Established – 1961 Subject: OSDBMS

SEVA SADAN’S

R. K. TALREJA COLLEGE

OF

ARTS, SCIENCE & COMMERCE ULHASNAGAR – 421 003

CERTIFICATE

This is to certify that Mr <u>IRFAN KAYUM MANIYAR</u> of S.Y. Information Technology (SYIT) Roll No. <u>2542023</u> has satisfactorily completed the Open
Source DataBase Management System Mini Project entitled
<u>Stock Transaction Record .</u> During the academic year 2025 – 2026, as a part of the practical requirement. The project work is found to be satisfactory and is approved for submission.

PROF. INCHARGE **HEAD OF DEPT**

INDEX

CHAPTER 1: INTRODUCTION

A database system is an organized collection of data that allows efficient storage, retrieval, and management of information. It plays an important role in handling large volumes of structured data in various fields such as banking, healthcare, education, and finance. Database systems reduce manual effort, avoid duplication of data, and help maintain accuracy and consistency.

The Stock Transaction Record System is a database-based system designed to manage stock trading information such as user details, stock information, buy and sell transactions, and portfolio records. In many small organizations or academic environments, stock transaction data is maintained manually using spreadsheets or paper records. This approach often leads to errors, missing records, and difficulty in tracking stock ownership.

This project provides a structured relational database solution using MySQL to store and manage stock transaction data efficiently. All stock-related information is stored in multiple related tables such as User, Stock, Transaction, and Portfolio. These tables are connected using primary and foreign keys to maintain proper relationships.

Database constraints such as PRIMARY KEY, FOREIGN KEY, NOT NULL, and UNIQUE are applied to prevent duplicate records and ensure data integrity. Transaction management is also used to guarantee that important operations such as buying or selling stocks are completed correctly without partial updates.

MySQL is chosen as the database management system for this project because it is open-source, reliable, and widely used. It supports standard SQL queries, transaction control, and security features. It is also easy to install and suitable for academic projects.

This project demonstrates how a database system can be used to manage stock transaction records in a simple and effective manner while giving practical exposure to core DBMS concepts.

CHAPTER 2: PROBLEM DEFINITION

Many small organizations and students maintain stock transaction data using manual methods or unstructured files such as spreadsheets. These methods are inefficient and become difficult to manage as the volume of data increases.

Due to the absence of a proper database-driven system, the following problems arise:

- Data Redundancy – Same stock or user data is stored multiple times.
- Data Inconsistency – Updates made in one place are not reflected everywhere.
- Duplicate Records – Multiple entries for the same user or stock may exist.

- Incorrect Portfolio Calculation – Manual calculations may produce wrong results.
- Partial Data Updates – Buy or sell operations may be recorded incompletely.
- Poor Data Integrity – Transactions may exist without valid user or stock references.
- Lack of Security – Sensitive trading data is not protected properly.
- Difficulty in Tracking Transactions – Transaction history becomes hard to maintain.
- Slow Data Retrieval – Searching records takes more time.
- Limited Reporting Capability – Reports cannot be generated easily.
- Risk of Data Loss – Data can be lost due to file corruption or human error.

CHAPTER 3: OBJECTIVES OF THE PROJECT

The main objectives of the Stock Transaction Recording System are as follows:

To design and develop a structured relational database using MySQL for managing stock transaction records.

- To design and develop a structured database for managing stock transaction records using MySQL.
- To store user, stock, transaction, and portfolio data efficiently.

- To implement SQL queries for inserting, updating, deleting, and retrieving stock data.

- To ensure data accuracy and consistency using database constraints.

- To prevent duplicate and conflicting records in the database.

- To maintain portfolio balance using transaction management.

- To retrieve meaningful reports using SQL joins and queries.

- To gain practical experience with MySQL as an open-source DBMS

- To understand real-world application of relational database concept

CHAPTER 4: SCOPE OF THE PROJECT

The scope of the Stock Transaction Recording System defines the boundaries and limitations of the project.

This project focuses on building a database system to store and manage stock transaction data efficiently. It is designed mainly for academic and learning purposes.

4.1 Application Areas

The system can be used in the following areas:

- Educational institutions for teaching database concepts
- Small businesses for maintaining stock trading records
- Training programs related to finance and databases
- Academic mini-projects

4.2 Users of the System

The main users of the system are:

- Administrator – Manages database and stock records.
- Data Entry Operator – Records transactions and stock data.
- Students – Use the system for learning DBMS concepts.
- Small Business Users – Can use it to maintain stock records.

4.4 Future Expandability

The system can be extended in future to include:

- Web-based interface
- Real-time stock price updates
- Advanced reporting and analytics
- User authentication and dashboards

CHAPTER 5: REQUIREMENT SPECIFICATION

This chapter describes the hardware and software requirements needed to implement the Stock Transaction Recording System.

5.1 Hardware Requirements :

Component	Specification
Processor	Intel i3 or above
RAM	Minimum 4 GB
Hard Disk	Minimum 20 GB free space

Input Devices	Keyboard, Mouse
Output Devices	Monitor

5.2 Software Requirements :

Software	Purpose
MySQL Server	Database management
MySQL Workbench	SQL query execution and database design
Operating System	Windows / Linux
SQL	Query language
MS Word	Documentation preparation

CHAPTER 6: SYSTEM DESIGN

System design describes how the database system works conceptually.

6.1 System Architecture (Conceptual):

The system consists of the following components:

- User
- MySQL Workbench (Interface)
- MySQL Server (Database Engine)

Working:

1. The user enters SQL queries using MySQL Workbench.
2. The queries are sent to the MySQL Server.
3. The server processes the queries.
4. Results are returned to the user.
5. Data is stored in relational tables.

6.2 Data Flow Description

User enters stock or transaction data.

Data is validated by constraints.

Data is stored in tables.

Reports are generated using SELECT queries.

6.3Process Flow (Textual)

User → SQL Query → MySQL Server → Database Tables → Result Output
PROJECT SCHEDULE (GANTT CHART)

A Gantt Chart is used to represent the project timeline and activities.

Table 6.1: Project Schedule (Gantt Chart) Activity

Table 6.1: Project Schedule (Gantt Chart)

Activity	Week 1	Week 2	Week 3	Week 4
Topic Selection	✓			
Requirement Analysis	✓	✓		

Database Design		✓	✓	
Table Creation			✓	
SQL Implementation			✓	✓
Testing				✓
Documentation	✓	✓	✓	✓

CHAPTER 7: DATABASE DESIGN

Database design is one of the most important phases of this project. It defines how data is structured, stored, and related to each other. A welldesigned database reduces redundancy and improves data integrity.

The Stock Transaction Recording System uses a relational database model. The database is designed using MySQL and consists of four main tables: User, Stock, Transaction, and Portfolio.

7.1 Entity Description

1. User Entity Attributes:

    1. user_id
    2. name
    3. email
    4. phone 2. Stock Entity

Attributes: stock_id

1. stock_name

2. symbol
3. price

3. Transaction Entity

1. transaction_id
2. user_id
3. stock_id
4. transaction_type
5. quantity
6. transaction_date

4.. Portfolio Entity

Attributes:

1. portfolio_id
2. user_id
3. stock_id
4. total_quantity 7.2 Table Structure

Table 7.1: User Table

Field Name	Data Type	Description
User_id	INT (PK)	Unique ID of user
Name	VARCHAR(50)	Name of user
Email	VARCHAR(50)	Email of user
Phone	VARCHAR(15)	Contact number

Table 7.2: Stock Table

Field Name	Data Type	Description
Stock_id	INT (PK)	Unique ID of stock
Stock_name	VARCHAR(50)	Name of stock
Symbol	VARCHAR(10)	Stock symbol
Price	DECIMAL(10,2)	Stock price

Table 7.3: Transaction Table

Field Name	Data Type	Description
Transaction_id	INT (PK)	Unique ID of transaction
User_id	INT (FK)	References User(user_id)
Stock_id	INT (FK)	References Stock(stock_id)
Transaction_type	ENUM(“BUY”,”SELL”)	Buy or Sell
Quantity	INT	Number of shares
Transaction_date	DATE	Date of transaction

Table 7.4: Portfolio Table

Field Name	Data Type	Description
Portfolio_id	INT (PK)	Unique ID of portfolio
User_id	INT (FK)	References User(user_id)
Stock_id	INT (FK)	References Stock(stock_id)
Total_quantity	INT	Total shares owned

7.4 Constraints Used Constraints are used to maintain data accuracy and consistency.

Primary Key (PK)

Used to uniquely identify each record in a table.

Example: user_id in User table.

Foreign Key (FK)

Used to establish relationship between tables.

Example: user_id in Transaction references User(user_id).

NOT NULL

Ensures that a column cannot have empty values.

Example: stock_name, transaction_type.

UNIQUE

Ensures that values are not duplicated.

Example: email in User table.

ENUM

Restricts role and status to predefined values

Example: Transaction_type in Transaction table

7.5 Transaction for Buy/Sell Operation

In this system, every buy or sell activity is treated as a database transaction.
When a user buys a stock:

A new record is inserted into the Transaction table. The Portfolio table is updated by increasing the stock quantity.

When a user sells a stock:

A new record is inserted into the Transaction table with type SELL. The Portfolio quantity is reduced accordingly.
Both operations are executed together as a single transaction to ensure data accuracy.

7.6 Balance / Portfolio Consistency

Portfolio consistency means that the quantity stored in the Portfolio table must always match the total quantity derived from buy and sell transactions.

This is ensured by:

- Using constraints

- Updating portfolio only after valid transactions

- Preventing selling more stocks than available If a user attempts to sell more stocks than present in the portfolio, the operation is rejected, ensuring that incorrect data is not stored.

7.7 Rollback Example

If an error occurs during a buy or sell operation (such as an invalid stock ID or system failure), the transaction is rolled back.
Rollback cancels all changes made during the operation. This ensures that:

- No partial data is stored

- Portfolio and transaction tables remain consistent

- Database integrity is preserved

Thus, either all steps succeed or none are applied.

CHAPTER 8: UML DIAGRAMS

Unified Modeling Language (UML) diagrams are used to represent the structure and behavior of the system in a graphical form. These diagrams help in understanding how different components of the system interact with each other and how data flows through the system.

For the Stock Transaction Record System, the following UML diagrams are used:

- ER Diagram

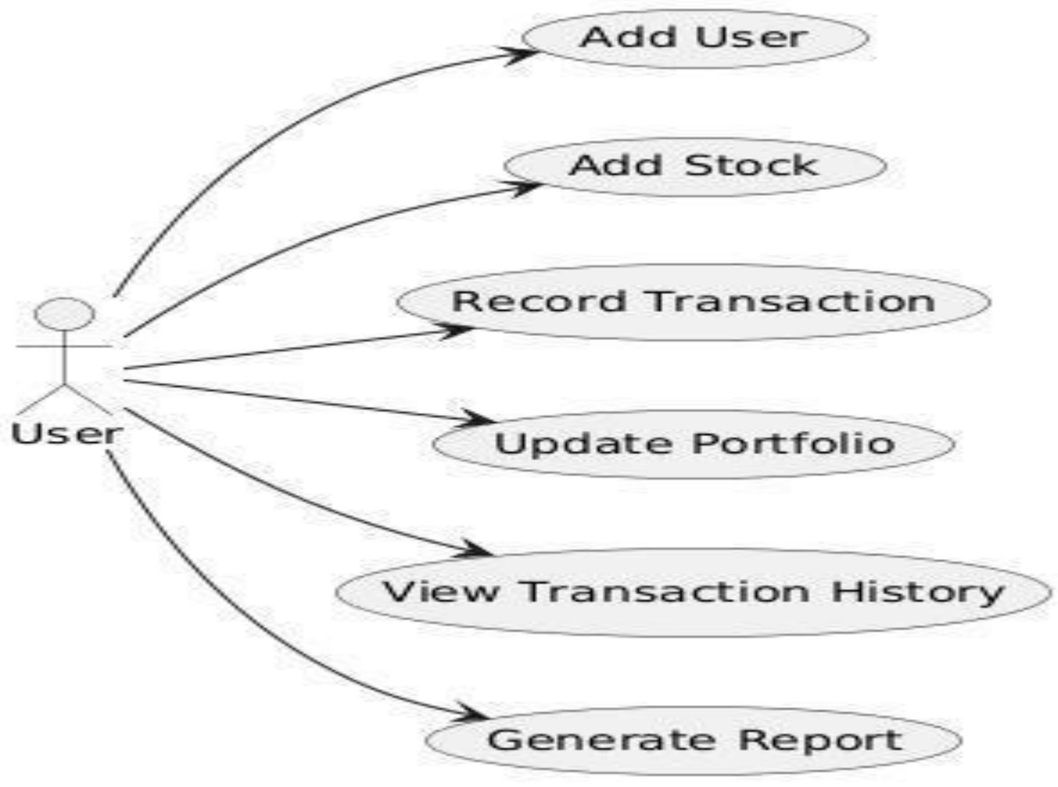
- Use Case Diagram

- Sequence Diagram

- Activity Diagram

These diagrams provide a clear visualization of the database structure, user interactions, and transaction flow.

- **USE CASE DIAGRAM:**

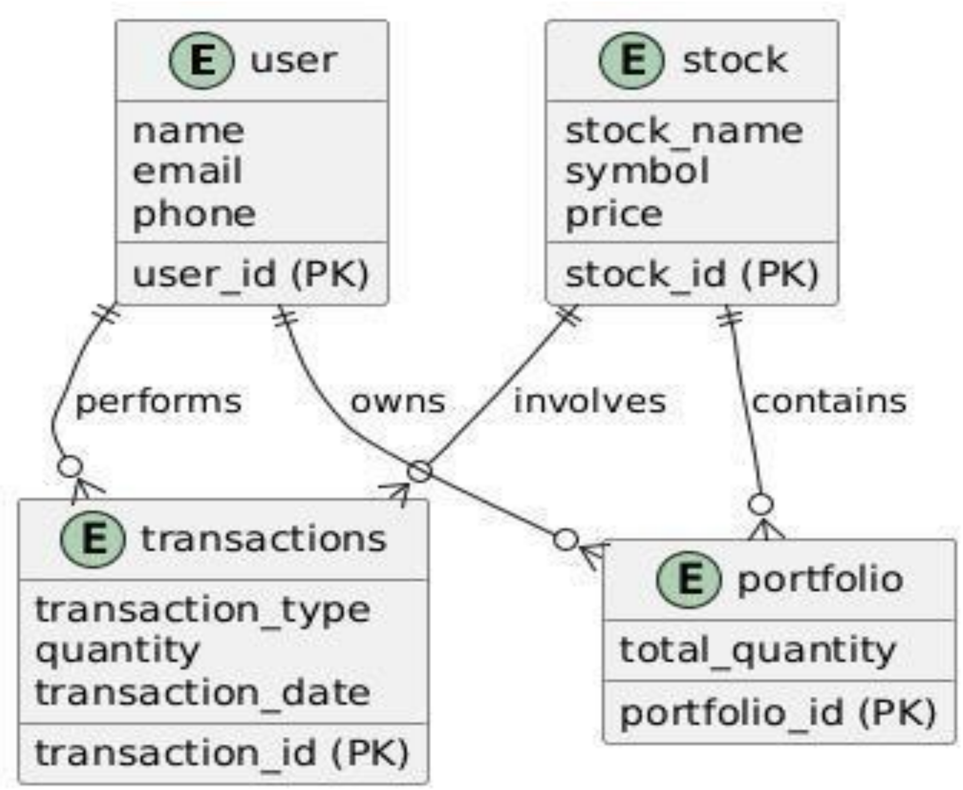


- **The Use Case Diagram represents the interaction between the user and the system. It shows the different operations that the user can perform on the system.**
- *Actor:*
- *• User (Administrator / Operator)* • *Use Cases:*
- *• Add User*
- *• Add Stock*
- *• Buy Stock*

- • *Sell Stock*
- • *View Portfolio*
- • *View Transaction History*
- • *Generate Report*
- ***This diagram indicates that the user can perform both buy and sell operations and can view portfolio and transaction reports.***

ER RELATIONSHIP :

STOCK TRANSACTION RECORDING



The ER (Entity Relationship) Diagram represents the database structure of the system and the relationships between different entities.

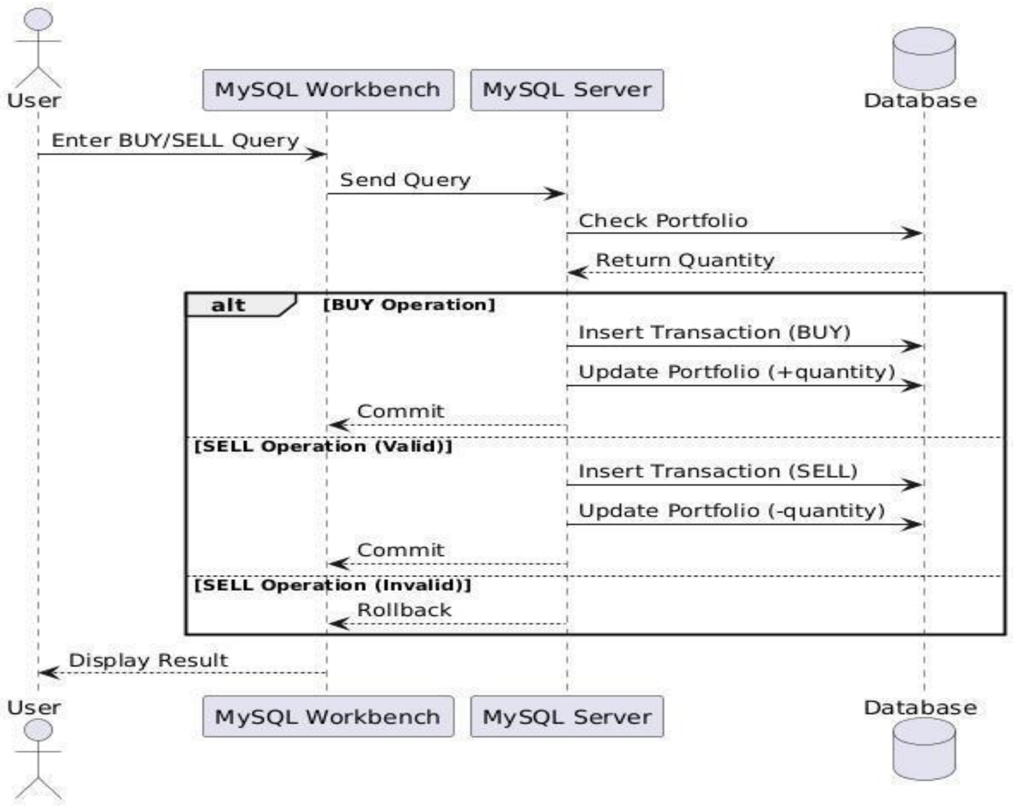
Entities:

- User – Stores information about users who perform stock transactions.
- Stock – Stores details of available stocks.
- Transactions – Stores records of all buy and sell operations.
- Portfolio – Stores the current stock balance of each user.

Relationships:

- One user can perform many transactions.
- One stock can be involved in many transactions.
- One user can have many portfolio records.
- One stock can appear in many portfolio records.

The ER diagram shows how the primary keys and foreign keys connect the tables and ensure data integrity in the database. **SEQUENCE DIAGRAM :**



Sequence Diagram shows how objects interact in a sequence of time.

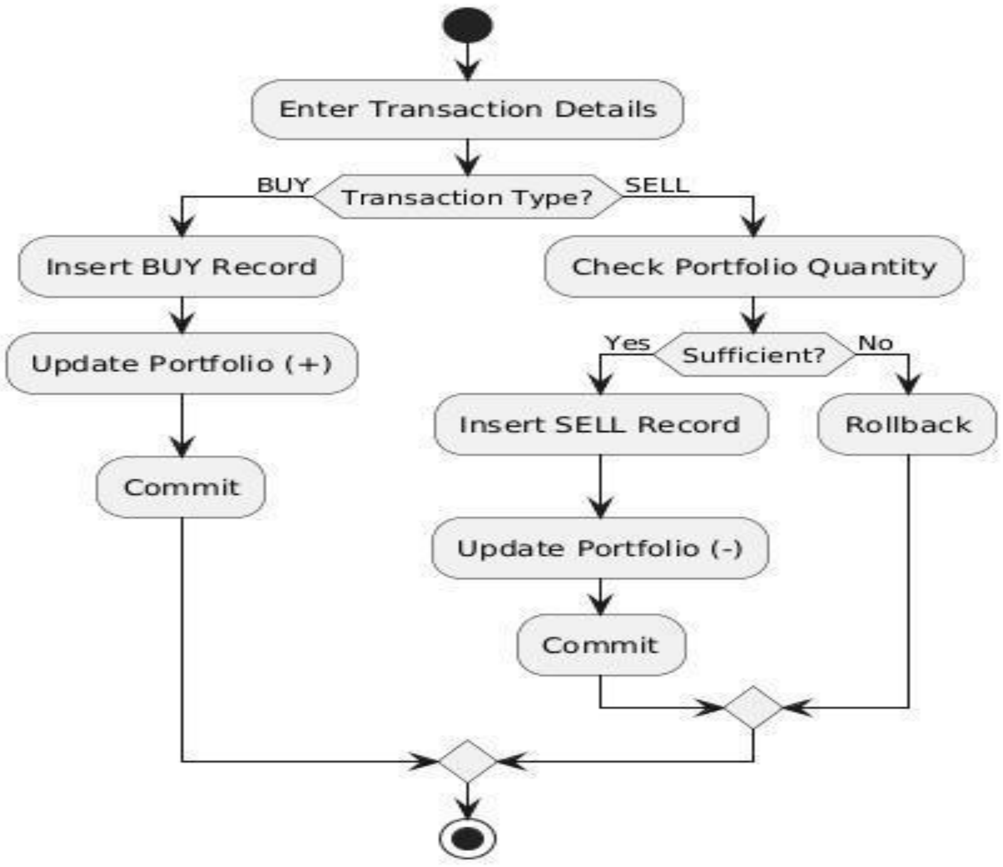
Objects:

- User
- MySQL Workbench
- MySQL Server
- Database Tables

Interaction Flow:

1. User sends SQL query
2. MySQL Workbench forwards query
3. MySQL Server processes query
4. Database tables are accessed
5. Result is returned
6. Output is shown to user

- **ACTIVITY DIAGRAM**



An Activity Diagram represents the flow of activities in the system from start to end.

Steps in Activity Flow:

Start

User enters SQL query

System validates data

If data is valid → Insert/Update records

If data is invalid → Show error

Update portfolio

Store transaction record

Display result

End

CHAPTER 9: SQL IMPLEMENTATION

This chapter describes the SQL commands used to implement the database for the Stock Transaction Record System. SQL is used to create the database, define tables, insert records, retrieve data, and manage transactions. Transaction control commands such as COMMIT and ROLLBACK are used to maintain data consistency during buy and sell operations.

9.1 DATABASE CREATION

```
CREATE DATABASE stock_db;

USE stock_db;
```

Explanation:

The above commands create a database named stock_db and select it for further operations.

9.2 TABLE CREATION

9.2.1 User Table

```
CREATE TABLE user (
    User_id INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(50) NOT NULL,
    Email VARCHAR(50) UNIQUE NOT NULL,
    Phone VARCHAR(15) NOT NULL
);
```

Explanation:

This table stores user information. The user_id uniquely identifies each user.

9.2.2 Stock Table

```
CREATE TABLE stock (
    Stock_id INT PRIMARY KEY AUTO_INCREMENT,
    Stock_name VARCHAR(50) NOT NULL,
    Symbol VARCHAR(10) UNIQUE NOT NULL,
    Price DECIMAL(10,2) NOT NULL
);
```

Explanation:

This table stores stock details such as stock name, symbol, and price.

9.2.3 Transactions Table

```
CREATE TABLE transactions (
    Transaction_id INT PRIMARY KEY AUTO_INCREMENT,
    User_id INT,
    Stock_id INT,
    Transaction_type ENUM(‘BUY’,’SELL’) NOT NULL,
    Quantity INT NOT NULL,
    Transaction_date DATE NOT NULL,
    FOREIGN KEY (user_id) REFERENCES user(user_id),
    FOREIGN KEY (stock_id) REFERENCES stock(stock_id)
);
```

Explanation:

This table stores buy and sell transaction records.

The transaction_type field specifies whether the operation is BUY or SELL.

9.2.4 Portfolio Table

```
CREATE TABLE portfolio (
    Portfolio_id INT PRIMARY KEY AUTO_INCREMENT,
    User_id INT,
    Stock_id INT,
    Total_quantity INT NOT NULL,
    UNIQUE(user_id, stock_id),
    FOREIGN KEY (user_id) REFERENCES user(user_id),
    FOREIGN KEY (stock_id) REFERENCES stock(stock_id)
);
```

Explanation:

This table stores the current stock balance of each user.

The UNIQUE constraint ensures that a user can have only one record per stock.

9.3 DATA INSERTION

Insert into User Table

```
INSERT INTO user(name,email,phone)
VALUES
(‘Irfan Maniyar’,’irfan@gmail.com’,’9876543210’),
(‘Subham Mahanty’,’subham088@gmail.com’,’9899143210’),
(‘Divya Tripathi’,’darkshadow04@gmail.com’,’8882143210’),
(‘Arpit Mishra’,’arpit01@gmail.com’,’9871243210’),
(‘Neev Sabban’,’neev03@gmail.com’,’7113438810’);
```

Insert into Stock Table

```
INSERT INTO stock(stock_name,symbol,price)
VALUES
(‘TCS’,’TCS’,3500),
(‘Infosys’,’INFY’,1500);
```

9.4 DATA RETRIEVAL QUERIES

View all users

```
SELECT * FROM user;
```

View all stocks

```
SELECT * FROM stock;
```

View all transactions

```
SELECT * FROM transactions;
```

Join Query (User + Stock + Transactions)

```
SELECT u.name, s.stock_name, t.transaction_type, t.quantity,
t.transaction_date

FROM transactions t

JOIN user u ON t.user_id = u.user_id

JOIN stock s ON t.stock_id = s.stock_id;
```

Explanation:

This query displays the transaction history of users along with stock names.

9.5 TRANSACTION MANAGEMENT (BUY OPERATION)

The BUY operation must insert a transaction record and update the portfolio table in a single transaction.

```
START TRANSACTION;
```

```
INSERT INTO transactions(user_id, stock_id, transaction_type, quantity,
transaction_date)

VALUES (1,1,'BUY',10,CURDATE());


INSERT INTO portfolio(user_id, stock_id, total_quantity)

VALUES (1,1,10)

ON DUPLICATE KEY UPDATE

Total_quantity = total_quantity + 10;


COMMIT;
```

Explanation:

If both the INSERT and UPDATE statements are successful, the transaction is committed and changes are saved permanently.

9.6 TRANSACTION MANAGEMENT (SELL OPERATION)

Before selling a stock, the system must verify whether the user has sufficient quantity available in the portfolio.

```
START TRANSACTION;
```

```
SELECT total_quantity

FROM portfolio

WHERE user_id = 1 AND stock_id = 1

FOR UPDATE;
```

If the available quantity is greater than or equal to the selling quantity, then:

```
INSERT INTO transactions(user_id, stock_id, transaction_type, quantity, transaction_date)

VALUES (1,1,’SELL’,5,CURDATE());
```

```
UPDATE portfolio

SET total_quantity = total_quantity – 5

WHERE user_id = 1 AND stock_id = 1;
```

```
COMMIT;
```

If sufficient quantity is not available:

```
ROLLBACK;
```

Explanation:

The ROLLBACK command cancels all changes made during the transaction if the user does not have enough stock to sell. This ensures data consistency.

9.7 ADVANCED QUERIES

GROUP BY Query

```
SELECT stock_id, SUM(quantity) AS TotalShares

FROM transactions

GROUP BY stock_id;
```

HAVING Clause

```
SELECT stock_id, SUM(quantity) AS TotalShares
```

```
FROM transactions
GROUP BY stock_id
HAVING SUM(quantity) > 5;
```

Subquery

```
SELECT name
FROM user
WHERE user_id IN (
    SELECT user_id
    FROM transactions
    WHERE quantity > 5
);
```

9.8 VIEW CREATION

```
CREATE VIEW TransactionReport AS

SELECT u.name, s.stock_name, t.transaction_type, t.quantity

FROM transactions t

JOIN user u ON t.user_id = u.user_id

JOIN stock s ON t.stock_id = s.stock_id;
```

Display View

```
SELECT * FROM TransactionReport;
```

Thus, the Stock Transaction Record System ensures data integrity and consistency using transaction management techniques.

CHAPTER 10: SYSTEM TESTING AND RESULT

System testing is performed to verify whether the developed database system works correctly and produces accurate results. It ensures that all SQL queries, constraints, and transactions operate as expected.

The Stock Transaction Recording System is tested using different SQL queries for insertion, retrieval, updating, and deletion of data. Sample data is used to validate the working of the system.

10.1 Testing Strategy

The following testing methods are used:

- Query execution testing
- Data validation testing
- Constraint testing
- Transaction testing

• Result verification

Each module of the system is tested independently and then tested as an integrated system.

10.2 Test Cases

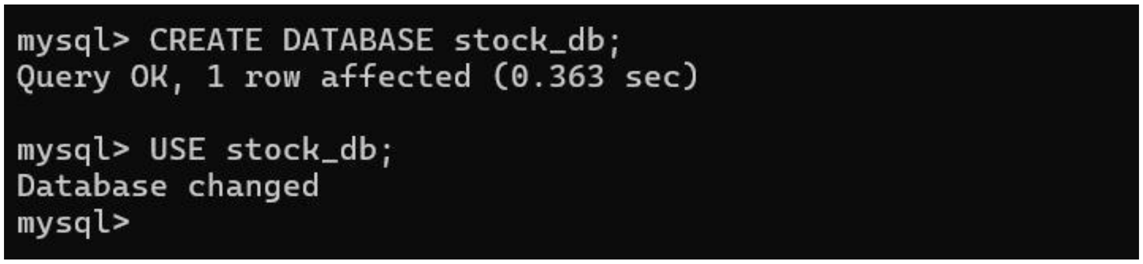
Table 10.1: Test Cases:

Test Case ID	Operation	Input	Expected Result	Status
TC01	Insert User	Valid user data	Record inserted successfully	Pass
TC02	Insert Stock	Valid stock data	Stock record inserted	Pass
TC03	Buy Transaction	Quantity ≤ available	Transaction recorded and portfolio updated	Pass
TC04	Sell Transaction	Quantity > available	Transaction rejected	Pass

TC05	View Report	SELECT query	Correct transaction list displayed	Pass

10.3 Sample Output (Result Tables):

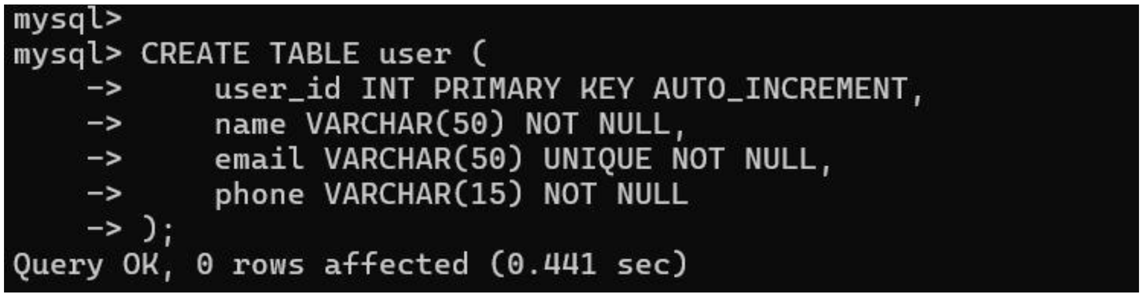
10.3.1 Database Creation and Use Database.



```
mysql> CREATE DATABASE stock_db;
Query OK, 1 row affected (0.363 sec)

mysql> USE stock_db;
Database changed
mysql>
```

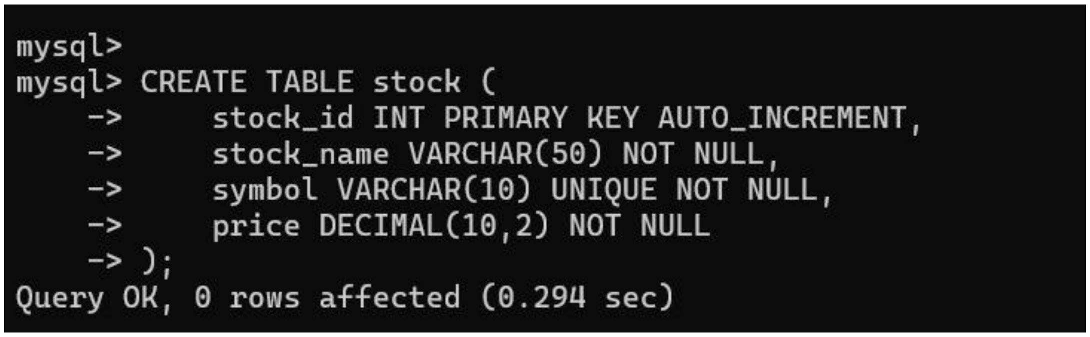
10.3.2 Tables Creation (user, stock, transaction , portfolio) A. User

Table:



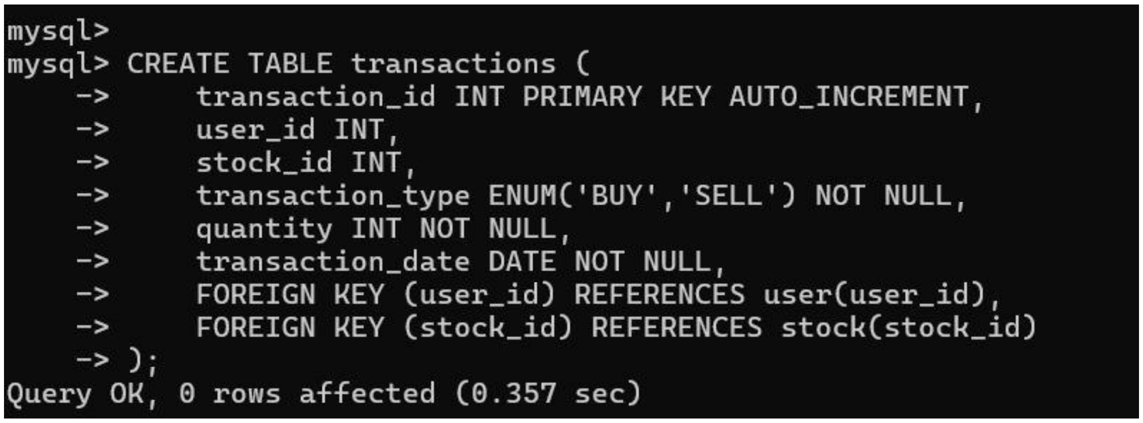
```
mysql>
mysql> CREATE TABLE user (
    ->     user_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     name VARCHAR(50) NOT NULL,
    ->     email VARCHAR(50) UNIQUE NOT NULL,
    ->     phone VARCHAR(15) NOT NULL
    -> );
Query OK, 0 rows affected (0.441 sec)
```

B. Stock Table:



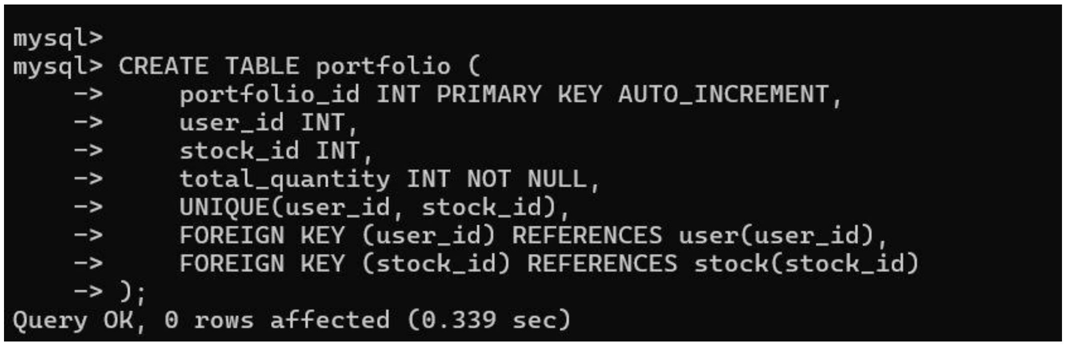
```
mysql>
mysql> CREATE TABLE stock (
    ->     stock_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     stock_name VARCHAR(50) NOT NULL,
    ->     symbol VARCHAR(10) UNIQUE NOT NULL,
    ->     price DECIMAL(10,2) NOT NULL
    -> );
Query OK, 0 rows affected (0.294 sec)
```

C. Transaction Table(Buy/Sell Operation):



```
mysql>
mysql> CREATE TABLE transactions (
    ->     transaction_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     user_id INT,
    ->     stock_id INT,
    ->     transaction_type ENUM('BUY','SELL') NOT NULL,
    ->     quantity INT NOT NULL,
    ->     transaction_date DATE NOT NULL,
    ->     FOREIGN KEY (user_id) REFERENCES user(user_id),
    ->     FOREIGN KEY (stock_id) REFERENCES stock(stock_id)
    -> );
Query OK, 0 rows affected (0.357 sec)
```

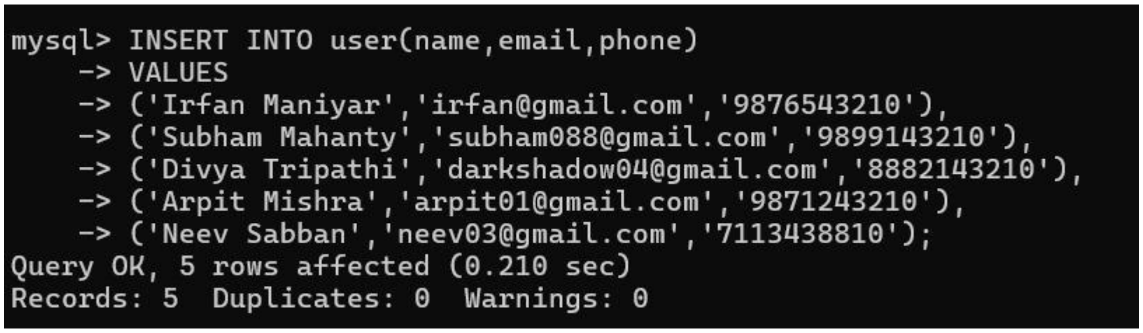
D. Portfolio Table:



```
mysql>
mysql> CREATE TABLE portfolio (
    ->     portfolio_id INT PRIMARY KEY AUTO_INCREMENT,
    ->     user_id INT,
    ->     stock_id INT,
    ->     total_quantity INT NOT NULL,
    ->     UNIQUE(user_id, stock_id),
    ->     FOREIGN KEY (user_id) REFERENCES user(user_id),
    ->     FOREIGN KEY (stock_id) REFERENCES stock(stock_id)
    -> );
Query OK, 0 rows affected (0.339 sec)
```

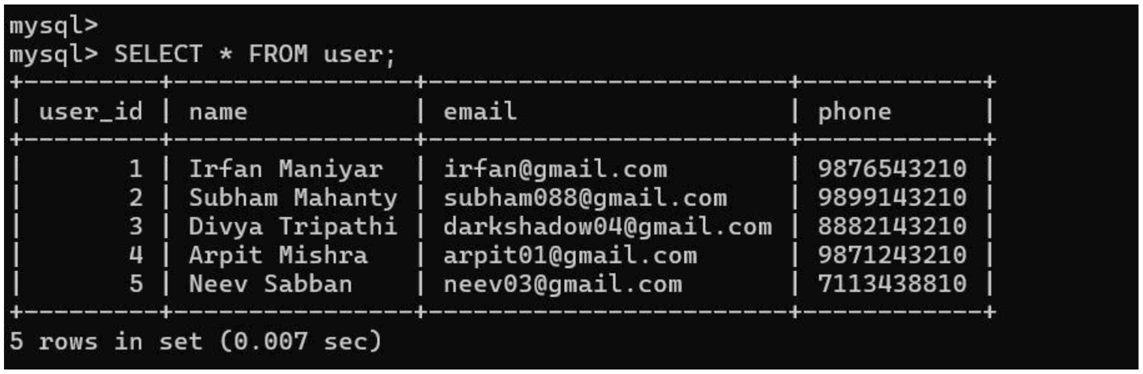
10.3.3 Table Value Insertion and Display:

A. Insert into User table:



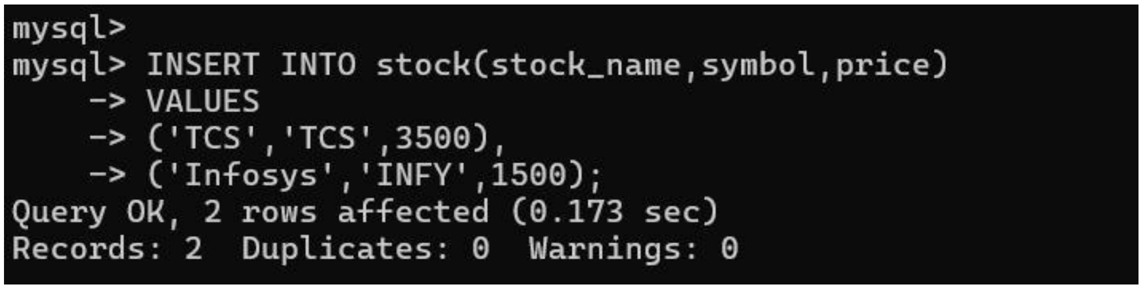
```
mysql> INSERT INTO user(name,email,phone)
    -> VALUES
    -> ('Irfan Maniyar','irfan@gmail.com','9876543210'),
    -> ('Subham Mahanty','subham088@gmail.com','9899143210'),
    -> ('Divya Tripathi','darkshadow04@gmail.com','8882143210'),
    -> ('Arpit Mishra','arpit01@gmail.com','9871243210'),
    -> ('Neev Sabban','neev03@gmail.com','7113438810');
Query OK, 5 rows affected (0.210 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

- Display User Table:



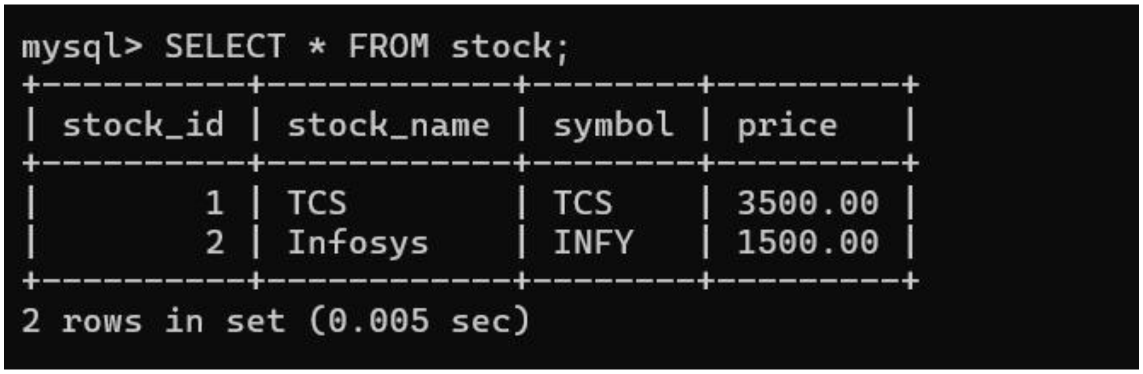
```
mysql>
mysql> SELECT * FROM user;
+---------+----------------+------------------------+------------+
| user_id | name           | email                  | phone      |
+---------+----------------+------------------------+------------+
|       1 | Irfan Maniyar  | irfan@gmail.com        | 9876543210 |
|       2 | Subham Mahanty | subham088@gmail.com    | 9899143210 |
|       3 | Divya Tripathi | darkshadow04@gmail.com | 8882143210 |
|       4 | Arpit Mishra   | arpit01@gmail.com      | 9871243210 |
|       5 | Neev Sabban    | neev03@gmail.com       | 7113438810 |
+---------+----------------+------------------------+------------+
5 rows in set (0.007 sec)
```

B. Insert into Stock Table:



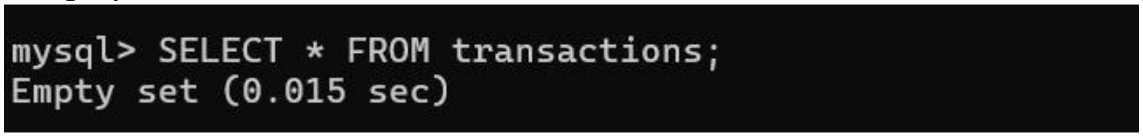
```
mysql>
mysql> INSERT INTO stock(stock_name,symbol,price)
    -> VALUES
    -> ('TCS','TCS',3500),
    -> ('Infosys','INFY',1500);
Query OK, 2 rows affected (0.173 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

- Display Stock Table:



```
mysql> SELECT * FROM stock;
+----------+------------+--------+---------+
| stock_id | stock_name | symbol | price   |
+----------+------------+--------+---------+
|        1 | TCS        | TCS    | 3500.00 |
|        2 | Infosys    | INFY   | 1500.00 |
+----------+------------+--------+---------+
2 rows in set (0.005 sec)
```

- Display Transaction Table:

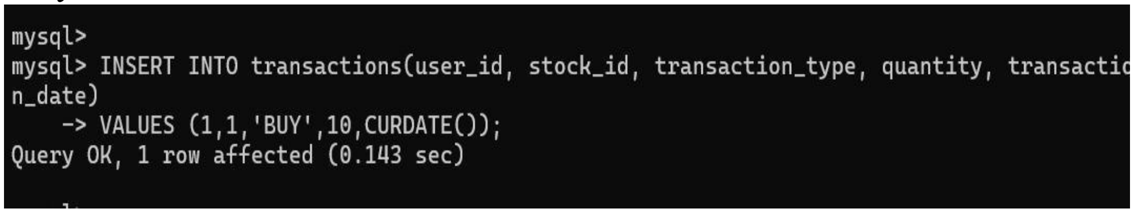


```
mysql> SELECT * FROM transactions;
Empty set (0.015 sec)
```

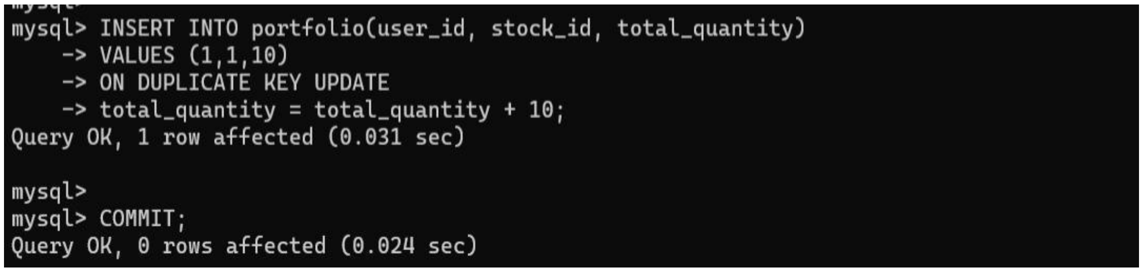
- ***<u>This proves that no buy and sell transaction have been recorded.now insert the values in the transaction Tables.</u>***

C. Insert into Transaction Table(BUY OPERATION):

- Buy Transaction:

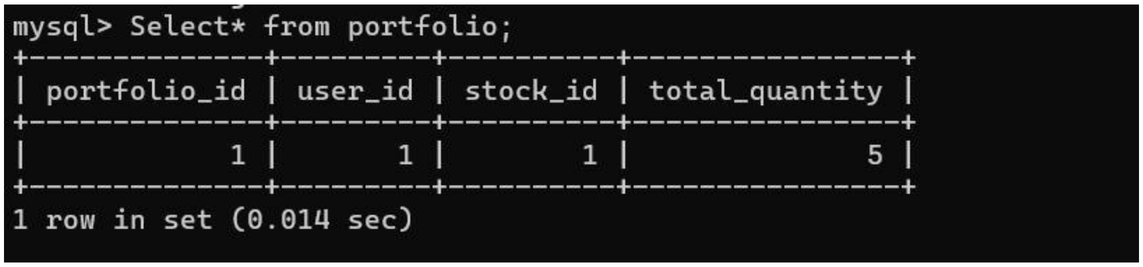


```
mysql>
mysql> INSERT INTO transactions(user_id, stock_id, transaction_type, quantity, transactio
n_date)
    -> VALUES (1,1,'BUY',10,CURDATE());
Query OK, 1 row affected (0.143 sec)
```



```
mysql> INSERT INTO portfolio(user_id, stock_id, total_quantity)
    -> VALUES (1,1,10)
    -> ON DUPLICATE KEY UPDATE
    -> total_quantity = total_quantity + 10;
Query OK, 1 row affected (0.031 sec)

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.024 sec)
```

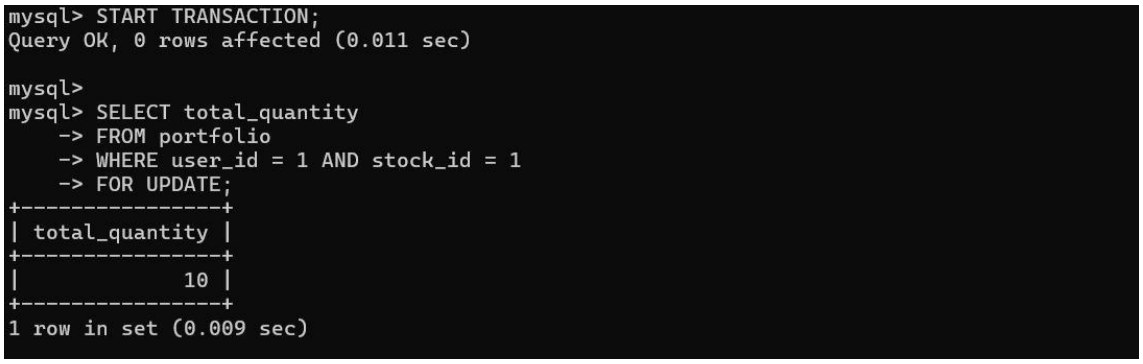
- Portfolio after buy:



```
mysql> Select* from portfolio;
+--------------+---------+----------+----------------+
| portfolio_id | user_id | stock_id | total_quantity |
+--------------+---------+----------+----------------+
|            1 |       1 |        1 |              5 |
+--------------+---------+----------+----------------+
1 row in set (0.014 sec)
```

D. Insert into Transaction Table(SELL OPERATION):

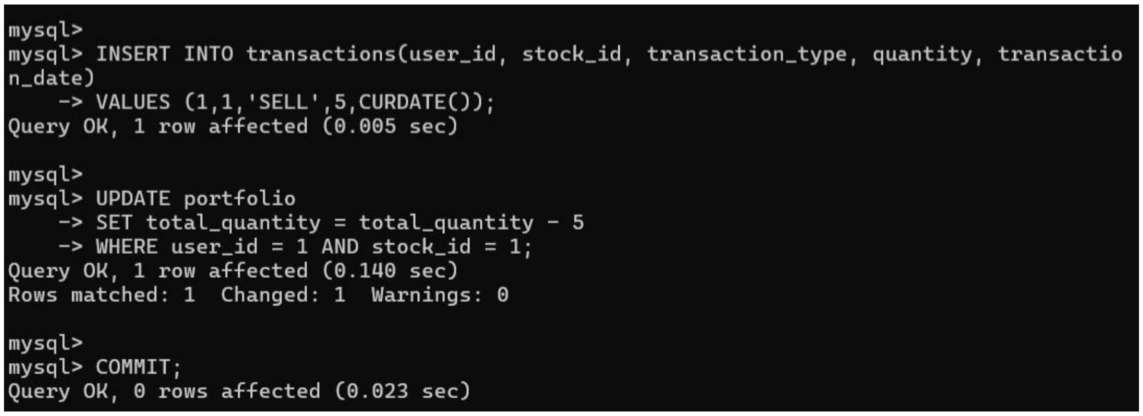
- Start Sell Transaction:



```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.011 sec)

mysql>
mysql> SELECT total_quantity
    -> FROM portfolio
    -> WHERE user_id = 1 AND stock_id = 1
    -> FOR UPDATE;
+----------------+
| total_quantity |
+----------------+
|             10 |
+----------------+
1 row in set (0.009 sec)
```

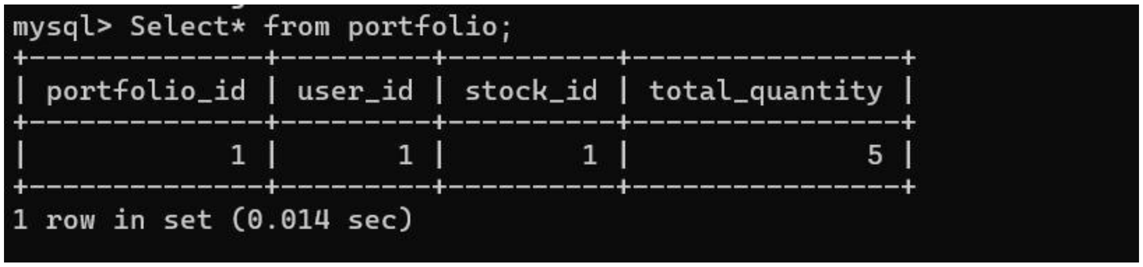
C. Result of Transaction Table: •

Sell Transaction

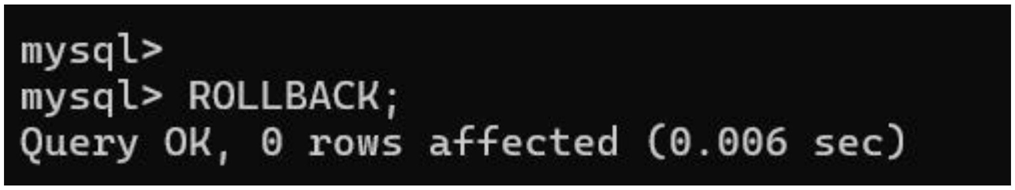


```
mysql>
mysql> INSERT INTO transactions(user_id, stock_id, transaction_type, quantity, transactio
n_date)
    -> VALUES (1,1,'SELL',5,CURDATE());
Query OK, 1 row affected (0.005 sec)

mysql>
mysql> UPDATE portfolio
    -> SET total_quantity = total_quantity - 5
    -> WHERE user_id = 1 AND stock_id = 1;
Query OK, 1 row affected (0.140 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.023 sec)
```

- Portfolio After Sell:

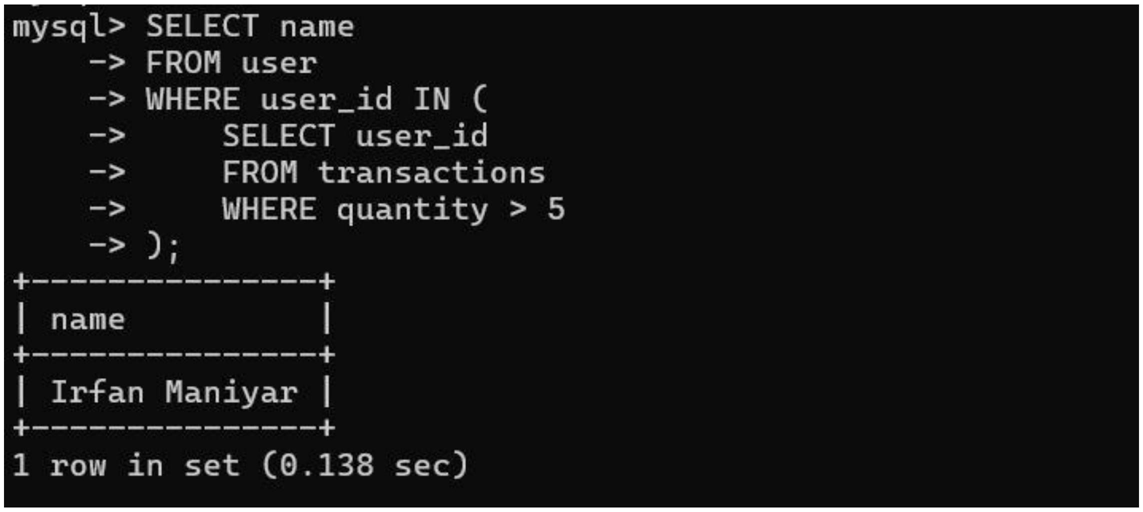

```
mysql> Select* from portfolio;
+--------------+---------+----------+----------------+
| portfolio_id | user_id | stock_id | total_quantity |
+--------------+---------+----------+----------------+
|            1 |       1 |        1 |              5 |
+--------------+---------+----------+----------------+
1 row in set (0.014 sec)
```

- Rollback:


```
mysql>
mysql> ROLLBACK;
Query OK, 0 rows affected (0.006 sec)
```

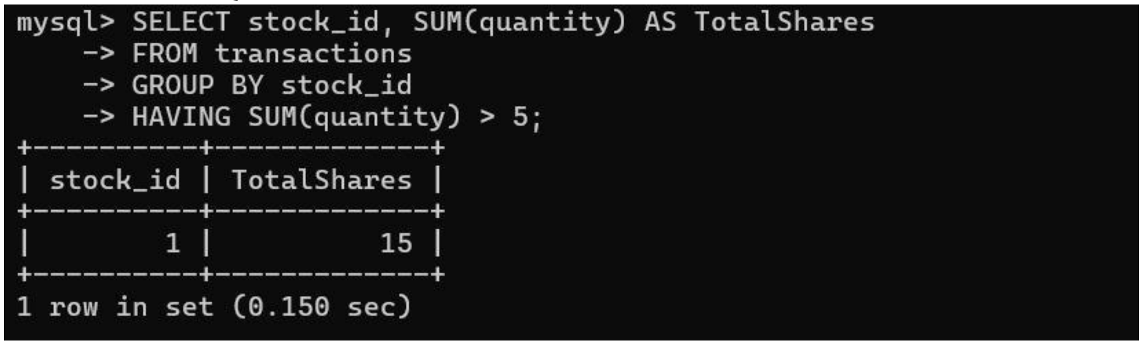
10.3.4 Aggregate Query (GROUP BY + HAVING)

- GROUP BY QUERY:



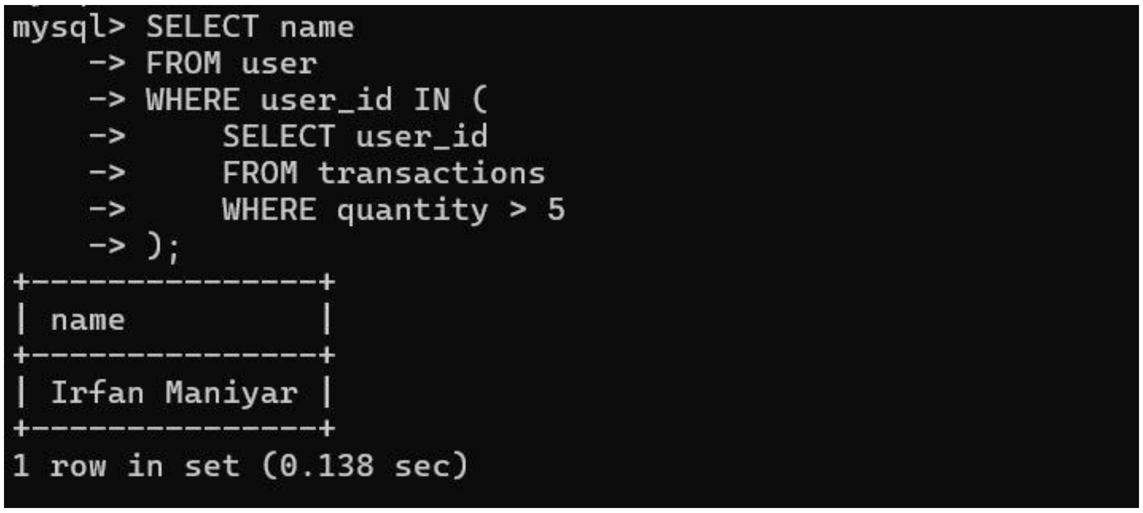
```
mysql> SELECT name
    -> FROM user
    -> WHERE user_id IN (
    ->     SELECT user_id
    ->     FROM transactions
    ->     WHERE quantity > 5
    -> );
+---------------+
| name          |
+---------------+
| Irfan Maniyar |
+---------------+
1 row in set (0.138 sec)
```

- HAVING QUERY:



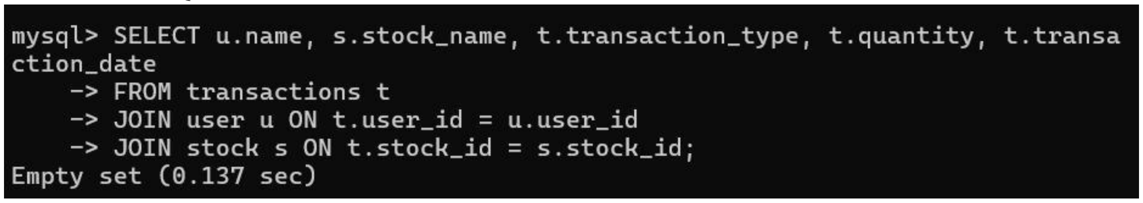
```
mysql> SELECT stock_id, SUM(quantity) AS TotalShares
    -> FROM transactions
    -> GROUP BY stock_id
    -> HAVING SUM(quantity) > 5;
+----------+-------------+
| stock_id | TotalShares |
+----------+-------------+
|        1 |          15 |
+----------+-------------+
1 row in set (0.150 sec)
```

- SUBQUERY:



```
mysql> SELECT name
    -> FROM user
    -> WHERE user_id IN (
    ->     SELECT user_id
    ->     FROM transactions
    ->     WHERE quantity > 5
    -> );
+---------------+
| name          |
+---------------+
| Irfan Maniyar |
+---------------+
1 row in set (0.138 sec)
```

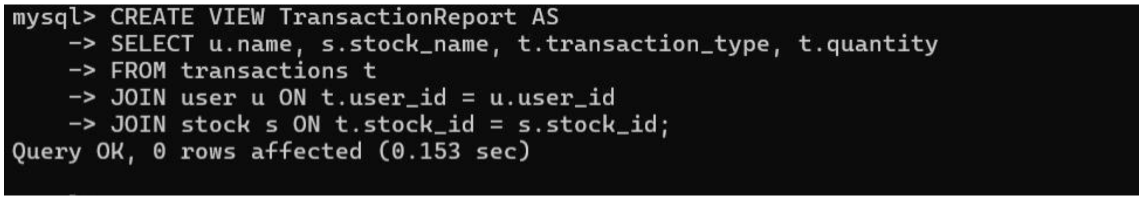
- JOIN QUERY:



```
mysql> SELECT u.name, s.stock_name, t.transaction_type, t.quantity, t.transa
ction_date
    -> FROM transactions t
    -> JOIN user u ON t.user_id = u.user_id
    -> JOIN stock s ON t.stock_id = s.stock_id;
Empty set (0.137 sec)
```

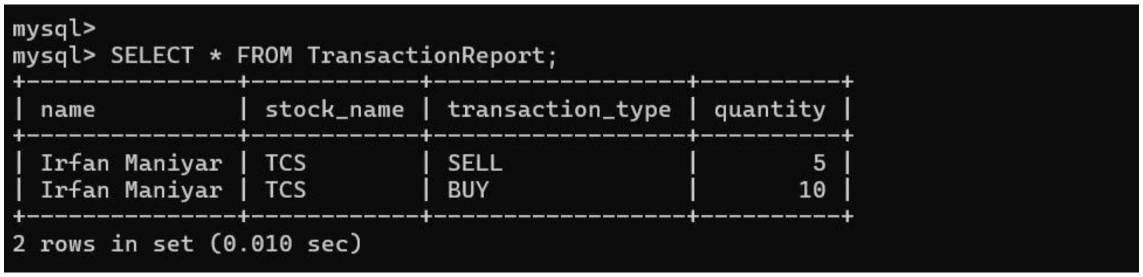
10.3.5 VIEW REPORT:

CREATE VIEW:



```
mysql> CREATE VIEW TransactionReport AS
    -> SELECT u.name, s.stock_name, t.transaction_type, t.quantity
    -> FROM transactions t
    -> JOIN user u ON t.user_id = u.user_id
    -> JOIN stock s ON t.stock_id = s.stock_id;
Query OK, 0 rows affected (0.153 sec)
```

VIEW OUTPUT:



```
mysql>
mysql> SELECT * FROM TransactionReport;
+---------------+------------+------------------+----------+
| name          | stock_name | transaction_type | quantity |
+---------------+------------+------------------+----------+
| Irfan Maniyar | TCS        | SELL             |        5 |
| Irfan Maniyar | TCS        | BUY              |       10 |
+---------------+------------+------------------+----------+
2 rows in set (0.010 sec)
```

10.4 Result Analysis

The results show that:

- Data is inserted successfully into all tables.
- Foreign key relationships are maintained.
- Portfolio quantity is updated after each transaction.
- Invalid sell transactions are not allowed.
- Transaction history is retrieved correctly.

The system performs accurately and maintains data consistency.

CHAPTER 11: SECURITY, BACKUP AND RECOVERY

Database security and data protection are essential to prevent unauthorized access and data loss. This system uses MySQL's built-in security and backup features to ensure safe storage of stock transaction records.

11.1 Security

Security is maintained using the following methods:

- User authentication through MySQL login
- Role-based access control
- Granting limited privileges to users
- Use of passwords for database access

Example of User Privilege Control:

```
CREATE USER ‘dbuser’@’localhost’ IDENTIFIED BY ‘password123’;
GRANT SELECT, INSERT, UPDATE ON StockTransactionDB.* TO
‘dbuser’@’localhost’;
```

This ensures that only authorized users can access and modify the database.

11.2 Backup

Backup is the process of creating a copy of the database to prevent data loss due to system failure or accidental deletion.

MySQL provides a tool called mysqldump for database backup.

Backup Command:

```
Mysqldump -u root -p StockTransactionDB > backup.sql
```

This command creates a backup file named backup.sql.

11.3 Recovery

Recovery is the process of restoring data from a backup file.

Restore Command:

```
Mysql -u root -p StockTransactionDB < backup.sql
```

This restores the database to its previous state.

CHAPTER 12: FUTURE SCOPE AND CONCLUSION

12.1 Future Scope

The Stock Transaction Recording System can be enhanced in future with the following features:

- Integration with a web-based user interface
- Real-time stock market data connection
- Graphical dashboards for reports
- User login system with encryption
- Advanced analytics and forecasting
- Mobile application integration

These improvements will make the system more practical and suitable for real-world stock management applications.

12.2 Conclusion

The Stock Transaction Recording System is a successful implementation of a database-driven application using MySQL. It efficiently stores and manages user, stock, transaction, and portfolio data in a structured format.

The project demonstrates important database concepts such as:

- Table creation and relationships

- Use of primary and foreign keys
- SQL queries for data manipulation
- Transaction control (COMMIT, ROLLBACK)
- Security and backup mechanisms

This project has provided valuable practical experience in database design and SQL programming. It also enhances understanding of how real-world financial data can be managed using a relational database system.

CHAPTER 13: REFERENCES

1. MySQL Official Documentation https://dev.mysql.com/doc/

2. Korth, H. F., Silberschatz, A., Sudarshan, S.
Database System Concepts, McGraw-Hill.

3. Elmasri, R., Navathe, S.
Fundamentals of Database Systems, Pearson Education.

4. Online SQL learning resources
https://www.w3schools.com/sql/

CHAPTER 14: GLOSSARY

Term.	Description
1. DBMS	Database Management System

2.	SQL.	Structured Query Language
3.	MySQL	Open-source relational database
4.	Primary Key	Unique identifier for a table
5.	Foreign Key	Field linking two tables
6.	Transaction.	A single logical unit of work
7.	Commit.	Saves changes permanently
8.	Rollback	Cancels changes
9.	ER Diagram	Entity Relationship Diagram
10.	Gantt Chart	Project scheduling chart