Project Documentation

Project Title: Retail Store & Inventory Management System
Team Members: Sarsembay Aibat, Yeskaliyev Nurbol, Seitkhan Zhannur, Nurzhanov Beket
University: SDU University

---

1. Data Dictionary

1)**inventory**

Stores information about the quantity of each product available in each store.

- **inventory_id: Unique identifier for the inventory record.**
- **product_id: Foreign key to the product table.**
- **store_id: Foreign key to the store table.**
- **quantity: Amount of product available in the store.**

---

2) **feedback**

Stores customer reviews about products.

- **feedback_id: Unique identifier for the feedback.**
- **customer_id: Foreign key to the customer who left the feedback.**
- **product_id: Foreign key to the reviewed product.**
- **rating: Numeric rating from 1 to 5.**
- **comment: Text of the review.**
- **date: Date when the feedback was submitted.**

---

3) **employee**

Contains information about the store employees.

- **employee_id: Unique identifier of the employee.**
- **name: Full name of the employee.**
- **position: Job position or role.**
- **store_id: Foreign key to the store where the employee works.**

---

4) **discount**

Tracks active discounts for products.

- **discount_id: Unique identifier of the discount.**

- **product_id: Foreign key to the discounted product.**
- **start_date: Start date of the discount.**
- **end_date: End date of the discount.**
- **persentage: Discount percentage applied.**

---

5) **customer**

Stores information about store customers.

- **customer_id: Unique identifier of the customer.**
- **name: Full name.**
- **phone: Phone number.**
- **email: Email address.**

---

6) **city**

Represents cities where stores are located.

- **city_id: Unique identifier of the city.**
- **name: City name.**

---

7) **purchaseOrder**

Represents orders placed by stores to suppliers.

- **order_id: Unique order ID.**
- **supplier_id: Foreign key to the supplier.**
- **store_id: Store that placed the order.**
- **order_date: Date the order was made.**
- **status: Current status of the order (e.g. pending, delivered).**

---

8) **purchaseOrderItem**

Describes individual items in each purchase order.

- **item_id: Unique item ID.**
- **order_id: Foreign key to the purchase order.**
- **product_id: Product being ordered.**
- **quantity: Quantity of the product ordered.**

---

9) **productCategory**

Defines categories for organizing products.

- **category_id: Unique identifier of the category.**
- **name: Name of the category.**

---

10) **product**

Contains product details.

- **product_id: Unique identifier of the product.**
- **name: Name of the product.**
- **price: Price per unit.**
- **category_id: Foreign key to the product category.**

---

11)**payment**

Tracks how sales were paid.

- **payment_id: Unique payment ID.**
- **sale_id: Related sale transaction.**
- **method: Method of payment (cash, card, etc.).**
- **amount: Total payment amount.**

---

12) **SupplierContact**

Stores contact information for suppliers.

- **contact_id: Unique contact record ID.**
- **supplier_id: Foreign key to the supplier.**
- **phone: Contact phone number.**
- **email: Contact email address.**

---

13) **Supplier**

Contains information about product suppliers.

- **supplier_id: Unique supplier ID.**
- **name: Name of the supplier.**

---

14) **store**

Describes physical store locations.

- **store_id: Unique store ID.**
- **name: Store name.**
- **street_number: Number of the building.**
- **street_name: Name of the street.**
- **city_id: Foreign key to the city.**

---

15) saleItem

Represents products sold in each sale transaction.

- **sale_item_id: Unique sale item ID.**
- **sale_id: Foreign key to the sale.**
- **product_id: Product that was sold.**
- **quantity: Quantity sold.**

---

16) sale

Stores complete sale transactions.

- **sale_id: Unique sale ID.**
- **customer_id: Buyer's ID.**
- **employee_id: Employee who handled the sale.**
- **store_id: Store where the sale took place.**
- **sale_date: Date of the sale.**

---

17) return

Keeps track of returned products.

- **return_id: Unique return record ID.**
- **sale_item_id: Item that was returned.**
- **return_date: Date when the product was returned.**

2. SQL Queries Explanation

This section describes the key SQL queries implemented in the project. They provide valuable statistics and analytics to support efficient store and inventory management.

---

1. Store List with Addresses

Purpose: Displays all stores with their full addresses.
How it works: Joins the store and city tables to include the city name.
Result: Returns store ID, name, street number, street name, and city name.

---

2. Product List with Categories

Purpose: Lists all products along with their categories.
How it works: Joins product and productCategory tables.
Result: Shows product ID, product name, and category name.

---

3. Employee List with Positions

Purpose: Retrieves names and positions of all employees.
How it works: Simple SELECT query from the employee table.
Result: Returns employee ID, name, position, and the store they work at.

---

4. Supplier List with Contact Information

Purpose: Retrieves suppliers along with their contact details.
How it works: Joins supplier and supplierContact tables.
Result: Displays supplier ID, name, phone number, and email.

---

5. Customer Purchase Count

Purpose: Counts total purchases per customer.
How it works: Uses a LEFT JOIN between customer and sale to include customers with no purchases.
Result: Shows customer name and number of purchases.

---

6. Highest Sales Revenue by Store

Purpose: Analyzes which store generates the highest revenue.
How it works: Joins store, sale, and saleItem tables, then calculates total revenue.
Result: Returns store name, month, total revenue, and number of sales.

---

7. Product Performance by Category

Purpose: Evaluates which product categories bring in the most revenue.
How it works: Joins productCategory, product, and saleItem, then groups by category.
Result: Displays number of products and total sales for each category.

---

8. Customer Purchase Behavior

Purpose: Analyzes customer shopping patterns.
How it works: Joins customer, sale, and saleItem, groups by customer.
Result: Shows frequency of purchases, total spent, and average order value.

---

9. Payment Method Analysis

Purpose: Identifies which payment methods are most commonly used.
How it works: Queries only the payment table, groups by method.
Result: Returns volume, total value, and percentage for each payment type.

---

10. Customer Feedback by Product

Purpose: Analyzes how customers rate products.
How it works: Joins product, productCategory, and feedback, then groups by product.
Result: Shows average rating and percentage of positive reviews.

11. Stores with Above-Average Total Sales

Purpose: Identifies which stores perform better than the average.
How it works:

- **Inner subquery calculates total sales for each store.**
- **Then calculates the average of all store sales.**
- **Outer query filters stores whose total sales are above this average.**
  Result: Returns store names with sales higher than average.

---

12. Most Active Suppliers

Purpose: Finds suppliers with the highest number of purchase orders.
How it works:

- **Counts the number of records in the purchaseOrder table per supplier_id.**
- **Uses GROUP BY and ORDER BY DESC to rank them.**
  Result: Shows suppliers and their order counts.

---

13. Average Order Amount per Customer

Purpose: Understands customer order behavior and average spending.
How it works:

- **First subquery calculates total spent by each customer.**
- **Outer query computes the average of those totals.**
  Result: Returns average purchase amount per customer.

---

14. Customers Who Shop at Multiple Stores

Purpose: Finds customers who buy from more than one store.
How it works:

- **Uses DISTINCT store_id per customer in the sale table.**
- **Filters those with more than one unique store.**
  Result: Lists loyal or flexible customers across locations.

---

15. Top-Rated Products

Purpose: Lists products with a high number of positive reviews.
How it works:

- **Joins feedback and product.**
- **Filters reviews with ratings 4 or 5 and count >= 5.**
  Result: Shows well-reviewed, popular products.

---

16. Products on Active Discount

Purpose: Displays products that are currently on sale.
How it works:

- **Subquery checks discount start and end dates vs. current date.**
- **Outer query fetches matching products.**
  Result: Lists all products with active discounts.

---

17. Customers with Product Returns

Purpose: Identifies customers who returned products.
How it works:

- **Nested subqueries follow the chain: return → saleItem → sale → customer.**
- **Final query returns customer names.**
  Result: Focuses on return behavior for customer analysis.

---

18. Most Expensive Product in Each Category

Purpose: Finds the highest-priced item per product category.
How it works:

- **Subquery gets max price per category.**
- **Outer query matches products to that max.**
  Result: Helps analyze premium products by category.

---

19. Employees Who Have Never Made a Sale

Purpose: Detects inactive sales staff.
How it works:

- **Subquery lists all employees present in sale.**
- **Outer query returns those who are not in that list.**
  Result: Identifies staff who may need training or work in other roles.

---

20. Unsold Products

Purpose: Identifies products that have never been sold.
How it works:

- **Subquery finds product IDs in saleItem.**
- **Outer query selects products not in that list.**
  Result: Supports inventory cleanup and sales strategy.

3.User Manual

This section describes how different users interact with the Retail Store & Inventory Management System.

Customer Interface

Customers can:

- **View the list of products and their categories.**
- **See product prices and current discounts.**
- **Make purchases by selecting products and quantities.**
- **Leave feedback and rate products they've bought.**
- **View their purchase history and return items if needed.**

This allows customers to easily browse, shop, and give opinions about their experience.

Employee Interface

Employees have access to more features:

- **Add, update, or delete products in the system.**
- **Manage stock levels by updating inventory.**
- **Process customer sales and handle returns.**
- **View basic customer data (names and contact info).**
- **Monitor their own sales performance.**

This helps staff to handle day-to-day operations smoothly.

Admin / Manager Interface

Admins or managers have full control of the system. They can:

- **Add and manage stores, cities, and employees.**
- **Register and update suppliers and their contact details.**
- **Create and manage product categories.**
- **Place purchase orders from suppliers and track status.**
- **Analyze detailed sales reports, customer behavior, and inventory statistics.**
- **Monitor discount effectiveness and return rates.**
- **Use SQL queries and visual reports for business insights.**

This role is responsible for business decision-making and overall management.

4. Triggers Explanation

In this project, multiple SQL triggers were implemented to automate database actions and ensure data consistency. Below is a description of each trigger and its purpose.

---

1. Auto-Set Return Date (BEFORE INSERT on return)

Purpose: Automatically fills in the return_date if it's missing.
How it works:

- **Trigger activates before a new row is inserted into the return table.**
- **If the user did not specify a return date, it assigns the current system date (SYSDATE).**
  Result: Ensures every return record has a valid date without user error.

---

2. Delete Discounts with Product (AFTER DELETE on product)

Purpose: Deletes all discounts related to a deleted product.
How it works:

- **Trigger activates after a product is deleted.**
- **It removes all entries in the discount table where product_id matches.**
  Result: Prevents orphan discounts from staying in the system.

---

3. Clean Up Inventory on Product Deletion (AFTER DELETE on product)

Purpose: Removes inventory records of deleted products.
How it works:

- **After a product is deleted, it deletes related rows from inventory.**
- **Matches by product_id.**
  Result: Keeps the inventory table clean and accurate.

---

4. Prevent Negative Inventory Quantity (BEFORE INSERT OR UPDATE on inventory)

Purpose: Blocks entries with negative inventory.
How it works:

- **Before inserting or updating a row in inventory, it checks the quantity.**
- **If the value is less than 0, it raises an error with a custom message.**
  Result: Maintains data integrity and avoids illogical states.

---

5. Update Inventory After Sale (AFTER INSERT on saleItem)

Purpose: Automatically subtracts sold quantity from inventory.
How it works:

- **After a new saleItem is added, this trigger runs.**
- **It uses PRAGMA AUTONOMOUS_TRANSACTION to perform an update independently.**
- **The product's quantity in inventory is reduced accordingly.**
  Result: Keeps inventory levels synchronized with sales in real-time.

These triggers improve automation, data accuracy, and reliability of the system without manual intervention.

5. Testing and Performance Considerations

This section explains how the system was tested and what performance improvements were considered during development.

Testing the Database

We performed the following types of testing:

- **Functionality Testing:**
  We checked that all CRUD operations (Create, Read, Update, Delete) work correctly across all main tables such as product, sale, customer, inventory, etc.
- **Trigger Testing:**
  Each trigger was tested by manually performing actions like inserting returns, deleting products, and making sales. The trigger logic executed correctly, as expected.
- **Query Testing:**
  All SQL queries were tested using realistic data. Special attention was paid to aggregate queries, subqueries, and JOIN operations to ensure accurate results.

- **Edge Case Testing:**
  We checked edge cases, such as:
  - **inserting a negative quantity (trigger correctly blocked it),**
  - **returning products without setting a date (trigger filled it in),**
  - **deleting a product with related records (cleanup triggers worked).**

Performance Optimization and Scalability

We implemented several techniques to ensure that the database will work efficiently, even with large volumes of data:

- **Normalization up to 3NF:**
  The schema was fully normalized, reducing data redundancy and improving update performance.
- **Indexes on Foreign Keys and IDs:**
  Indexes were created on all primary keys and foreign key columns to speed up joins and lookups (e.g., product_id, customer_id, store_id).
- **Efficient SQL Queries:**
  We avoided SELECT *, used appropriate GROUP BY and WHERE clauses, and applied LIMIT where needed to reduce data loads.
- **Use of Mock Data (100+ rows):**
  We populated the database with over 100 rows using mock data to simulate real-world conditions.

Conclusion

In this project, we designed and implemented a fully functional Retail Store & Inventory Management System. The system includes a well-structured database with 17 normalized tables, over 30 analytical SQL queries, and 5+ powerful triggers that automate key processes like inventory updates, product returns, and data cleanup.

We successfully:

- **Built an Entity-Relationship model with real-life connections.**
- **Populated the database with realistic test data.**
- **Created complex queries to analyze customer behavior, product performance, and store revenue.**
- **Ensured data integrity and automation using SQL triggers.**
- **Considered performance and scalability by applying normalization and indexing.**

This project helped us better understand real-world database design, team collaboration, and the importance of analytics in business decision-making.