

DBMS-Driven Financial and Investment Management System

Ilakiyaselvan N¹, Jayanthi Sharma², Riddhi.B³ and Sinchan Shetty⁴
Associate Professor, School of Computer Science and Engineering

Vellore Institute of Technology, Chennai, India

ilakiyaselvan.n@vit.ac.in

UG Scholar, School of Computer Science and Engineering

Vellore Institute of Technology, Chennai, India

²jayanthi.sharma2022@vitstudent.ac.in

³riddhi.b2022@vitstudent.ac.in

⁴sinchan.nagaraj2022@vitstudent.ac.in

Abstract: This project aims to build and execute an Investment Management System that allows users to explore various investment schemes, including Fixed Deposits (FD), Public Provident Funds (PPF), and other investment choices. Customers may pick a suitable scheme, provide pertinent information, input their financial details, and receive real-time calculations of the potential returns on their investments including the final amount they will receive after interest is accrued over the selected period. The system's goal is to provide an easy-to-use interface for users to enter financial data and get interest-augmented results while utilizing Entity-Relationship modeling for successful database architecture.

1 Introduction

In the digital era, banks have much diversity in financial services. These services have to be maintained duly. The traditional banking systems were mainly concerned with loan management. However, investment schemes like Fixed Deposit, Recurring Deposit, Mutual Funds and Public Provident Fund gained more interest among people due to their stable returns. With the increasing demand for these services, it is now high time to have an integrated system that makes selection, tracking, and calculation of returns on investment easier.

The proposed system is the development of an investment management system that will enable the user to view different available investment schemes and input their financials to get calculated returns in real time. This system manages and stores the financial data through a DBMS efficiently and allows access to the data with security. Digital tools now constitute the bedrock of modern banking, and this initiative will further reinforce customer experience and trust in the services provided, ensuring that they are delivered in an accurate, transparent, user-friendly manner in financial management.

2 Background

Current banking management systems largely focus on handling basic financial services like loans, deposits, and withdrawals. However, these systems often overlook the growing demand for effective tools that manage various investment schemes, such as Fixed Deposits (FDs), Public Provident Funds (PPF), and other long-term investment plans. As customer interest in secure investment options grows, there's a need for a system that prioritizes these schemes.

This project aims to address that gap. Instead of treating loans as the central entity in the system, the focus shifts towards managing a wide range of banking schemes. The challenge lies in designing a system that can efficiently handle multiple investment schemes, each with different interest rates, compounding frequencies, and withdrawal options. The primary goal is to create a user-friendly and accurate platform that not only simplifies the management of these schemes but also empowers customers to make informed investment decisions.

3 Literature review

[1] analyses the results and approaches of studies that compare the performance of different database management systems (DBMSs) to find the most efficient one and provide recommendations for both industry and research based on these comparisons. The paper involves a systematic review of literature from sources like ACM Digital Library, IEEE Xplore, and ScienceDirect, focusing on studies published between 2000 and March 2022 that compare query execution performance across different DBMSs, including Relational, NoSQL, NewSQL, and others. Major findings of the paper include that performance tests often didn't reflect real-world scenarios, in most papers, tests lacked details for replication, and results were inconsistent due to varying test conditions. It was found that Yahoo! Cloud Serving Benchmark (YCSB) was the most widely used tool, and noted a lack of studies in major data management forums like ACM SIGMOD or VLDB, highlighting a gap in rigorous research. The paper concludes that current DBMS performance comparisons are insufficient for making definitive judgments about which system is best. It advises industry practitioners to interpret performance results cautiously and researchers to follow testing guidelines, consider various testing approaches, and explore different use cases beyond simple DBMS comparisons.

[2] compares relational and NoSQL databases, exploring the motivations behind NoSQL adoption, areas of application, and security concerns. It highlights the trends driving NoSQL's popularity and contrasts them with traditional relational databases, especially in terms of handling large data volumes and scalability. The paper involves a literature review, examining the characteristics, advantages, and limitations of both types of databases. It compares aspects such as data models, transaction reliability, scalability, cloud support, big data handling, complexity, crash recovery, and security features. In the literature, several key differences between relational and NoSQL databases have been highlighted. Relational databases provide reliable transactions through ACID properties and use structured schemas, while NoSQL databases offer flexible models like key-value, graph, and document stores, with eventual consistency through BASE or ACID. NoSQL databases are better suited for horizontal scalability, cloud environments, and big data. However, relational databases have more mature security mechanisms, and NoSQL is still evolving in this area. The paper concludes that, although NoSQL databases have existing security gaps, their popularity continues to rise because of their scalability and capacity to manage unstructured data, with ongoing improvements expected to address these challenges.

[3] Focuses on developing a Bank Customer Management System (BCMS) using a database management system (DBMS) to enhance customer satisfaction and bank profitability. The system, built with VB.NET and Microsoft Access, allows customers to manage accounts and transactions online securely. The database plays a crucial role in organizing customer information, ensuring data integrity, and supporting efficient data retrieval and updates. The design process includes Information Level Design to gather requirements and Physical Level Design tailored to the specific DBMS for optimal data handling. With a modular and expandable approach, the system improves customer relationship management, using multi-level security for data protection, ultimately boosting customer retention and bank efficiency.

[4] The paper on "Bank Management System" generates a computerized solution in order to automate and secure the transactions of finances by reducing the requirement of physical bank branches and manual processing. The system provides a unique customer account number for the identity and thus, facilitates secured transactions like transferring money and withdrawal with the updation of accounts through Aadhar linking or changing the branch location, etc. The proposed system provides great efficiency, accuracy, and security attributes; some of the attributes are multilevel authentication and online data updating, with an intention to enhance customer satisfaction. It motivates further growth into a better and more efficient system in the banking industry.

[5] This paper focuses on research in the area of the database management system so well supports designing a computer-based management system that deals with customer account information and generates monthly statements. The introduction of user-friendliness into the updation, maintenance, and searching in record is perfectly aligned to the literature on user-centric design by Nielsen, 2000; Norman, 2008 and the modern database systems by Aronov et al., 2008; Kotelnikov, 2017; Moller et al., 2011. Literature related to the advantage of automation in banking systems also supports this aspect by providing justification towards making use of GUI-based systems for enhancing user-friendliness and productivity and the purpose of the project is to increase productivity and make effective utilization of working hours with minimum manpower.

Table 1: Related works comparison table

Reference Number	Author and Year	Methodology	Dataset used	Research Findings	Performance measurement
[1]	Taipalus, T. et al. 2024	A systematic literature review approach, searching databases for articles comparing query execution performance across different DBMSs, published between 2000 and March 2022. A total of 117 studies were selected.	ACM Digital Library, IEEE Xplore, ScienceDirect, and Google Scholar	<ul style="list-style-type: none"> * Most performance tests are conducted in non-realistic environments. * Performance test conditions vary, leading to inconsistent results. * Insufficient detail in studies for replicating results. * Commonly compared systems: MongoDB, Cassandra, Redis. 	Test environments, execution details, and parameters varied widely across studies, affecting the consistency of performance results.

[2]] Mohamed, M. A., Altrafi, O. G., and Ismail M. O. et al. 2014	Literature survey, comparing relational and NoSQL databases based on key factors like data models, transaction reliability, scalability, cloud support, big data handling, complexity, crash recovery, and security features.	The paper relies on existing research and studies from various sources to perform a comparative analysis.	<p>*Transaction Reliability: Relational databases support ACID properties; NoSQL varies between BASE and ACID models.</p> <p>*Data Model: Relational uses structured schemas; NoSQL supports flexible, unstructured models.</p> <p>*Scalability: Relational databases struggle with horizontal scaling; NoSQL excels in this.</p> <p>*Cloud & Big Data: NoSQL is better suited for cloud environments and big data applications.</p> <p>*Security: Relational databases have mature security features, while NoSQL is still evolving.</p>	Scalability and flexibility are key strengths of NoSQL, whereas transaction reliability and security remain stronger in relational databases.
-----	--	---	---	---	---

[3]	Dr. Pankaj Kumar et al. 2018	The system was developed using VB.NET for the front end and Microsoft Access for the back end. The database design followed two stages: Information Level Design (gathering user requirements independent of any DBMS) and Physical Level Design (tailored to the specific DBMS, focusing on data integrity and independence).	It includes customer information such as account details, transaction history, loan info, and personal details for managing customer interactions and banking operations.	Implementing the BCMS improved customer relationship management, enhanced security through multi-level protections, and led to higher customer satisfaction and retention. It also reduced the need for in-person banking and improved operational efficiency.	Data integrity, security (multi-level access control), customer satisfaction, reduced processing time for transactions, and increased efficiency in managing customer data and interactions.
[4]	MD.Faizan, MD.Aquil Amwar, Masrurul Haque et al. 2021	This will be laying down the methodology of bringing on board a multi-tier architecture for the presentation layer, utilizing MVC for the service layer, keeping as its focus function automation by banking with multilevel security and user authentication.	The dataset includes general columns for customer information, account balances, and transaction histories required for validating inputs and performing secure banking operations.	<ul style="list-style-type: none"> * Its facilities for automatic transfers of money, change in accounts and branches enable it to perform more efficiently. * Multilevel authentication provides for security as well as fraud prevention at a high rate. * The banking system reduces manual efforts and prevents the entry of human error into banking operations. * It enables users to link their Aadhar, update personal details, and facilitate transactions online. 	<ul style="list-style-type: none"> *Security: Other checks include integrity-checking, secure logins, as well as encryption protocols of data to ensure security. * The system boasts an usability feature; it is designed with an extremely user-friendly interface so that the customers can update their personal details and conduct transactions with ease.

				<p>* The improved solution will enhance customer satisfaction in terms of faster and more reliable services in banking.</p>	<p>*Maintainability: The unit is designed with a modularity of modules that can be easily upgraded and updated for the systems thus ensuring long-term sustainability as well as adaptability.</p>
[5]	Dr. Rolly Gupta, Harshit Batra, Shivansh Kaul	<p>The above project includes the design of a computer-based management system involving handling customer account information. The methodology to be used in this project is going to be of both qualitative and quantitative approach, involving literature review, requirements analysis, database design, and GUI-based system development, including the testing, deployment, and maintenance of the system using tools such as MySQL and Java, over a period of 18 weeks.</p>	<p>The dataset represents a banking system, encompassing information about banks, branches, customers, accounts, and loans. It stores details like bank codes, names, addresses, branch numbers, customer names, social security numbers, phone numbers, addresses, account numbers, balances, account types, loan numbers, amounts, and loan types.</p>	<p>* The system automates monthly statement calculations, ensuring accurate data retrieval and generation from a dedicated database.</p> <p>* User-friendly features simplify record searching and updating, allowing easy access to customer details via account numbers.</p> <p>* The system optimizes working hours, reducing manpower needs and accelerating service delivery.</p> <p>* Designed for adaptability, the system allows future upgrades with minimal changes to existing components.</p> <p>*: The GUI-based interface enables users to operate the system easily, even</p>	<p>* Accuracy Rate: Percentage of accurate monthly statements generated (target: 99% accuracy).</p> <p>* User Satisfaction: Customer and employee satisfaction ratings, collected through surveys (target: 90% satisfaction).</p> <p>* Transaction Processing Time: Average time taken to complete transactions or generate statements (target: under 5 seconds per transaction).</p> <p>* Operational Efficiency:</p>

				without technical knowledge.	prior	Reduction in manpower required for statement processing and customer inquiries (target: 30% reduction). * System Uptime: Percentage of time the system is operational and accessible (target: 99.9% uptime). * Adaptability: Number of successful upgrades or new feature implementations per year (target: at least 2 significant upgrades annually).
--	--	--	--	------------------------------	-------	--

4 Methodology

4.1. Requirement Analysis

The project begins with a comprehensive analysis of the requirements for various banking schemes, such as Fixed Deposits (FD), Public Provident Fund (PPF), and Recurring Deposits (RD). Key input parameters, including the principal amount, rate of interest, and time period, are identified for each scheme. This analysis ensures that the system can handle diverse financial products and their unique features.

4.2. Database Design

The database is designed to capture and manage the data associated with different banking schemes.

- **Entities:** The core entities in the system include Customer, Account, Investments, and Returns.
- **Attributes:** Each banking scheme has attributes such as Policy_ID, Principal_Amount, Interest_Rate, and interest rate. These attributes facilitate the management of financial data and interest calculations.

- **Relationships:** The system supports a one-to-many relationship between Customer and Banking Scheme, allowing a single customer to avail multiple schemes. Each scheme involves interest calculations based on its parameters.

4.3. Setting Up the Local Development Environment

The implementation of the project begins with the installation and configuration of XAMPP, a cross-platform web server solution. XAMPP provides an integrated environment with Apache for hosting web applications and MySQL for database management. Once installed, Apache and MySQL services are launched from the XAMPP control panel to set up a local server.

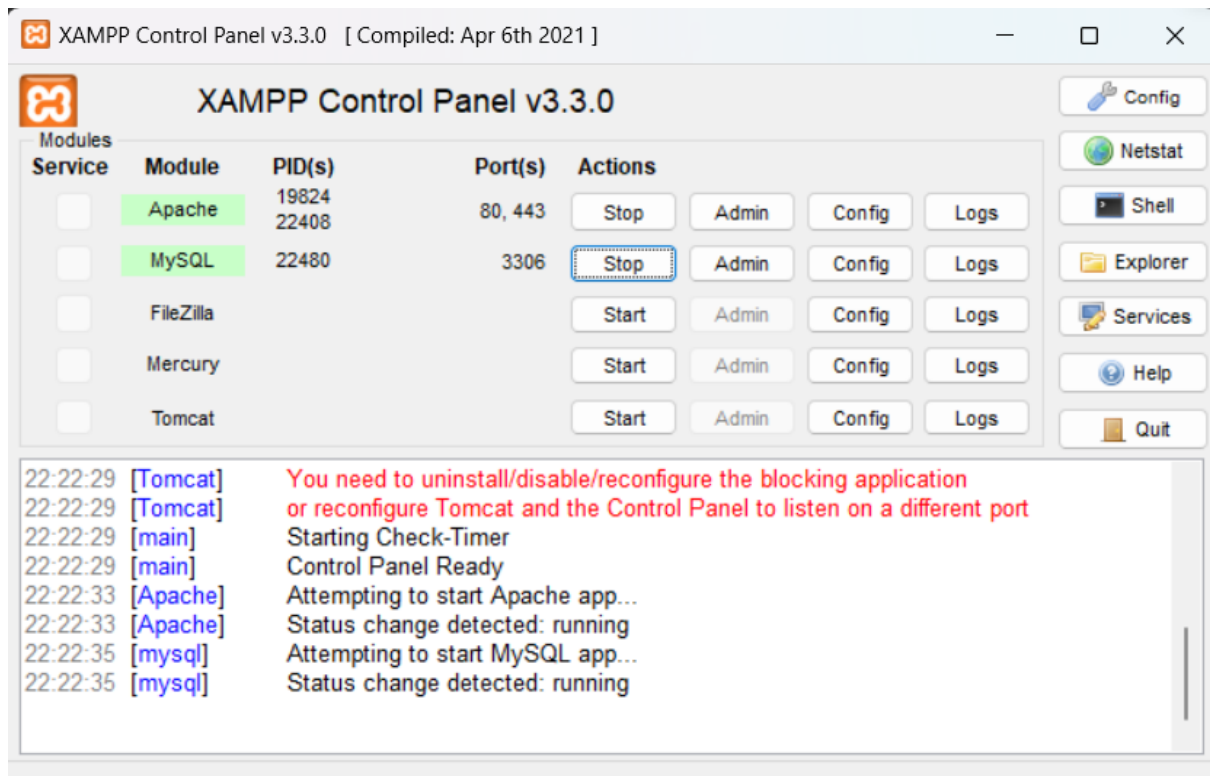


Fig 1 Starting Xampp Platform

4.4. Database Management Using phpMyAdmin

phpMyAdmin, a web-based tool included in XAMPP, is used for managing MySQL databases through an intuitive graphical interface. By navigating to localhost/phpmyadmin, the database for the project, named **investment management**, is created and structured. This database contains multiple tables, including users, transactions and investment. The tool simplifies database operations such as creating tables, inserting data, and executing queries without the need for manual SQL scripting, thereby streamlining the development process.

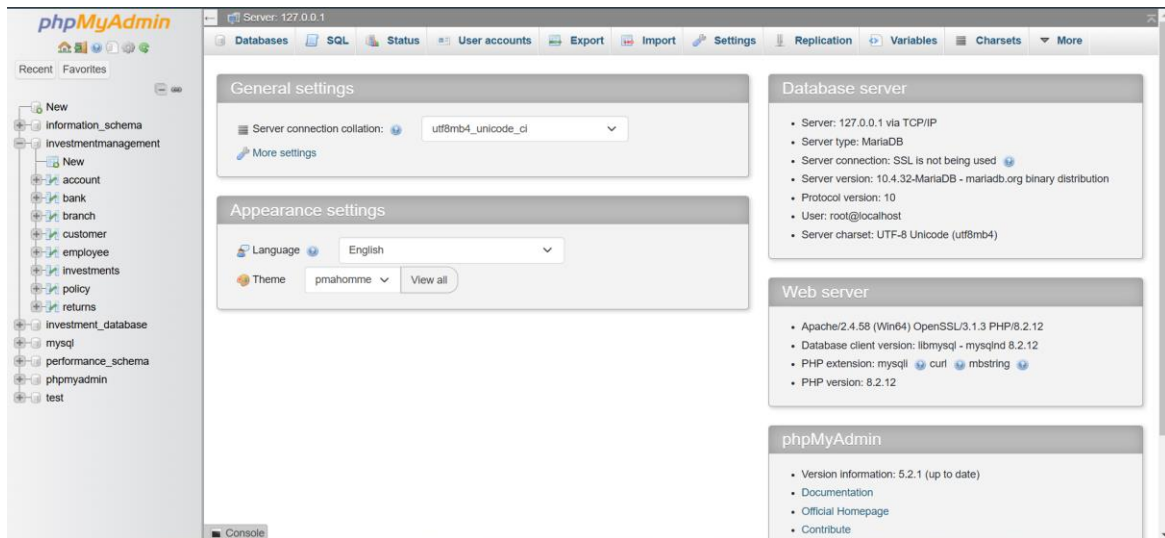


Fig 2 Initial Database State

4.5. Backend Development

The backend functionality is implemented using Python and the Flask framework. A script named app.py serves as the core backend logic, handling HTTP requests and database interactions. The script connects to the MySQL database using a driver like mysql-connector.. Within app.py, various routes and endpoints are defined to process user actions, such as querying balances or updating transaction records. Flask facilitates the development of a lightweight server environment that seamlessly manages data flow between the frontend and the database.

4.6. User Interaction and Workflow

The project's user interface is a web application accessible through a browser. Users interact with the system by performing tasks such as adding funds, viewing transaction history, or managing investments. These actions generate HTTP requests that are processed by the Flask server. The server sends corresponding SQL queries to the MySQL database, retrieves the results, and returns them to the frontend for display. This setup ensures real-time data updates and a smooth user experience.

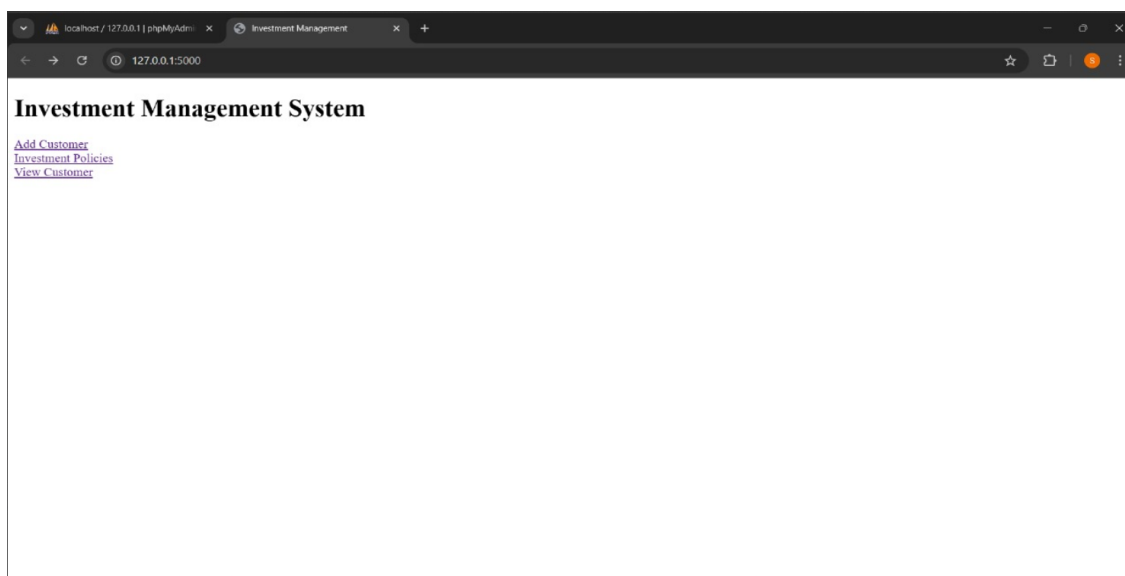


Fig 3 User Interaction Interface

4.7. System Testing and Demonstration

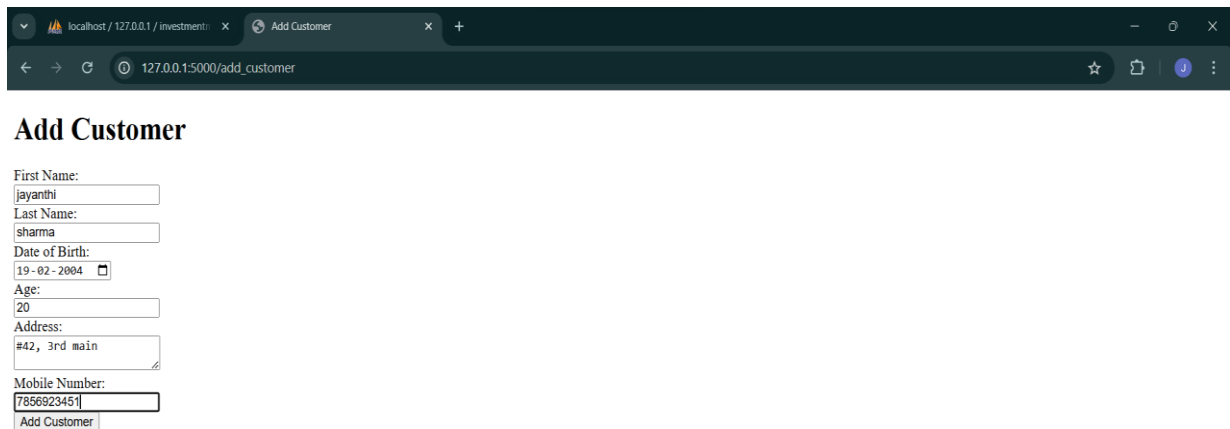
To validate the system, XAMPP services are launched, and database configurations are managed via phpMyAdmin. The Flask server is then started by running app.py. Various user scenarios, including updating investment records and retrieving transaction details, are tested to ensure the system's accuracy and responsiveness. The demonstration highlights the integration of phpMyAdmin, Flask, and MySQL in handling data-driven operations efficiently.

4.8. Project Demonstration

The project demonstration begins with launching the XAMPP control panel to start the Apache and MySQL services. With these services running, the database and corresponding tables, previously created using phpMyAdmin, are ready for use. The system comprises Python and HTML files that handle the backend and frontend, respectively. To start the application, the app.py file is executed, initiating the Flask web server.

4.8.1 Adding a New Customer

The first step involves adding a new customer. Upon entering customer details through the user interface, the data is processed by app.py and inserted into the customers table in the database. The updated customers table can be viewed in phpMyAdmin, confirming the successful addition of the new customer.



The screenshot shows a web browser window with the address bar displaying 'localhost / 127.0.0.1 / investment' and 'Add Customer'. The page title is 'Add Customer'. The form contains the following fields and values:

- First Name: jayanthi
- Last Name: sharma
- Date of Birth: 19-02-2004
- Age: 20
- Address: #42, 3rd main
- Mobile Number: 7856923451

An 'Add Customer' button is located at the bottom of the form.

Fig 4 Adding a New Customer

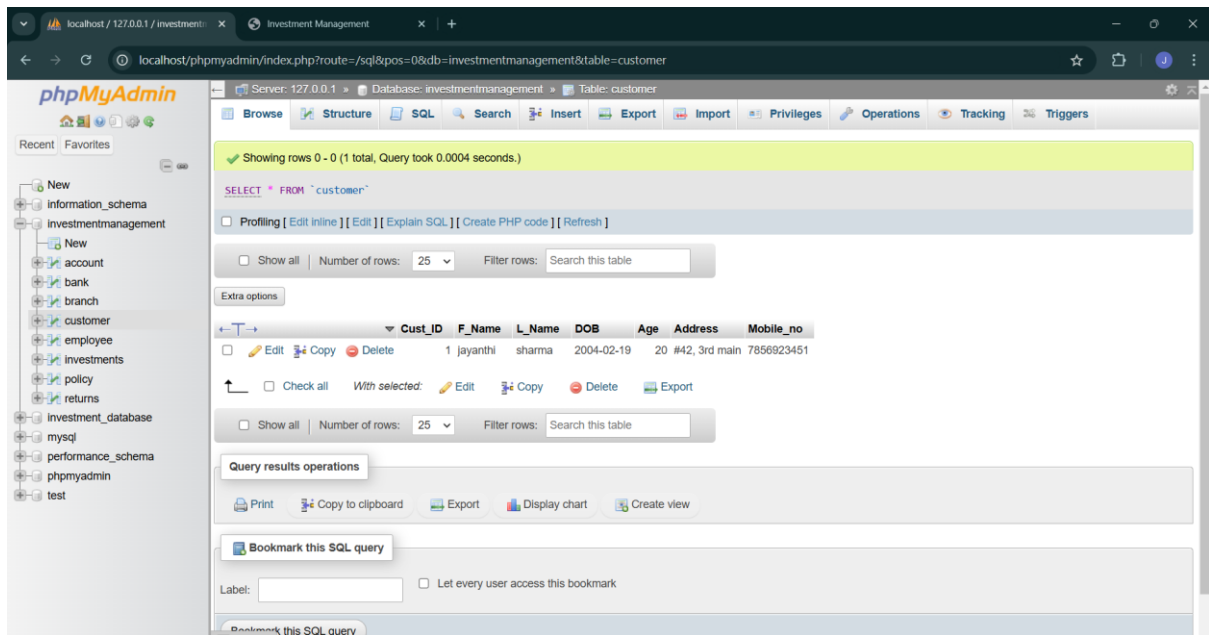


Fig 5 Updates in the Customer Table in the Database

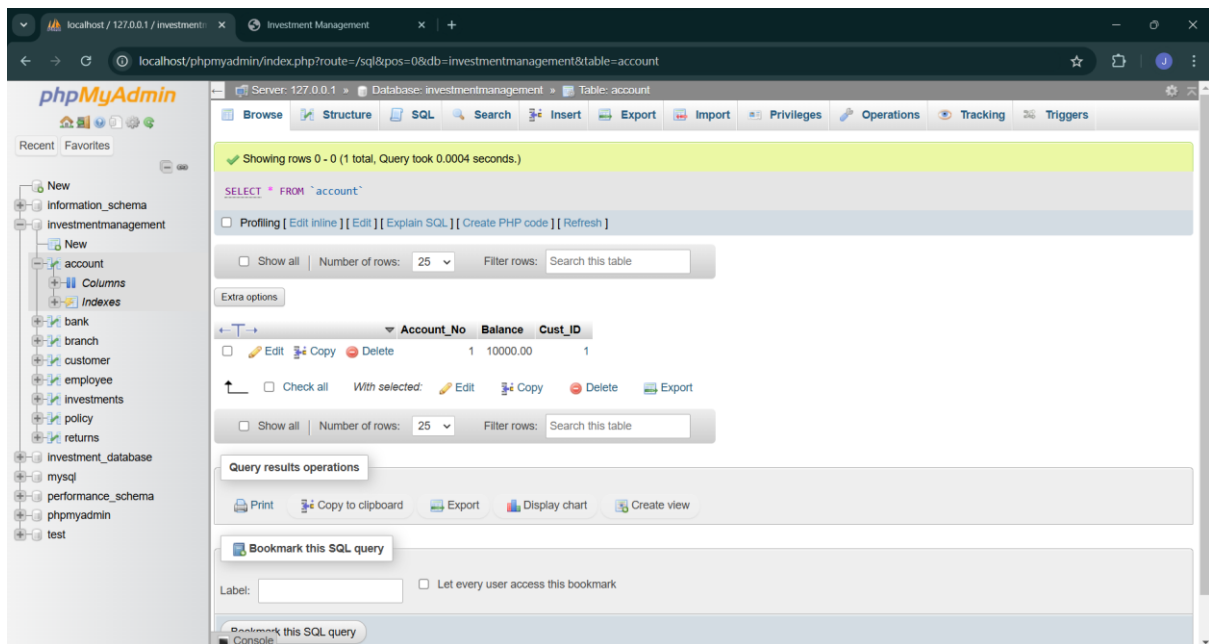


Fig 6 Updates in the Account Table in the Database

4.8.2 Investment Policy Demonstration

Next, the system allows users to explore different investment policies. For example, to view the returns for a Fixed Deposit (FD), the user inputs the policy name, ID, and the investment amount. The backend calculates the potential returns using the compound interest formula and displays the result on the frontend.

Investment Policies

ID	Policy Name	Duration (months)	Interest Rate
1	Fixed Deposit	12	6.5%
2	Recurring Deposit	24	5.8%
3	Mutual Funds	36	8.2%

Make an Investment

Account ID:

Select Policy:

Investment Amount:

Calculate Return

Customer ID:

Select Policy:

Principal Amount:

Return Estimate

Policy: 1

Principal: 90000.0

Estimated Return Amount: 160200.0

Monthly Payment: 13350.0

Lock-in Period: 12 months

[Back to Home -->](#)

Fig 7 Calculating Returns for Fixed Deposit

4.8.3 Making an Investment

After reviewing the returns, the customer decides to invest in the policy. By clicking the "Invest" button, the investment details are processed and recorded in the database. This updates multiple tables, including policies and investments.

Investment Policies

ID	Policy Name	Duration (months)	Interest Rate
1	Fixed Deposit	12	6.5%
2	Recurring Deposit	24	5.8%
3	Mutual Funds	36	8.2%

Make an Investment

Account ID:

Select Policy:

Investment Amount:

Calculate Return

Customer ID:

Select Policy:

Principal Amount:

Return Estimate

Policy: 1

Principal: 90000.0

Estimated Return Amount: 160200.0

Monthly Payment: 13350.0

Lock-in Period: 12 months

[Back to Home -->](#)

Fig 8 Customer 1 Making an Investment in Fixed Deposit Policy

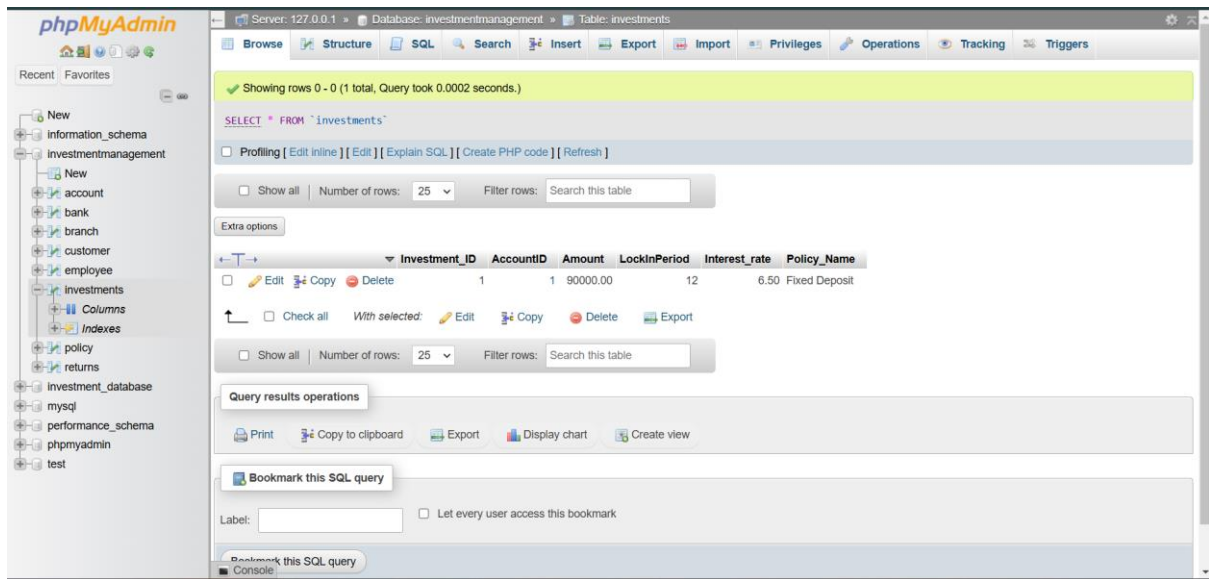


Fig 9 Updates in the Investments Table of Database

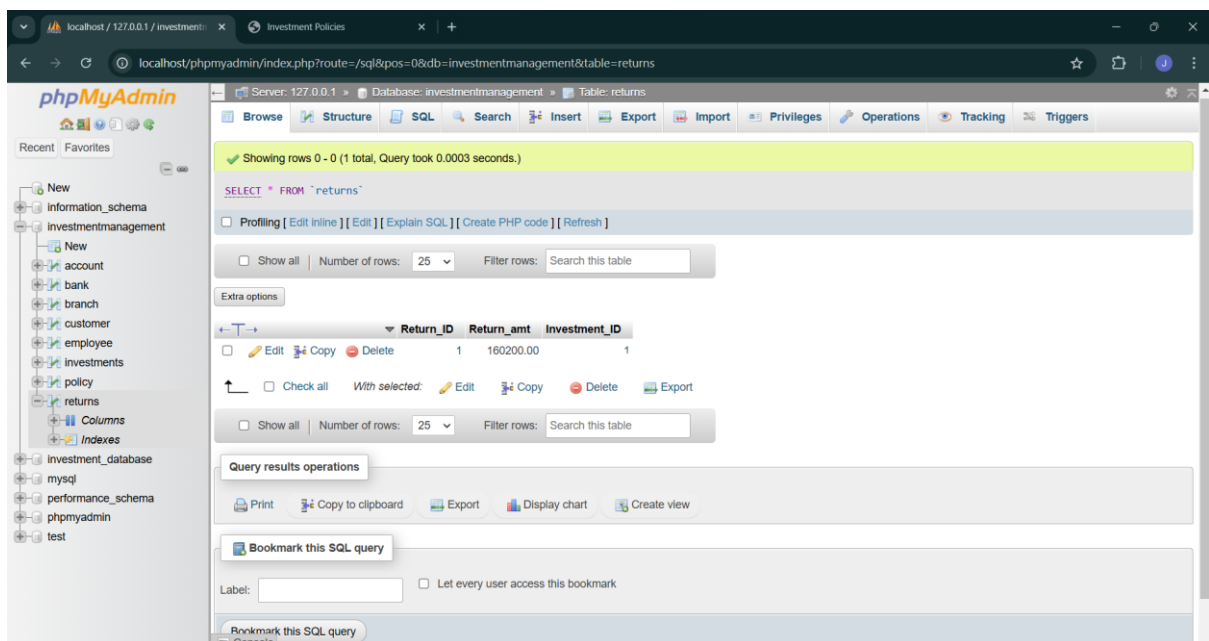


Fig 10 Updates in the Returns Table of Database

4.8.4 Viewing Customer Details

Finally, the system provides a "View Customer" feature to display the customer's investment history. The updated database tables, including customers, policies, and investments, are shown, confirming the accurate recording of all transactions and data updates. The demonstration highlights the seamless integration of the frontend and backend, along with real-time updates to the database, ensuring a smooth and reliable user experience.

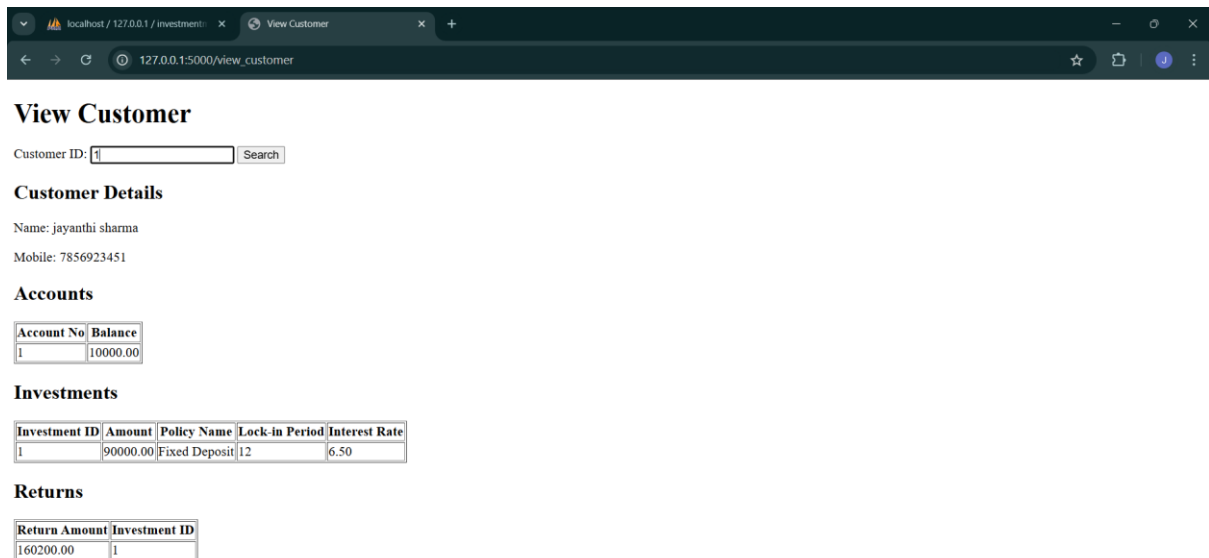


Fig 11 Viewing the Customer Details

5 Results and Discussions

The developed system successfully meets the project objectives by providing a seamless interface for managing customer investments and calculating returns on various banking schemes. The following results were observed:

1. **Database Operations:**
 - New customer records can be added efficiently, and the customers table is updated in real-time.
 - Investment policies chosen by customers are accurately recorded, with corresponding updates in related tables such as policies and investments.
2. **Interest Calculation Accuracy:**
 - The system calculates future values for various schemes like Fixed Deposits, Recurring Deposits, and Public Provident Fund accurately using the compound interest formula.
 - The results displayed for different investment scenarios matched the expected outputs, validating the correctness of the implemented logic.
3. **User Interaction:**
 - The user-friendly web interface enables smooth data entry, policy selection, and investment management.
 - The "View Customer" feature provides an overview of the customer's investment portfolio, ensuring transparency and ease of use.
4. **System Integration:**
 - The integration of Flask, MySQL, and phpMyAdmin facilitated seamless communication between the frontend and backend, ensuring data consistency.
 - The system handled concurrent updates efficiently, with no data loss or integrity issues.

The project highlights the importance of integrating a robust backend with a dynamic frontend for managing complex financial data. The use of XAMPP provided a convenient local environment for development and testing, while phpMyAdmin simplified database management. The system's ability to calculate compound interest for various schemes adds significant value, enabling users to make informed investment decisions.

Moreover, the modular design of the backend using Flask ensures scalability and maintainability, allowing for future enhancements such as adding more investment schemes or incorporating advanced analytics for financial forecasting. However, the system's reliance on a local server limits its accessibility. Deploying the system on a cloud-based platform could enhance its usability by enabling remote access. Additionally, implementing more robust error handling and security features, such as encryption for sensitive data, would further improve the system's reliability and security.

6 Conclusion

This project successfully demonstrates the development of a comprehensive system for managing customer investments and calculating returns on various banking schemes. By leveraging XAMPP for local server setup, MySQL for robust database management, and Flask for backend development, the system ensures efficient handling of user inputs and real-time data updates. The use of phpMyAdmin streamlines database operations, while the web-based interface enhances user experience by providing a simple and intuitive way to interact with the system.

The implementation of the `calculate_returns` allows accurate financial projections, empowering customers to make informed investment decisions. The project validates the functionality and reliability of the system through rigorous testing, confirming that it meets the requirements for accurate data management and calculation.

The project hence provides an effective solution for various banking schemes and their interest return estimation to customers. Using Banking Scheme, the system offers easy access to customers through bringing a sense of wisdom into investment by effective calculation of interest. The project is likely to improve customer satisfaction through making it easier to plan for long-term savings and investments in various financial schemes. This system can be further extended to include more sophisticated financial instruments and integrate with real-time banking data for further development.

7 Future Work

In future iterations, the system could be enhanced by deploying it on a cloud platform for broader accessibility, incorporating additional security measures, and expanding the range of supported financial products. Overall, the project serves as a solid foundation for developing scalable and user-friendly financial management applications.

8 References

- [1] Taipalus T. Database management system performance comparisons: A systematic literature review. *J Syst Softw.* 2024;208:111872. doi:10.1016/j.jss.2023.111872
- [2] Mohamed MA, Altrafi OG, Ismail MO. Relational vs. NoSQL databases: A survey. *Int J Comput Inf Technol.* 2014;3(3):598-601
- [3] Sene M, Moreaux P, Haddad S. Performance evaluation of distributed database - a banking system case study. *IFAC Proc Vol.* 2006;39(3):373–8. doi:10.3182/20060517-3-fr-2903.00200.
- [4] MD F, MD AA, Haque M, School of Computer Science and Engineering, Galgotias University, Greater Noida, India. Bank management system. *Int J Eng Res Comput Sci Eng.* 2021;8(8). Available from: https://www.technoarete.org/common_abstract/pdf/IJERCSE/v8/i8/Ext_82410.pdf
- [5] Studocu. Bank management system - Bank management system: A project report submitted by Harshit. Available from: <https://www.studocu.com/in/document/srm-institute-of-science-and-technology/dbms/bank-management-system/56848417>.

9 Journals

IEEE Transactions on Knowledge and Data Engineering - Impact Factor: 6.0+
This journal is known for high-impact work in data engineering, this journal could be ideal since our project focuses on data handling techniques in financial systems.

ACM Transactions on Database Systems - Impact Factor: ~2.6
This journal provides high-quality research on database management topics, including performance and security, which are crucial for financial DBMS applications.

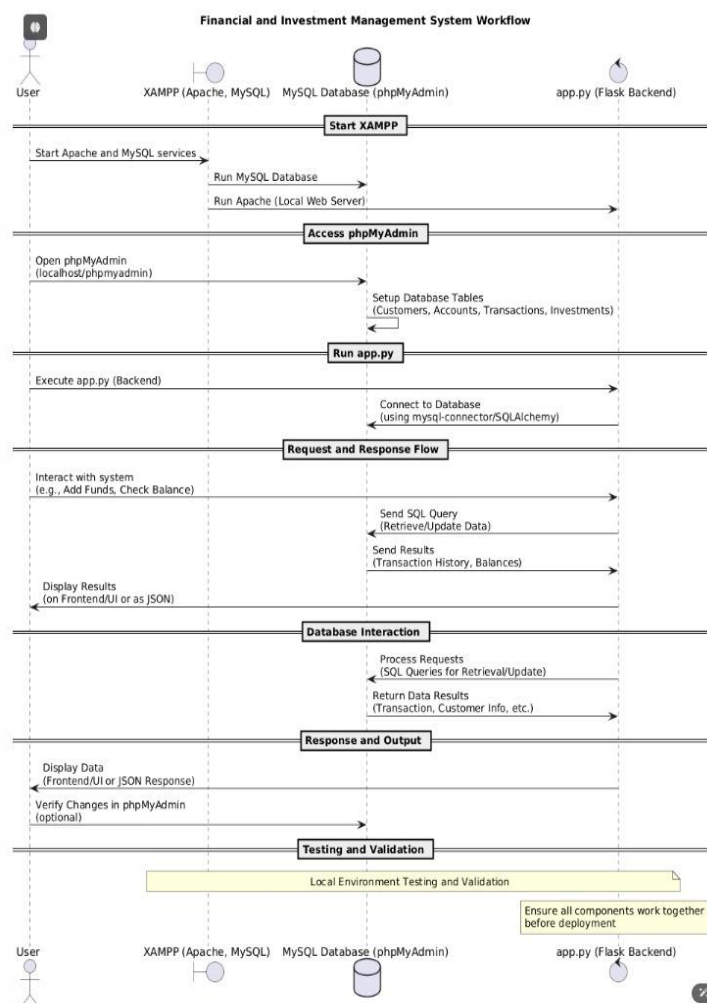
Information Systems Journal - Impact Factor: 4.0 (approximately)

This journal addresses information systems design and implementation, including real-time database systems in finance. This focus on secure and efficient data handling could support the real-time processing needs of our investment system.

Journal of Database Management (JDM) - Impact Factor: 1.3 (2023)
JDM covers advancements in database management, focusing on data architecture and efficiency, which aligns well with DBMS integration in financial applications. This journal's scope is relevant to the foundational database elements of our project, though its impact factor is moderate

Journal of Financial Data Science - Impact Factor: N/A (as of 2023)
This journal is highly relevant due to its focus on financial analytics, predictive modeling, and the role of data management systems within investment platforms. It aligns closely with the data-driven aspect of your system.

High-Level Diagram for our Project



Overview of the above system workflow

The financial and investment management system provides for the easy management of customers' accounts, investments, and any related financial information. The whole process of workflow entails engagement between a user and the backend, thereby encompassing some services, database operations, and responses in real-time. The high-level process involved includes elements such as XAMPP for local management of the server, phpMyAdmin for managing databases, and Flask as the backend framework.

1. Start XAMPP

The process starts by executing XAMPP, which boots the necessary services.

Apache: Used to activate the local web server.

MySQL: administers the MySQL database server. These services are integral because they help the system to run. One is able to access the local website once Apache is running and back-end connectivity of their data transactions with MySQL running.

2. Access phpMyAdmin

Once inside XAMPP, the user gets to phpMyAdmin by accessing “localhost/phpmyadmin”. In this section a user can create and manage the required tables that will be for the database of the system. These include Customer, Account, Investments, and Transactions. This step puts together the database schema and sets it up for interaction with the backend application.

3. Run app.py

The core of the backend is the file app.py, which is written within the Flask framework. When this file app.py is executed, it starts a local server and binds to this server via libraries such as mysql-connector, which then interact with the database. It means that when SQL operations are carried out on the MySQL database using app.py, it verifies all possible connections.

4. Request and Response Flow

Once app.py is running, the user can interact with the system through a frontend interface or API client. Users requests like checking balances and adding funds send requests over to app.py. App.py processes such actions by sending corresponding SQL queries for the database. For instance, whenever a request for a transaction history comes from a user, it requires the app.py to call transactions table, get the data, and return it to be used on the frontend.

5. Database Interaction

This is where the MySQL database will interact with the application in the file app.py. Real-time processing of SQL queries in terms of fetching, inserting, and updating will be done.

6. Response and Output

The system will generate a response based on the processed data. The results will be presented back to the user using the frontend UI or returned as JSON data in API-based usage.

7. Testing and Validation

Lastly, there is the local environment presented by XAMPP. The environment allows comprehensive testing and validation before deployment. In this case, the developer can test data transactions, monitor the UI, and ensure all components integrate well. This helps reduce possible errors in the operation of the system once deployed.

Code Details

App.py

Flask application managing an investment management system. It will connect to a MySQL database named InvestmentManagement and will be able to handle many routes for customer- and investment-related operations. Some of the routes include adding new customers, managing investment policies, calculating returns on investments, and viewing customer data, including their investments and accounts. Add customer route, where customer's information, like first name, last name, date of birth, etc. and will get added to the database through investment_policies route, which mainly controls two types of functionalities: Firstly, it will display previously defined investment policies, and, on the other hand, manage investments. Once investment forms are submitted, there will be a need to calculate returns and update account balances accordingly. Lastly, the view_customer route allows fetching user-related customer information, like accounts and investments from the database.

The calculate_return route will now be used to compute returns from an investment by a simple formula that considers principal, interest rate, and lock-in period. MySQL queries within the application guarantee data retrieval as well as updates in real-time. Along with these, the application offers simple error handling as well as form processing just to ensure that the application cooperates well with the user interface and the database. Actually, it's a pretty simple yet executable web application that is meant to assist in the monitoring of customers' investments and further computation of possible return on investment through several policies.

SQL QUERIES:

The SQL code provided defines the database structure for a **Financial and Investment Management System** designed to handle customer accounts, bank and branch details, employees, investment information, and returns. This structure aligns with the project's goal to manage and analyze investment data and customer interactions. Below is a breakdown of each table and its purpose:

Customer Table:

- Stores information about each customer, including their first name, last name, date of birth, age, address, and mobile number.
- **Primary Key:** Cust_ID - uniquely identifies each customer.
- This table is essential for tracking individual customers and linking them to their accounts.

Account Table:

- Represents the financial accounts owned by each customer, with details like the balance.
- **Primary Key:** Account_No - uniquely identifies each account.
- **Foreign Key:** Cust_ID - links each account to a specific customer, establishing a one-to-many relationship between Customer and Account (one customer can have multiple accounts).

Investments Table:

- Stores details about investments made by customers, such as the amount invested, lock-in period, interest rate, and inception date.
- **Primary Key:** Investment_ID - uniquely identifies each investment.
- **Foreign Key:** AccountID - links each investment to a specific account, establishing a one-to-many relationship between Account and Investments (one account can hold multiple investments).

Returns Table:

- Tracks returns generated from each investment, with details like the return amount and return date.
- **Primary Key:** Return_ID - uniquely identifies each return record.
- **Foreign Key:** Investment_ID - links each return record to a specific investment, establishing a one-to-many relationship between Investments and Returns (one investment can have multiple returns over time).

Policy Table:

- Defines the types of investment policies available, such as Fixed Deposit, Recurring Deposit, and Mutual Funds, with details about policy duration and interest rate.
- **Primary Key:** Policy_ID - uniquely identifies each policy type.
- This table is used to categorize investments under specific policies, allowing for flexible investment options.

Biography:

1 **Riddhi Bandyopadhyay**- A passionate and motivated computer science student who concentrates on DBMS, software development, algorithms, and emerging technologies, with a desire to implement knowledge into real-life problem-solving.

Role: DBA, Back-end developer and documentation, Flask code handling



2 **Sinchan Shetty** - Pursuing Computer Science (Pre-Final) and an AI-enthusiast with growing interest in DBMS and related tech. Strong background in web development, thus allowing to create dynamic and interactive applications.

Role: DBA, Front-end Developer, documentation, phpMyAdmin Integration



3 **Jayanthi Sharma** - A dedicated third-year computer science student with a keen interest in database management systems (DBMS) and software development. Proficient in C and C++, with a strong foundation in problem-solving and algorithms. Passionate about leveraging technology to create real-world solutions.

Role: DBA, Full-stack developer and documentation, XAMPP environment handling

My major contribution to this project was design and implementation of backend using flask which was focused on routing and dynamic content rendering. I also managed relational databases using XAMPP for seamless integration and data storage. other than these, I contributed to table creation and query handling.



Registration Details

1. Jayanthi Sharma - 22BCE1758
2. Riddhi Bandyopadhyay- 22BCE1068
3. Sinchan Shetty- 22BCE5238