**Internship Project Title:**

**Financial Management System (FMS) – Database System Design**

****

**REPORT**

**Name: Dev Shankar**

**Course BCA**

**(Tecnia Institute of advance studies)**

**YEAR-> 2023-26**

**Date->19-06-2025**

**Introduction**

This document outlines the **design and structure** of a comprehensive **Financial Management System (FMS)** intended for use in modern banking or fintech environments. The purpose of this system is to **efficiently manage customer accounts, transactions, loans, investments, and administrative controls** within a secure and scalable relational database model.

The Financial Management System is developed with a strong emphasis on **data integrity, modularity, and clarity of relationships** between key entities. It is designed to support both day-to-day banking operations and long-term financial tracking, ensuring institutions can offer seamless and transparent financial services to their users.

**2. Purpose of the System**

The primary objective of this system is to:

* Provide a centralized database for **customer information management**.
* Allow tracking and categorization of **multiple account types and transactions**.
* Facilitate **loan issuance and repayment tracking**.
* Monitor **investment activities and portfolio growth**.
* Empower administrators to manage operations without risking data exposure to unauthorized personnel.

The database structure ensures **fast access, efficient queries**, and logical relationships, enabling analysts and developers to build high-performance financial applications, dashboards, and reports

**3. Scope**

This Financial Management System supports a range of operational and analytical functions across:

* **Retail Banking**: Savings, current, and fixed accounts.
* **Transaction Logging**: Real-time updates of withdrawals, deposits, and transfers.
* **Loan Processing**: From application to EMI payments and closure.
* **Investment Management**: Tracking user investments in various asset classes.
* **Administrative Controls**: Role-based access and audit tracking.

The system’s architecture makes it suitable for small- to mid-sized financial institutions aiming for digital transformation and operational transparency.

**4. Target Audience**

This documentation is intended for:

* **Database Architects & Engineers** – to implement or customize the schema.
* **Developers** – to integrate the database into applications or APIs.
* **Business Analysts & Data Scientists** – to run analytics and reporting workflows.
* **Auditors & Compliance Officers** – to understand data flow and validations.
* **Stakeholders** – to assess business coverage and suggest features.

**Entity Overview**

This section describes the core entities (tables) used in the Financial Management System and their respective purposes:

1. **Users**  
   This table stores essential information about registered users of the system. Each user has a unique identifier (user\_id) along with details like full name, email, phone number, and account creation date.  
   *Purpose:* Acts as the central entity for linking user-related data such as accounts, loans, and investments.
2. **Accounts**  
   This table contains information about users’ bank accounts, such as account type (e.g., savings, current), account status, and current balance. It is linked to the Users table via user\_id.  
   *Purpose:* Enables tracking and management of multiple accounts held by a user.
3. **Transactions**  
   This table records every financial transaction associated with an account. Each transaction stores the type (credit or debit), amount, and date, and links to the relevant account using account\_id.  
   *Purpose:* Maintains a complete transaction history for each account for monitoring and auditing.
4. **Loans**  
   The Loans table stores data related to loans taken by users, including the loan type, principal amount, interest rate, and loan duration. It references the Users table through user\_id.  
   *Purpose:* Facilitates management and tracking of different loans issued to users.
5. **Payments**  
   This table holds records of individual repayments made toward specific loans. Each payment entry includes the loan reference, payment amount, method, and date. It connects to the Loans table via loan\_id.  
   *Purpose:* Monitors the repayment schedule and outstanding balances of loans.
6. **Investments**  
   This table captures information about various investment types made by users (e.g., mutual funds, fixed deposits). It includes the investment amount and current status, and is linked to the Users table.  
   *Purpose:* Allows systematic tracking of user investments and their performance.
7. **Admins**  
   the Admins table maintains login credentials and roles for system administrators. It includes fields for username, password (ideally hashed in practice), and role.  
   *Purpose:* Controls access for administrative tasks like user management and system oversight.

**Table Descriptions (with Fields)**

**Users Table**

This table stores the profile information of all users registered in the system.

* user\_id (Primary Key): A unique identifier for each user (usually auto-incremented).
* full\_name: The full name of the user.
* email: User’s email address used for communication and login.
* phone: Contact phone number of the user.
* created\_at: Timestamp marking when the user account was created.

**Accounts Table**

Stores bank account details for each user. A user can have multiple accounts.

* account\_id (Primary Key): Unique ID for each account.
* user\_id (Foreign Key): Refers to user\_id in the Users table to associate the account with a specific user.
* account\_type: Type of account, e.g., Savings, Current, Fixed Deposit.
* balance: Current balance maintained in the account.
* status: Indicates whether the account is active, inactive, or closed.

**Transactions Table**

Maintains a log of all financial transactions linked to accounts.

* transaction\_id (Primary Key): Unique identifier for each transaction.
* account\_id (Foreign Key): Links to account\_id in the Accounts table.
* type: Type of transaction (e.g., Credit or Debit).
* amount: Monetary value involved in the transaction.
* date: Date and time of the transaction.

***Loans Table***

Holds details about loans availed by users.

* loan\_id (Primary Key): Unique identifier for each loan.
* user\_id (Foreign Key): Links the loan to a user.
* loan\_type: Specifies the type of loan (e.g., Home, Personal, Education).
* principal\_amount: Initial amount borrowed.
* interest\_rate: Rate of interest applicable to the loan.
* duration\_months: Loan duration in months.

#### **Payments Table**

Tracks repayments made for loans by users.

* payment\_id (Primary Key): Unique ID for each payment record.
* loan\_id (Foreign Key): Refers to the related loan in the Loans table.
* payment\_date: Date on which the payment was made.
* amount: Amount paid in the installment.
* payment\_method: Mode used for repayment (e.g., UPI, Net Banking, Cash).

**Investments Table**

Stores information on various investment types linked to users.

* investment\_id (Primary Key): Unique ID for each investment record.
* user\_id (Foreign Key): Refers back to the Users table.
* investment\_type: Type of investment (e.g., Mutual Fund, Fixed Deposit, Stock).
* amount\_invested: Total amount invested.
* status: Indicates whether the investment is active, matured, or withdrawn.

**Admins Table**

Manages administrator credentials for system access.

* admin\_id (Primary Key): Unique ID for each admin.
* username: Admin’s login name.
* password: Encrypted password (stored securely).
* role: Specifies the admin’s role (e.g., Super Admin, Auditor, Manager).

**SQL Table Creation Scripts**

This section provides the SQL statements used to create the database and its tables.

CREATE DATABASE Financial\_management\_systems;

USE Financial\_management\_systems;

-- Users Table

CREATE TABLE Users (

user\_id INT PRIMARY KEY AUTO\_INCREMENT,

full\_name VARCHAR(255) NOT NULL,

email VARCHAR(255) UNIQUE NOT NULL,

phone\_number VARCHAR(20) UNIQUE,

address VARCHAR(255),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Accounts Table

CREATE TABLE Accounts (

account\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT NOT NULL,

account\_type VARCHAR(255) NOT NULL,

balance DECIMAL(10, 2) DEFAULT 0.00,

opened\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status VARCHAR(255) NOT NULL,

FOREIGN KEY (user\_id) REFERENCES Users(user\_id) ON DELETE CASCADE

);

-- Transactions Table

CREATE TABLE Transactions (

transaction\_id INT PRIMARY KEY AUTO\_INCREMENT,

account\_id INT NOT NULL,

transaction\_type ENUM('Deposit', 'Withdrawal', 'Transfer') NOT NULL,

amount DECIMAL(10, 2) NOT NULL,

transaction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

description TEXT,

FOREIGN KEY (account\_id) REFERENCES Accounts(account\_id) ON DELETE CASCADE

);

-- Loans Table

CREATE TABLE Loans (

loan\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT NOT NULL,

loan\_type VARCHAR(50) NOT NULL,

amount DECIMAL(10, 2) NOT NULL,

interest\_rate DECIMAL(5, 2) NOT NULL,

start\_date DATE NOT NULL,

end\_date DATE NOT NULL,

status ENUM('Approved', 'Pending', 'Rejected') NOT NULL,

FOREIGN KEY (user\_id) REFERENCES Users(user\_id) ON DELETE CASCADE

);

-- Investments Table

CREATE TABLE Investments (

investment\_id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT NOT NULL,

investment\_type VARCHAR(50) NOT NULL,

amount\_invested DECIMAL(10, 2) NOT NULL,

date\_invested DATE NOT NULL,

current\_value DECIMAL(10, 2) DEFAULT 0.00,

FOREIGN KEY (user\_id) REFERENCES Users(user\_id) ON DELETE CASCADE

);

-- Payments Table

CREATE TABLE Payments (

payment\_id INT PRIMARY KEY AUTO\_INCREMENT,

loan\_id INT NOT NULL,

amount\_paid DECIMAL(10, 2) NOT NULL,

payment\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

payment\_method VARCHAR(50) NOT NULL,

FOREIGN KEY (loan\_id) REFERENCES Loans(loan\_id) ON DELETE CASCADE

);

-- Admins Table

CREATE TABLE Admins (

admin\_id INT PRIMARY KEY AUTO\_INCREMENT,

username VARCHAR(255) UNIQUE NOT NULL,

password\_hash VARCHAR(255) NOT NULL,

role ENUM('SuperAdmin', 'Manager', 'Auditor') NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**Data Insertion in Tables**

-- Sample Users

INSERT INTO Users VALUES

(1, "Dev Shankar", "devshankar3@gmail.com", "7894568978", "Delhi", "2024-05-19 08:40:50"),

(2, "Rahul", "rahul05@gmail.com", "9130258549", "South west delhi", "2025-03-19 10:43:40"),

(3, "Amit Gond", "amitji@gmail.com", "8585697489", "North Delhi", "2025-03-21 13:11:03"),

(4, "Zahid Ali", "zahidali34@gmail.com", "7859652541", "East Delhi", "2025-04-01 07:40:47"),

(5, "Nitin kumar", "sainiNitin10@gmail.com", "8595784589", "South west delhi", "2025-04-05 11:51:31");

-- Sample Accounts

INSERT INTO Accounts VALUES

(78945869, 1, "Savings", 18000.56, "2024-05-19 08:40:50", "Active"),

(78523658, 2, "Business", 50000.85, "2025-03-19 10:43:40", "Active"),

(79025484, 3, "Savings", 10000.80, "2025-03-21 13:11:03", "Inactive"),

(79177859, 4, "Business", 200000.90, "2025-04-01 07:40:47", "Active"),

(79258967, 5, "Business", 195266.66, "2025-04-05 11:51:31", "Inactive");

-- Transactions

INSERT INTO Transactions VALUES

(1107854698, 78945869, 'Deposit', 1000, "2024-06-19", "Deposit amount"),

(1125689578, 78523658, 'Transfer', 5000, "2025-05-19", "Salary Payment"),

(1120549854, 79025484, 'Withdrawal', 8000, "2025-03-21", "Property Purchase"),

(1158978546, 79177859, 'Deposit', 6000, "2025-04-01", "Deposit amount"),

(1178542146, 79258967, 'Transfer', 9000, "2025-04-05", "Game Purchase");

-- Loans

INSERT INTO Loans VALUES

(1102546897, 1, 'Home Loan', 120000.00, 7.25, '2024-08-15', '2034-01-15', 'Approved'),

(1225468978, 2, 'Car Loan', 500000.00, 8.50, '2025-07-01', '2028-07-01', 'Pending'),

(1358965745, 3, 'Personal Loan', 200000.00, 10.75, '2025-05-10', '2027-02-10', 'Rejected'),

(1458468957, 4, 'Education Loan', 750000.00, 6.95, '2025-08-20', '2029-08-20', 'Approved'),

(1568587458, 5, 'Business Loan', 150000.00, 9.25, '2025-06-01', '2030-06-01', 'Pending');

-- Investments

INSERT INTO Investments VALUES

(0017854963, 1, 'Stocks', 100000.00, '2025-05-10', 125000.00),

(0025896587, 2, 'Mutual Fund', 50000.00, '2025-09-01', 56500.00),

(0038446854, 3, 'Fixed Deposit', 200000.00, '2025-06-15', 206000.00),

(0046897854, 4, 'Real Estate', 800000.00, '2025-09-20', 950000.00),

(0056845987, 5, 'Crypto', 30000.00, '2025-12-05', 27500.00);

-- Payments

INSERT INTO Payments VALUES

(11102547, 1102546897, 125000.00, "2030-02-15", 'Bank Transfer'),

(11265895, 1225468978, 57000.00, "2026-03-22", 'UPI'),

(11358946, 1358965745, 280000.00, "2027-02-10", 'Credit Card'),

(11425689, 1458468957, 754000.00, "2029-08-20", 'NEFT'),

(11574859, 1568587458, 158000.00, "2030-06-01", 'Debit Card');

-- Admins

INSERT INTO Admins VALUES

(1101, 'admin\_Dev', 'dev123', 'SuperAdmin', "2024-05-19"),

(1102, 'manager\_Raj', 'raj870', 'Manager', "2025-03-19"),

(1103, 'audit\_neha', 'neha345', 'Auditor', "2025-03-21"),

(1104, 'manager\_amit', 'amit456', 'Manager', "2025-04-01"),

(1105, 'audit\_kiran', 'kiran789', 'Auditor', "2025-04-05");

**Relationships Explanation**

***1. Users → Accounts***

* **Type:** One-to-Many
* **Description:** A single user can have multiple accounts, such as savings, current, or fixed deposit accounts.
* **Linking Field:** user\_id in the Accounts table references user\_id in the Users table.

Users (user\_id) ───▶ Accounts (user\_id )

***2*. Accounts → Transactions**

* **Type:** One-to-Many
* **Description:** Each account can have multiple financial transactions recorded, including credits and debits.
* **Linking Field:** account\_id in the Transactions table references account\_id in the Accounts table.

Accounts (account\_id) ───▶ Transactions (account\_id as FK)

**3. Users → Loans**

* **Type:** One-to-Many
* **Description:** A user can take multiple loans with different terms and conditions.
* **Linking Field:** user\_id in the Loans table references user\_id in the Users table.

Users (user\_id) ───▶ Loans (user\_id as FK)

**4. Loans → Payments**

* **Type:** One-to-Many
* **Description:** Each loan can have multiple payments made against it over time.
* **Linking Field:** loan\_id in the Payments table references loan\_id in the Loans table.

Loans (loan\_id) ───▶ Payments (loan\_id as FK)

📌 Combined Chain: A single user may have many loans, and each loan can have multiple repayments.

Users ───▶ Loans ───▶ Payments

**5. Users → Investments**

* **Type:** One-to-Many
* **Description:** A user can invest in different instruments such as mutual funds, stocks, and fixed deposits.
* **Linking Field:** user\_id in the Investments table references user\_id in the Users table.

**Normalization & Design Logic**

The Financial Management System database follows a **normalized relational design**, primarily up to the **Third Normal Form (3NF)**. This ensures data redundancy is minimized and the integrity of relationships between entities is maintained.

**Normalization Benefits Applied:**

1. **Avoiding Data Repetition**
   * Instead of storing user details multiple times with each account, loan, or investment, all basic user data is stored in the Users table.
   * Other tables like Accounts, Loans, and Investments use user\_id as a foreign key, which removes duplication.
2. **Atomic Data Structure**
   * All fields are broken down into the smallest possible unit of information.
   * For example, in Transactions, each record has just one amount, one type, and one date. This keeps each record atomic.
3. **Functional Dependencies Eliminated**
   * In the Payments table, repayment amount depends only on loan\_id, not the full user profile, which is stored separately.
   * This separation ensures that non-dependent fields don’t exist in the same table.
4. **Data Integrity through Foreign Keys**
   * The use of foreign key relationships (e.g., user\_id, account\_id, loan\_id) ensures that related data is linked properly.
   * This design avoids inconsistencies like orphaned transactions or payments with no valid parent record.

**Design Logic Summary:**

* The system adopts a **modular structure**, with each table focusing on one entity or activity.
* **Separate tables for Accounts, Transactions, Loans, Payments, and Investments** allow for independent expansion and data management.
* By using this normalization approach:
  + **Data anomalies** are reduced (no duplication or partial updates).
  + **Storage efficiency** is improved.
  + **Query performance** increases due to well-indexed and linked data.

**Conclusion**

The design and implementation of the Financial Management System database demonstrate a strong foundation in relational database principles, with a clear focus on **data organization, integrity, scalability, and security**. Through normalization up to the Third Normal Form (3NF), the system ensures that each entity and attribute is stored efficiently without unnecessary duplication, thereby minimizing redundancy and preventing data anomalies.

The schema encompasses a comprehensive range of real-world financial operations, including user registration, account creation, transactional activity, loan management, investment tracking, and administrative oversight. Each of these functional areas is represented by its own dedicated table — such as Users, Accounts, Transactions, Loans, Investments, Payments, and Admins — and interconnected via carefully constructed foreign key relationships. These relational links allow the database to maintain high referential integrity while enabling complex queries and data retrieval processes.

From a technical standpoint, the structure supports modular growth. New features like audit logging, advanced user roles, budgeting tools, or automated alerts can be integrated without disrupting the existing schema. Moreover, the use of industry-standard SQL practices — including constraints, timestamps, data validation types like ENUM, and default values — adds both precision and professionalism to the schema’s implementation.

On the administrative side, the Admins table ensures that system access is securely managed through role-based permissions. This offers a scalable mechanism for implementing multiple levels of system control and ensuring that sensitive financial data is only accessible by authorized personnel.

In summary, this Financial Management System database offers a **robust, adaptable, and real-world-ready architecture** capable of serving as the backbone for a full-fledged financial application. Whether deployed in a banking, investment, or personal finance context, the system is designed to support efficient operations, accurate record-keeping, secure data handling, and future expansion.

**Entity Relationship Diagram**

