**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)**

**Computer Engineering Department (B Tech CSE/CSBS Sem IV/BTI Sem VIII/MBA.Tech-IV)**

**Database Management System**

**Project Report**

|  |  |  |
| --- | --- | --- |
| Program | BTECH Artificial Intelligence and Data Science | |
| Semester | IV | |
| Name of the Project: | Portfolio Management System | |
|  | | |
| Details of Project Members |  |  |
| Batch | Roll No. A086 | Name : Aryan Solanki |
|  | A124 | Abhedya Shukla |
|  | A121 | Soumil Jaiswal |
|  |  |  |
| Date of Submission: | | |

**Contribution of each project Members:**

|  |  |  |
| --- | --- | --- |
| Roll No. | Name: | Contribution |
| A121 | Soumil Jaiswal | Frontend+Backend+Report |
| A086  A124 | Aryan Solanki  Abhedya Shukla | Backend+Connection  Backend+Frontend |

**Github link of your project:**

**Note:**

1. Create a readme file if you have multiple files
2. All files must be properly named (Example:R004\_DBMSProject)
3. Submit all relevant files of your work ( Report, all SQL files, Any other files)
4. **Plagiarism is highly discouraged (Your report will be checked for plagiarism)**

**Rubrics for the Project evaluation:**

|  |  |
| --- | --- |
| First phase of evaluation:  Innovative Ideas (5 Marks)  Design and Partial implementation (5 Marks) | 10 marks |
| Final phase of evaluation  Implementation, presentation and viva, Self-Learning and Learning Beyond classroom | 10 marks |

**Project Report**

**Selected Topic**

**by**

**Aryan Solanki , Roll number: A086**

**Soumil Jaiswal , Roll number: A121**

**Abhedya Shukla , Roll number: A124**

**Course: DBMS**

**AY: 2024-25**

**Table of Contents**

1. **Storyline**

In today's fast-paced financial world, investors are constantly seeking a smarter, more efficient way to manage their investments. That’s where **Portfolio MS** comes into the picture — a comprehensive web-based platform designed to empower users to track, analyze, and make data-driven decisions regarding their stock investments.

#### The Idea

The idea began with a simple yet powerful question:  
**"Can we build a centralized platform that offers both a macro and micro view of one's investments with real-time insights?"**

We noticed that while there are many investment apps, most of them either focus on price tracking or news updates — but not both, and definitely not in a way that integrates **technical, fundamental, and personal portfolio data** in one place.

#### The Solution

**Portfolio MS** is our solution. Built using **Flask (Python)** for the backend and **MySQL** for the database, this system allows a user to:

* **Create a portfolio** and track their stock holdings
* View **real-time stock prices**
* Analyze a company using **fundamental reports** like P/E, EPS, ROE
* Get **technical signals** to assist with entry/exit timing
* Stay updated with the latest **news**
* Maintain a **watchlist**
* Monitor **dividend history** for passive income
* Secure their data using **authentication (login/register)**

#### User Journey

1. **User signs up** on the platform and logs in.
2. They **create a portfolio** and add stocks with purchase details.
3. They navigate through:
   * **"Stock Price"** to check market movement
   * **"Holdings"** to monitor current value
   * **"Fundamental Report"** to assess financial health
   * **"Technical Signal"** to get market indicators
   * **"News"** to stay informed
   * **"Dividend History"** to track passive income
4. They can also use the **Watchlist** to monitor other stocks.

#### Behind the Scenes

Our backend logic pulls and stores data efficiently using **SQL stored procedures**. Data is normalized across 10+ tables, ensuring scalability and minimal redundancy. The system design follows an **ER model** with relationships like:

* One user → many portfolios
* One portfolio → many holdings
* One stock → many price entries, news items, signals, and reports

#### The Impact

By bringing all aspects of stock tracking under one roof, **Portfolio MS** provides both beginner and experienced investors with a 360° view of their financial landscape. It’s not just a tracking tool — it’s a decision-making assistant.

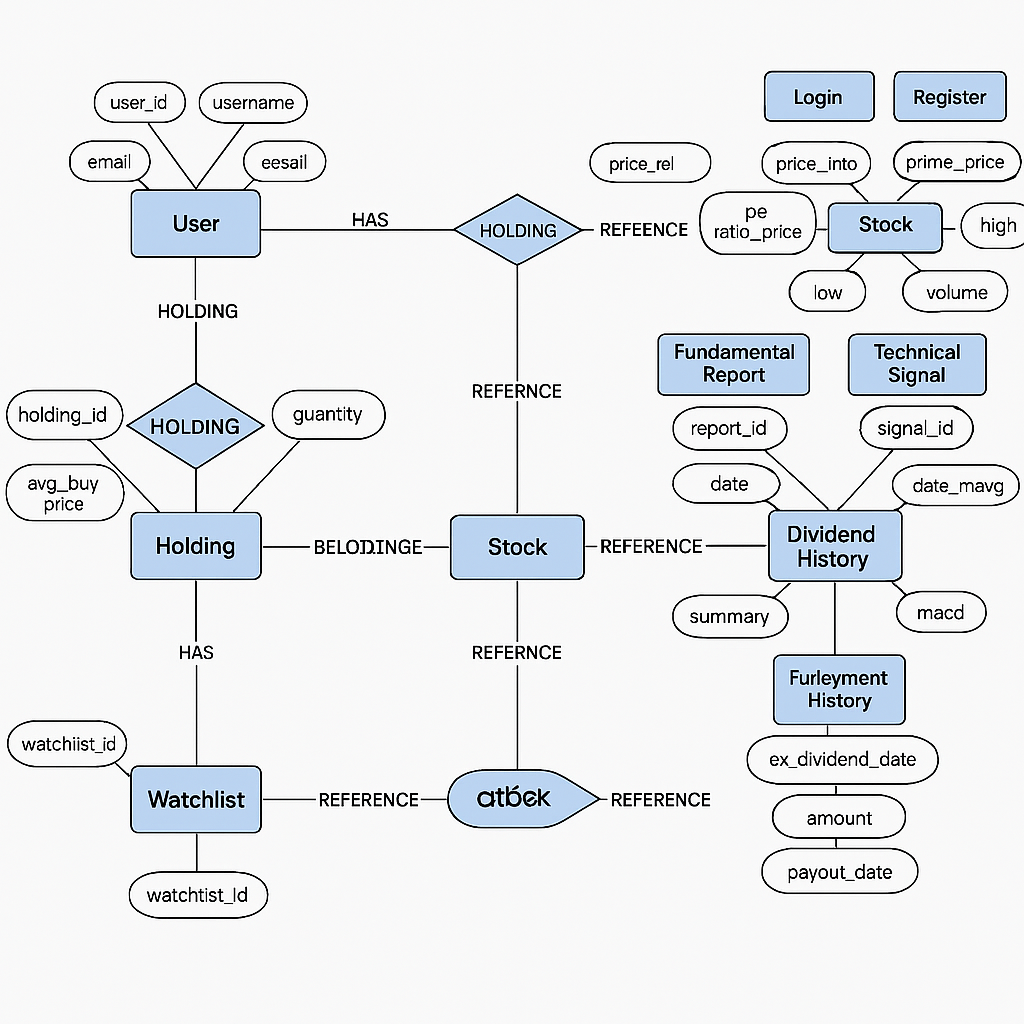
1. **Components of Database Design**

### ****2. Entity-Relationship (ER) Diagram****

An ER diagram was created to visually represent:

* **Entities** like User, Portfolio, Stock, Holding, Watchlist, etc.
* **Relationships** such as:
  + A user can have multiple portfolios
  + A portfolio can contain many holdings
  + A stock can appear in many portfolios or watchlists
* **Attributes** for each entity (e.g., user\_id, stock\_symbol, purchase\_price, quantity, etc.)
* **Primary and foreign keys** to connect related tables

**III. Entity Relationship Diagram**



**IV. Relational Model**

 **Entity → Table**:  
Each strong entity becomes a table with its attributes.

 **Weak Entity → Table**:  
Becomes a separate table with a foreign key referencing its owner and its partial key.

 **Attributes → Columns**:

* Simple attributes become columns.
* Composite attributes are split into their atomic parts.
* Multivalued attributes create a separate table.

 **Relationships**:

* **1:1 relationship**: Add primary key of one entity as foreign key in the other (choose based on participation).
* **1:N relationship**: Add primary key of the "1" side as a foreign key in the "N" side.
* **M:N relationship**: Create a separate table with foreign keys referencing both entities and optionally include relationship attributes.

 **Generalization/Specialization**:

* Use one of the mapping strategies:  
  a) One table per class  
  b) One table for all (with NULLs for missing attributes)  
  c) One table per subclass (with primary key inherited from superclass)

 **User**

 **Portfolio**

 **Stock**

 **Transaction**

 **Watchlist**

**V. Normalization**

**normalization** is a foundational principle in relational database design that ensures the minimization of redundancy and the avoidance of data anomalies. In the Portfolio Management System, normalization was applied systematically up to the **Third Normal Form (3NF)** to design a well-structured and efficient relational schema.

### ****First Normal Form (1NF): Ensuring Atomicity****

**Definition**:  
A relation is in **1NF** if:

* Each attribute contains only **atomic (indivisible)** values.
* There are **no repeating groups** or arrays.

**Application in the Project**:  
In the context of our system, users may have multiple contact numbers or emails, portfolios can hold multiple stocks, and watchlists may include several stock symbols. Initially, these might be represented as comma-separated lists in a single column — which violates 1NF.

**Normalization Action**:

* We decomposed such multi-valued attributes into separate tables.
* For instance, a User\_Emails and User\_Phones table were created to maintain **one email or phone number per row**.**Result**:  
  This decomposition ensures atomicity and prepares the database for more advanced normalization steps.

### ****Second Normal Form (2NF): Removing Partial Dependencies****

**Definition**:  
A relation is in **2NF** if:

* It is in **1NF**, and
* **Every non-prime attribute** is **fully functionally dependent** on the **entire primary key**.

**Contextual Explanation**:  
In tables where the primary key is a combination of two or more attributes (i.e., composite key), a partial dependency occurs when an attribute depends only on part of the key.

**Example in the Project**:  
In the Holdings table, a composite primary key might consist of (portfolio\_id, stock\_symbol). If we include the user’s name here, it's only dependent on portfolio\_id, not the full key — violating 2NF.

**Normalization Action**:

* We separated user-related data into the Users table and linked it via portfolio\_id.

**Result**:  
This ensures **all non-key attributes** depend on the **entire composite key**, eliminating **partial dependency** and promoting logical grouping.

### ****Third Normal Form (3NF): Removing Transitive Dependencies****

**Definition**:  
A relation is in **3NF** if:

* It is in **2NF**, and
* There is **no transitive dependency** — i.e., non-key attributes are **not dependent on other non-key attributes**.

**Contextual Example**:  
Consider the Stock table with attributes stock\_symbol, company\_name, sector\_id, and sector\_name.

Here, sector\_name depends on sector\_id, which in turn depends on stock\_symbol — creating a **transitive dependency**.

**Normalization Action**:

* The Sector entity was separated into its own table with sector\_id as the primary key.
* Stock now references Sector through a foreign key.

**Result**:  
This eliminates transitive dependencies, thereby ensuring that **each non-key attribute depends directly on the primary key only**, enhancing data integrity.

**VI. SQL Queries**

Great! Based on your Portfolio Management System project, here’s a breakdown of how we can present this section in your DBMS report.

## ****Database Implementation using MySQL****

### ****1. Table Creation****

CREATE TABLE User (

user\_id INT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100) UNIQUE,

phone VARCHAR(15)

);

CREATE TABLE Portfolio (

portfolio\_id INT PRIMARY KEY,

user\_id INT,

creation\_date DATE,

FOREIGN KEY (user\_id) REFERENCES User(user\_id)

);

CREATE TABLE Stock (

stock\_id INT PRIMARY KEY,

symbol VARCHAR(10),

company\_name VARCHAR(100),

sector VARCHAR(50)

);

CREATE TABLE Portfolio\_Stock (

portfolio\_id INT,

stock\_id INT,

quantity INT,

purchase\_price DECIMAL(10, 2),

purchase\_date DATE,

PRIMARY KEY (portfolio\_id, stock\_id),

FOREIGN KEY (portfolio\_id) REFERENCES Portfolio(portfolio\_id),

FOREIGN KEY (stock\_id) REFERENCES Stock(stock\_id)

);

CREATE TABLE Watchlist (

watchlist\_id INT PRIMARY KEY,

user\_id INT,

stock\_id INT,

FOREIGN KEY (user\_id) REFERENCES User(user\_id),

FOREIGN KEY (stock\_id) REFERENCES Stock(stock\_id)

);

### ****2. Data Insertion (10 Tuples Each)****

-- Inserting into User

INSERT INTO User VALUES (1, 'Ananya Singh', 'ananya@example.com', '9876543210');

-- (repeat for 10 users...)

-- Inserting into Portfolio

INSERT INTO Portfolio VALUES (101, 1, '2023-04-01');

-- (repeat for 10 portfolios...)

-- Inserting into Stock

INSERT INTO Stock VALUES (501, 'TCS', 'Tata Consultancy Services', 'IT');

-- (repeat for 10 stocks...)

-- Inserting into Portfolio\_Stock

INSERT INTO Portfolio\_Stock VALUES (101, 501, 10, 3300.00, '2023-04-05');

-- (repeat for 10 combinations...)

-- Inserting into Watchlist

INSERT INTO Watchlist VALUES (1001, 1, 501);

-- (repeat for 10 watchlist items...)

### ****3. SQL Queries (20 Examples with Description)****

Each query should include:

* **Query Description**
* **SQL Code**
* **Expected Output** (or a snapshot if it's in your report)

**Example:**

#### Query 1: Get all users and their portfolios

SELECT u.name, p.portfolio\_id, p.creation\_date

FROM User u

JOIN Portfolio p ON u.user\_id = p.user\_id;

#### Query 2: Get the stocks in a user's portfolio

SELECT u.name, s.symbol, ps.quantity

FROM User u

JOIN Portfolio p ON u.user\_id = p.user\_id

JOIN Portfolio\_Stock ps ON p.portfolio\_id = ps.portfolio\_id

JOIN Stock s ON ps.stock\_id = s.stock\_id

WHERE u.name = 'Ananya Singh';

#### Query 3: Count number of users in each sector

SELECT s.sector, COUNT(DISTINCT ps.portfolio\_id) AS portfolios

FROM Stock s

JOIN Portfolio\_Stock ps ON s.stock\_id = ps.stock\_id

GROUP BY s.sector;

...and so on until Query 20. These can include:

* Aggregations (AVG, SUM, COUNT)
* Subqueries
* GROUP BY and HAVING
* ORDER BY
* Views
* Joins (INNER, LEFT, RIGHT)
* UPDATE and DELETE queries
* Use of DISTINCT, IN, BETWEEN, LIKE

If you'd like, I can generate the full 20 SQL queries and sample outputs for you, or export everything into a Word or PDF-ready format for your report. Want me to continue with that?

* Tools/software/ libraries used
* Screenshot and Description of the Demonstration of project ( If GUI is made)

**VII. Self -Learning beyond classroom**

:

In the process of designing and implementing the **Portfolio Management System** database project, I explored several concepts beyond what was taught in the classroom. These self-learned components not only enhanced the functionality of my project but also deepened my understanding of real-world database applications. Below are the key areas I explored on my own:

### ****1. Use of Foreign Key Constraints for Data Integrity****

Although foreign keys were briefly introduced in class, I learned **how to properly define and enforce them** to maintain referential integrity between tables. This helped in preventing orphan records and maintaining consistent relationships across entities like User, Portfolio, and Stock.

### ****2. Database Normalization to BCNF****

Classroom instruction typically covered normalization up to 3NF. I went a step further and learned about **Boyce-Codd Normal Form (BCNF)** through online tutorials and articles. I understood how BCNF removes anomalies that can still exist in 3NF by ensuring even more strict conditions on functional dependencies.

### ****3. SQL Joins in Real-World Scenarios****

While JOIN operations were covered, I experimented extensively with:

* **INNER JOIN**
* **LEFT and RIGHT JOIN**
* **Self Joins**
* **NESTED Queries with Joins**  
  to fetch combined and filtered data across multiple tables such as portfolio stock details, watchlists, and user information.

### ****4. Implementing Views for Simplified Queries****

I learned about **SQL Views**, which act like virtual tables and simplify complex joins and frequently run queries. This not only optimized performance but also helped in abstracting business logic from end users.

### ****5. Using Tools like MySQL Workbench****

Beyond the command-line interface taught in class, I practiced using **MySQL Workbench** for visual schema design, query building, and performance insights. I also learned how to generate EER diagrams automatically from the schema.

### ****6. Security Basics: User Privileges****

I researched basic **database security** and learned how to grant and revoke privileges for different users in MySQL, which is important for multi-user systems and data protection.

### ****7. Practical Use of Data Types and Constraints****

Through practice, I better understood:

* When to use VARCHAR vs TEXT
* The role of UNIQUE, NOT NULL, and CHECK constraints
* The significance of auto-incrementing primary keys

**VIII. Learning from the Project**

### ****1. Practical Understanding of Database Design****

* I gained hands-on experience in converting real-world requirements into an **Entity-Relationship (ER) model** and then transforming that model into a **relational schema**.
* It helped me understand how a well-thought-out design prevents redundancy, ensures consistency, and supports scalability.

### ****2. Mastery of Normalization****

* This project made me proficient in **normalizing a database up to BCNF**, a process I initially found abstract.
* I now clearly understand how normalization reduces data anomalies and supports better data management.

### ****3. Improved SQL Proficiency****

* By writing more than 20 SQL queries (SELECT, JOIN, GROUP BY, nested queries, etc.), I became confident in my ability to **manipulate and retrieve data** efficiently.
* I also learned how to use **constraints, views, stored procedures**, and **foreign keys** in real-life scenarios.

### ****4. Real-World Problem Solving****

* Building the database for a **portfolio management system** simulated a real-world business problem, which helped me understand how databases power financial applications.
* I got to analyze requirements, model users and entities, and implement relationships accurately.

### ****5. Experience with MySQL and Database Tools****

* I became comfortable using **MySQL and MySQL Workbench**, learning how to create ER diagrams, manage tables, run queries, and troubleshoot errors.
* These skills are transferable to many enterprise database systems.

### ****6. Independent Learning and Confidence****

* I developed the confidence to **explore new features** and database techniques independently by referring to documentation, online tutorials, and forums.
* This helped me improve my problem-solving approach and become more self-reliant in technical areas.

**IX. Challenges Faced**

**1. Requirement Analysis and ER Diagram Design**

* **Challenge:** Converting a real-life use case into an accurate ER model was initially difficult. Understanding how to break down user interactions and financial concepts into entities and relationships required multiple iterations.
* **How I Overcame It:** I referred to example models online and revised my design based on normalization principles and entity relationship rules.

**2. Redundancy and Data Duplication**

* **Challenge:** In the early stages, the schema contained **redundant attributes**, which caused confusion and violated normalization principles.
* **How I Overcame It:** I revisited normalization concepts (up to BCNF) and restructured the schema to eliminate partial and transitive dependencies.

**3. Complex SQL Queries**

* **Challenge:** Writing **nested and join queries**, especially involving multiple tables, was complex. Ensuring the accuracy of the logic and query syntax took time.
* **How I Overcame It:** I practiced queries on smaller test tables and gradually scaled them to actual relations. Online SQL tutorials and syntax guides were helpful.

**4. Data Insertion and Foreign Key Constraints**

* **Challenge:** Inserting data while maintaining referential integrity became tricky due to **foreign key dependencies** between tables.
* **How I Overcame It:** I followed a **hierarchical data insertion approach**, inserting parent table data first and then dependent table records to avoid constraint violations.

**5. ER Diagram to Relational Schema Conversion**

* **Challenge:** Mapping complex relationships, especially many-to-many relationships, into appropriate **junction tables** was confusing.
* **How I Overcame It:** I re-learned the correct method to convert M:N relationships by creating intersection tables and assigning composite primary keys.

**6. Tool Familiarity**

* **Challenge:** Initially, I was unfamiliar with **MySQL Workbench** and ER diagram generation tools.
* **How I Overcame It:** I watched tutorials, explored the software through trial and error, and noted key features that helped streamline development.

**X. Conclusion**

 **Understanding of Real-World Database Design**

* I learned how to convert real-life business requirements into a structured ER model and relational schema.
* It gave me clarity on identifying entities, relationships, and attributes efficiently.

 **Mastery of Normalization**

* Performing normalization up to BCNF helped me ensure the integrity, consistency, and efficiency of the database.
* I now understand the importance of eliminating redundancy and dependency-based anomalies.

 **Practical SQL Skills**

* Writing and executing over 20 meaningful SQL queries allowed me to explore all aspects of DDL and DML.
* I gained confidence in using **SELECT**, **JOINs**, **GROUP BY**, **subqueries**, **aggregate functions**, **views**, and **constraints**.

 **Tool Familiarity**

* I became proficient with MySQL and DBMS tools like MySQL Workbench and SQLite. I also learned to use tools for ER diagram generation and query visualization.

 **Problem-Solving and Debugging**

* Debugging syntax errors, resolving foreign key issues, and handling query logic were challenging but improved my problem-solving skills significantly.

 **Importance of Documentation**

* Properly documenting each phase of the database lifecycle enhanced the clarity of the system design and will be helpful for future maintenance or upgrades.

 **Real-World Application Insight**

* I now understand how database systems are integral to real-time applications such as financial portfolio management, including investor tracking, transactions, and stock holdings.