# UNIVERSITY OF VISVESVARAYA COLLEGE OF ENGINEERING

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

### K.R. Circle, Bengaluru – 560001

A DBMS Mini-Project Report on

# FARMCONNECT: AN AGRICULTURAL SUPPLY CHAIN MANAGEMENT SYSTEM

Submitted by

## Sanjana C K (U25UV22T064049)

## Y Lohith (U25UV22T064065)

### V SEM, B. TECH (ISE)

Under the guidance of

**Dr. Kumaraswamy**

**Associate Professor**

**Department of Computer Science and Engineering**

**UVCE**

March-2025

# UNIVERSITY OF VISVESVARAYA COLLEGE ENGINEERING

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**K.R. Circle, Bengaluru – 560001**

# CERTIFICATE

This is to certify that **Sanjana C K** and **Y Lohith** of V Semester, B. Tech, Information Science and Engineering, bearing the register number **U25UV22T064049** and **U25UV22T064065** respectively has submitted the DBMS Mini-Project Report on **“FARMCONNECT: AN AGRICULTURAL SUPPLY CHAIN MANAGEMENT SYSTEM”**, in partial fulfilment for the DBMS Lab, prescribed by the University of Visvesvaraya College of Engineering for the academic year 2024-25.

|  |  |
| --- | --- |
| **Dr. Kumaraswamy** | **Dr. Thriveni J** |
| Associate Professor | Professor & Chairperson |
| Dept. of CSE, | Dept. of CSE, |
| UVCE | UVCE |

### Examiners:

1. ………………………… 2. ...……………………….

# ACKNOWLEDGEMENT

I take this opportunity to thank our institution **University of Visvesvaraya College of Engineering** for having given me an opportunity to carry out this project.

I would like to thank **Prof. Subhasish Tripathy, Director, UVCE**, for providing us all the facilities to work on this project. I am indebted to him for being my pillar of strength and inspiration.

I wish to place my grateful thanks to **Dr. Thriveni J, Professor and Chairperson, Department of Computer Science and Engineering, UVCE,** who helped me to make my project a great success.

It gives me great pleasure to express my gratitude to **Dr. Kumaraswamy, Associate Professor, Department of Computer Science Engineering, UVCE** for her valuable guidance and supervision in this course of project.

I express my sincere thanks to all teaching and non-teaching staff, Department of Computer Science and Engineering, UVCE for all the facilities that they have provided me for successfully completing this project.

I also thank my parents and friends for their continuous support and encouragement.

**Sanjana C K** and **Y Lohith (U25UV22T006003** and **U25UV22T006021)**

# ABSTRACT

FarmConnect is a web-based Agricultural Supply Chain Management System developed to bridge the gap between farmers and buyers in the agricultural sector. The system streamlines the process of crop listing, order management, and supply chain tracking, ensuring efficient delivery of agricultural products from farm to market.

The project utilizes modern web technologies including PHP for the backend, MySQL for database management, and HTML/CSS/JavaScript for the frontend. It implements a comprehensive supply chain tracking system that monitors order status from pending to delivery, manages farmer earnings, and handles buyer payments securely.

Key features include real-time order status updates, automated earnings calculations, inventory management, and detailed transaction history tracking. The system emphasizes data security, user experience, and efficient supply chain operations, making it a valuable tool for both farmers and buyers in the agricultural sector.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **Title** | **Pg no.** |
| **1. Introduction** |  |
| 1.1 Introduction | 1 |
| 1.2 Objective | 1 |
| 1.3 Functionality | 2 |
| 1.4 Database Management System | 3 |
| 1.5 MySQL | 4 |
| **2. Literature Review** |  |
| 2.1 Survey of Existing System | 8 |
| 2.2 Developed System | 8 |
| 2.3 Software Requirements | 9 |
| 2.4 Frontend Technologies | 9 |
| 2.5 Backend Technologies | 10 |
| 2.6 Entity Relationship Model | 11 |
| 2.7 Relational Schema Design | 13 |
| 2.8 Normalization | 16 |
| 2.9 API Development | 18 |
| 2.10 Advantages of the Developed System | 20 |
| **3. Proposed Work** |  |
| 3.1 Entity-Relationship (ER) Model | 21 |
| 3.2 Relational Model | 23 |
| 3.3 Normalization | 25 |
| **4. Results** |  |
| 4.1 Screenshots | 28 |
| 4.1.1 Welcome Page | 29 |
| 4.1.2 Login/Signup Page | 29 |
| 4.1.3 Customer Dashboard | 30 |
| 4.1.4 Transactions Page | 30 |
| 4.1.5 Loan Page | 31 |
| 4.1.6 Employee Dashboard | 31 |
| **5. Conclusion** | 33 |
| **6. Bibliography** | 34 |

# CHAPTER 1 INTRODUCTION

## Introduction

The **Bank Database Management System (BDMS)** is a digital platform designed to manage various day-to-day banking activities in an organized and efficient manner. It provides functionalities such as handling customer data, managing account information, processing transactions, and tracking loans. The system aims to replace traditional manual record-keeping methods with a centralized database, ensuring data consistency, security, and easy accessibility.

BDMS is developed as a web-based application using **ReactJS** for the frontend, **Node.js with Express** for the backend, and **MySQL** as the database. It supports multiple user roles, including customers, employees, and administrators, each with specific access rights to perform their respective operations. By automating routine banking processes, the system improves overall operational efficiency, reduces errors, and enhances user experience for all stakeholders involved.

## Objective

* To create a centralized system for managing customer, account, and transaction data efficiently.
* To automate core banking processes like account creation, fund transfers, and loan management.
* To reduce manual errors and improve operational accuracy.
* To implement role-based access for enhanced security and data control.
* To generate reports and summaries for financial tracking and analysis.
* To ensure data consistency and integrity using a well-structured relational database.

### User Roles in the System

* + **Employee**: Manage customers, accounts, and loans
  + **Customer**: View account status, perform transactions, request loans

### Views Available in the System

The user interface provides the following **features and functionalities**:

* + **Welcome Page** – Displays an introduction to the Banking System.
  + **Login Page** – Secure login for Customer and Employee.
  + **Customer Dashboard** – Manages all accounts, loans and transactions.
  + **Employee Dashboard –** Includes all the customer details and corresponding banking activities.
  + **Accounts** – Consisting of accounts like savings, checking, recurring.
  + **Loans** – Comprises of all the availed loans and their payments
  + **Transactions** – Listing all the credits and debits from one account to another

## Functionality

The **Bank Database Management System** aims to improve the efficiency of the user interactions with the bank and making it easier to operate through the following **functional modules**:

### Account Management

* Allows customers to open, modify, or close bank accounts (savings, checking, recurring).
* Displays account balances, details, and interest rates.
* Manages account linking, nominees, and status updates.

### Loan Management

* Enables customers to apply for personal, home, or auto loans.
* Tracks loan approval, disbursement, and EMI payments.
* Allows employees to review, approve, or reject loan requests.
* Displays active, pending, and closed loans with schedules.

### Transaction Management

* Processes fund transfers between accounts (intra-bank and inter-bank).
* Lists all credits and debits with filters for date, amount, and type.
* Generates downloadable account statements.
* Monitors transactions for fraud or unusual activity.

### Customer Management

* + Maintains customer profiles with personal and banking details.
  + Allows registration, login, and account recovery features.
  + Tracks customer activities and service requests.
  + Supports KYC verification and contact updates.

### Employee Management

* + Grants employees access to customer data and banking operations.
  + Enables management of loans, accounts, and approvals.
  + Tracks employee actions and permissions.
  + Supports role-based access control.

### Authentication & Security

* + Provides secure login for both customers and employees.
  + Implements password encryption and session management.
  + Includes multi-factor authentication (optional).
  + Logs all login attempts and unusual access patterns.

### Report Generation

* + Creates detailed reports on transactions, loans, accounts, and customer activity.
  + Exports data in PDF or Excel format.
  + Generates summaries for auditing and compliance.
  + Tracks outstanding dues and payments.

## Database Management System

A **Database Management System (DBMS)** is essential for managing large datasets efficiently. Our project uses **MySQL** to ensure **data consistency, security, and easy retrieval**.

### Why Use a DBMS?

* **Eliminates data redundancy** – Avoids duplicate records.
* **Ensures data integrity** – Prevents inconsistencies.
* **Provides structured queries** – Allows fast data retrieval.

### 1.41. Characteristics of DBMS

* **Self-describing nature** – Metadata is stored within the database.
* **Data redundancy control** – Eliminates duplicate records.
* **Enforces data integrity rules** – Ensures that relationships between tables remain valid.
* **Supports multiple views** – Different users see only relevant data.
* **Handles multiple transactions** – Allows multiple operations without conflicts.

### 1.4.2. Advantages of DBMS

* **Data Security:** Ensures access control and prevents unauthorized access.
* **Data Consistency:** Provides reliable data storage and retrieval.
* **Efficient Query Processing:** Allows advanced queries to fetch related data.
* **Scalability:** Can be extended as the coaching center grows.

## MySQL

### Overview

**MySQL** is a **relational database management system (RDBMS)** used for structured data storage. It is widely adopted due to its **performance, security, and scalability**.

### Why MySQL for ABCC-DBMS?

* + - * **Efficient Data Management** – Handles structured records efficiently.
      * **Query Optimization** – Supports complex queries with high-speed performance.
      * **Secure Transactions** – Ensures data integrity with ACID compliance.
      * **Scalability** – Supports increasing data and users.

### SQL Commands Used in ABCC-DBMS

**Creating the Student table**

CREATE TABLE student (

Student\_ID INT PRIMARY KEY AUTO\_INCREMENT, Name VARCHAR(100), Age INT, Batch\_ID INT, Contact VARCHAR(15), FOREIGN KEY (Batch\_ID) REFERENCES batch(Batch\_ID));

### Inserting a New Student Record

INSERT INTO student (Name, Age, Batch\_ID, Contact) VALUES ('John Doe', 18, 3, '9876543210');

### Retrieving Student Details

SELECT \* FROM student WHERE Batch\_ID = 3;

### Updating Student Information

UPDATE student SET Age = 19 WHERE Student\_ID = 1;

### Deleting a Student Record

DELETE FROM student WHERE Student\_ID = 5;

### SQL Statements

SQL provides essential commands to manage and manipulate relational databases. The key statements include:

1. **SELECT** – Retrieves information from a table.

SELECT \* FROM student WHERE Batch\_ID = 3;

1. **INSERT** – Adds new records to a table.

INSERT INTO student (Name, Age, Batch\_ID, Contact) VALUES ('John Doe', 18, 3, '9876543210');

1. **DELETE** – Removes one or more existing records from a table. DELETE FROM student WHERE Student\_ID = 5;
2. **UPDATE** – Modifies existing values in a table.

UPDATE student SET Age = 19 WHERE Student\_ID = 1;

### SQL Aggregate Functions

Aggregate functions perform calculations on a set of values and return a single result.

1. **COUNT** – Returns the number of records.

SELECT COUNT(\*) FROM student;

1. **SUM** – Computes the total sum of values in a column.

SELECT SUM(Amount) FROM payment;

1. **MAX** – Returns the maximum value in a column. SELECT MAX(Age) FROM student;
2. **MIN** – Returns the minimum value in a column. SELECT MIN(Age) FROM student;
3. **AVG** – Calculates the average value in a column. SELECT AVG(Age) FROM student;

### SQL Constraints

Constraints ensure data integrity and enforce rules in a database.

1. **NOT NULL** – Ensures a column cannot have NULL values. CREATE TABLE student ( Student\_ID INT PRIMARY KEY AUTO\_INCREMENT, Name VARCHAR(100) NOT NULL );
2. **PRIMARY KEY** – Ensures a column has unique, non-null values. CREATE TABLE student ( Student\_ID INT PRIMARY KEY, Name VARCHAR(100) );
3. **UNIQUE** – Ensures all values in a column are **distinct** (no duplicates allowed). CREATE TABLE student ( Email VARCHAR(100) UNIQUE, Name VARCHAR(100));
4. **FOREIGN KEY -** Creates a link between two tables by referencing the primary key of another table.

CREATE TABLE enrollment ( Enrollment\_ID INT PRIMARY KEY, Student\_ID INT, FOREIGN KEY (Student\_ID) REFERENCES student(Student\_ID));

1. **CHECK -** Ensures that values in a column meet a specific condition.

CREATE TABLE student (Age INT CHECK (Age >= 18), Name VARCHAR(100));

1. **DEFAULT -** Assigns a default value to a column when no value is provided.

CREATE TABLE student (Name VARCHAR(100), Country VARCHAR(100) DEFAULT

'India');

1. **AUTO\_INCREMENT -** Automatically generates a unique number when a new record is inserted (used mainly for primary keys).

CREATE TABLE student ( Student\_ID INT AUTO\_INCREMENT PRIMARY KEY, Name VARCHAR(100)**);**

# CHAPTER 2 LITERATURE REVIEW

This chapter focuses on the already existing systems for managing sports coaching centers and establishes the software requirements for the project.

## Survey of Existing Systems

Banking operations have traditionally been managed using manual methods such as paper records, physical ledgers, and basic spreadsheet software. While these methods served the purpose in the past, they posed several limitations in terms of speed, accuracy, and scalability. Manual systems often led to data duplication, misplaced records, unauthorized access, and delayed services, especially as the volume of customer data and transactions increased over time.

Modern banks require dynamic systems that can process real-time data, secure sensitive financial information, and support multi-user access with defined roles. However, many smaller or regional banks still lack efficient and integrated database management systems tailored to their specific needs. This gap results in fragmented data handling and inefficient service delivery.

### Disadvantages of Manual Banking Systems:

* + - High chances of human error and manual miscalculations
    - Time-consuming data entry and retrieval processes
    - Poor security and access control for sensitive financial data
    - Repetition of data leading to inconsistencies
    - Lack of real-time tracking of transactions and loan activities
    - Difficulty in generating accurate financial summaries and audit reports

The proposed **Bank Database Management System** aims to overcome these limitations by offering a centralized, automated, and secure solution to manage all essential banking operations.

## Developed System

In this project, we are using **React.js** for the front-end, **Node.js with Express** for the back-end, and **MySQL** as the database management system. React.js is a powerful JavaScript library that allows for the creation of dynamic and responsive user interfaces, enhancing user interaction and experience. Node.js, along with Express, is a lightweight and efficient backend framework that supports asynchronous operations and handles API requests effectively. MySQL is a popular open-source relational database

system known for its speed, scalability, and strong data integrity features.

The developed system efficiently manages core banking activities by centralizing operations such as account creation, transaction processing, loan management, customer and employee data handling, and report generation. It also implements role-based access, allowing customers, employees, and administrators to perform specific tasks based on their roles. By replacing traditional manual workflows with an integrated, real-time digital platform, the system improves operational efficiency, reduces human errors, and enhances the overall security and accessibility of banking services.

## Software Requirements

**Operating System:** Windows / Linux / MacOS **Frontend:** React.js, HTML, CSS, JavaScript **Backend:** Node.js with Express (JavaScript) **Database Management System:** MySQL

**Web Server:** Express.js development server (Node-based)

## Frontend Technologies

The frontend acts as the primary interface through which users interact with the banking system. It is responsible for ensuring a smooth, responsive, and intuitive user experience across different devices and browsers. The system’s frontend is built using a combination of modern web technologies.

### HTML

Hypertext Markup Language (HTML) is used to structure the content displayed on the web pages. It defines the layout of elements like headings, forms, tables, buttons, and inputs that are crucial for user interaction in a banking environment.

### CSS

Cascading Style Sheets (CSS) are used to style and visually organize the HTML elements. It enhances the user experience by controlling the look and feel of the web application, including fonts, colors, spacing, and responsive layout design.

### JavaScript

JavaScript adds interactivity to the web application. In this system, it enables real-time updates, form validation, input checking, and dynamic rendering of components based on user roles such as customer or employee.

### React.js

React.js is a modern JavaScript library used to build the dynamic user interface of the BDMS. It follows a component-based architecture, allowing for reusable and scalable UI components. React’s virtual DOM ensures efficient updates and smooth navigation within the banking dashboard.

## Backend Technologies

The backend of the Bank Database Management System handles the core logic of the application, including processing client requests, managing authentication, performing database operations, and sending appropriate responses to the frontend. It ensures that all transactions and operations are securely and accurately executed.

### Node.js with Express Framework.

Node.js is a JavaScript runtime built on Chrome's V8 engine that allows server-side execution of JavaScript code. Express is a minimal and flexible Node.js web application framework that simplifies the development of backend logic and RESTful APIs. It follows the MVC architecture and provides efficient routing and middleware support, making it ideal for scalable and high-performance applications.

Node.js with Express provides:

* + A fast and lightweight runtime environment
  + Routing and middleware for managing API endpoints
  + Easy integration with MySQL for data access and manipulation
  + Support for JSON-based communication with the frontend
  + Secure handling of user authentication and sessions

### MySQL

MySQL is an open-source relational database management system used to store structured banking data. It organizes data in the form of tables and supports complex queries for efficient data retrieval and processing. MySQL is reliable, scalable, and supports transactions, which is crucial for handling sensitive financial operations.

MySQL offers:

* + Structured storage of account, customer, loan, and transaction data
  + Foreign key constraints to ensure referential integrity
  + Secure login and access control features
  + Optimized performance for concurrent database operations
  + Scalability to accommodate growing banking activities

## Entity-Relationship (ER) Model

The Entity-Relationship (ER) model is a high-level conceptual data model that provides a systematic approach to database design by defining the data elements, their attributes, and the interrelationships between them. It serves as a blueprint for constructing relational databases and is particularly useful in capturing the requirements of complex systems such as a banking environment.

Purpose of the ER Model

The main goal of the ER model is to:

* + - Represent the data structure in a way that is easily understood by both technical and non-technical stakeholders.
    - Identify the entities that need to be represented in the database.
    - Define the relationships between these entities.
    - Specify the attributes of each entity and relationship.
    - Establish integrity constraints such as keys and cardinalities. Core Components of the ER Model

The ER model consists of three fundamental components: entities, attributes, and relationships.

### Entities

Entities are objects or concepts that are distinguishable from other objects. Each entity represents a set of real-world objects that share the same properties. In the banking domain, typical entities might include Customer, Account, Branch, Employee, and Loan.

* + Strong Entity: Has a primary key that uniquely identifies each instance (e.g., Customer, Account).
  + Weak Entity: Cannot be uniquely identified without a related strong entity (e.g., Payment may depend on Loan).

Entities are represented as rectangles in an ER diagram.

### Attributes

Attributes are the properties or characteristics of an entity or relationship. For example, a Customer

entity may have attributes such as Customer\_ID, First\_Name, Last\_Name, Address, and Phone\_Number.

### Types of attributes include:

* + Simple (Atomic): Cannot be divided further (e.g., Customer\_ID).
  + Composite: Can be broken down into smaller sub-parts (e.g., Name → First\_Name and Last\_Name).
  + Derived: Can be calculated from other attributes (e.g., Age from Date\_of\_Birth).
  + Multivalued: May have more than one value for an entity instance (e.g., multiple phone numbers). Attributes are typically represented as ellipses in an ER diagram, connected to their respective entity or relationship.

### Relationships

Relationships describe how two or more entities are associated with each other. In the banking domain, examples of relationships include:

* + A Customer owns one or more Accounts.
  + An Employee works at a Branch.
  + A Loan is repaid by Payments.
  + A Branch belongs to a Bank.

Relationships are depicted as diamonds in an ER diagram, connected to the participating entities. Each relationship can have:

* + **Cardinality constraints**: Define the number of instances of one entity that can or must be associated with instances of another entity.
    - One-to-One (1:1): Each entity instance in the relationship will have exactly one related instance.
    - One-to-Many (1:N): One instance of an entity is associated with multiple instances of another.
    - Many-to-Many (M:N): Multiple instances of one entity are related to multiple instances of another.
  + **Participation constraints**: Specify whether all instances of an entity must participate in the relationship.
    - Total participation: Every instance of the entity must be involved in the relationship.
    - Partial participation: Some instances may not participate.

### Keys and Integrity Constraints

* + Primary Key (PK): An attribute or set of attributes that uniquely identifies an entity instance.
  + Foreign Key (FK): An attribute that creates a link between entities by referring to the primary key of another entity.
  + Referential Integrity: Ensures that relationships between tables remain consistent (e.g., a foreign key must match an existing primary key or be null).

### Advanced Concepts in the ER Model

* + Generalization: The process of extracting shared characteristics from two or more entities and creating a generalized entity (e.g., combining Manager and Clerk into Employee).
  + Specialization: The opposite of generalization, where a higher-level entity is divided into sub- entities based on some distinguishing characteristics.
  + Aggregation: A higher-level abstraction that treats a relationship set as an entity set for the purpose of participating in other relationships.
  + Weak Entity and Identifying Relationships: Used when an entity cannot be uniquely identified by its own attributes and relies on a related entity and a partial key.

### Benefits of Using an ER Model in Banking Systems

* + Facilitates clear understanding of the system’s data structure.
  + Ensures logical consistency and accuracy of data.
  + Helps in identifying redundant data and eliminating anomalies through normalization.
  + Acts as a reference model during database implementation.
  + Supports better maintenance, scalability, and security by providing a well-organized schema.

## Relational Schema Design

The **Relational Schema** is a representation of the database structure in terms of **relations (tables)**, where data is organized in rows and columns. It is derived from the conceptual design represented in the ER model and forms the **logical structure** that is eventually implemented in a relational database system.

Each entity and relationship in the ER model is translated into one or more relations, and constraints such as primary keys, foreign keys, and cardinality are enforced through appropriate schema definitions. The goal is to create a **normalized, non-redundant**, and **integrity-preserving** relational schema that accurately models the domain.

## 7-Step Approach to Convert ER Model to Relational Schema

Converting an ER model to a relational schema is a systematic process. The following seven steps outline a structured approach for this transformation:

### Step 1: Mapping Regular (Strong) Entity Sets

Each strong entity in the ER model becomes a separate relation (table).

* All **simple** and **composite** attributes are included as columns.
* A **primary key** is chosen to uniquely identify each tuple.
* **Composite attributes** are flattened into individual attributes.

### Example:

Entity: Customer (Customer\_ID, Name, Phone, Address) Relational Schema:

Customer(Customer\_ID PRIMARY KEY, First\_Name, Last\_Name, Phone, Address)

### Step 2: Mapping Weak Entity Sets

Weak entities do not have a primary key of their own and depend on a strong entity for identification.

* Create a relation for the weak entity.
* Include the **partial key** and **foreign key** referencing the strong entity's primary key.
* The **combination** of the foreign key and partial key becomes the **composite primary key**. **Example:**

Entity: Payment (Pay\_ID) dependent on Loan Relational Schema:

Payment(Loan\_ID, Pay\_ID, Pay\_Amount, Pay\_Date, PRIMARY KEY(Loan\_ID, Pay\_ID), FOREIGN KEY(Loan\_ID) REFERENCES Loan)

### Step 3: Mapping Binary 1:1 Relationships

For one-to-one relationships, the foreign key can be placed in either of the participating entities’ tables, preferably the one with **total participation**.

* Ensure **unique constraint** on the foreign key.
* If participation is total on one side, foreign key goes to that side.

### Example:

Employee assigned to Branch (1:1)

Employee(Emp\_ID PRIMARY KEY, ..., Branch\_ID UNIQUE, FOREIGN KEY(Branch\_ID) REFERENCES Branch)

### Step 4: Mapping Binary 1:N Relationships

For one-to-many relationships, the foreign key is placed in the relation on the **‘many’ side** of the relationship.

* The foreign key references the primary key of the ‘one’ side.
* Ensures referential integrity.

### Example:

Branch (1) — (N) Employee

Employee(Emp\_ID PRIMARY KEY, ..., Branch\_ID, FOREIGN KEY(Branch\_ID) REFERENCES Branch)

### Step 5: Mapping Binary M:N Relationships

Many-to-many relationships require a **separate relation**.

* Create a new relation to represent the relationship.
* Include **foreign keys** referencing the primary keys of both participating entities.
* The **combination** of both foreign keys becomes the **primary key** of the new relation.
* Include any **attributes** of the relationship as additional columns.

### Example:

Customer (M:N) Account

Customer\_Account(Customer\_ID, Acc\_No, Role, PRIMARY KEY(Customer\_ID, Acc\_No), FOREIGN KEY(Customer\_ID) REFERENCES Customer, FOREIGN KEY(Acc\_No) REFERENCES Account)

### Step 6: Mapping Multivalued Attributes

Multivalued attributes are represented by creating a **separate relation**.

* The new relation includes the multivalued attribute and the primary key of the entity it belongs to.
* The combination of these becomes the **composite primary key**. **Example:**

Customer with multivalued attribute Phone\_No

Customer\_Phone(Customer\_ID, Phone\_No, PRIMARY KEY(Customer\_ID, Phone\_No), FOREIGN

KEY(Customer\_ID) REFERENCES Customer)

### Step 7: Mapping Ternary (and Higher) Relationships

For relationships involving **three or more entities**, create a new relation.

* Include the primary keys of all participating entities as **foreign keys**.
* These foreign keys together form the **composite primary key**.
* Add any **attributes** of the relationship to the relation.

### Example:

Relationship: Customer applies for Loan at Branch

Loan\_Application(Customer\_ID, Loan\_ID, Branch\_ID, Date\_Applied, PRIMARY KEY(Customer\_ID, Loan\_ID, Branch\_ID), FOREIGN KEYs to all 3 entities)

This 7-step method ensures a systematic and consistent transformation of the ER model into a fully functional relational schema. It preserves:

* **Data integrity** through keys and constraints,
* **Logical consistency** by maintaining relationships, and
* **Efficiency** through normalization and redundancy elimination.

The resulting relational schema becomes the foundation for physical database implementation in systems such as MySQL, PostgreSQL, or Oracle, allowing robust handling of banking operations like account management, transactions, loan tracking, and customer service.

## Normalization

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update, and Deletion Anomalies. Normalization rules divide larger tables into smaller tables and link them using relationships.

Normalization of data can be looked upon as a process of analyzing the given relation schemas based on their Functional Dependencies and primary keys to achieve the desirable properties of:

* + - Minimizing redundancy.
    - Minimizing the insertion, deletion, and update anomalies.

### Functional Dependency

The Functional Dependency, denoted by **X → Y**, between two sets of attributes X and Y that are subsets

of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that for any two tuples **t1** and **t2** in r that have **t1[X] = t2[X]**, the values of the Y component of a tuple in r depend on or are determined by the values of the X components. Alternatively, the values of the X component of a tuple uniquely determine the values of the Y component.

Here is a list of Normal Forms in SQL:

* + - 1NF (First Normal Form)
    - 2NF (Second Normal Form)
    - 3NF (Third Normal Form)
    - BCNF (Boyce-Codd Normal Form)
    - 4NF (Fourth Normal Form)
    - 5NF (Fifth Normal Form)

### First Normal Form (1NF)

It states that the domain of an attribute must include only atomic values and that the value of any attribute in a tuple must be a single value from the domain of the attribute. Hence, **1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple**.

### Second Normal Form (2NF)

This normal form is based on **full functional dependency**. A functional dependency **X → Y** is fully functional if the removal of any attribute **A** from **X** means that the dependency does not hold anymore. A relation schema **R** is in **2NF** if every non-prime attribute **A** in **R** is fully functionally dependent on the primary key of **R**.

### Third Normal Form (3NF)

**Third Normal Form (3NF)** is based on the concept of **transitive dependency**. A functional dependency **X → Y** in a relation schema **R** is a transitive dependency if there is a set of attributes **Z** that is neither a candidate key nor a subset of any key of **R**, and both **X → Z** and **Z → Y** hold.

A relation schema **R** is in **3NF** if it satisfies **2NF** and no non-prime attribute of **R** is transitively dependent on the primary key.

### Boyce-Codd Normal Form (BCNF)

A relation schema R is in BCNF if, for every one of its functional dependencies X → Y, X is a super key. BCNF is a stricter version of 3NF where there should not be any partial or transitive dependencies.

### Fourth Normal Form (4NF)

A relation is in 4NF if it is in BCNF and does not have multi-valued dependencies. Multi-valued dependencies occur when one attribute in a table uniquely determines another attribute, independent of other attributes.

### Fifth Normal Form (5NF)

A relation is in 5NF if it is in 4NF and does not have join dependencies that are not implied by the candidate keys. This form deals with eliminating redundancy caused by overcomplicated relationships within the data.

## API Development

To enable seamless communication between the frontend and backend, the Bank Database Management System uses a set of **RESTful APIs** developed using **Express.js**. These APIs act as the bridge between the user interface (built using React.js) and the backend database (MySQL), allowing real-time data exchange for various banking operations.

The backend server handles CRUD (Create, Read, Update, Delete) operations and ensures secure access to banking data through authentication and role-based authorization. JSON is used as the data format for all requests and responses, enabling fast and structured data transfer between the client and server.

### Sample API Endpoints

1. **Get All Customer: GET /api/customers, Returns a list of all customers in the system.**

app.get('/api/customers', (req, res) => {

db.query("SELECT \* FROM CUSTOMER", (err, result) => { if (err) throw err;

res.json(result);

});

});

### Create New Account: POST /api/accounts, Creates a new bank account and links it to a

**customer.**

app.post('/api/accounts', (req, res) => {

const { acc\_number, balance, acc\_type } = req.body;

const sql = "INSERT INTO ACCOUNT (Acc\_Number, Balance, Acc\_Type) VALUES (?, ?,

?)";

db.query(sql, [acc\_number, balance, acc\_type], (err, result) => { if (err) throw err;

res.send("Account created successfully");

});

});

### Loan Payment Entry: POST /api/payments, Records a new loan payment for a specific loan.

app.post('/api/payments', (req, res) => {

const { pay\_id, pay\_amount, pay\_date, loan\_id } = req.body;

const sql = "INSERT INTO PAYMENT (Pay\_ID, Pay\_Amount, Pay\_Date, Loan\_ID) VALUES (?, ?, ?, ?)";

db.query(sql, [pay\_id, pay\_amount, pay\_date, loan\_id], (err, result) => { if (err) throw err;

res.send("Payment recorded successfully");

});

});

1. **Get Account Details by Customer ID: GET /api/customer/:id/accounts Fetches all accounts associated with a specific customer.** app.get('/api/customer/:id/accounts', (req, res) => {

const customerId = req.params.id; const sql = `

SELECT ACCOUNT.Acc\_Number, ACCOUNT.Balance, ACCOUNT.Acc\_Type FROM ACCOUNT

JOIN CUSTOMER\_ACCOUNT ON ACCOUNT.Acc\_Number =

CUSTOMER\_ACCOUNT.Acc\_Number

WHERE CUSTOMER\_ACCOUNT.Customer\_ID = ?`;

db.query(sql, [customerId], (err, result) => { if (err) throw err;

res.json(result);

});

});

## Advantages of the Developed System

The developed Bank Database Management System (BDMS) offers a wide range of advantages over traditional, manual banking processes. By utilizing modern web technologies and a structured relational database, the system ensures improved performance, security, and user experience.

Below are some key benefits of the developed system:

* + - Automation of Core Operations

The system automates major banking tasks such as account management, transaction processing, and loan handling, significantly reducing manual effort.

* + - Improved Accuracy and Efficiency

Database-driven operations eliminate human errors and provide fast, accurate data retrieval, especially for transaction histories and customer records.

* + - Enhanced Data Security

Role-based access control ensures that customers, employees, and administrators can only access functionalities relevant to their roles, maintaining confidentiality and data protection.

* + - Real-Time Data Access

Users can perform transactions, check balances, and manage payments in real-time, improving responsiveness and convenience.

* + - Scalable and Modular Design

The system can be easily expanded to include features such as online banking, credit score integration, or mobile banking without restructuring the entire backend.

* + - Centralized Database Management

All data is stored in a single, well-organized MySQL database, reducing redundancy and ensuring data consistency across all modules.

* + - Custom Reporting and Analytics

The system supports financial report generation for audits, customer summaries, loan statistics.

# CHAPTER 3 PROPOSED WORK

This chapter outlines the design and structure of the proposed Bank Database Management System (BDMS), detailing how the entities interact, how data is modeled, and how normalization principles are applied. The design focuses on efficiently organizing and managing various banking functions such as customer registration, account management, loan disbursal, and payment tracking.

### Entity Relationship (ER) Model

The Entity-Relationship Diagram (ERD) is a visual representation of the database structure that shows entities, their attributes, and relationships. The BDMS ER model includes core entities such as **Bank**, **Branch**, **Customer**, **Account**, **Loan**, **Employee**, **Loan Account, Transactions, Payment**. Relationships are established using foreign keys and many-to-many relationship tables.

Fig 3.1.1 shows the ER-Diagram of the Bank Database Management System, representing the entities, their attributes, and the relationships among them.

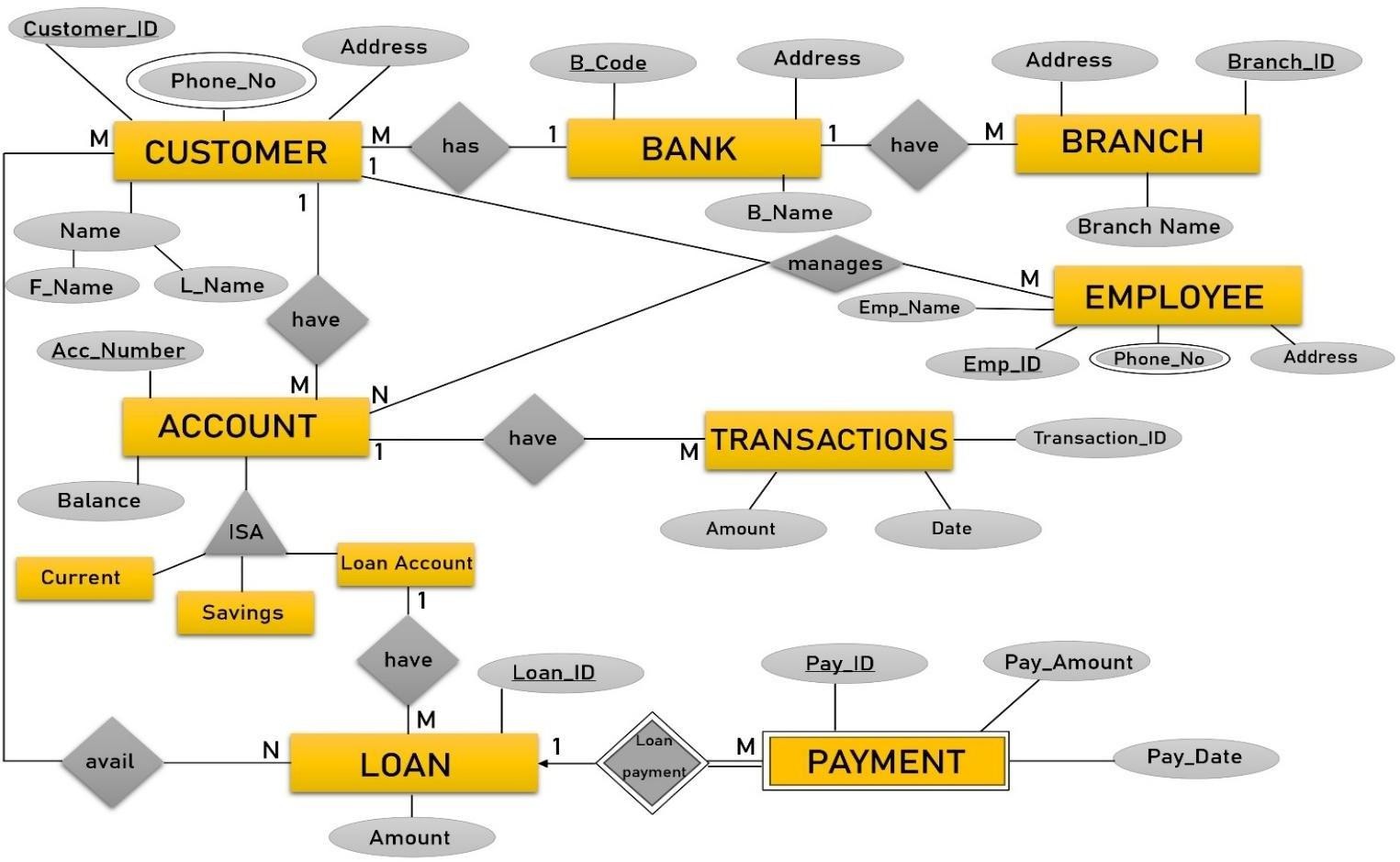


Fig 3.1.1. ER-Diagram of Bank Database Management System

### Entities and their Attributes

The proposed system consists of several entities that play a critical role in managing the core functionalities of a banking environment. These entities include banks, branches, customers, employees, accounts, loans, transactions, payments, and multiple relationship tables. Each entity is defined with specific attributes that store relevant information required for smooth and secure operations.

* + - **Bank**: Represents the central banking institution.

**Fields include**: B\_Code, B\_Name, and Address.

* + - **Branch**: Represents the different branches under the bank.

**Fields include**: Branch\_ID, Branch\_Name, Address, and B\_Code (FK).

* + - **Customer**: Represents individuals who hold accounts or loans in the bank.

**Fields include**: Customer\_ID, F\_Name, L\_Name, Phone\_no, and Address.

* + - **Employee**: Represents the staff responsible for customer and account management.

**Fields include**: Emp\_ID, F\_Name, L\_Name, Emp\_Name, Phone\_no, and Address.

* + - **Account**: Represents the various types of accounts customers hold.

**Fields include**: Acc\_No, Balance, and Acc\_Type (CHECK constraint for 'Savings' or 'Current').

* + - **Loan**: Represents the loan data issued to customers.

**Fields include**: Loan\_ID and Amount.

* + - **Loan\_Account**: Represents additional details associated with a loan, such as its type.

**Fields include**: Loan\_ID (FK) and Type.

* + - **Transactions**: Represents the transaction activity on customer accounts.

**Fields include**: Transaction\_ID, Amount, Date, and Acc\_No (FK).

* + - **Payment**: Represents the loan repayment details.

**Fields include**: Pay\_ID, Pay\_Amount, Pay\_Date, and Loan\_ID (FK).

* + - **Manages**: Relationship table connecting employees with the accounts they manage.

**Composite Key**: Emp\_ID and Acc\_No.

* + - **Avail**: Relationship table linking customers with loans they have availed.

**Composite Key**: Customer\_ID and Loan\_ID.

All these entities are stored in a MySQL database and are interlinked through primary and foreign keys. This structured relational design allows for seamless data access, real-time updates, and consistent operations while minimizing redundancy and maintaining data integrity throughout the system.

## Relational Model

The relational model defines the structure of the database in terms of **tables**, their **attributes**, and the **relationships** between them. The **Bank Database Management System (BDMS)** is built using relational tables to efficiently store, retrieve, and manage various types of banking data. These tables are normalized and interconnected using **primary keys** and **foreign keys**, ensuring data integrity and minimizing redundancy.

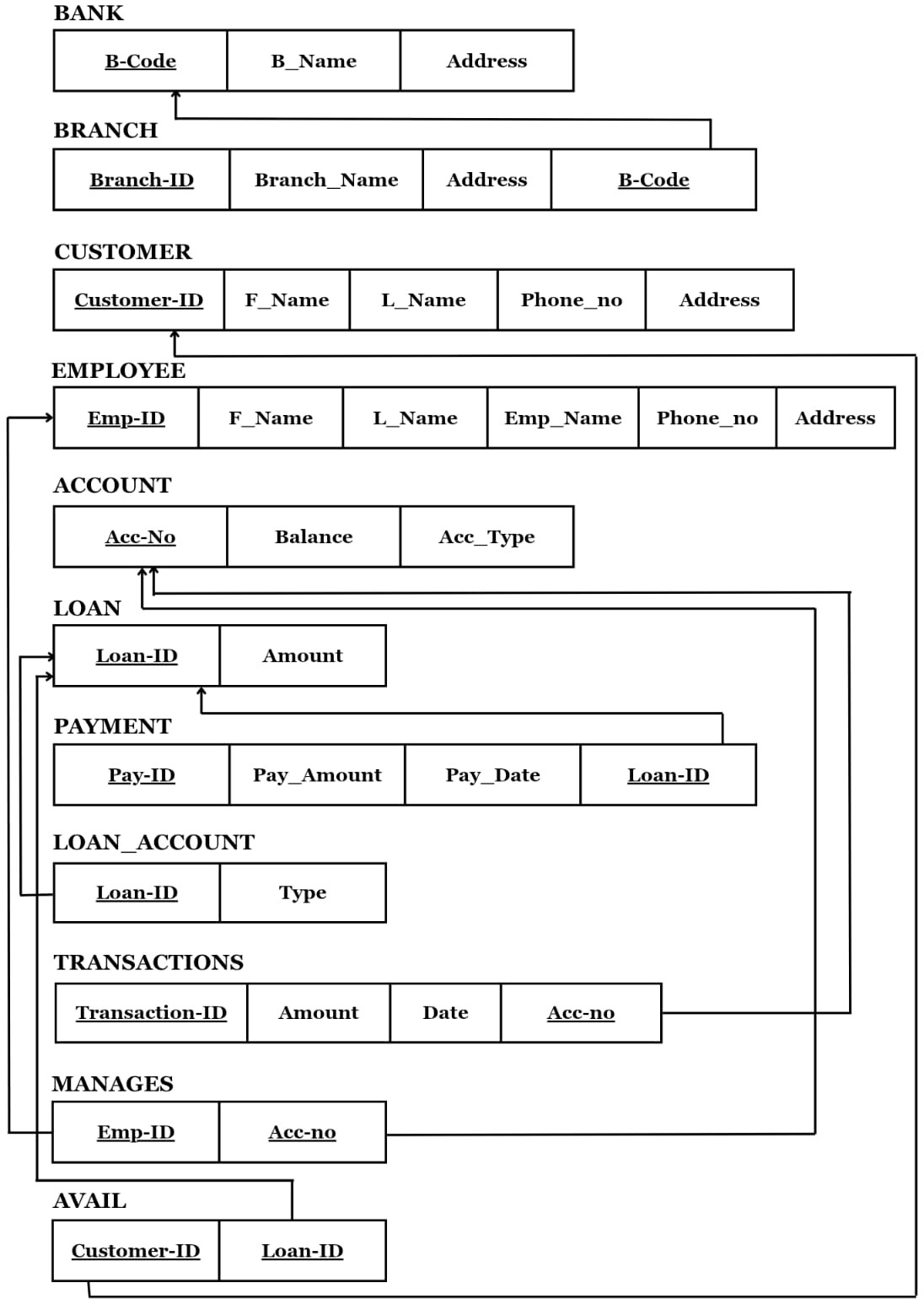


Fig 3.2.1 Shows the Relational Model of the Bank Database Management System

## Relationships and Cardinalities

### A bank has multiple branches (BANK → BRANCH) (1:M)

* + A bank can have multiple branches, but each branch belongs to only one bank.

### A customer is associated with one bank (BANK → CUSTOMER) (1:M)

* + A bank can have many customers, but each customer is associated with a single bank.

### A branch employs multiple employees (BRANCH → EMPLOYEE) (1:M)

* + A branch can have many employees, but each employee works in only one branch.

### A customer owns multiple accounts (CUSTOMER → ACCOUNT) (1:M)

* + A customer can hold multiple accounts, but each account belongs to only one customer.

### A customer can avail multiple loans (CUSTOMER → LOAN) (M:N)

* + A customer may take several loans, and a loan can be shared or co-avail by multiple customers. (Resolved using the **AVAIL** relationship table)

### An employee manages multiple accounts (EMPLOYEE → ACCOUNT) (M:N)

* + An employee can manage several accounts, and an account can be managed by multiple employees.

(Resolved using the **MANAGES** relationship table)

### An account has multiple transactions (ACCOUNT → TRANSACTIONS) (1:M)

* + An account can have many transactions, but each transaction is associated with one account.

### A loan has a loan account type (LOAN → LOAN\_ACCOUNT) (1:1)

* + Each loan has one corresponding loan account record defining the type of the loan.

### A loan has multiple payments (LOAN → PAYMENT) (1:M)

* + A loan can have multiple payments made over time, but each payment is linked to a single loan.

These relationships define the logical connections between entities in your Bank Database Management System. They form the foundation for implementing foreign key constraints and relationship tables in your MySQL schema.

## Normalization

Normalization is a key principle in database design that eliminates redundancy and ensures data consistency. By organizing data into structured tables and enforcing relationships through keys, normalization minimizes anomalies during insertions, deletions, and updates. The **Bank Database Management System (BDMS)** adheres to normalization rules up to **Fifth Normal Form (5NF)** to ensure a clean, reliable, and scalable database structure.

### First Normal Form (1NF)

A table is in 1NF if:

* + It contains only atomic (indivisible) values.
  + There are no repeating groups or multi-valued fields.
  + Each row is uniquely identifiable using a primary key. BDMS Compliance:
  + All tables in BDMS already satisfy 1NF.
  + Each field contains atomic values, and all rows are uniquely identified by primary keys.
  + No additional normalization was required at this stage.

### Second Normal Form (2NF)

A table is in 2NF if:

* + It is already in 1NF.
  + All non-key attributes are fully dependent on the entire primary key (especially in composite key tables).

BDMS Compliance:

Normalization Applied To:

* + MANAGES, AVAIL

These tables use composite primary keys. 2NF was enforced by ensuring that all non-key attributes (if any) depend on both parts of the composite key.

Normalization Not Required For:

* + CUSTOMER, EMPLOYEE, ACCOUNT, LOAN, LOAN\_ACCOUNT, TRANSACTIONS, PAYMENT, BANK, BRANCH

These tables use single-column primary keys and already satisfy 2NF by design.

### Third Normal Form (3NF)

A table is in 3NF if:

* + It is in 2NF.
  + There are no transitive dependencies (non-key attributes should not depend on other non-key attributes).

BDMS Compliance:

Normalization Applied To:

* + CUSTOMER, EMPLOYEE, BRANCH

Ensured that all fields such as Phone\_no, Address, etc., directly depend only on their primary keys (Customer\_ID, Emp\_ID, Branch\_ID) with no indirect or transitive relationships.

Normalization Not Required For:

* + ACCOUNT, LOAN, LOAN\_ACCOUNT, TRANSACTIONS, PAYMENT, BANK

These tables already had non-key attributes directly dependent on the primary key without transitive dependencies.

### Boyce-Codd Normal Form (BCNF)

A table is in BCNF if:

* + It is in 3NF.
  + For every functional dependency X → Y, X is a super key. BDMS Compliance:
  + All entity and relationship tables already satisfy BCNF.
  + No non-superkey determines another attribute in any table.
  + No additional normalization was necessary.

### Fourth Normal Form (4NF)

A table is in 4NF if:

* + It is in BCNF.
  + It contains no multi-valued dependencies. BDMS Compliance:

Normalization Applied To:

* + Multi-valued relationships such as:
* A customer having multiple accounts (via CUSTOMER → ACCOUNT)
* A customer availing multiple loans (via AVAIL)
* An account being managed by multiple employees (via MANAGES)

These were normalized into dedicated relationship tables to avoid multi-valued dependencies.

Normalization Not Required For:

* + Tables like ACCOUNT, LOAN, PAYMENT, TRANSACTIONS, BANK, and

LOAN\_ACCOUNT have no multi-valued dependencies.

### Fifth Normal Form (5NF)

A table is in 5NF if:

* + It is in 4NF.
  + All join dependencies are implied by candidate keys, and decomposition does not lead to data loss.

BDMS Compliance:

* + Join dependencies are properly handled through bridge tables like MANAGES and AVAIL.
  + No further decomposition was required.
  + The structure ensures lossless joins and data consistency.

# CHAPTER 4 RESULT

The Bank Database Management System (BDMS) was successfully implemented and tested using sample banking data. The system demonstrated its ability to manage customer records, account creation, employee assignment, loan processing, and payment tracking with efficiency and accuracy. By automating these operations, BDMS significantly reduces manual workload and minimizes the risk of errors in financial data handling.

Test users found the system interface to be intuitive and user-friendly, allowing for smooth navigation through modules such as customer management, account handling, loan monitoring, and transaction history. Real-time feedback and clear workflows enabled both employees and administrators to access and manage data with ease.

The system efficiently generates reports on customer accounts, active loans, and repayment schedules, offering valuable insights into branch performance and financial activity. These features support quick decision-making and ensure that administrators have full visibility over all operations.

The backend is built using Node.js with Express, while the frontend is developed using React.js, ensuring responsive and interactive user experiences. The database is powered by MySQL, which ensures data consistency, high availability, and secure relational storage.

Overall, the BDMS enhances core banking operations by improving data organization, increasing process speed, and reducing redundancy. It provides a centralized, secure, and scalable platform that benefits customers, employees, and banking administrators alike.

## 4.1 Screenshots

The following is a series of screenshots of the developed application.

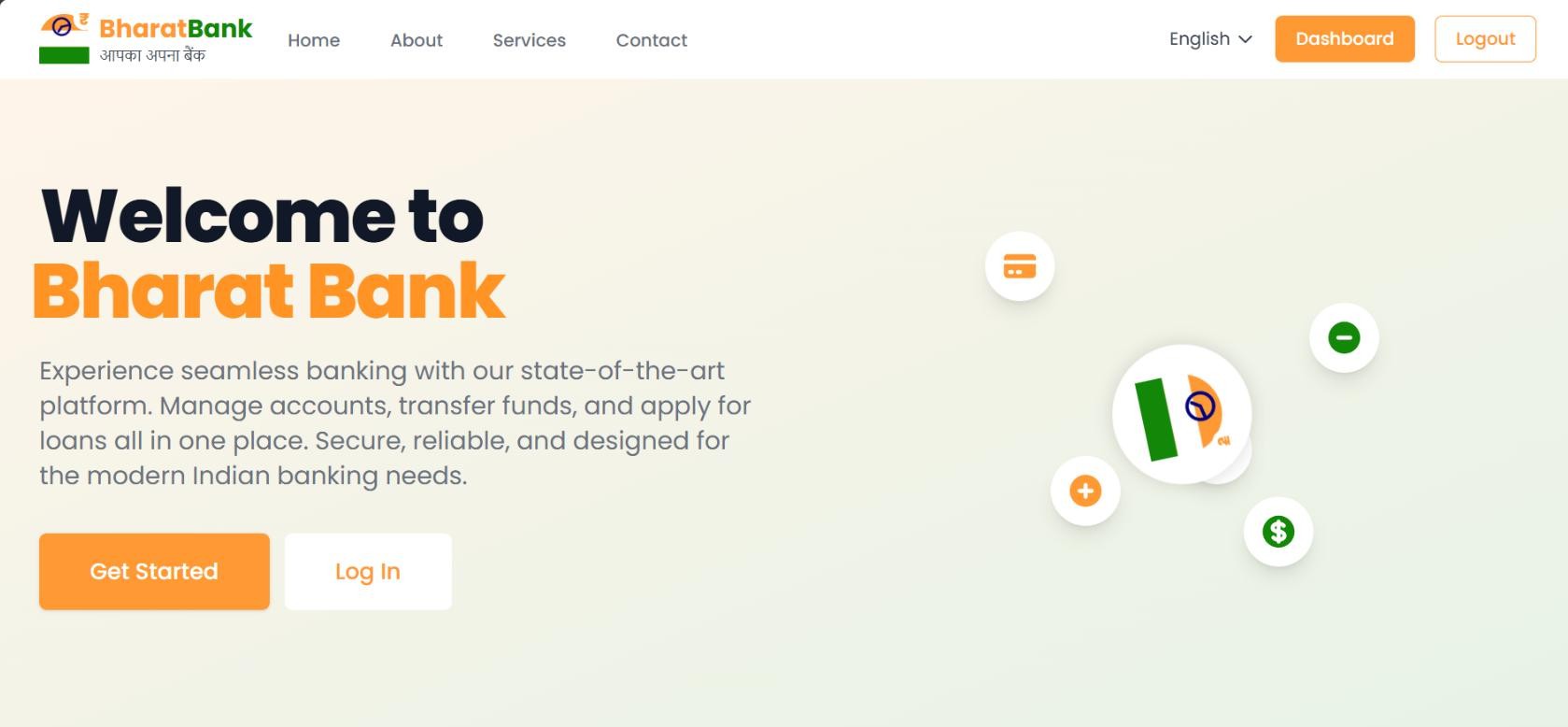


Fig 4.1.1 Welcome Page

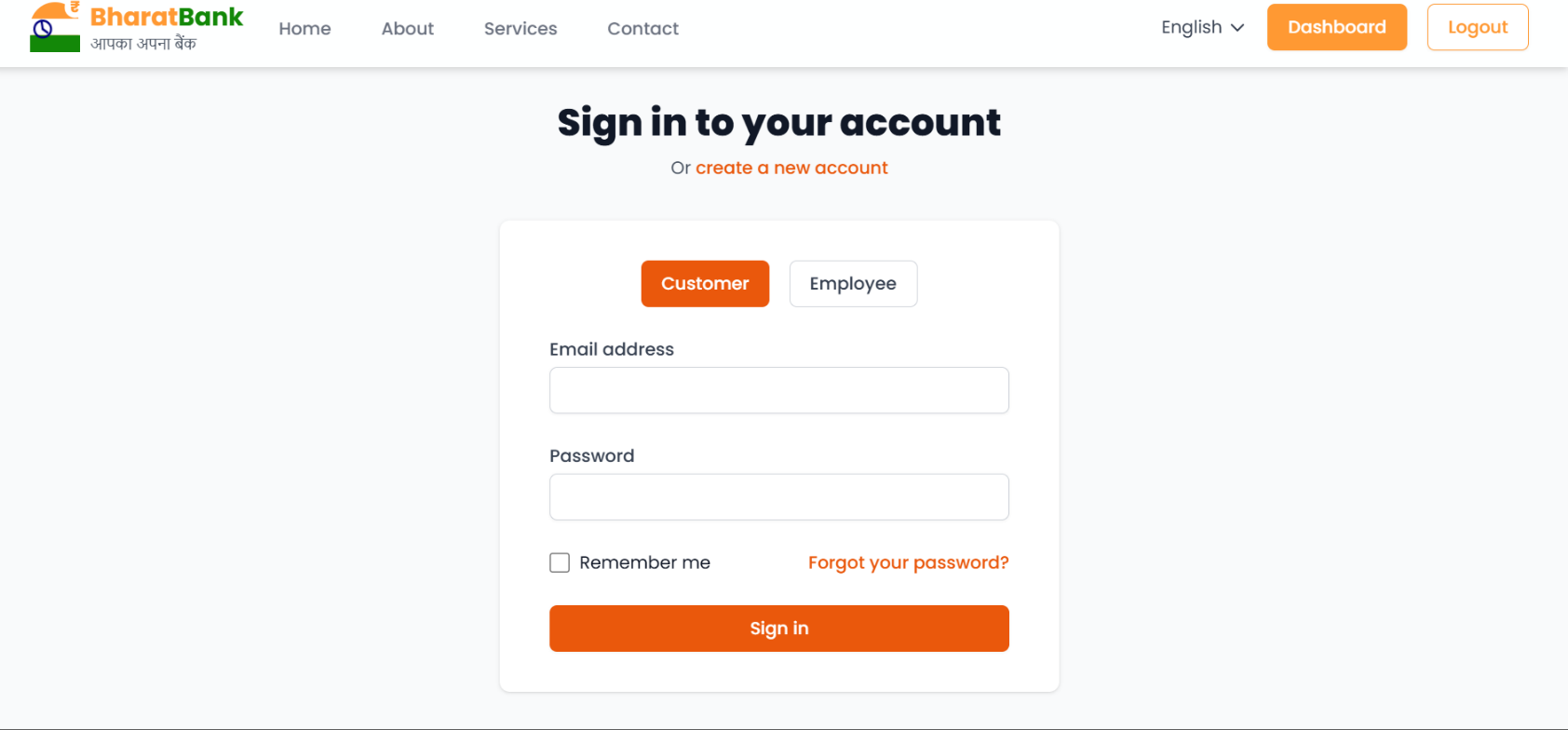


Fig 4.1.2 Login/Signup Page

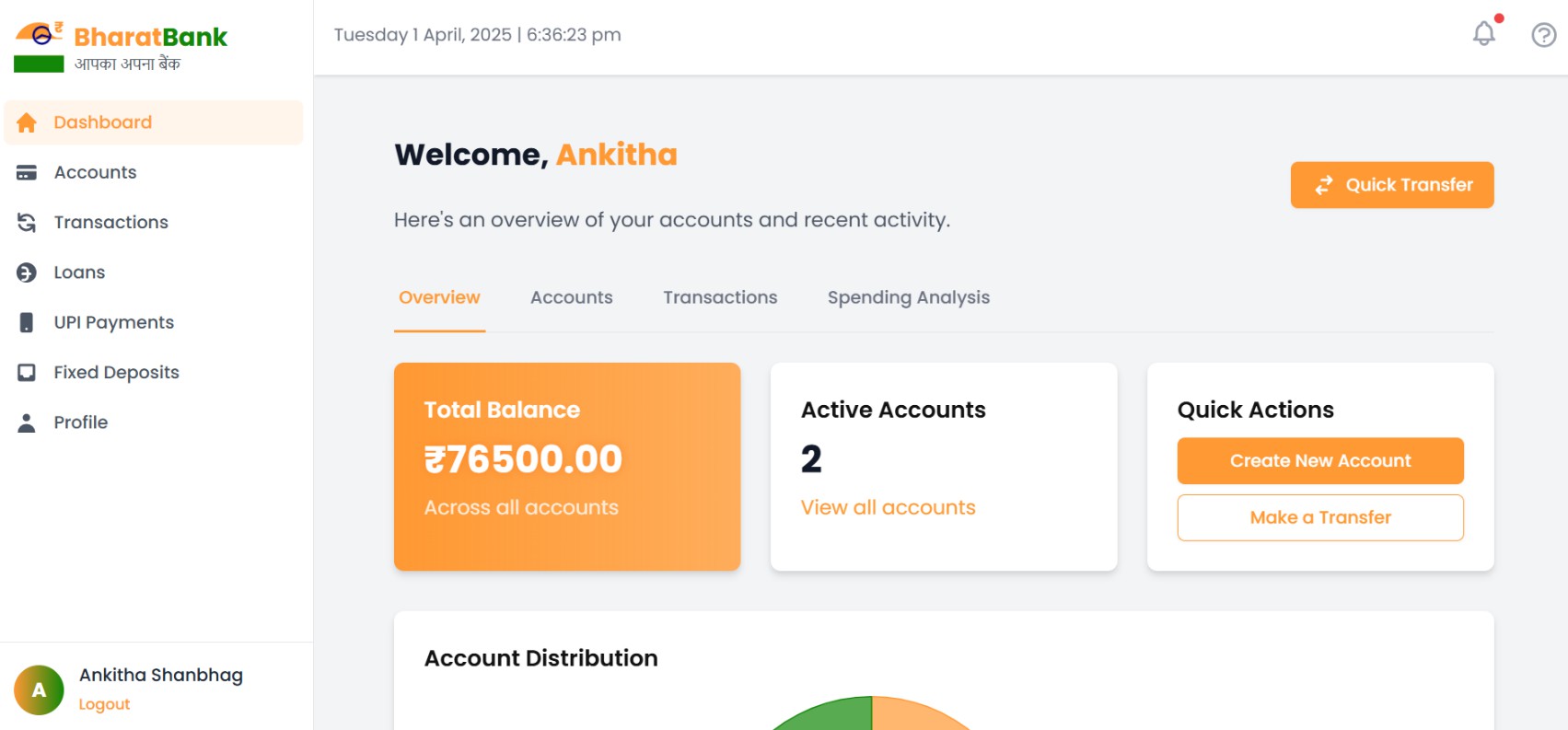


Fig 4.1.3 Customer Dashboard

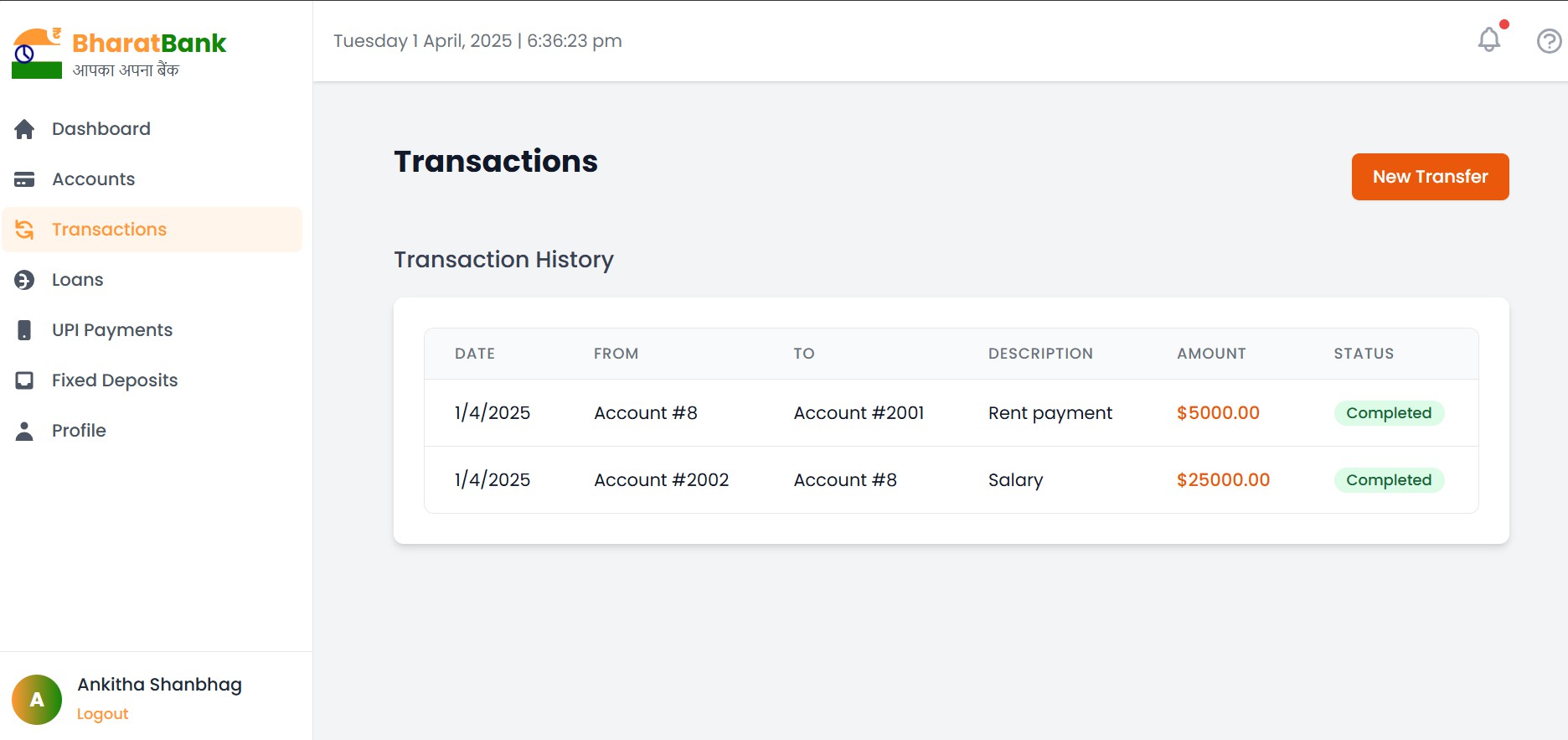


Fig 4.1.4 Transactions Page

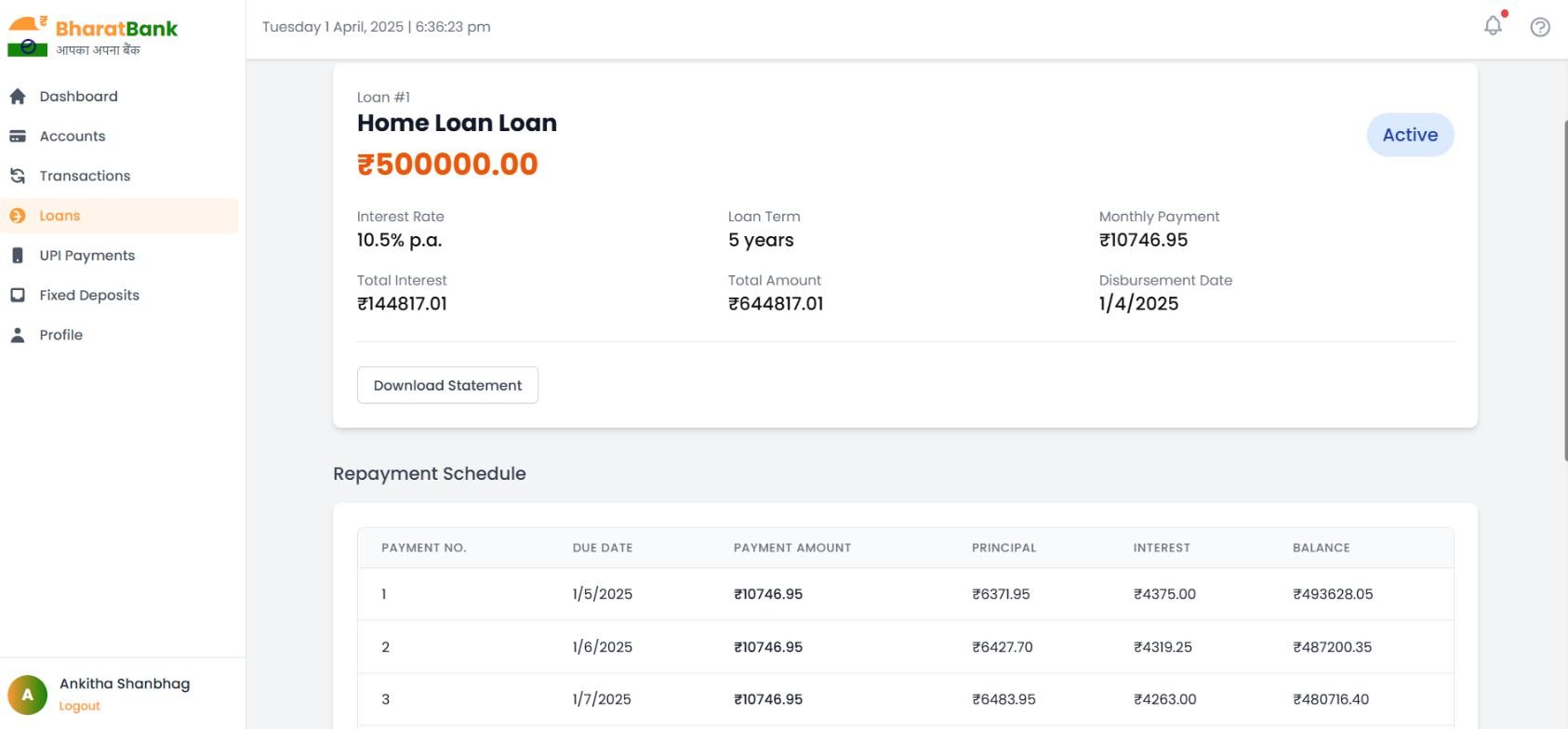


Fig 4.1.5 Loan Page

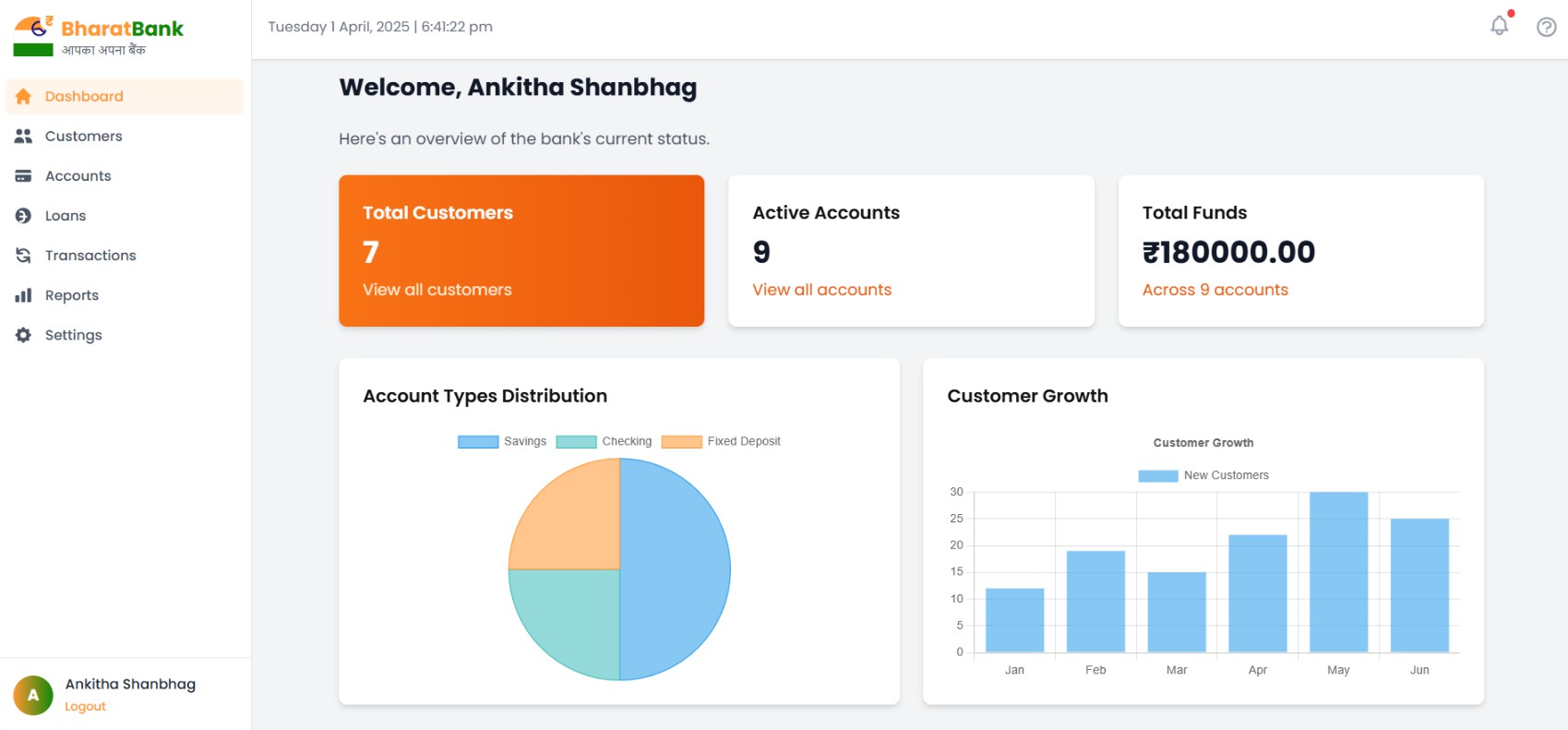


Fig 4.1.6 Employee Dashboard

* **Figure 4.1.1: Welcome Page** – Introduces users to the Bharat Bank with an engaging interface.
* **Figure 4.1.2: Login/Signup Page** – Allows both Customer and Employee to login or create new accounts respectively.
* **Figure 4.1.3: Customer Dashboard**– Interface where all the details of the customer is displayed
* **Figure 4.1.4: Transactions Page** – Details of all the transactions done by the customer is displayed
* **Figure 4.1.5: Loan Page** – Number of loans the customer had taken, and their payments is displayed
* **Figure 4.1.6: Employee Dashboard** – Details of all the Customers and their accounts, loans and transactions id displayed

# CONCLUSION

The project aimed to develop the **Bank Database Management System (BDMS)** to efficiently manage and organize key operations within a banking environment. The system was implemented using a **relational database management system (MySQL)** and features a user-friendly interface developed using **React.js** for the frontend and **Node.js with Express** for the backend, enabling ease of use for customers, employees, and administrators.

BDMS allows for the secure handling of **customer records, account creation, loan issuance, payment tracking, and employee-account management**, while automating essential banking processes such as transactions and loan repayments. The database design adheres to **third normal form (3NF)** and beyond, minimizing data redundancy and ensuring consistency across all modules. The overall structure promotes efficient data retrieval, streamlined workflows, and centralized control over financial operations.

The integration between the frontend, backend, and database enables smooth communication across all system layers, allowing real-time access to account information, transaction histories, and loan details. The system supports role-based access, enabling differentiated permissions for customers, employees, and administrators to safeguard sensitive data.

## Further Improvements

While the current implementation successfully handles core banking functionality, future enhancements could include:

* Integration with **online payment gateways** for real-time fund transfers.
* Implementation of **customer authentication via OTP or biometric login**.
* A **dashboard for credit score tracking** and loan eligibility analysis.
* Addition of a **notification system** to alert users about low balances, EMI due dates, and successful transactions.
* A **mobile-friendly version** or Android/iOS app for on-the-go banking access.
* Export options for **PDF/CSV reports** for audits and customer summaries.

# BIBLIOGRAPHY

1. Ramez Elmasri and Shamkant B Navathe, *Fundamentals of Database Systems*, 6th ed. Addison Wesley, 2009.
2. Abraham Silberschatz, Henry F. Korth, S. Sudarshan, *Database System Concepts*, 6th ed. Tata McGraw-Hill, 2010.
3. Alex Banks and Eve Porcello, *Learning React: Modern Patterns for Developing React Apps*, 2nd ed. O'Reilly Media, 2020.
4. Miguel Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed. O'Reilly Media, 2018.
5. React Documentation
6. Flask Documentation
7. MySQL Documentation