Final Project: ShopLite MySQL – Deployed Relational Website

Objective

Build and publicly deploy a small e-commerce style website that uses **only MySQL** as its database. No authentication, no transactions, and no concurrency control are required.

Functional Requirements

- **Public Site:**
  - Product catalog with search and category filter.
  - Product detail page.
  - Cart (client-side is fine) with add/remove/update quantities.
  - Mock checkout that records an order and its items in the database.
  - Order confirmation page showing the saved order.
- **Data Tools Page (no authentication):**
  - The Data Tools Page is a special admin-like page on your website, and it does not require any login. Its main purpose is to provide quick tools to keep the store's data accurate and up to date. One button should recompute order totals, which means it will go through all the orders in the database and make sure the total amount for each order is correct. For example, if an order has two products at $10 each, the system will recalculate and confirm that the total is $20. This helps fix any errors or mismatches that might happen when data is stored.
  - Another button should refresh the 90-day sales summary, which means it will update the report of sales from the last 90 days. This ensures that the system always shows the most recent and correct numbers, such as how many products were sold and how much money was earned. In short, this admin-like page is a simple way to run maintenance tasks that keep business data clean, accurate, and ready for generating reports.
- **Reporting:**
  - See which products brought in the most money during a time period you pick, like the last 30 days or the last 90 days.

  - A daily sales chart that shows how sales go up or down over time. To make the pattern easier to see, the chart can also include a simple trend line or a moving average line.
  - A warning that pops up when the number of items you have left is getting close to the minimum level you set for reordering. This helps you know when it's time to restock.

Technical Constraints

- Database: **MySQL**.
- App: Any backend (Node.js, Python, Java) and any frontend framework or plain HTML/JS.
- All data must reside in the relational database (no NoSQL or spreadsheets).
- No user authentication or login flows.
- Use primary keys, foreign keys, unique constraints, and basic value checks for data integrity.

Deployment Requirements

- Public URL for the website.
- Seed the database with realistic demo data (catalog, inventory, customers, orders).

API Requirements

- Get products (with optional search/filter).
- Get single product by ID.
- Create order (accept customer info and cart items; save to DB).
- Get order by ID.
- Get reports (top products, sales series, low-stock).
- Trigger data tools (recompute totals, refresh summaries).

Deliverables

- Live URLs for the website and API.
- Source repository with:
  - DB scripts.
  - API code.

  - Frontend code.
  - README with setup instructions, deployment steps, API documentation, and an ER diagram.

- Short demo video ($\leq$ 3 minutes) showing:
  - Browsing and adding products to cart.
  - Checkout flow and order confirmation.
  - Viewing reports.
  - Running data tools and observing effect.
- One-page design brief explaining relational model, constraints, and indexing plan.
- Brief performance note demonstrating that indexes improve at least one query.

Workflow Checklist for Students

1. Provision a managed MySQL database.
2. Design the relational schema and implement tables with constraints.
3. Load seed data (products, inventory, customers, orders).
4. Implement backend API for data operations and reports.
5. Build frontend for catalog, product detail, cart, checkout, confirmation, reports, and tools.
6. Deploy database, API, and frontend; configure environment variables.
7. Test all functionality and record the demo video.
8. Submit URLs, repository link, README, design brief, and performance note.