event log of stock changes (receipts, sales issues, transfers, adjustments).
- `inventory_level`: current on-hand quantity per `(product, location)` pair.
- `audit_log`: optional audit entries that can record user actions and descriptions.

DDL SCRIPTS WITH KEYS & CONSTRAINTS

```sql
-- Category
CREATE TABLE Category (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    description TEXT
);

-- Product
CREATE TABLE Product (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    category_id INT NOT NULL,
    name VARCHAR(150) NOT NULL,
    sku VARCHAR(50) NOT NULL UNIQUE,
    unit_price DECIMAL(10,2) NOT NULL CHECK (unit_price >= 0),
    reorder_level INT NOT NULL CHECK (reorder_level >= 0),
    status ENUM('active', 'inactive') DEFAULT 'active',
    FOREIGN KEY (category_id) REFERENCES Category(category_id)
);
```

```sql
-- User & Role
CREATE TABLE Role (
    role_id INT AUTO_INCREMENT PRIMARY KEY,
    role_name VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE App_User (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role_id INT NOT NULL,
    FOREIGN KEY (role_id) REFERENCES Role(role_id)
);
```

```sql
-- Location
CREATE TABLE Location (
    location_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    type ENUM('warehouse', 'store') NOT NULL,
    status ENUM('active', 'inactive') DEFAULT 'active'
);
```

```sql
-- Inventory Level (composite key, no duplicate stock records)
CREATE TABLE InventoryLevel (
    product_id INT NOT NULL,
    location_id INT NOT NULL,
    quantity_on_hand INT NOT NULL CHECK (quantity_on_hand >= 0),
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (product_id, location_id),
    FOREIGN KEY (product_id) REFERENCES Product(product_id),
    FOREIGN KEY (location_id) REFERENCES Location(location_id)
);
```

INVENTORY CONSISTENCY DESIGN

SIPMS records each inventory change as a row in ***stock_movement***. This provides:

- a complete **audit trail** of what changed, when, where, and by whom,
- a unified **model for multiple operations** (purchase receipt, sales issue, transfer, adjustment),
- a reliable **base for reporting** and troubleshooting.

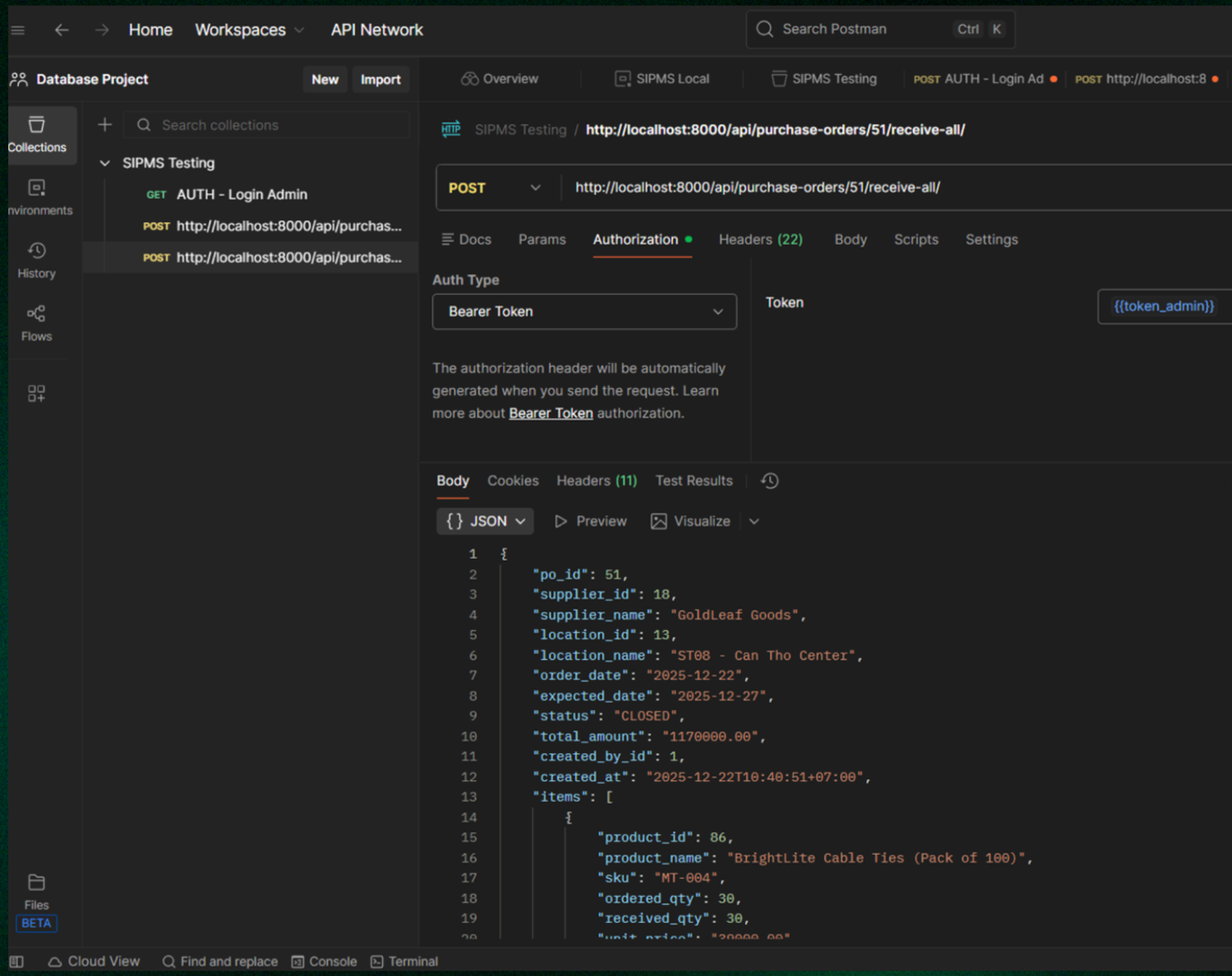


Figure 6.4: Receive-all workflow: stock receipt succeeds and PO status is updated.

DATABASE VIEWS FOR REPORTING

Implemented views

- ***view_stock_per_location***: including reorder-level flags & stock value.
- ***view_low_stock_products***: below reorder level (supports “Low Stock” dashboard/report).
- ***view_top_selling_products_last_30_days***

Benefits

- Reporting **logic** (joins and filters) is centralized in the database.
- The frontend/backend can query a **stable interface for analytics needs**.
- **Performance is improved** when combined with appropriate indexes.

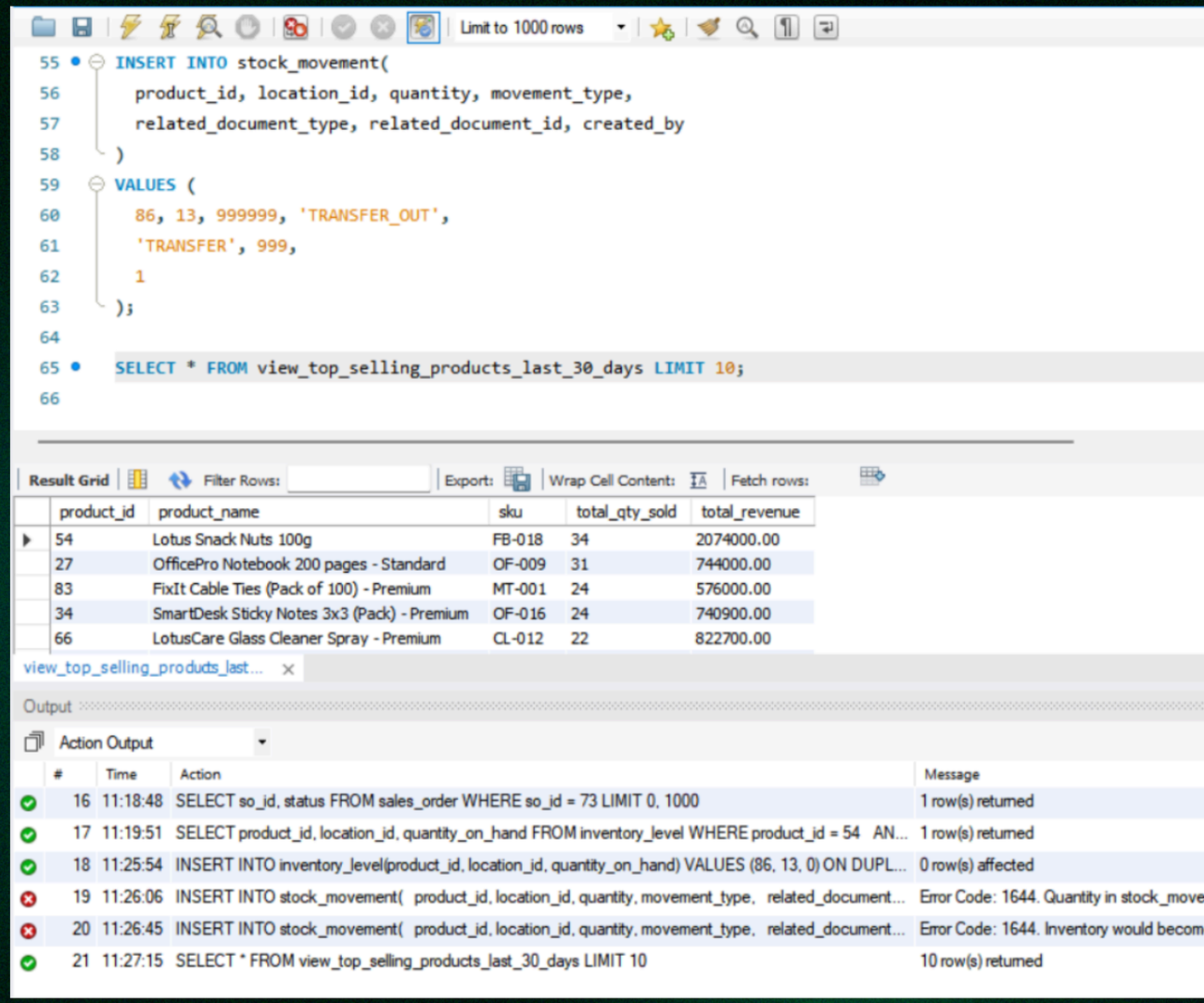


Figure 6.10: Workbench validation: view_low_stock_products returns products below reorder level.

STORED PROCEDURES VIEWS FOR WORKFLOWS

Implemented views

- ***sp_create_purchase_order***
- ***sp_confirm_sales_order***

Error Handling

- Procedure signals **clear validation failures** (e.g., missing sales order or insufficient stock).
- Errors are propagated to the **API layer** and displayed as **user-facing validation** messages.

6.3 SQL Listing: procedures.sql

```sql
-- 04_procedures.sql
USE sipms;

DROP PROCEDURE IF EXISTS sp_create_purchase_order;
DROP PROCEDURE IF EXISTS sp_confirm_sales_order;

-- 1. Create new purchase order
DELIMITER $$

CREATE PROCEDURE sp_create_purchase_order (
    IN p_supplier_id  INT,
    IN p_location_id  INT,
    IN p_order_date   DATE,
    IN p_expected_date DATE,
    IN p_total_amount DECIMAL(12,2),
    IN p_created_by   INT
)
BEGIN
    INSERT INTO purchase_order(
        supplier_id,
        location_id,
        order_date,
        expected_date,
        status,
        total_amount,
        created_by,
        created_at
    )
    VALUES (
        p_supplier_id,
        p_location_id,
        p_order_date,
        p_expected_date,
        'APPROVED',          -- modify if enum values
           APPROVED / PARTIALLY_RECEIVED / CLOSED / C
        p_total_amount,
        p_created_by,
        NOW()
    );
    -- return the newly created po_id
    SELECT LAST_INSERT_ID() AS po_id;
END$$

-- 2. Confirm sales order: check inventory + create stock m
CREATE PROCEDURE sp_confirm_sales_order (
    IN p_so_id   BIGINT,
    IN p_user_id INT
)
BEGIN
    DECLARE v_location_id INT;
    DECLARE v_not_enough_stock INT DEFAULT 0;

    -- get the location of the sales order
    SELECT location_id INTO v_location_id
    FROM sales_order
    WHERE so_id = p_so_id
    FOR UPDATE;

    IF v_location_id IS NULL THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Sales order not found';
    END IF;

    -- check stock for each item
    SELECT COUNT(*) INTO v_not_enough_stock
    FROM sales_order_item si
    LEFT JOIN inventory_level il
           ON il.product_id  = si.product_id
          AND il.location_id = v_location_id
    WHERE si.so_id = p_so_id
      AND (il.quantity_on_hand IS NULL
           OR il.quantity_on_hand < si.quantity);

    IF v_not_enough_stock > 0 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'Not enough stock to confirm
              order';
    END IF;

    -- update status SO
    UPDATE sales_order
    SET status = 'CONFIRMED'    -- modify if enum values dif
    WHERE so_id = p_so_id;

    -- insert stock movements (each line for each item)
    INSERT INTO stock_movement (
        product_id,
        location_id,
        quantity,
        movement_type,
```

LIMITATIONS & IMPROVEMENTS

Strengthen transactional atomicity for complex workflows by ensuring multi-step status updates and stock movement inserts commit as a single unit in all execution paths.

Extend auditing by writing audit_log entries for key operations (e.g., confirm sales, receive goods, transfers) either in procedures/triggers or consistently at the API service layer.

Add additional analytical views (e.g., inventory valuation over time, supplier performance, gross margin by product).

WEB-BASED INTERFACE DEMO

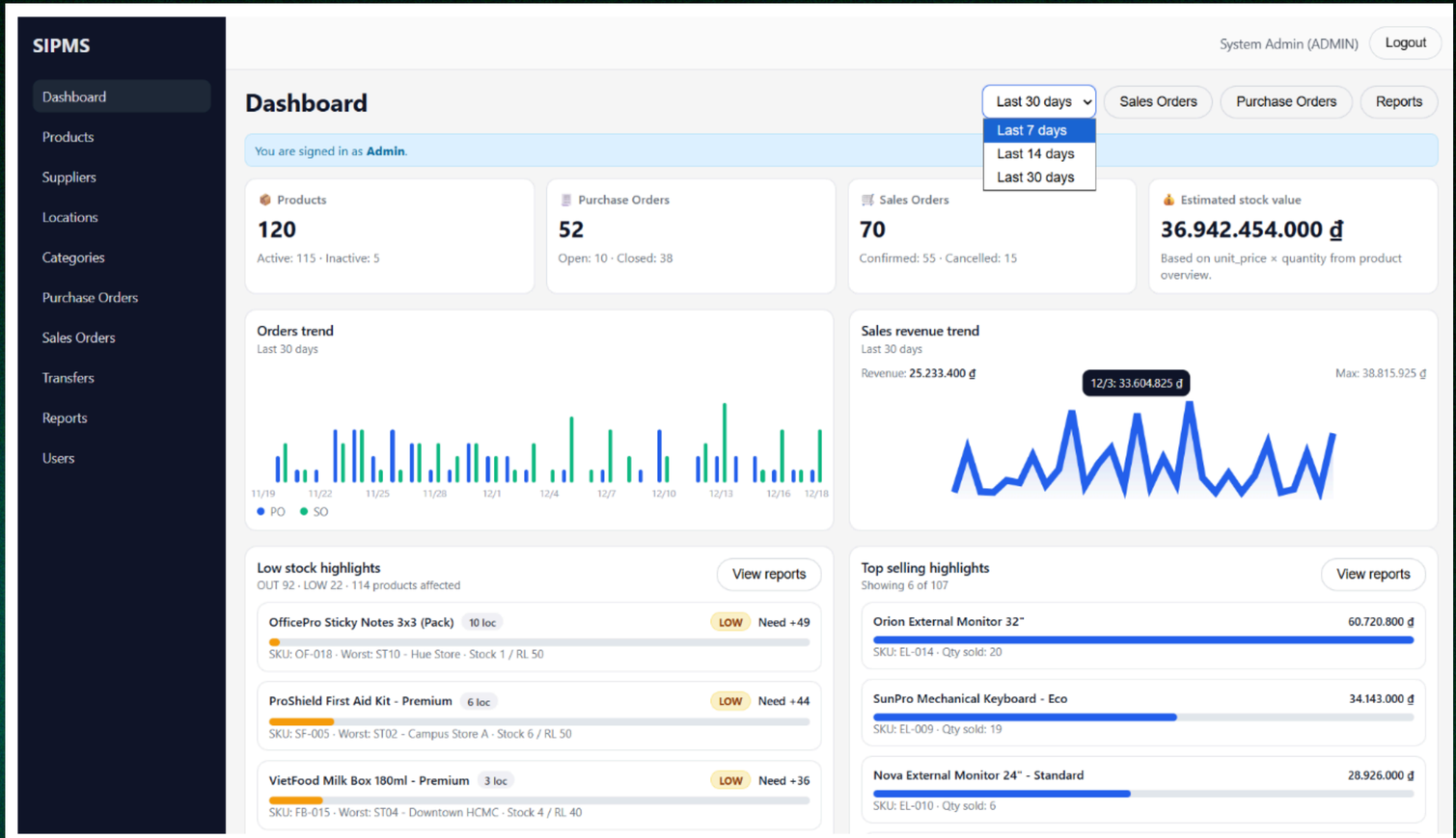


Figure 3.3: Interact with Chart and Option Selector for Time Range

PERFORMANCE TUNING

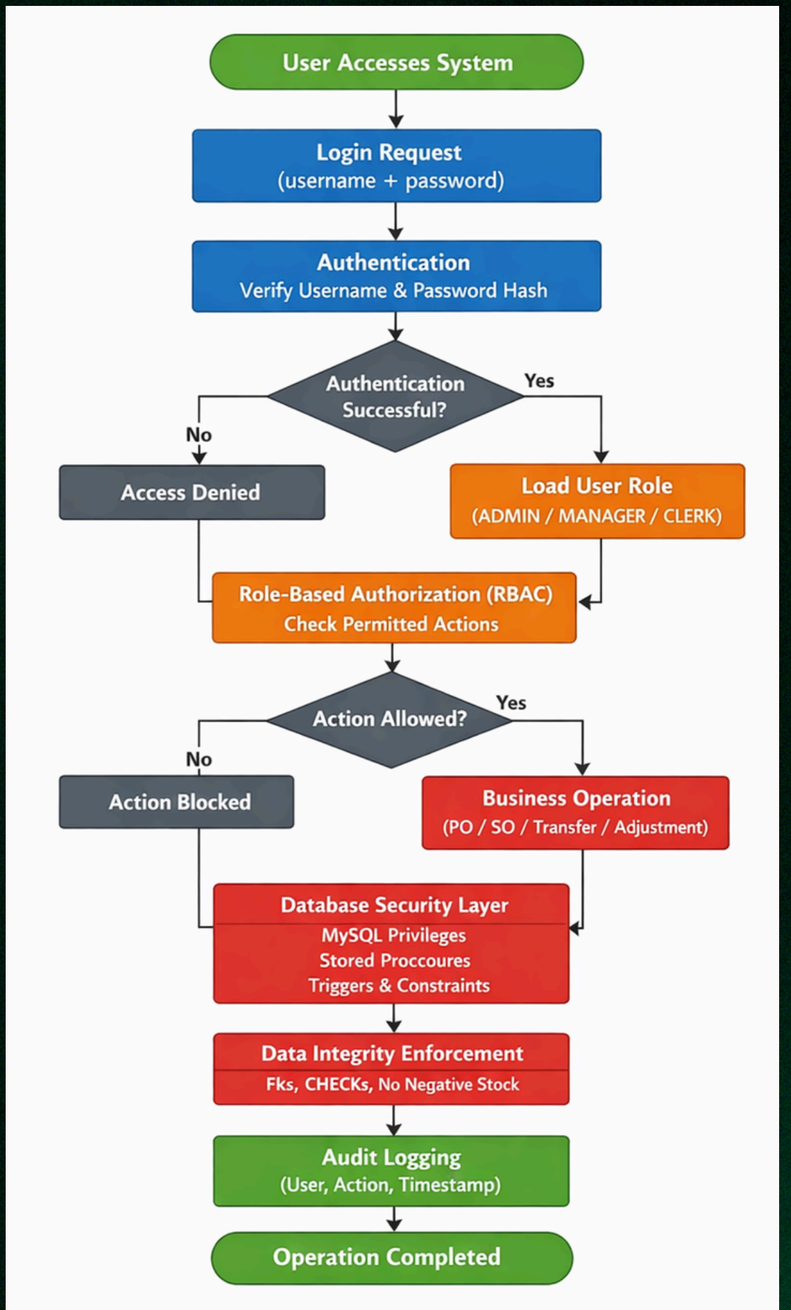
- Workload characteristics and query patterns
- Indexing strategy
- Query design optimizations at the application layer
- Use of views for reporting performance
- Performance verification (recommended evidence)

```sql
-- 07_indexes.sql
USE sipms;

-- Index for product lookup by SKU
ALTER TABLE product
  ADD INDEX idx_product_sku (sku);

-- Index for inventory_level by (product_id, location_id)
ALTER TABLE inventory_level
  ADD INDEX idx_inventory_product_location (product_id, location_id);

-- Index for stock_movement by (movement_date, product_id)
ALTER TABLE stock_movement
  ADD INDEX idx_stock_movement_date_product (movement_date, product_id
    );

-- Index for sales_order filtering by date/status (dashboard charts)
ALTER TABLE sales_order
  ADD INDEX idx_sales_order_order_date_status (order_date, status);

-- Index for purchase_order filtering by date/status (dashboard charts
  )
ALTER TABLE purchase_order
  ADD INDEX idx_purchase_order_order_date_status (order_date, status);
```

SECURITY CONFIGURATION

Additional practical hardening steps include:

- **rotate credentials** regularly and **restrict database access** by host/network;
- **avoid using admin** DB users for the application runtime connection;
- **enable audit logging** for sensitive actions (either at application layer or DB layer);
- ensure **backups and recovery procedures** are tested.



END-TO-END TESTING

Test Environment and Tools

- Web UI, Postman, MySQL Workbench

Deployment

1. `schema.sql` (create database `sipms`, tables, constraints)
2. `seed_data.sql` (demo/testing dataset)
3. `views.sql` (analytics/reporting views)
4. `procedures.sql` (stored procedures)
5. `triggers.sql` (inventory enforcement triggers)
6. `indexes.sql` (performance indexes)
7. `security.sql` (optional, DB user grants; passwords replaced before execution)

Test Setup Procedure

- Authentication & RBAC
- Workflow Correctness
- Database Integrity Enforcement

(quantity > 0, non-negative inventory)

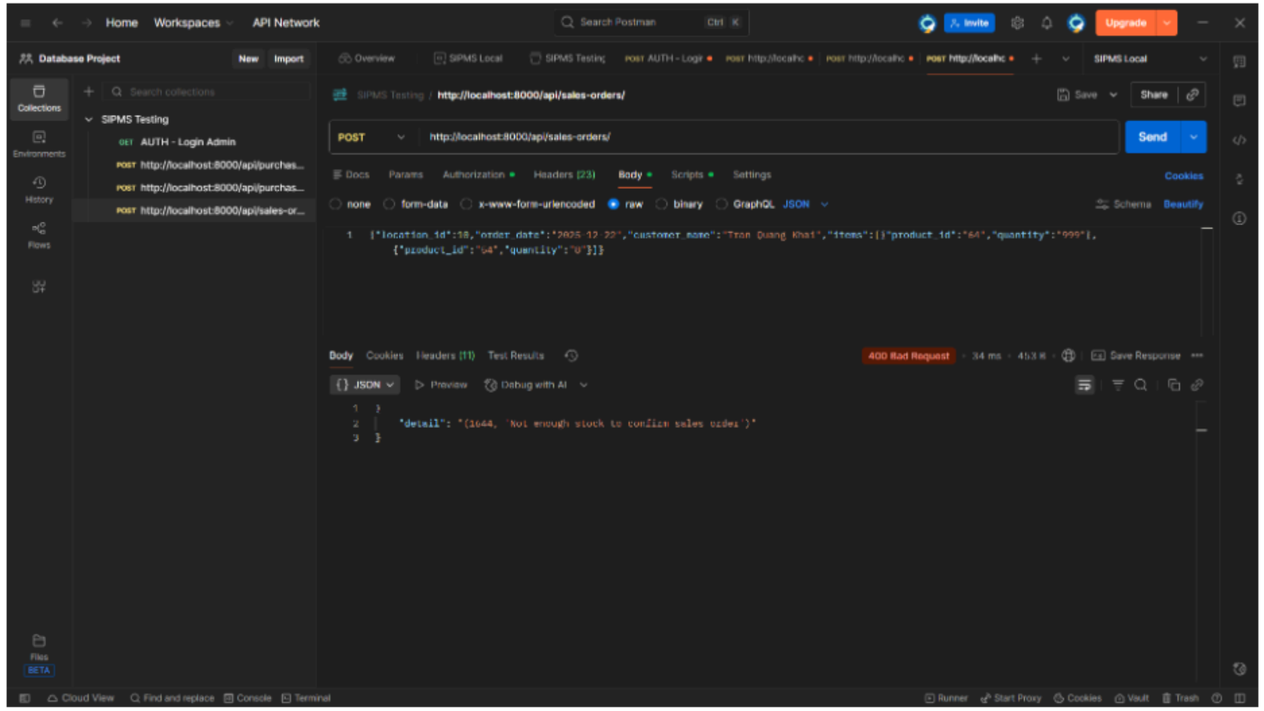


Figure 6.7: Sales order FAIL case: creation is rejected due to insufficient stock (HTTP 400 with DB error).

DIRECT TRIGGER VALIDATION

- **Quantity must be > 0**: invalid movements are blocked by a BEFORE INSERT trigger.
- **Non-negative inventory**: movements led to negative stock are rejected by the AFTER INSERT trigger.

```
55  INSERT INTO stock_movement(
56    product_id, location_id, quantity, movement_type,
57    related_document_type, related_document_id, created_by
58  )
59  VALUES (
60    86, 13, 999999, 'TRANSFER_OUT',
61    'TRANSFER', 999,
62    1
63  );
64
```

```
VALUES (
  86, 13, 0, 'ADJUSTMENT',
  'ADJUSTMENT', 999,
  1
);
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
15	10:55:42	SELECT * FROM inventory_level WHERE product_id = 86 AND location_id = 13 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
16	11:18:48	SELECT so_id, status FROM sales_order WHERE so_id = 73 LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
17	11:19:51	SELECT product_id, location_id, quantity_on_hand FROM inventory_level WHERE product_id = 54 AN...	1 row(s) returned	0.000 sec / 0.000 sec
18	11:25:54	INSERT INTO inventory_level(product_id, location_id, quantity_on_hand) VALUES (86, 13, 0) ON DUPL...	0 row(s) affected	0.000 sec
19	11:26:06	INSERT INTO stock_movement(product_id, location_id, quantity, movement_type, related_document...	Error Code: 1644. Quantity in stock_movement must be > 0	0.016 sec
20	11:26:45	INSERT INTO stock_movement(product_id, location_id, quantity, movement_type, related_document...	Error Code: 1644. Inventory would become negative	0.000 sec

OVERALL, ...

SIPMS testing confirms:
(i) **role-based access restrictions** are enforced at the API layer,
(ii) **business workflows** correctly generate stock movements and update inventory,
(iii) **triggers/procedures** prevent invalid inventory states (eg, negative stock).

Let’s Build Stock Future Together!

THANK YOU FOR YOUR ATTENTION!