Inventory Controlling system

Group 8: Jinzhao Li, Liuyi Yang, Teng Liu, Zeyu Li

Northeastern University

Abstract

We assume that we have our own e-commerce company and the system will be a solution to the problems of order management, inventory management, and revenue management. The goal is to create an easy-to-use, scalable inventory control system with some integrated accounting-related features that will be able to support automated functionality as our business grows.

Introduction

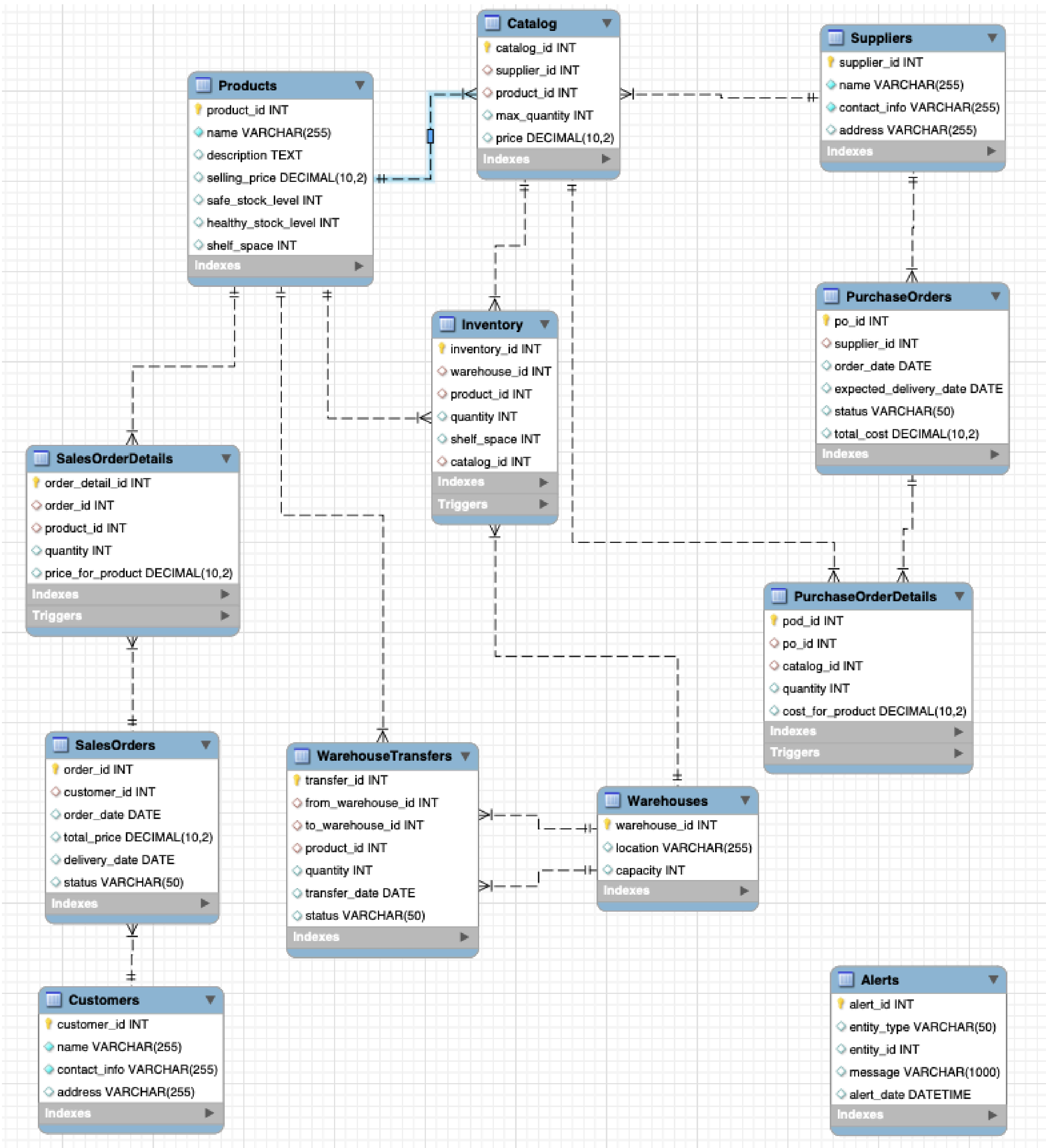
Our goal for this project is to create an inventory management system for an e-commerce company that has multiple warehouses serving numerous customers. This system aims to streamline and optimize inventory tracking, order management, and warehouse operations to ensure efficient and accurate management of products and orders.

Dozens of small e-commerce companies come and go every week because of a single problem that has always surrounded these businesses. Implementation of a full suite of supply chain, inventory, warehouse, and revenue management systems is too expensive and complex. That's why we want to build our mock inventory management system with some accounting features that are easy to understand. We also want to have the potential to add more advanced and automated features for our own e-commerce business.

We assume that our company has resources for different warehouses that can send products directly to customers when an order is placed (sales order in our perspective). Our warehouse will ship the products to the customer. The system will monitor the inventory level with different triggers to automate alerts and present data that are essential for the business.

Database Design

The core purpose of the database is to streamline the relationship between orders, inventories, products, warehouses, suppliers, and customers. The Inventory entity is a key join table connecting Products, Catalog, and Warehouses, ensuring accurate tracking of product quantities, pricing information, and storage locations. The Catalog details the products each supplier carries, including prices and production limits, which are vital for placing Purchase Orders (POs) and determining product availability and cost. POs are linked to Suppliers and detailed in the PurchaseOrderDetails table, which connects to the Catalog for supplier-specific pricing, essential for calculating the total cost of POs. SalesOrders and SalesOrderDetails capture the relationship between customers and the products they purchase, tracking sold products, quantities, and financial details. The Alerts table is a stand-alone table where all warnings or alerts are logged, allowing continuous monitoring and resolution of issues. This system supports efficient inventory and order management, facilitating smooth operations and informed decision-making.



Data Sources and Methods

The inventory management system is for an e-commerce company with multiple warehouses. We wanted to be flexible in designing and implementing the database and make sure the data could cover various test scenarios. So, instead of using real-world data, we decided to create our own.

First, we looked at some datasets related to logistics warehouse management. For example, the "Warehouse and Retail Sales" dataset includes monthly sales and movement data by item and department. We analyzed this data to figure out what features and information users might need. We also checked out the U.S. FDA's product classification codes to create more realistic product data.

Aside from analyzing existing datasets, we also looked at real-world works and products of the logistics and warehouse industry by reading several research papers on logistics warehouse management systems. For example, "Analysis and Design of Logistics Warehousing Management Information System based on RFID" gave us some examples of data structures that helped us better understand how to design our database, and "Research on Intelligent Warehousing and Logistics Management System of Electronic Market based on Machine Learning" includes several typical logistics processes we could reference. They helped us set some guidelines for our dataset to make sure our database design could support warehousing and logistics management needs.

Based on this research and the project's goals, we redesigned and created a set of simulated data. The structure is as follows:

Table	**Data Count**	**Description**
Suppliers	6 suppliers	Contains supplier ID, name, contact information, and address. Based on standard practices in supply chain management, making it closer to real-world applications.
Products	Approximately 50 products	Contains product ID, name, description, selling price, safe stock level, healthy stock level, and shelf space. Ensures reasonable and scientific classification based on market research and warehouse management needs.

Catalog	150 product and supplier relationships	Contains catalog ID, supplier ID, product ID, max quantity, and price. Helps manage the relationship between suppliers and products, ensuring accurate price and quantity information.
Customers	40 to 60 static entries or dynamic input	Contains customer ID, name, contact information, and address. Follows basic user behavior patterns, making the simulated data closer to real use cases.
Warehouses	5 warehouses	Contains warehouse ID, location, and capacity. Ensures realistic warehouse layout and capacity based on real-world examples from the references.
Inventory	Inventory levels for 5 warehouses	Contains inventory ID, warehouse ID, product ID, quantity, shelf space, and catalog ID. Ensures the stock level of each warehouse does not exceed its maximum capacity.
SalesOrders	60 sales orders	Contains order ID, customer ID, order date, total price, delivery date, and order status. Ensures the integrity and authenticity of the data based on standard procedures in warehouse management systems.
SalesOrder **Details**	Approximately 150 order details	Contains order detail ID, order ID, product ID, quantity, and price for each product. Ensures clarity in order processing and fulfillment.
PurchaseOrders	50 purchase orders	Contains purchase order ID, supplier ID, order date, expected delivery date, status, and total cost. Helps manage the supply chain and procurement process.
PurchaseOrder **Details**	Approximately 150 purchase order details	Contains purchase order detail ID, purchase order ID, catalog ID, quantity, and cost for each product. Helps track purchase specifics and financials.

Warehouse Transfers	Approximately 35 transfers	Contains transfer ID, from warehouse ID, to warehouse ID, product ID, quantity, transfer date, and status. Ensures the rational flow of inventory between warehouses.
Alerts	Based on system requirements	Contains alert ID, entity type, entity ID, message, and alert date and time. Helps promptly identify and address anomalies in the system.

Through these steps, we generated a complete logistics warehouse database, including multiple tables for suppliers, products, catalogs, customers, warehouses, inventory, orders, purchase orders, warehouse transfers, and alerts. This ensures that the data can effectively support the project's needs and user usage. Each table represents an independent entity or relationship with a clear structure, making it easy to expand and maintain. Foreign key constraints ensure data consistency and integrity. We made sure that the simulated data is practical and representative of realistic use cases by referencing standard processes and data models in real-world supply chain and warehouse management.

User Cases

User 1: Inventory Manager Anna Manages Inventory and Procurement

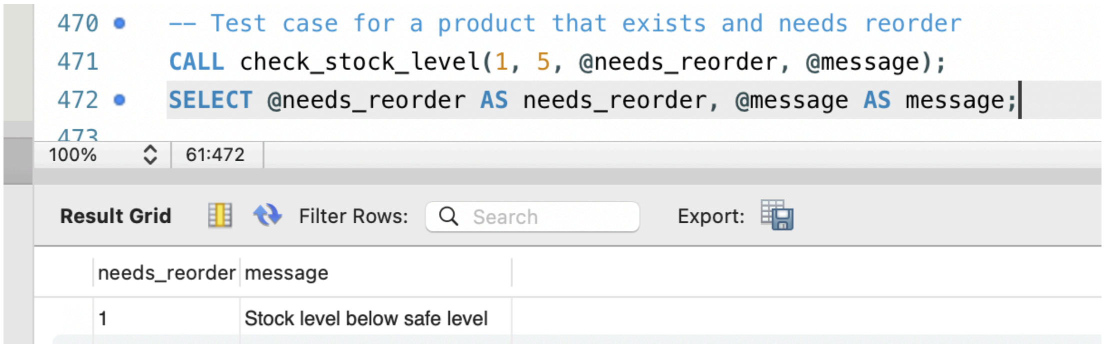
Background

A Khoury school postgraduate student, Anna, is also an inventory manager at a large retail company. Her daily tasks include monitoring inventory levels, ensuring stock does not fall below the safe stock level, and contacting suppliers to replenish inventory.

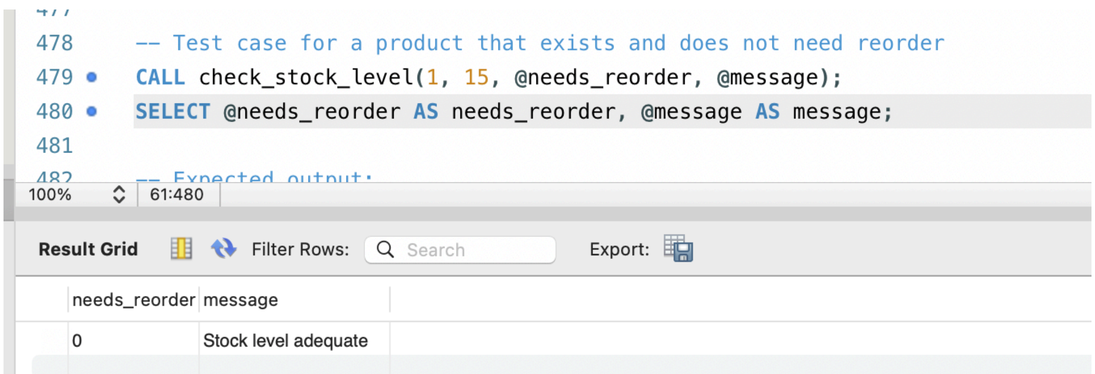
Use Case

1. **Check Products Below Safe Stock Level**
    - **Action:** Anna runs SQL queries to check which products are below the safe stock level.
    - **System Response:** The query results show which products are below the safe stock level.
    - **Related User Case:** Which products are below the safe stock level?
    - **SQL Query and/or result:**

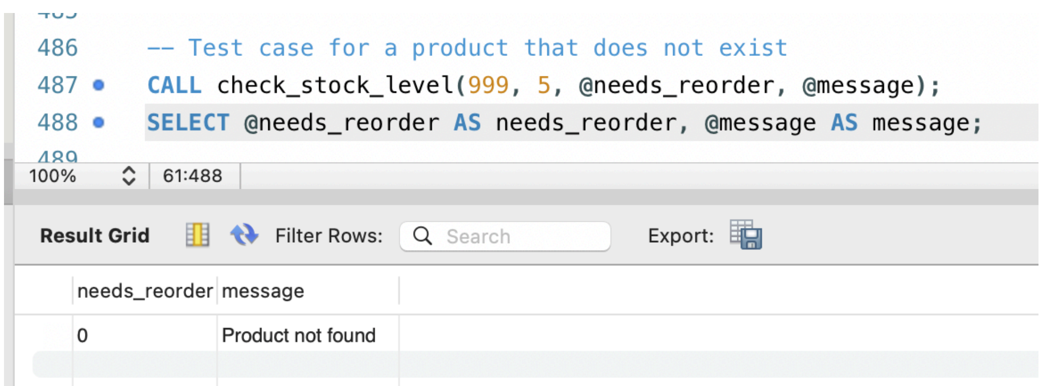
An example of checking for a low stock product and successfully triggering an alert



An example of checking for a product with sufficient stock and confirming that no alert is created



An example of checking a product that does not exist



An example of showing all products that are below the safe stock level products 1 × Statistics 1

CALL GetLowStockProducts() | Enter a SQL expression to filter results (use Ctrl+Space)

	product_id	product_name	safe_stock_level	current_stock	restock_needed
1	13	Electric Kettle	30	15	15
2	15	Vacuum Cleaner	5	2	3

2. **Trigger Alerts and View Low Stock Products**
   - **Action:** The system triggers an alert, notifying Anna that several products are below the safe stock level. Anna runs a query to view the details.
   - **System Response:** The query results display detailed information on the low-stock products, including current inventory, required replenishment quantity, and recommended suppliers.
   - **Related Trigger:** Alert triggered when inventory falls below the safe stock level.
   - **SQL Query and/or result:**

An example of a product that is below the safe stock level triggers an alert

```sql
-- Test case for low stock situation
-- 1. Insert SalesOrderDetails data to test low stock situation
-- For example, product_id 1 (Laptop) and quantity 20 to simulate low stock
INSERT INTO SalesOrderDetails (order_id, product_id, quantity, price_for_product) VALUES (34, 1,

-- 2. Check the Alerts table to confirm the alert was triggered
SELECT * FROM Alerts WHERE entity_type = 'Product' AND entity_id = 1;
```

100% 98:647

Result Grid Filter Rows: Search Edit: Export/Import:

alert_id	entity_type	entity_id	message	alert_date
1	Product	1	Stock level for product 1 in inventory ID 1 has fallen below the safety stock level.	2024-06-23 14:38:14
2	Product	1	Stock level for product 1 in inventory ID 16 has fallen below the safety stock level.	2024-06-23 14:38:15
3	Product	1	Insufficient stock for product 1 to fulfill sales order 1	2024-06-23 14:38:15
4	Product	1	Stock level for product 1 in inventory ID 1 has fallen below the safety stock level.	2024-06-23 14:38:28
5	Product	1	Stock level for product 1 in inventory ID 16 has fallen below the safety stock level.	2024-06-23 14:38:28
6	Product	1	Stock level for product 1 in inventory ID 32 has fallen below the safety stock level.	2024-06-23 14:38:28

A new purchase order was created to fulfill the safe stock

po_id	supplier_id	order_date	expected_delivery_d...	status	total_cost
34	1	2024-06-23	2024-06-30	Pending	57500.00
NULL	NULL	NULL	NULL	NULL	NULL

pod_id	po_id	catalog_id	quantity	cost_for_product
87	34	1	50	1150.00
NULL	NULL	NULL	NULL	NULL

3. **View Supplier Prices and Choose a Supplier**
   - **Action:** Anna runs a query to view the recommended suppliers and their prices for the low stock products and selects the supplier offering the best price for replenishment.
   - **System Response:** The query results show the suppliers sorted by price from the catalog, recommending the cheapest supplier.
   - **Related User Case:** Compare prices for the same product from different suppliers.
   - **SQL Query and/or result:**

An example of checking on low stock products

```
CALL GetLowStockProducts();
```

product_id	product_name	safe_stock_le...	current_sto...	stock_defi...
13	Electric Kettle	30	15	15
15	Vacuum Cleaner	5	2	3

A query for finding the best supplier for the product with product id = 13, based on the lowest price

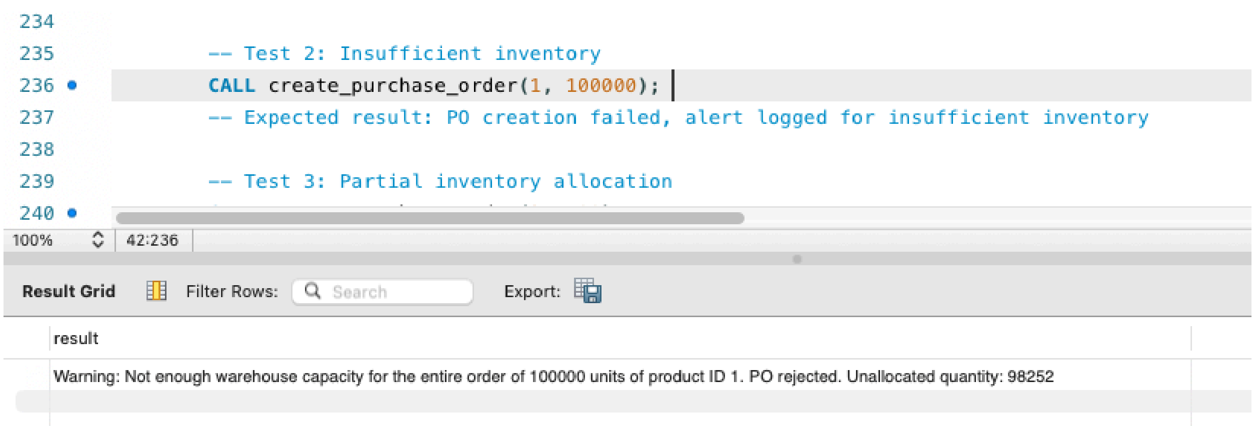
```
-- Test for product 13
DELIMITER //
CALL find_cheapest_suppliers(13, @cheapest_supplier_id1, @cheapest_catalog_id1, @cheapest_price1);
SELECT @cheapest_supplier_id1 AS cheapest_supplier_id, @cheapest_catalog_id1 AS cheapest_catalog_id, @che
DELIMITER ;
```

cheapest_supplier_id	cheapest_catalog_id	cheapest_price
1	37	35.00

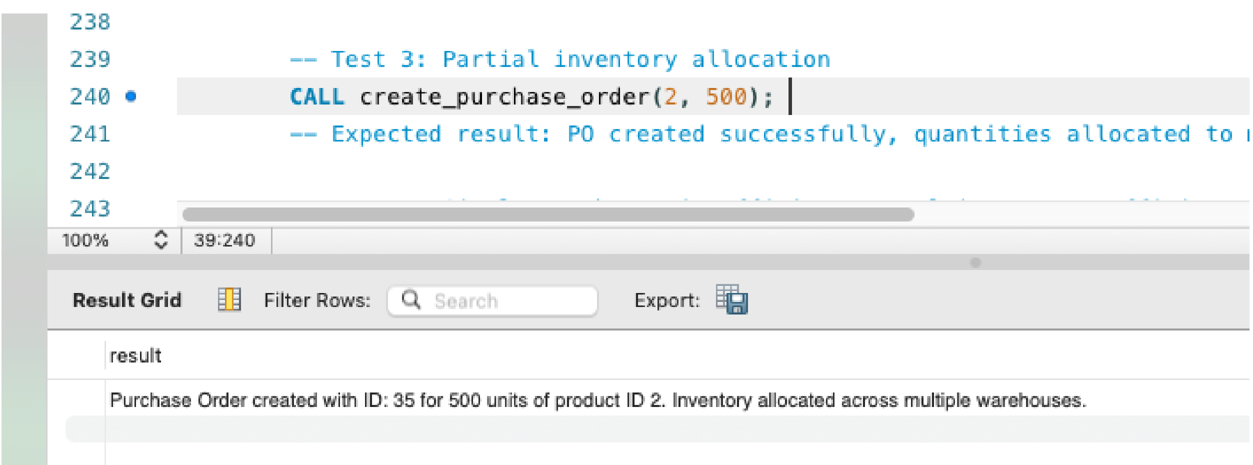
4. **Create a Purchase Order (PO)**

- **Action:** After selecting a supplier, Anna uses SQL queries to create a purchase order in the system.
- **System Response 1:** If the ordered quantity exceeds the total supplier supply limit, the system only purchases the available quantity. The cost is calculated based on the actual quantity purchased.
    - A warning should be displayed and added to the alerts table, but is not fully implemented yet.
- **System Response 2:** The system checks the warehouse capacity. If a warehouse does not have sufficient stock for an entire purchase order, the system begins to check other warehouses to see if the purchase order can be split and stored in different locations. If all available space is insufficient, an alert is generated.
- **Related Trigger:** Check warehouse capacity after inventory changes.
- **SQL Query and/or result:**

An example of a Failed PO creation attempt is purchasing too much for all warehouse's total available capacity



An example of a successful PO creation attempt is purchasing 500 pcs of product 2.



Example of how the Purchase Order Is updated after one failed PO Attempt and one successful PO entry attempt.

po_id	supplier_id	order_date	expected_delivery_d...	status	total_cost
35	4	2024-06-23	NULL	Add to Inventory	146000.00
34	1	2024-06-23	NULL	Rejected	0.00
33	3	2024-03-14	2024-03-24	Delivered	59500.00

5. **Update Inventory Information**
    - **Action:** When the new inventory arrives at the warehouse, the inventory will be automatically updated which needs Anna to monitor if the inventory distribution is correct.
    - **System Response:** The system will add a success message to the Alerts table.
    - **Related Trigger:** rg_after_insert_purchase_order_details will be triggered if the item does not exist in the inventory table.

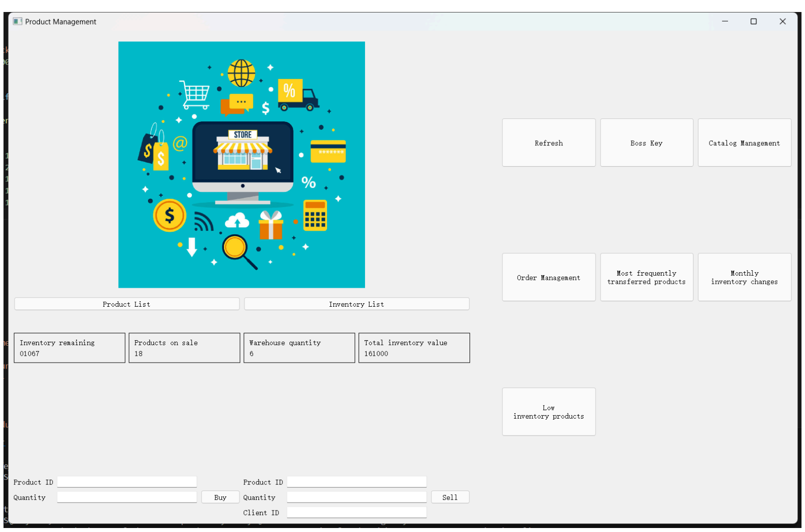
User 2: Purchasing Manager Sam Optimizes Procurement Strategy

Background

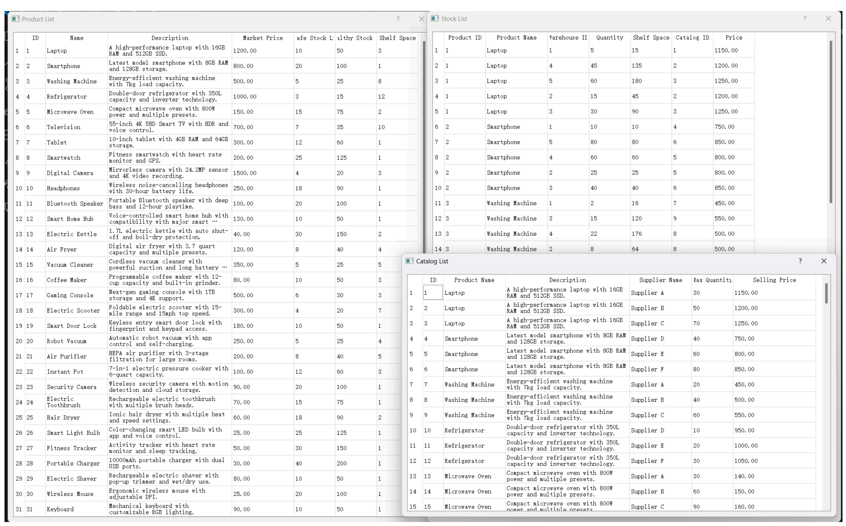
A Khoury school graduate, Sam, is a purchasing manager at an electronics company. His job includes dealing with multiple suppliers to procure needed products at the best prices and maintaining a smooth supply chain.

Use Case

1. **Open the System and Check Inventory**
    - **Action:** Sam opens the inventory management system and views the overview on the inventory dashboard. Sam can click a button to query the features that he is interested in, and He can enter the product ID and quantity to buy inventory or ship goods (not ready in-app, ready in the database system).
    - **App Screenshot:**



2. **Optimize Supplier Selection and Create Purchase Order**
    - **Action:** Sam runs SQL queries to view the prices for the same products from different suppliers and selects the supplier offering the best price. After selecting the best supplier, Sam creates a purchase order in the system using SQL queries.
    - **System Response:** The query results display sorted prices from different suppliers, helping Sam choose the best procurement option. The system checks the warehouse capacity. If the ordered quantity exceeds the total supplier supply limit, the system only purchases the available quantity. The cost is calculated based on the actual quantity purchased.
    - **Related User Case:** Compare prices for the same product from different suppliers; Check warehouse capacity after inventory changes.
    - **SQL Query and/or result:** See Anna's case 1 and 2 above.
3. **Compare prices offered by suppliers**
    - **Action:** By opening windows, Sam can compare prices offered by suppliers in real time and make good deals. Sam can get a lot of combined information by opening and combining different windows.
    - **System Response:** The query results display the order status in a window. A User can open multiple windows at the same time. The built-in buttons in the app are an integration of common commands.
    - **App Screenshot:**



4. **Optimize Transfer Strategy**
    - **Action:** Sam reviews the most frequently transferred products between warehouses over the past few months to optimize storage strategies.
    - **System Response:** The system shows the most frequently transferred products between warehouses.
    - **Related User Case:** Table shows the most frequently transferred products**.**
    - **SQL Query and/or result:**

products 1 × Statistics 1

CALL MostTransferredProducts()

	warehouse_id	product_id	product_name	total_transferred
3	1	29	Electric Shaver	12
4	1	5	Microwave Oven	8
5	2	27	Fitness Tracker	14
6	2	18	Electric Scooter	12
7	2	7	Tablet	10
8	2	17	Gaming Console	9
9	3	18	Electric Scooter	12
10	3	19	Smart Door Lock	10
11	3	38	Space Heater	10
12	3	3	Washing Machine	9
13	4	29	Electric Shaver	12

5. **Optimize Procurement Strategy**
    - **Action:** Sam reviews the inventory changes and the most frequently transferred products between warehouses over the past few months to optimize procurement strategies.
    - **System Response:** The system shows the inventory changes for each warehouse.
    - **Related User Case:** Table shows the inventory changes in warehouse transfer for each warehouse**.**
    - **SQL Query and/or result:**

	warehouse_id	product_id	product_name	month_and_year	quantity_change
6	1	11	Bluetooth Speaker	2024-04	9
7	1	13	Electric Kettle	2024-02	7
8	1	13	Electric Kettle	2024-05	-6
9	1	14	Air Fryer	2024-04	-5
10	2	1	Laptop	2024-01	7
11	2	2	Smartphone	2024-01	-5
12	2	5	Microwave Oven	2024-05	4
13	2	7	Tablet	2024-02	-10
14	2	10	Headphones	2024-02	1
15	2	11	Bluetooth Speaker	2024-04	-9

Conclusions

In this project, we designed and implemented a comprehensive logistics warehouse management system database. The database includes several key entities such as suppliers, products, catalog, customers, warehouses, inventory, purchase orders, warehouse transfers, and alerts. By referencing standard processes in real-world supply chain, logistics and warehouse management, we generated simulation data that closely mocks real-world applications, ensuring data integrity and consistency.

Major accomplishments

1. Data Integrity: Foreign key constraints ensure data consistency and integrity.
2. Automation: As a step toward a fully automated system with human intervention, we attempted to use triggers to create interactions between each action.
3. Realism: By referencing standard processes and data models in the actual world, our simulation data is more realistic and practical.
4. Frontend Development: We also created a frontend interface to display and interact with the data in the database. The frontend interface is intuitive and user-friendly, further enhancing the practicality of the system.

Some limitations

1. Data Simulation: Although we generated realistic simulation data, it is not actual operational data and may not cover all real-world business scenarios.
2. Process Automation: Even though we have some great ideas, it's hard for us to fully implement the functionality. The automation is only partially working, and more improvements need to be made.
3. Performance Optimization: For large-scale data queries and processing, our stored procedures and database design may require further optimization to improve performance.
4. Scalability: Further expansion and adjustments may be needed in actual applications based on specific business needs.

Building this system and applying the knowledge learnt through our course, we reinforced our understanding of course materials, while also exploring a lot of additional data manipulation skills and expanding our vision of database management. The team worked closely together with an online study room for real-time communication, where everyone contributed actively and demonstrated their leadership and cooperative skills. We intend to continue improving the database and the existing client application in the future, aiming to make it a project of real-world significance.

Author Contributions

If you worked in a team, describe how each member of your group contributed to the success of your project.

Zeyu Li

1. **Report Writing:** Worked on the abstract, Introduction, and database Design section of the report.

2. **Presentation**: Prepared content for the database section of the presentation and PPT.
3. **Database:** The main designer of the database with some adjustments from the group discussion and the main constructor for creating the database.
4. **SQL Code:** Wrote 7 SQL procedures, including logic, and triggers about processes related to purchase orders (PO) and sales orders (SO).
5. **Application Code:** I did not contribute to the application code.

Teng Liu

1. **Report Writing:** Completed some examples in the user cases section of the report.
2. **Presentation:** Prepared content for the functionality and user cases section of the PPT.
3. **Database:** Discussed the database structure with team members. Stimulated the creation of data for the database system, except for WarehouseTransfers.
4. **SQL Code:** Wrote test code for some procedures and improved the documentation's format.
5. **Application Code:** I did not contribute to the application code.

Liuyi Yang

1. **Report Writing:** I was primarily responsible for drafting the Data Sources and Methods, User Cases, Conclusions, and References sections
2. **Presentation:** I select and adjust templates for our team to use, structured the outline, and drafted the section 1 content
3. **Database:** I participated in team discussions and created the simulated data for WarehouseTransfers
4. **SQL Code:** I wrote 4 SQL procedures
5. **Application Code:** I did not contribute to the application code directly

Jinzhao Li

1. **Report Writing:** Completed some examples in the user cases section of the report (the application part).
2. **Presentation**: Prepared content for the demonstration and future application section for the presentation.

3. **Database:** Not contribute to the actual database code. In the early stages of model building, proposed the main macro-architecture of the system and the achievable functions.

4. **SQL Code:** No SQL Code written in the database, transplanted and modified multiple SQL procedures and codes in the application.

5. **Application Code:** The main designer of the application. Achieved tasks such as connecting to the database, organizing tables, designing UI, etc.

References

1. Affairs, Office of Regulatory. “Product codes and product code builder.” *U.S. Food And Drug Administration*, 12 May 2021, www.fda.gov/industry/import-program-tools/product-codes-and-product-code-builder.

2. Data.gov. “Montgomery County of Maryland - Warehouse and retail sales.” *Data.gov*, catalog.data.gov/dataset/warehouse-and-retail-sales/resource/b1ecfb86-7b0a-4619-a263-ffbc37f4e3e0.

3. Ordonez, Carlos, et al. “Extending ER models to capture database transformations to build data sets for data mining.” *Data & Knowledge Engineering*, vol. 89, Jan. 2014, pp. 38–54. https://doi.org/10.1016/j.datak.2013.11.002.

4. Xu, Da Wei, et al. “Analysis and design of Logistics Warehousing Management Information System based on RFID.” *Applied Mechanics and Materials*, vol. 644–650, Sept. 2014, pp. 6200–03. https://doi.org/10.4028/www.scientific.net/amm.644-650.6200.

5. Zhang, Ruifeng, et al. “Research on Intelligent Warehousing and Logistics Management System of Electronic Market based on Machine Learning.” *Computational Intelligence and Neuroscience*, vol. 2022, Mar. 2022, pp. 1–14. https://doi.org/10.1155/2022/2076591.