🏪 Store Information System – Documentation

Overview

This Store Information System is designed to manage a comprehensive retail or inventory-based environment with support for multiple roles: **Admin**, **Employees**, and **Customers**. The system supports real-time product tracking, customer billing, vendor procurement, and service-level warranty management — all backed by PostgreSQL and optional MongoDB integrations.

🧱 Core Data Structure

1. Products

The `products` table stores all information about the items available in the store. It includes SKU identifiers, pricing (purchase and selling), stock levels, and warranty periods.

- `name`: Name of the product
- `sku`: Unique product identifier
- `purchase_price`: What the store paid to acquire it
- `selling_price`: Retail price
- `stock_quantity`: Current quantity in stock
- `warranty_period_months`: Warranty duration (in months)
- `vendor_id`: Foreign key linking to the vendor that supplies it

---

2. Vendors

The `vendors` table maintains supplier data for product sourcing and replenishment.

- `name`, `contact_email`, `contact_phone`, `address`: Vendor contact details
- Used for tracking vendor purchases and payment history

---

3. Customers

The `customers` table tracks store clients — either regular walk-in users or registered users with billing history.

- `name`, `email`, `phone`, `address`
- Linked to bills and purchases to track customer behavior

---

4. Bills

The `bills` table captures customer transactions:

- Each bill is associated with a customer
- Contains `total_amount`, `bill_date`, and `payment_status`
- Acts as the parent record for `sales`

---

5. Sales

The `sales` table logs product-level sales associated with a bill:

- Links products sold under each bill
- Captures quantity and price at time of sale

---

6. Product Warranty Status

Tracks the **warranty coverage** for every product sold:

- Connected to individual `sales`
- Fields include: `warranty_start_date`, `warranty_end_date`, and `status` (`Active`, `Expired`, `Claimed`)

---

7. Vendor Purchases

Captures inventory sourcing transactions from vendors:

- Tracks what was purchased, at what cost and quantity
- Also maintains payment status: `Unpaid`, `Partial`, `Paid`

---

8. Vendor Payments

Logs payments made to vendors, either full or partial:

- Includes `payment_method`, `amount_paid`, and notes
- Tied to specific purchase orders

🔐 Permissions and Roles

The system uses a centralized `permissions` table to control what each role can **read**, **insert**, **update**, or **delete**.

Role	Access
Admin	Full CRUD on all tables

Employee	Can view products and customers only
Customer	Can view only product listings and their own purchases (read-only)

✅ These permissions are checked dynamically by querying the `permissions` table before executing any database operation.

🧠 Smart Features

- **LLM-Powered Queries:** Users can ask natural language questions, which are translated into SQL queries (based on role-based permissions).
- **Markdown Response Formatting:** Query results are returned in readable, structured Markdown format for chat or interface consumption.
- **Vector Search for Store Info:** General information like warranty policy or store rules is handled via a vector-backed QA system (`qa_advanced`).

💡 Use Cases

- **Admin** checks which vendors need payment and restocks inventory
- **Employee** looks up product availability for customers
- **Customer** checks their past orders and warranty status

🛠️ Tech Stack

- **Database**: PostgreSQL (primary), MongoDB (optional)
- **LLM Integration**: For query classification and response generation
- **RBAC**: Enforced at the query layer using the `permissions` table

- **Frontend/Chat Interface**: Accepts natural queries and shows markdown-formatted answers