1. Introduction

The goal of this project is to design and implement a complete relational database for a simplified online retail system. Modern information systems rely on well-structured databases to store, manage and retrieve data efficiently and this project aims to demonstrate the full process of building such a system from the ground up. The work follows the standard lifecycle of database development: requirement analysis, conceptual modeling, logical design, normalization, physical implementation and query formulation.

The first stage consisted of identifying the core components of an online retail environment. The system must represent customers, registered users, items offered for sale, item categories, bank accounts and the relationships between them such as purchases,based on and provide. Based on these requirements, an Entity Relationship (EER) model was created to visually capture the entities, their attributes and the cardinalities of the relationships. Special attention was given to representing different types of customers through specialization (Customer → Registered Customer), many-to-many relationships (such as Registrated Customer–BankAccount) and associative entities (such as Provide).

After defining the conceptual schema, it was translated into a Relational Schema by identifying primary keys, foreign keys and constraints that enforce referential integrity. The schema was then evaluated through normalization principles, ensuring that all relations satisfy the conditions of Third Normal Form (3NF). This step eliminates data redundancy and prevents update anomalies contributing to a consistent and stable database structure.

The database was subsequently implemented in MySQL. All tables were created using SQL command and sample data was inserted to test the design. Finally, a set of SQL queries including joins, nested queries and aggregate operations was developed to demonstrate how the database supports meaningful data extraction and analysis.

This project illustrates not only the technical aspects of database construction but also the underlying logic behind modeling decisions, integrity constraints and real-world applicability. The final result is a coherent, normalized and functional relational database aligned with the requirements of an online retail system.

ENTITIES AND THEIR ROLES

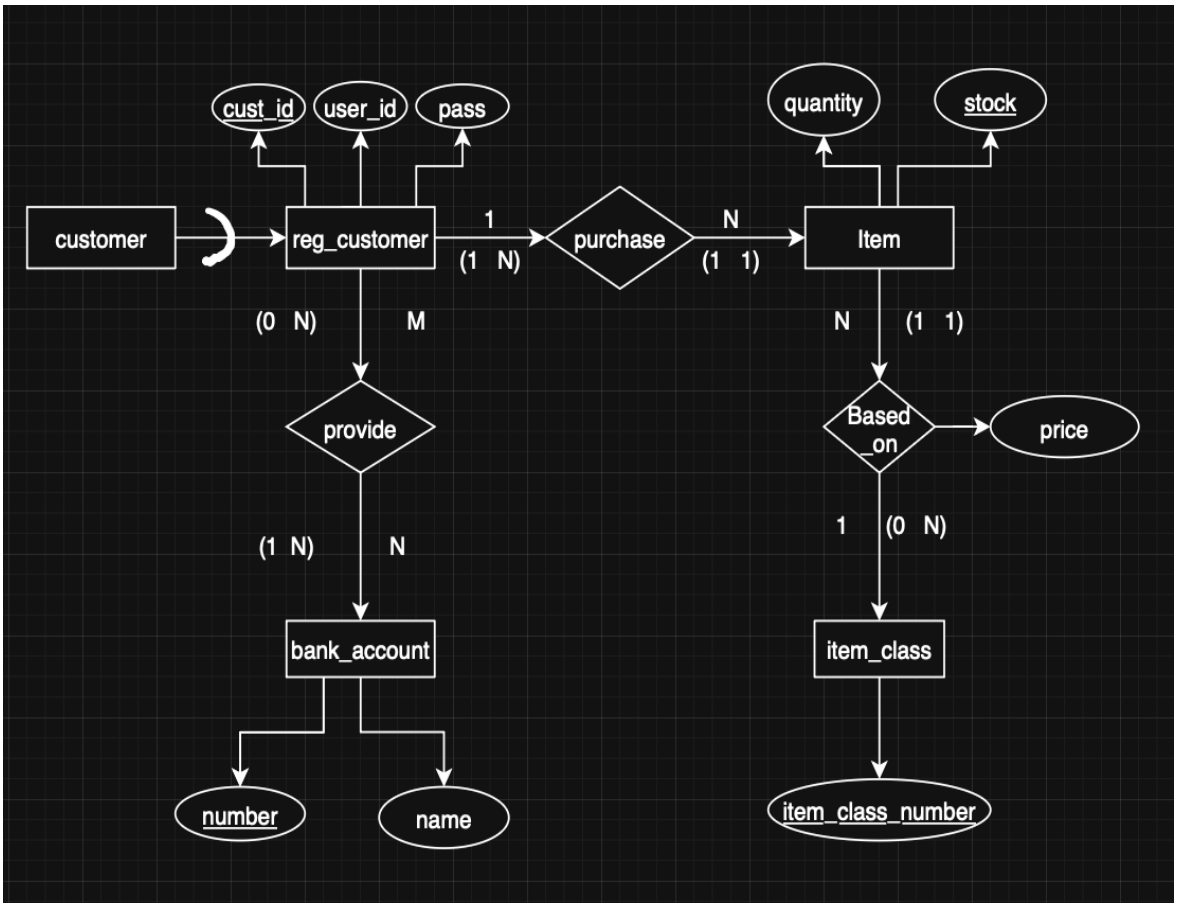
- Customer: Represents all customers in the system. This entity stores general customer information such as name, address and email.
- Registered Customer (reg_customer): This is a subtype of Customer. Only registered customers can log in and perform purchases. Therefore, this subtype contains additional authentication-related attributes such as user_id and password.
- Bank Account: Each registered customer can use one or more bank accounts. A bank account has a unique account number and a bank name.
- Item: Represents products sold in the retail system. Each item has a stock identifier ,a quantity available and a price.
- Item Class: Products are grouped into different categories such as Budget, Standard, Premium. This entity stores class identifiers and descriptions.

RELATIONSHIPS AND THEIR CONSTRAINTS

- Provide (Customer–Bank Account): A registered customer may provide several bank accounts and each bank account can belong to multiple customers. This is a (M:N) relationship

- Based On (Item–Item Class): Each item belongs to exactly one item class while each item class can contain many items (1:N).The relationship contains an attribute price, which following standard relational conversion rules is stored inside the item table.
- Purchase (Customer–Item):A registered customer can purchase zero or more items; an item can be purchased by zero or more customers.This is another M:N relationship is represented by the associative table purchase(cust_id, stock).

EER Diagram of the Online Retail Database:



2. Conceptual Design (EER Diagram Explanation)

The EER diagram represents the conceptual structure of the online retail database. Each entity and relationship was designed based on the logical requirements of the system and the cardinalities reflect real-world constraints.

Customer → Registered Customer (Specialization)

The system distinguishes between general customers and customers who create a registered account.

- customer represents all individuals in the system.
- reg_customer is a subset of customer and includes login-related attributes (user_id, password).
- It is also disjoint because a customer cannot belong to multiple subtypes.

This separation improves data integrity and reflects real-world scenarios where a retail system has both guest and registered users.

Regular_Customer – Purchase – Item Relationship

Purchases are modeled using the associative relationship purchase, because:

- One customer can buy many items.
- One item can be purchased by many different customers.

→ This naturally creates a 1:N relationship that represented as an associative entity with a primary key composed of (cust_id, stock).

The cardinalities:

Between customer and purchase:

- A customer can make 0 or many purchases (0 N).
- Each purchase belongs to exactly 1 customer (1 1).

Between item and purchase:

- Each purchase references exactly one item (1..1).

This design aligns with typical e-commerce systems.

Item – Item_Class (1:N Classification)

Each item belongs to one category (item_class), and each class can contain multiple items.

Cardinality (1 N) from item_class → item means:

- One class can describe many items.
- An item must belong to exactly one class.

The price attribute is modeled as dependent on the item_class through the Based_on relationship because the design assumes each class has its own base pricing structure. This also simplifies category-level queries.

Regular Customer – Bank Account – Provide Relationship (M:N)

Customers can connect to multiple bank accounts

Thus, the relationship is many-to-many that represented through the associative entity provide.

Between customer and provide:

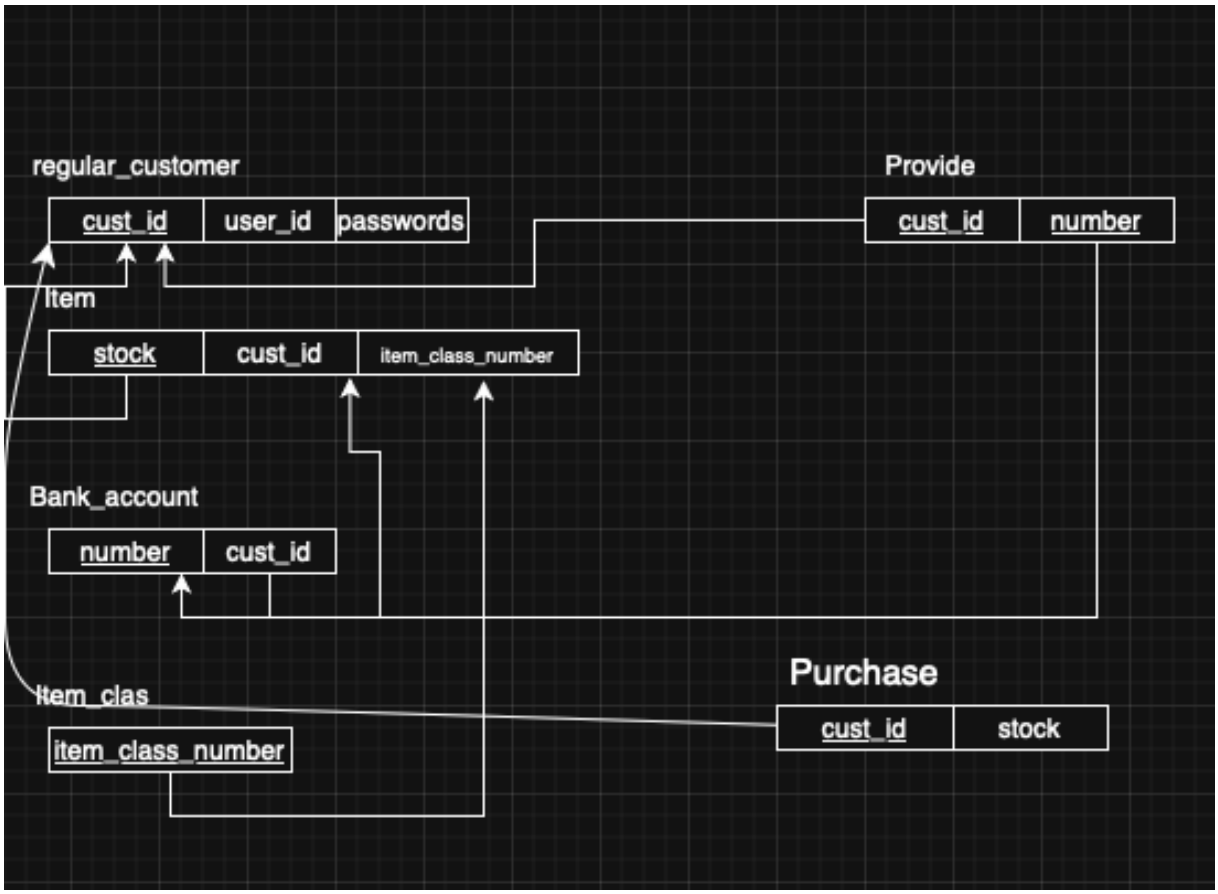
- A customer may have 0 or many connected accounts (0 N).
- Each provide relation belongs to exactly 1 customer (1 1).

Between bank_account and provide:

- A bank account can appear in multiple customer relations (1 N).
- Each provide entry connects exactly one account.

This design ensures flexibility while keeping transactional payment information normalized.Using natural keys for items and bank accounts maintains consistency, while surrogate integer keys simplify customer management.

Relational schema:



Normalization

The database schema was normalized to Third Normal Form (3NF) in order to eliminate redundancy, prevent update anomalies and ensure data consistency. The normalization process was carried out systematically by analyzing functional dependencies in each relation.

First Normal Form (1NF)

A relation is in First Normal Form (1NF) if:

- All attributes contain atomic (indivisible) values
- There are no repeating groups or multivalued attributes

Application in the model:

- All tables use atomic attributes (e.g., cust_name, price, quantity).
- Multivalued information such as bank accounts and purchases is handled through separate relations instead of repeated columns.
- Each table has a clearly defined primary key.

Conclusion:

All relations in the schema satisfy 1NF.

Second Normal Form (2NF)

A relation is in Second Normal Form (2NF) if:

- It is already in 1NF
- All non-key attributes are fully functionally dependent on the entire primary key

Analysis of composite-key tables:

provide (cust_id, number)

- Composite primary key: (cust_id, number)
- No additional non-key attributes.
- Fully dependent on the whole key.

Single-key tables:

- Tables such as customer, item, bank_account and item_class have single-attribute primary keys, so partial dependency cannot occur.

Conclusion:

All relations satisfy 2NF.

Third Normal Form (3NF)

A relation is in Third Normal Form (3NF) if:

- It is in 2NF
- There are no transitive dependencies (non-key attributes depending on other non-key attributes)

Key design decisions supporting 3NF:

- Authentication data is separated into regular_customer, preventing dependency on non-key attributes in customer.
- Item classification details are isolated in item_class, avoiding repeated category names in item.
- Bank account details are stored in bank_account instead of being duplicated per customer.
- Associative tables contain only key attributes and eliminating transitive dependencies.

Conclusion:

No non-key attribute depends on another non-key attribute.

All tables are in Third Normal Form (3NF).

Boyce–Codd Normal Form (BCNF)

A relation is in Boyce–Codd Normal Form (BCNF) if for every non-trivial functional dependency

X → Y, X is a superkey of the relation.

BCNF is a stricter version of 3NF and is used to eliminate remaining anomalies that may still exist even when a schema is in 3NF.

BCNF Analysis of the Schema

Each relation in the database was examined by identifying its functional dependencies and verifying whether the determinant of each dependency is a superkey.

customer

- Primary key: cust_id
- Functional dependencies:
- cust_id → cust_name, cust_address, cust_email
- cust_id is a superkey.

customer is in BCNF

regular_customer

- Primary key: cust_id
- Functional dependencies:
- cust_id → user_id, password
- user_id is defined as UNIQUE but it does not determine other attributes.
- All dependencies are determined by a superkey.

regular_customer is in BCNF

item

- Primary key: stock
- Functional dependencies:
- stock → quantity, price
- No attribute other than the primary key determines non-key attributes.

item is in BCNF

item_class

- Primary key: item_class_number
- Determinant is a superkey.

item_class is in BCNF

bank_account

- Primary key: number

- Functional dependencies:
- number → name
- No additional dependencies exist.

bank_account is in BCNF

provide

- Composite primary key: (cust_id, number)
- No dependency violates BCNF.

provide is in BCNF

Implementation (SQL)

The database was implemented using SQL by explicitly defining data types, constraints and relationships to enforce the design rules at the database level. Tables were created using CREATE TABLE statements where primary keys were defined to ensure uniqueness and fast access.

INT data types were used for identifiers such as cust_id and stock to support indexing and efficient joins, while AUTO_INCREMENT was applied to customer identifiers to automate key generation. Textual attributes were implemented using VARCHAR with bounded lengths (30, 50 and 100) to prevent unnecessary storage usage while allowing sufficient flexibility for real-world data.

Relationships were enforced using FOREIGN KEY constraints rather than relying on application logic. One-to-many relationships were implemented by placing foreign keys in dependent tables, whereas many-to-many relationships were implemented through junction tables with composite primary keys. This prevents duplicate relationship entries at the SQL level.

Constraints such as NOT NULL and UNIQUE were applied where required to enforce domain rules (unique usernames). Cascading delete rules (ON DELETE CASCADE) were used to maintain referential integrity and automatically remove dependent records when a parent record is deleted.

Sample data was inserted using INSERT statements to validate constraint behavior and ensure that joins, aggregations, and nested queries could be executed correctly.

QUERIES

1)Query Type:

INNER JOIN

Purpose:This query is used to identify which customers in the system are registered users. Since not all customers are required to have login credentials joining the customer table with the reg_customer table allows the system to distinguish registered customers from non-registered ones.

Result Explanation:The result includes only customers who exist in both tables, meaning customers who have created an account. It displays the customer ID, customer name, and the associated user IDand providing a clear list of all registered users in the system.

2) Query Type:JOIN query

Purpose:This query is used to display the bank accounts associated with each customer. Since customers and bank accounts have a many-to-many relationship, the associative table provide is used to correctly link customers to their accounts.

Result Explanation:The result shows each customer's name together with the bank account number and bank name they are linked to. Ordering the output by customer name improves readability and makes it easier to analyze customer–account relationships.

3) Query Type:JOIN query (LEFT JOIN)

Purpose:This query is used to determine how many bank accounts are associated with each customer. A left join is applied to ensure that customers without any linked bank accounts are also included in the result.

Result Explanation:The result shows each customer together with the total number of bank accounts they have. Customers with no associated accounts appear with a count of zero providing a complete overview of account distribution across all customers.

4) Query Type:JOIN query (LEFT JOIN)

Purpose:This query is used to display all items together with the class they belong to. A left join is used to ensure that items are still shown even if their class information is missing.

Result Explanation:The result lists each item's stock number, available quantity, price and the corresponding item class name. This provides an overview of the inventory and its categorization within the system.

5)Query Type:Nested query (EXISTS subquery)

Purpose:This query is used to identify customers who have made at least one purchase. The EXISTS subquery checks whether a corresponding record for the customer exists in the purchase table.

Result Explanation:The result returns only customers for whom at least one matching purchase record is found. It lists the customer ID and name effectively filtering out customers who have never made a purchase.

6) Query Type:JOIN query (INNER JOIN + LEFT JOIN)

Purpose:This query is used to display detailed information about purchases made by customers. By joining purchase, customer, item and item class tables it combines transactional and descriptive data into a single result.

Result Explanation:The result shows each customer together with the items they purchased including the item stock number, price and item class. Ordering the output by customer name improves readability and supports customer based analysis of purchases.

7)Query Type:Nested query (subquery with aggregate function)

Purpose:This query is used to identify the item or items with the highest price in the inventory. A subquery is applied to compute the maximum price across all items.

Result Explanation:The result returns the stock number and price of the most expensive item. If multiple items share the same maximum price all of them are included providing a complete view of top-priced products.

8) Query Type:Nested query (NOT IN subquery)

Purpose:This query is used to identify customers who have never made a purchase.

Result Explanation:The result lists customers whose IDs do not appear in the purchase table meaning they have not bought any items.

9) Query Type:Nested query (subquery with aggregate function)

Purpose:This query is used to find items that are more expensive than the average item price.

Result Explanation:The result shows items whose price is higher than the overall average price, allowing analysis of premium or high-priced products.

10) Query Type:JOIN query (LEFT JOIN)

Purpose:This query is used to determine how many items each customer has purchased. A left join is applied to include all customers even those who have not made any purchases.

Result Explanation:The result shows each customer together with the total number of items they have bought. Customers with no purchases appear with a count of zero providing a complete overview of purchasing activity across all customers.

11) Query Type:Nested query (multiple nested subqueries)

Purpose:This query is used to identify customers who purchased items belonging to a specific item class.

Result Explanation:The result returns customers who bought at least one item from the selected item class, demonstrating how nested subqueries can filter data across multiple relations.

12) Query Type:

JOIN query (INNER JOIN with aggregation)

Purpose:This query is used to analyze inventory levels at the category level. By grouping items by their class and summing their quantities that it provides an overview of how stock is distributed across item classes.

Result Explanation:The result shows each item class together with the total quantity of items available in that class. This helps evaluate which categories have higher or lower stock levels and supports inventory management decisions.

Conclusion

In this project, a complete relational database for an online retail system was designed and implemented using SQL. The work followed a structured database development process, starting from conceptual modeling with an EER diagram and continuing through relational schema design normalization and SQL implementation.

The database design successfully captures real-world relationships between customers, registered users, items, item classes, bank accounts and provide. By applying normalization principles up to BCNF, the schema avoids redundancy and ensures data consistency. Referential integrity is enforced through

primary and foreign key constraints while many-to-many relationships are correctly handled using associative tables.

A set of SQL queries, including join-based and nested queries were developed to demonstrate the practical usability of the database. These queries show how meaningful information such as registered customers, purchase behavior, inventory levels and category-based analysis can be extracted efficiently.

Overall, the project demonstrates a clear understanding of database modeling principles and their implementation in SQL. The final database is consistent, extensible and suitable as a foundation for a real-world online retail application.