



Project Proposal

Smart Shop Management System

Guided By

Anuj Kumar

Created By:-

Name : Aditya Pal

AFID: AF04971732

Batch Code : ANP-D2405

Course Code : ITPR

Name: Pranjal Khokhar

AFID: AF04991772

Batch Code: ANP-D2405

Course Code : ITPR

Table Of Contents

- 1. Introduction**
- 2. Objective**
- 3. Project Category**
- 4. Analysis**
 - Modules and Description
 - Database Design (Schema)
 - ER Diagram
 - Data Flow Diagram
- 5. Complete Structure**
 - Process Logical Diagram
- 6. Platform Used**
 - Hardware Requirement
 - Software Requirement
- 7. Future Scope**
- 8. Bibliography**



Introduction

In the current era of digital transformation, small and medium-scale retail businesses face significant challenges in managing their day-to-day operations manually. Traditional paper-based bookkeeping is prone to calculation errors, lacks real-time inventory tracking, and makes it difficult to adapt to changing tax regulations (like GST updates).

To address these issues, we have developed the **Smart Shop Manager**. This is a command-line application developed using **Java, JDBC, MySQL, and Maven**. This is a robust, Java-based Console Application designed to automate the entire retail lifecycle.

It simulates a real-world scenario where it allows the **Administrator** to manage the supply chain (Brands, Categories, Taxes), while allowing **Customers** to browse products, add items to a cart, and generate professional invoices dynamically. Advanced features such as **dynamic tax calculation (GST)**, **low-stock alerts**, and **sales history tracking** make the system realistic and feature-rich.

All functionalities operate through **Java's CLI (Command Line Interface)**, making the system lightweight, platform-independent, and easy to run in academic labs using **Eclipse IDE**.



Objective

The primary objective of this project is to build a fully functional console-based retail system that showcases real-world software engineering concepts such as **Data Normalization**, **Maven Build Management**, and **JDBC Transaction Handling**.

Specific objectives include:

1. To simulate real-world retail workflows The project recreates the complete cycle of store management:

- a. Configuring Master Data (Brands, Taxes, Categories).
- b. Adding products and updating stock.
- c. Customer cart management.
- d. Generating invoices with tax breakdown.

2. Functional Objectives:

- Automation: To eliminate manual calculations for billing, taxes, and discounts.
- Dynamic Configuration: To allow the shop owner to update Tax Rates (e.g., changing GST from 12% to 18%).
- Inventory Control: To prevent "Out of Stock" scenarios by strictly tracking quantity updates in real-time.

3. Technical Objectives:

- Database Normalization: To implement a complex 8-table schema that minimizes data redundancy using Foreign Keys.
- Java Core Concepts: To practically apply **JDBC** (for database connectivity), **Collection Framework** (ArrayList for the shopping cart), and **Exception Handling**.
- Build Management: To utilize **Maven** for managing project dependencies (MySQL Connector) and directory structure.

4. To provide a strong academic example The project serves as a comprehensive demonstration of:

- a. Core Java (Collections, Exceptions).
- b. JDBC Prepared Statements.
- c. SQL Joins and Normalization.
- d. Team Collaboration (2-Member development).

Project Category

1. **Application Type: Desktop Console Application (CLI)** This project is developed as a **Command Line Interface (CLI)** application. Unlike Graphical User Interfaces (GUI) that rely on windows and buttons, this system interacts with the user through standard input/output streams.

2. **JDBC-Based CRUD System:** Demonstrates professional JDBC usage including Connection Pooling and Transaction Management.



Analysis

Modules and Description :

The project is architected into 4 core functional modules:

Module 1: Product Management

- **1.1 Product Creation:** Admin adds items to the **products** table, linking them to **Categories**, **Brands**, and **Tax Slabs** via Foreign Keys.
- **1.2 Product Update:** Modify details like Price, Tax Rate, or Brand associations.
- **1.3 Product List:** View the complete catalog sorted by Category or Brand.
- **1.4 Product Details:** Fetch specific details including Tax percentage and Category ID.
- **1.5 Product Deletion:** Remove obsolete items (with validation checks against active sales).

Module 2: Inventory Management

- **2.1 Stock Status:** Check available quantity filtered by Brand or Category.
- **2.2 Stock Refill:** Update stock quantities when new shipments arrive.
- **2.3 Low Stock Alert:** Auto-highlight items below the defined threshold.
- **2.4 Stock Adjustment:** Auto-deduct stock immediately after a sale is committed.

Module 3: Cart & Customer Management

- **3.1 Customer Registration:** New users must create an account by providing their Name, Phone Number, and a secure Password.
- **3.2 Secure Login:** Existing customers must authenticate using their Phone Number and Password to access the shopping interface.
- **3.3 Catalog Browsing:** Once logged in, customers view products with clear, tax-inclusive pricing.
- **3.4 Cart Logic:** Uses a temporary **ArrayList** to store selected items. This allows users to add, remove, or modify quantities before finalizing the purchase.

Module 4: Billing & Sales Management

- **4.1 Total Calculation:** The system automatically calculates the Grand Total: **Sum of (Price + Tax) * Quantity** for all items in the cart.
- **4.2 Sale Generation:** The system creates a unique **Sale ID** in the **sales** table (The Bill Header).
- **4.3 Item Recording:** It iterates through the cart and saves individual items to the **cart_items** table (The Bill Content), linking them via the Sale ID
- **4.4 Inventory Update:** Deducts the sold quantity from the **products** table

Database Design :

The database design follows relational schema principles and uses **8 Normalized Tables** with proper Foreign Keys to maintain data integrity.

1. **users**: Stores login credentials for Admin/Staff.

```
mysql> desc users;
```

Field	Type	Null	Key	Default	Extra
user_id	int	NO	PRI	NULL	auto_increment
username	varchar(50)	NO	UNI	NULL	
password	varchar(50)	NO		NULL	
role	varchar(20)	NO		NULL	

2. **categories**: Groups products logically.

```
mysql> desc categories;
```

Field	Type	Null	Key	Default	Extra
category_id	int	NO	PRI	NULL	auto_increment
name	varchar(50)	NO		NULL	
description	varchar(100)	YES		NULL	

3. **brands**: Manufacturer lookup.

```
mysql> desc brands;
```

Field	Type	Null	Key	Default	Extra
brand_id	int	NO	PRI	NULL	auto_increment
brand_name	varchar(50)	NO		NULL	

4. **taxes**: Dynamic Tax Configurations.

```
[mysql]> desc taxes;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| tax_id | int | NO | PRI | NULL | auto_increment |
| tax_name | varchar(20) | YES | | NULL | |
| rate | double | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
```

5. **products**: Master Inventory (Links to above tables).

```
[mysql]> desc products;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| product_id | int | NO | PRI | NULL | auto_increment |
| name | varchar(100) | NO | | NULL | |
| price | double | NO | | NULL | |
| quantity | int | YES | | 0 | |
| category_id | int | YES | MUL | NULL | |
| brand_id | int | YES | MUL | NULL | |
| tax_id | int | YES | MUL | NULL | |
+-----+-----+-----+-----+-----+-----+
```

6. **customers**: Customer loyalty data.

```
[mysql> desc customers;
```

Field	Type	Null	Key	Default	Extra
customer_id	int	NO	PRI	NULL	auto_increment
name	varchar(100)	YES		NULL	
phone	varchar(15)	YES	UNI	NULL	
email	varchar(100)	YES		NULL	
password	varchar(50)	NO		NULL	

7. **sales**: Bill Header.

```
[mysql> desc sales;
```

Field	Type	Null	Key	Default	Extra
sale_id	int	NO	PRI	NULL	auto_increment
sale_date	datetime	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
total_amount	double	YES		NULL	
customer_id	int	YES	MUL	NULL	

8. **cart_items**: Bill Details.

```
[mysql> desc cart_items;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
sale_id	int	YES	MUL	NULL	
product_id	int	YES	MUL	NULL	
qty	int	YES		NULL	
price_at_sale	double	YES		NULL	

ER Diagram :

Relationships - derived directly from your 8-Table Database Schema and Java implementation.

1 : Master Data Relationships (Product Setup)

These relationships define how a product is categorized and taxed.

- Category —> Product
 - Relationship: Many Products (e.g., Mouse, Keyboard) belong to One Category (e.g., Electronics).
 - Cardinality: **N : 1** (Many-to-One)
 - Link: products.category_id references categories.category_id.
- Brand —> Product
 - Relationship: Many Products manufactured by One Brand (e.g., Samsung).
 - Cardinality: **N : 1** (Many-to-One)
 - Link: products.brand_id references brands.brand_id.
- Tax —> Product
 - Relationship: Many Products falls under One Tax Slab (e.g., GST 18%).
 - Cardinality: **N : 1** (Many-to-One)
 - Link: products.tax_id references taxes.tax_id.

2. Transaction Header Relationships (The Bill)

These relationships define "Who bought what, who sold it, and how they paid."

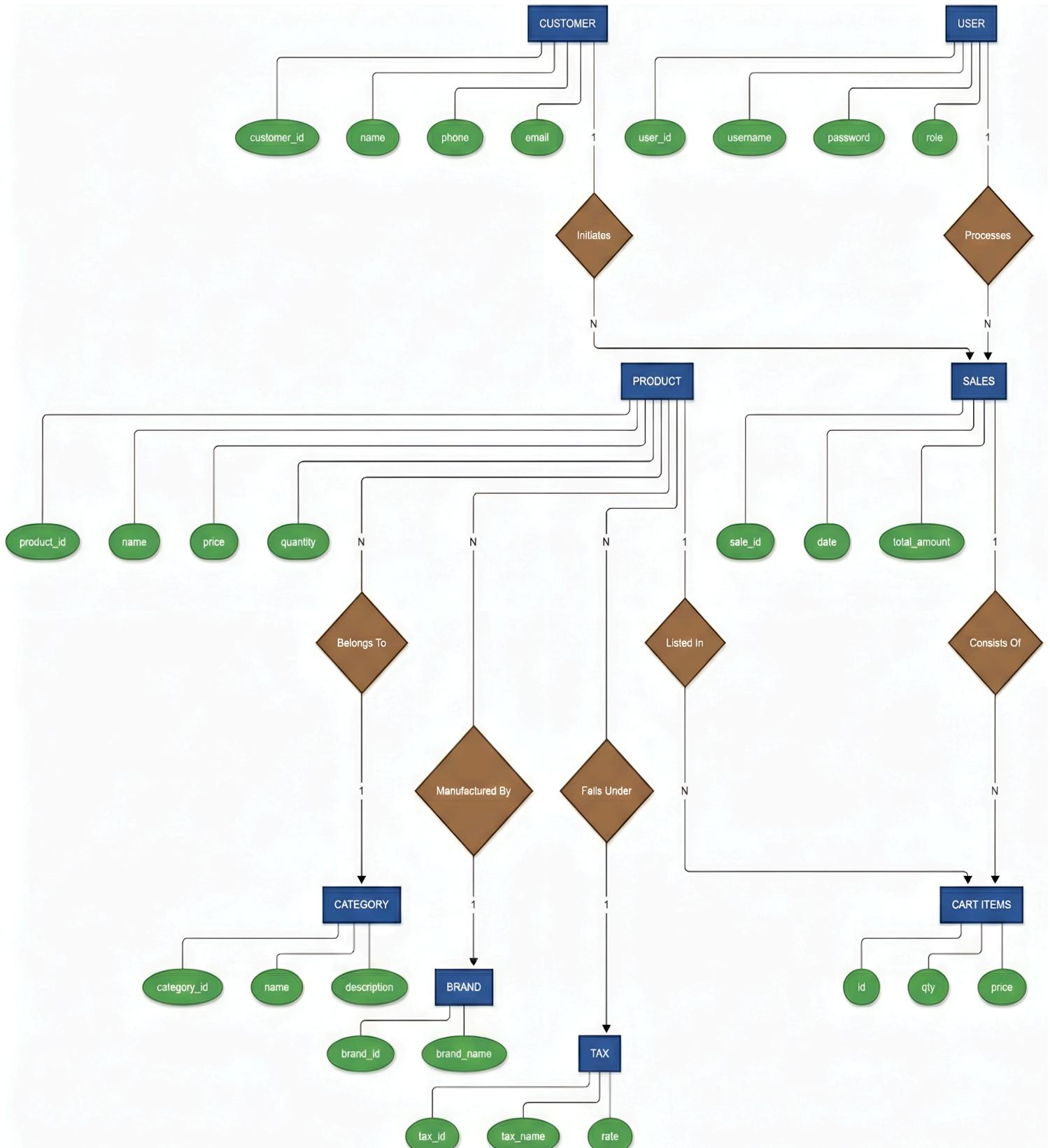
- User —> Sale
 - Relationship: One User (Admin) processes many Sales.

- Cardinality: **1 : N** (One-to-Many) (One cashier handles multiple sales).
 - Link: `sales.user_id` references `users.user_id`.
- Customer —> Sale
 - Relationship: One Customer *makes* many Sales (Purchases).
 - Cardinality: **1 : N** (One-to-Many)
 - Link: `sales.customer_id` references `customers.customer_id`.

3. Transaction Detail Relationships (The Items)

This is the bridge between the Bill and the Inventory.

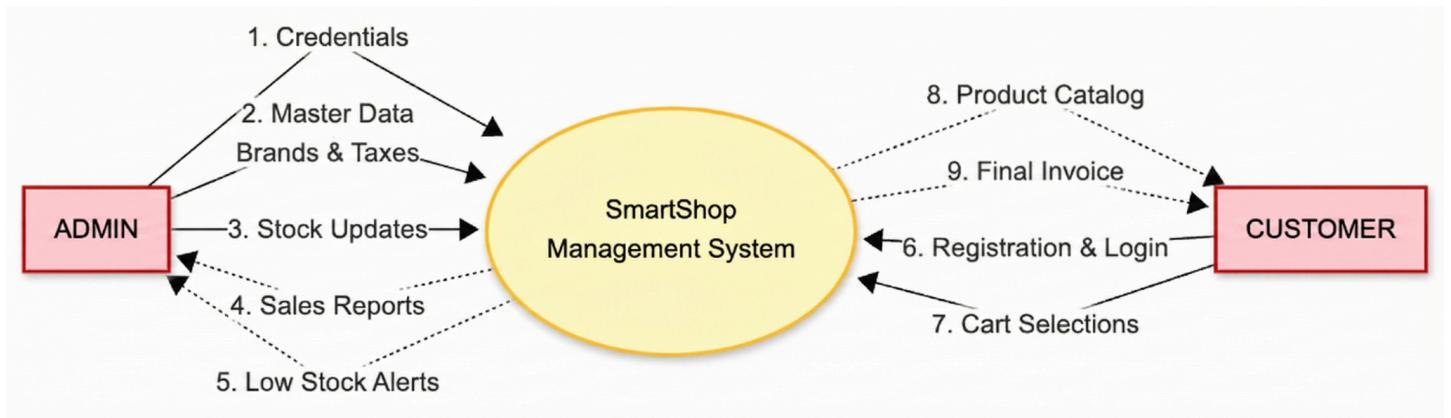
- Sale —> Cart_Items
 - Relationship: One Sale (Invoice) *contains* many Cart Items (Rows in the bill).
 - Cardinality: **1 : N** (One-to-Many)
 - Link: `cart_items.sale_id` references `sales.sale_id`.
- Product —> Sale_Item
 - Relationship: One Product *appears in* many cart Items (Across different bills).
 - Cardinality: **1 : N** (One-to-Many)
 - Link: `cart_items.product_id` references `products.product_id`.



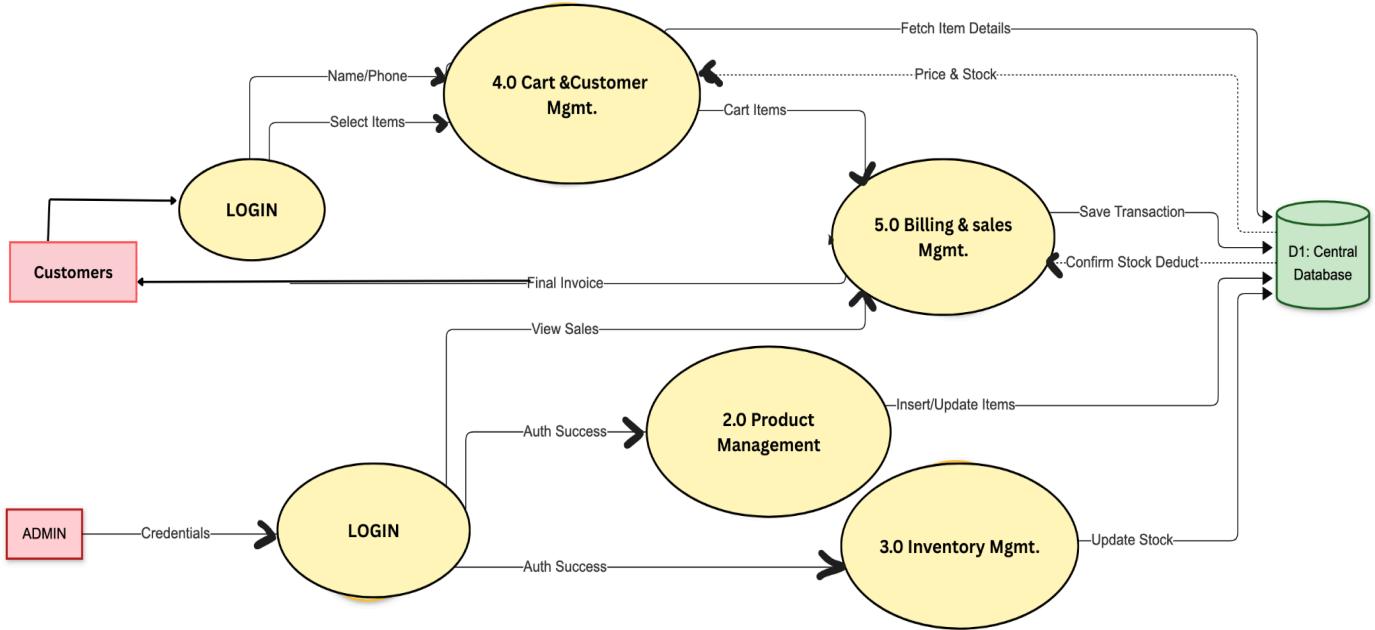
Data Flow Diagram (DFD):

1. **Input:** Admin inputs Master Data; Customer inputs Order.
2. **Process:** System checks Stock -> Fetches Tax from DB -> Calculates Total.
3. **Storage:** Transaction saved to sales/cart_items; Stock deducted in products.

Data Flow Diagram Level 0:

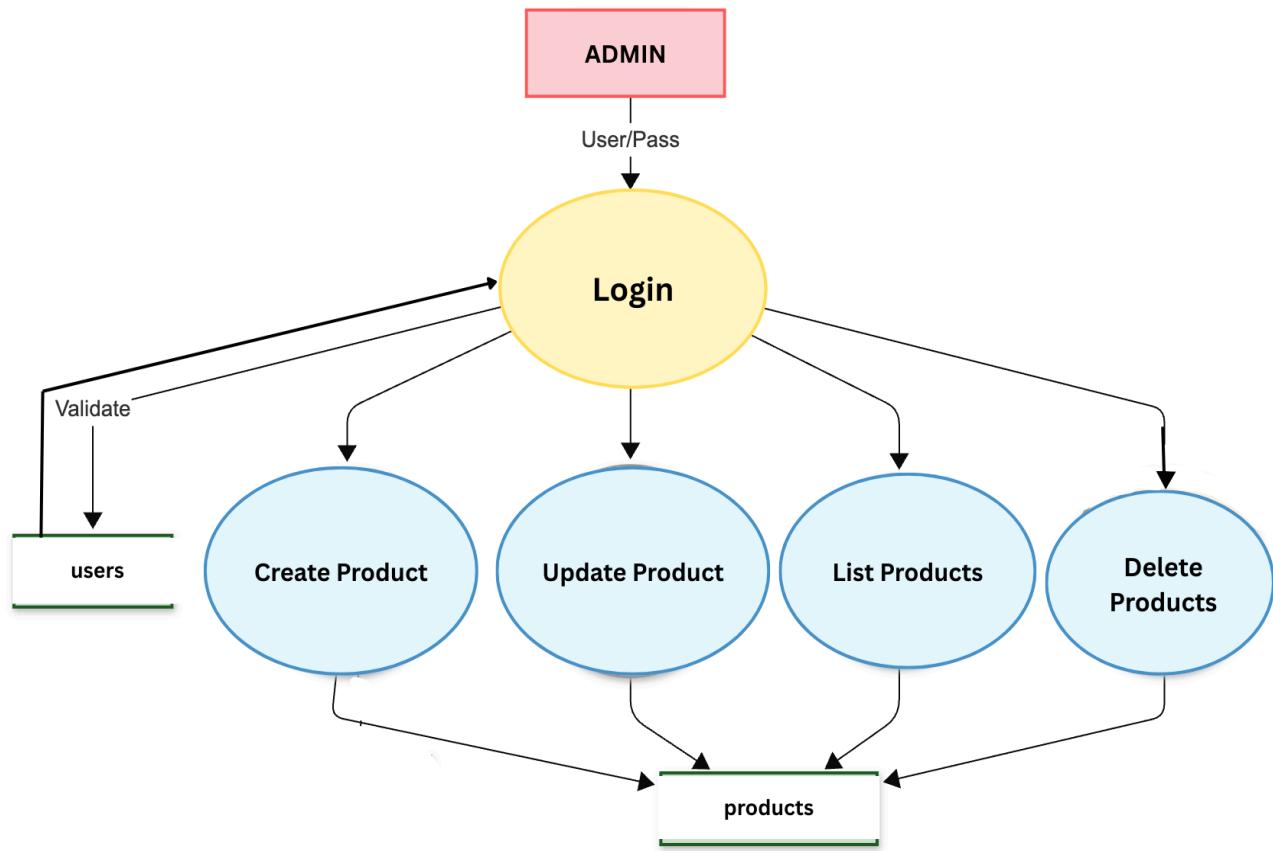


Data Flow Diagram Level 1:

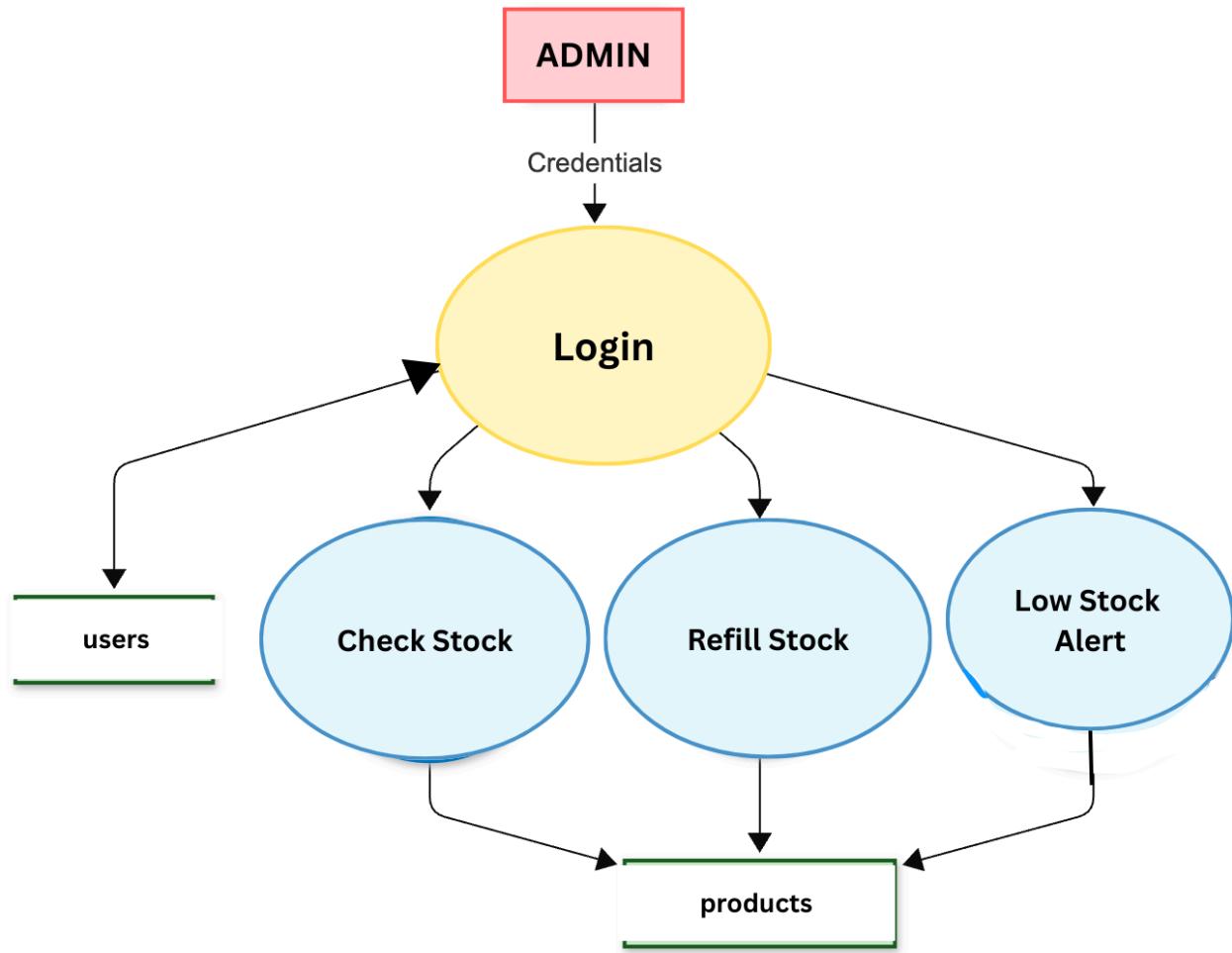


Data Flow Diagram Level 2:

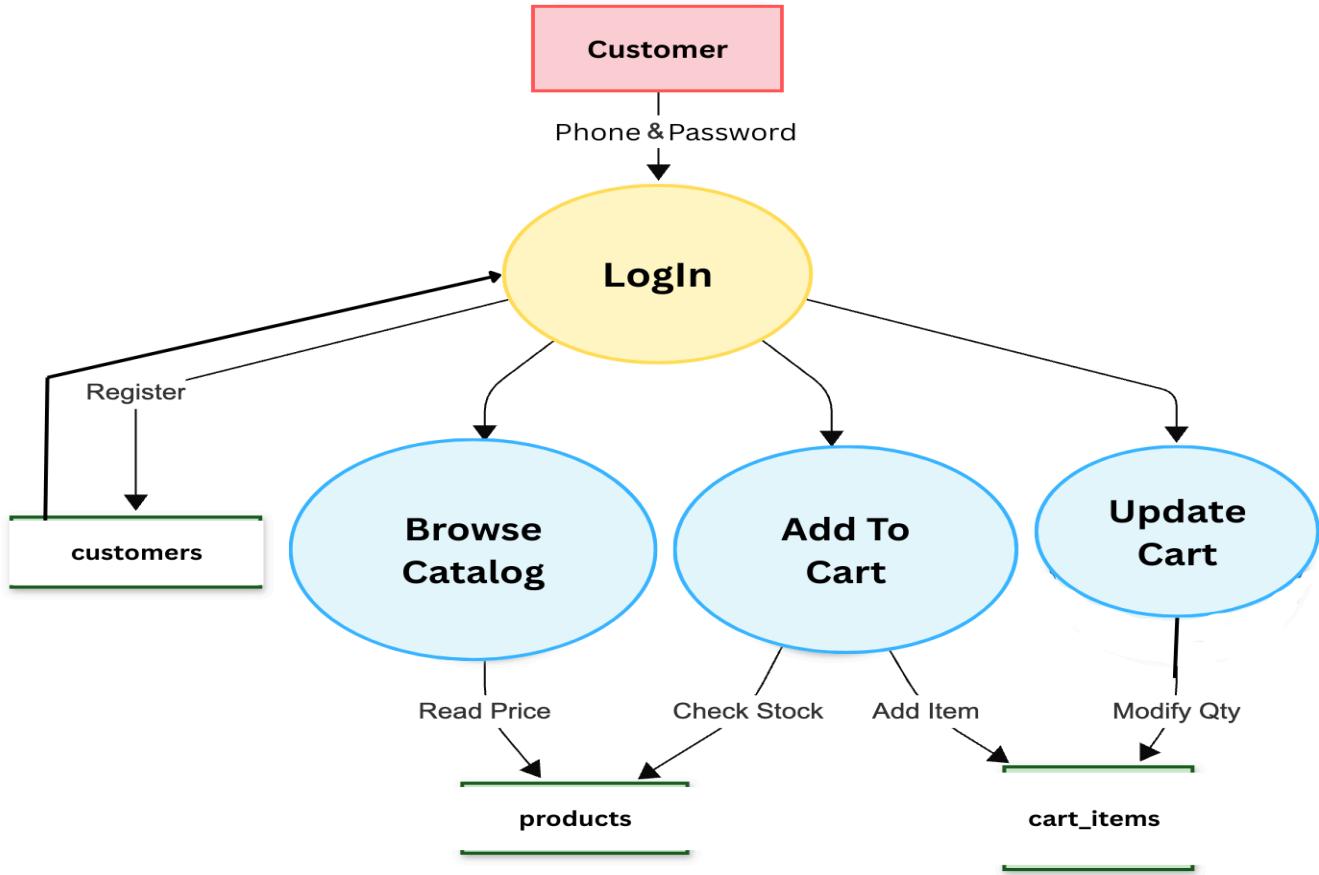
Module 1: Product Management



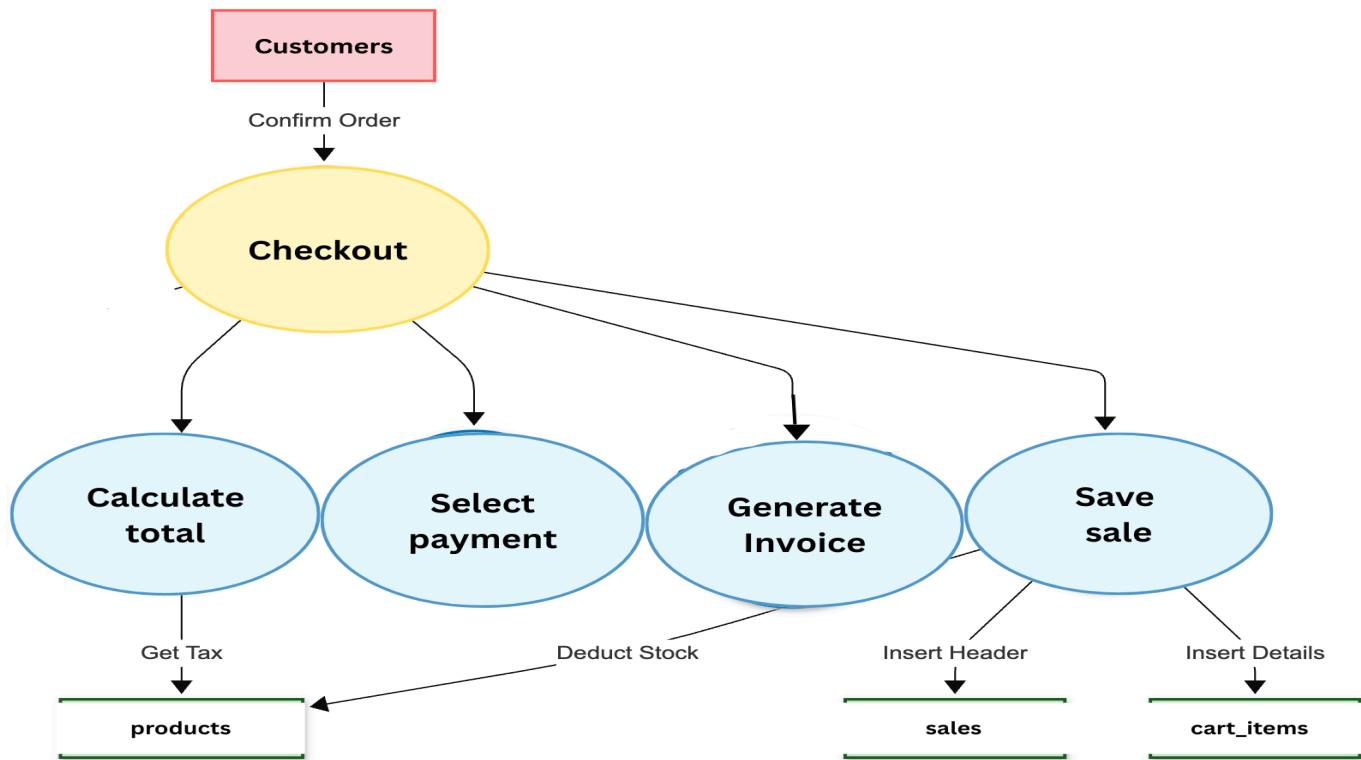
Module 2: Inventory Management



Module 3: Cart & Customer Management



Module 4: Billing & Sales Management



Complete Structure

This design pattern ensures a clean separation of concerns, where the data (Model), database logic (DAO), and user interaction (Controller) are kept independent. This structure makes the application easy to debug, test, and scale.

The project is divided into four primary packages/layers:

1. Model Layer

Each class acts as a digital mirror of a specific database table. These classes use private variables for data security (Encapsulation) and public Getter/Setter methods for access.

- Entities:
 - **User**: Represents Admin/Staff credentials and roles.
 - **Product**: Represents inventory items with Price, Quantity, and Foreign Keys.
 - **Customer**: Represents buyer details (Loyalty data)and Login.
 - **Sale**: Represents the Bill Header (Date, Total).
 - **cartItem**: Represents individual line items inside a bill.
 - **Category, Brand, Tax**: Represent the lookup data configuration.
- Purpose:
 - Ensures type safety within the Java application.
 - Facilitates the easy transfer of data between the Database and the UI.

2. DAO Layer

The DAO (Data Access Object) layer handles all direct communication with the MySQL database. It abstracts the SQL logic from the rest of the application.

- Primary DAO Classes:
 - UserDAO: Handles Login authentication logic.
 - ProductDAO: Manages CRUD operations for Inventory (Add, Update Stock, Delete).
 - SalesDAO: The most complex class; it handles the Atomic Transaction of generating a bill, saving items, and deducting stock simultaneously.
 - MasterDAO: Fetches dropdown lists for Categories, Brands, and Taxes.
 - CustomerDAO: Checks if a customer exists and registers new ones automatically.
- Responsibilities:
 - Executing SELECT, INSERT, UPDATE queries using JDBC PreparedStatements.
 - Managing SQL Exceptions.
 - Handling Database Transactions (Commit/Rollback).

3. Utility Layer

This layer contains infrastructure helper classes used system-wide to maintain connectivity.

- Core Class: DataBaseUtil.java

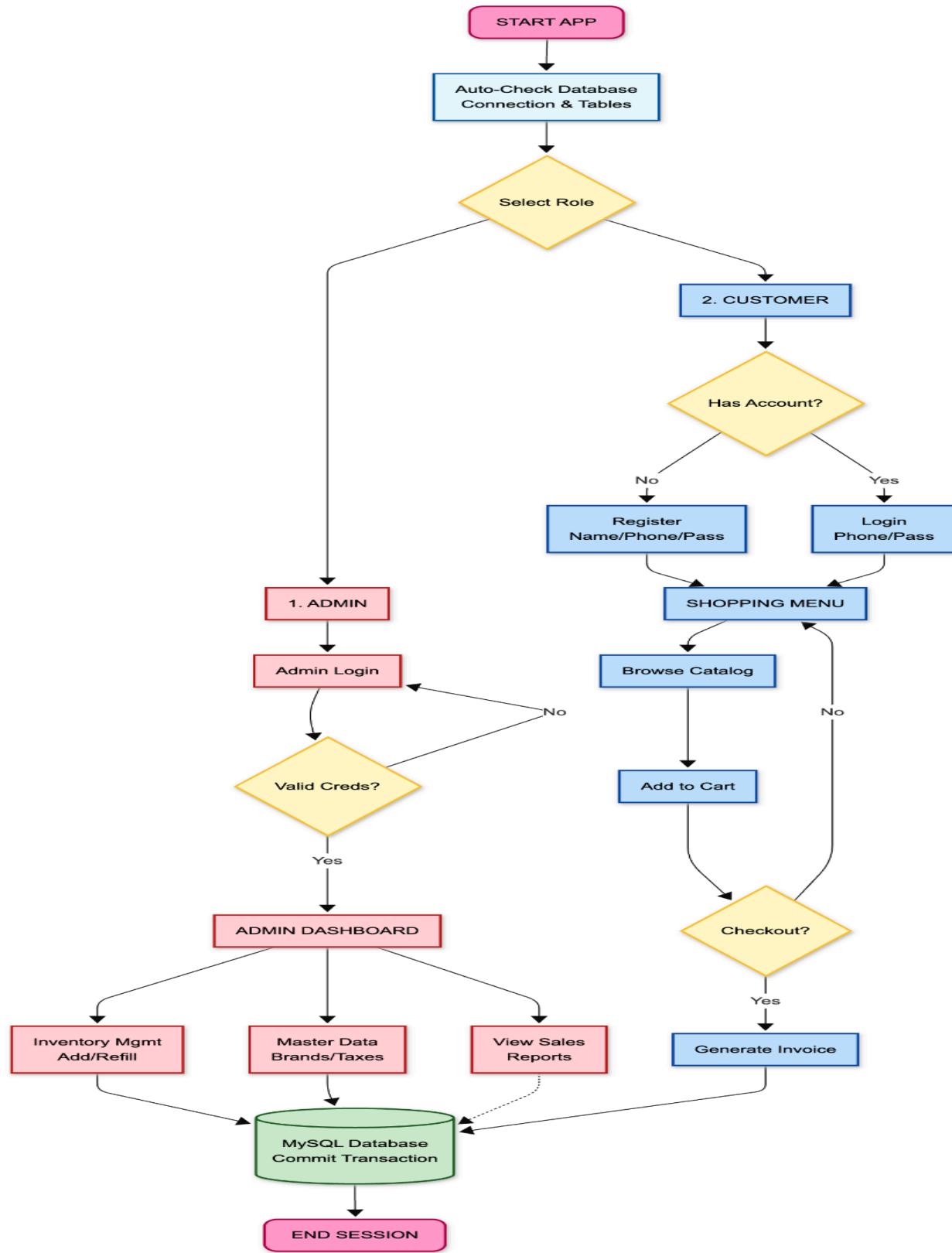
- Key Functions:
 - Connection Management: Loads the JDBC Driver and establishes a connection to MySQL.
 - Lazy Initialization: A unique feature of this project. On the very first run, it checks if the 8 tables exist. If not, it automatically creates the database structure and inserts default admin credentials. This makes the software "Plug-and-Play."

4. Controller Layer

The Main class serves as the entry point and the "Brain" of the application.

- Class: App . java
- Responsibilities:
 - UI/UX Flow: Displays the interactive Console Menu.
 - Input Handling: Uses the Scanner class to capture user choices.
 - Routing: Directs the user to the Admin Dashboard (if logged in as Admin) or the Customer Shopping Interface.
 - Business Logic: Validates inputs (e.g., ensuring a user cannot buy more quantity than available in stock) before calling the DAO layer.

Process Logical Diagram





Platform Used

Hardware Requirement

- **Processor:** Intel Core i3 or higher.
- **RAM:** 4 GB minimum.
- **Storage:** 500 MB free space.

Software Requirement

- **Operating System:** Windows 10/11, MacOS, or Linux.
- **Programming Language:** Java SE (JDK 1.8 or higher).
- **Build Tool:** Apache Maven 3.x.
- **IDE:** Eclipse IDE (Enterprise Java Developers edition).
- **Database:** MySQL Server 8.0.
- **Dependencies:** mysql-connector-j (Managed via pom.xml).



Future Scope

The current version is a fully functional CLI application. However, it provides a foundation for:

1. **Web-Based Frontend:** Migrating the Logic to a **Spring Boot** backend with a React JS frontend.
2. **Barcode Integration:** Adding a physical scanner to read Product IDs automatically.
3. **Payment Gateway:** Replacing simulated payments with real UPI/Stripe integration.
4. **Graphical Dashboard:** Adding charts for "Top Selling Products" using JavaFX.



Team Work Distribution

Name : Aditya Pal

Key Responsibilities:-

- Database Design
- Module 1 : Product Management
- Module 2 : Inventory Management
- DAO Implementation

Name : Pranjal Khokhar

Key Responsibilities:-

- Module 3:Cart And Customer Management
- Module 4: Billing & Sales Management
- Controller Layer



Bibliography

- Core Java: Concepts of Collections, JDBC, and OOPs.
- Build Tools: Apache Maven Documentation (maven.apache.org).
- Database: MySQL Normalization principles.
- MySQL Official Documentation – <https://dev.mysql.com/doc/>
- JDBC MySQL Connector :-
<https://dev.mysql.com/downloads/connector/j>
- GeeksforGeeks – Java & JDBC tutorials
- MySQL Official Documentation (Foreign Keys & Normalization).

