ZenDB

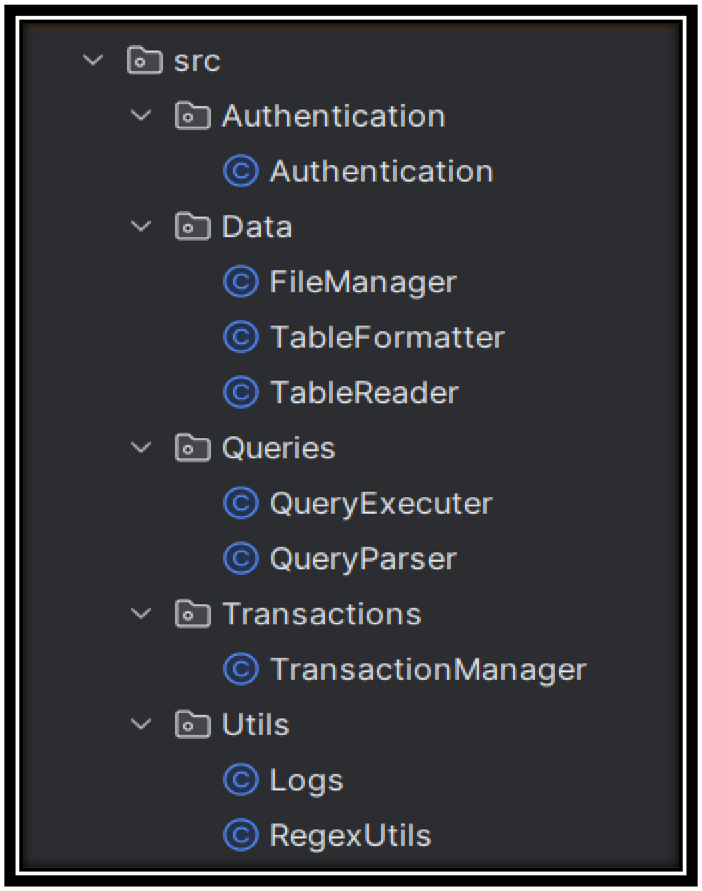
Table of Contents

Design Principles

Design principles guide the organization and arrangement of elements to create effective and aesthetically pleasing designs, ensuring consistency and clarity for better user experience. For the development of this project, solid principles have been followed and the elaboration of that is given below.

1. Single responsibility principle

- The idea behind the SRP[8] is that every class, module, or function in a program should have one responsibility/purpose in a program. As a commonly used definition, "every class should have only one reason to change".
- This principle is well followed in the code as all the functionalities are having their own separate class.



```
src
  Authentication
    Authentication
  Data
    FileManager
    TableFormatter
    TableReader
  Queries
    QueryExecuter
    QueryParser
  Transactions
    TransactionManager
  Utils
    Logs
    RegexUtils
```

Figure 2.1.1 : Evidence of SRP

- All the authentication related processes like user registration and authentication are handled in Authentication class.

- Reading and writing in files and all the other operation in the file are handled in FileManager class.
- Table needs to be formatted before displaying it to the user so all the formatting related stuff are handled in TableFormatter.
- In buffer the data is stored in the form of tables so TableReader is a class provided to facilitate that functionality.
- All the user input are in the form of queries so QueryParser is the class that manages to parse all the incomming queries to process and give relevant results.
- After parsing the query, it needs to be executed and for providing that functionality, QueryExecuter class is designed.

2. Open/Closed Principle

- The open-closed principle states that software entities should be open for extension but closed for modification[9].
- This states that we should be able to add new functionality without altering the existing code.
- To serve this purpose, I have created interfaces[10] so that new code can be added without altering the existing code.

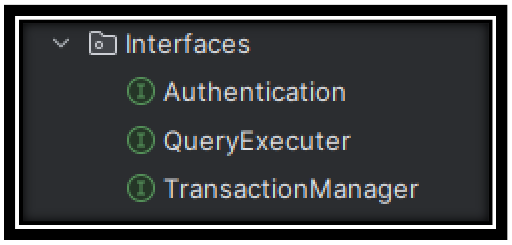


Figure 2.1.2 : Evidence of OCP

3. Liskov substitution principle

- This principle states that objects of a parent class shall be replaceable with the objects of its subclass without breaking the application[11].

4. Interface segregation principle

- This principle states that a class should not be forced to implement unnecessary interfaces.
- The Interface Segregation Principle[13] divides software into several independent components with the aim of minimising side effects and the frequency of necessary updates.
- Thus, as shows in the above diagram, I have created multiple interfaces so that no class should implement the interface which it is not supposed to.

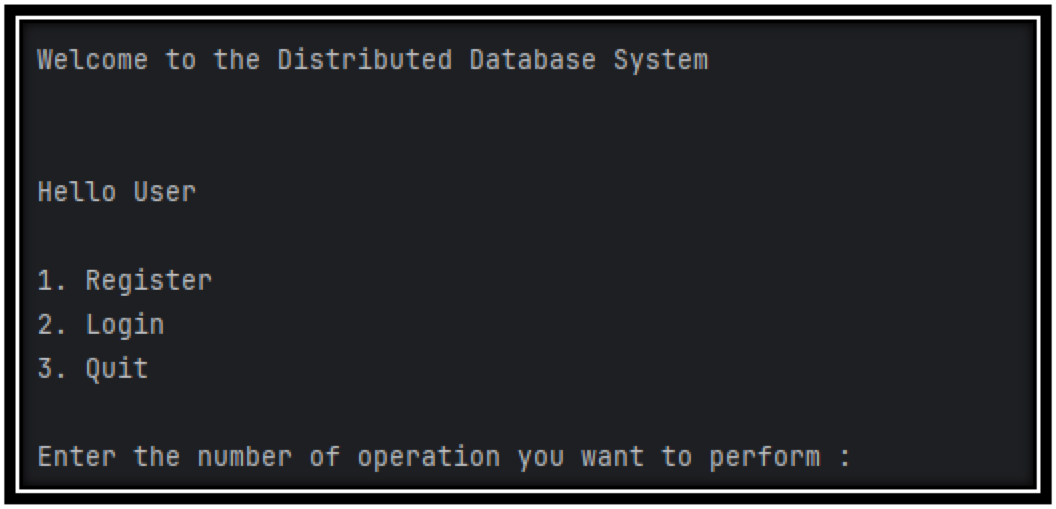
5. Dependency Inversion

- The Dependency Inversion Principle (DIP)[14] states that high-level modules should not depend on low-level modules but rather both should depend on abstractions.
- Additionally, it emphasizes that abstractions should not depend on details; details should depend on abstractions.

Authentication

1. Initial prompt:

- This is the welcome message and the initial prompt that the user is shown when entering the system.
- Firstly, we are shown authentication options where we can register ourselves or Login if we are already registered.

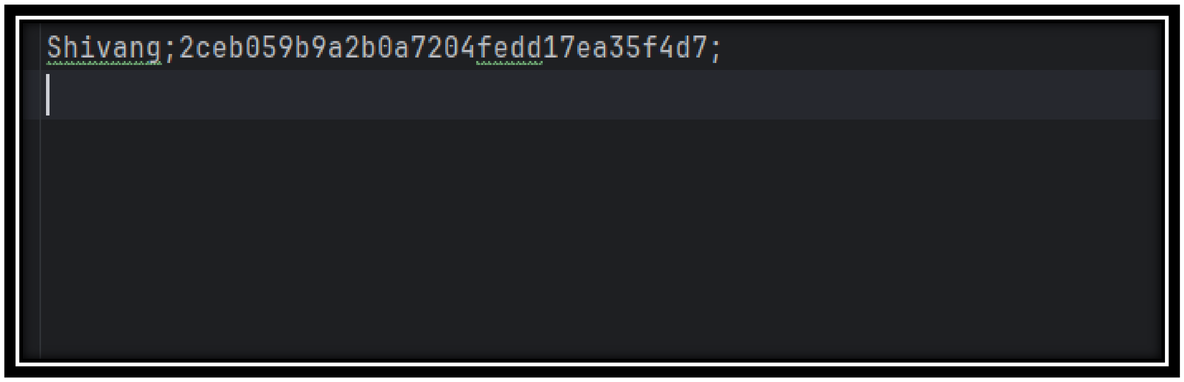


```
Welcome to the Distributed Database System


Hello User

1. Register
2. Login
3. Quit

Enter the number of operation you want to perform :
```

Figure 2.2.1.1 : Initial Prompt

2. Registration:

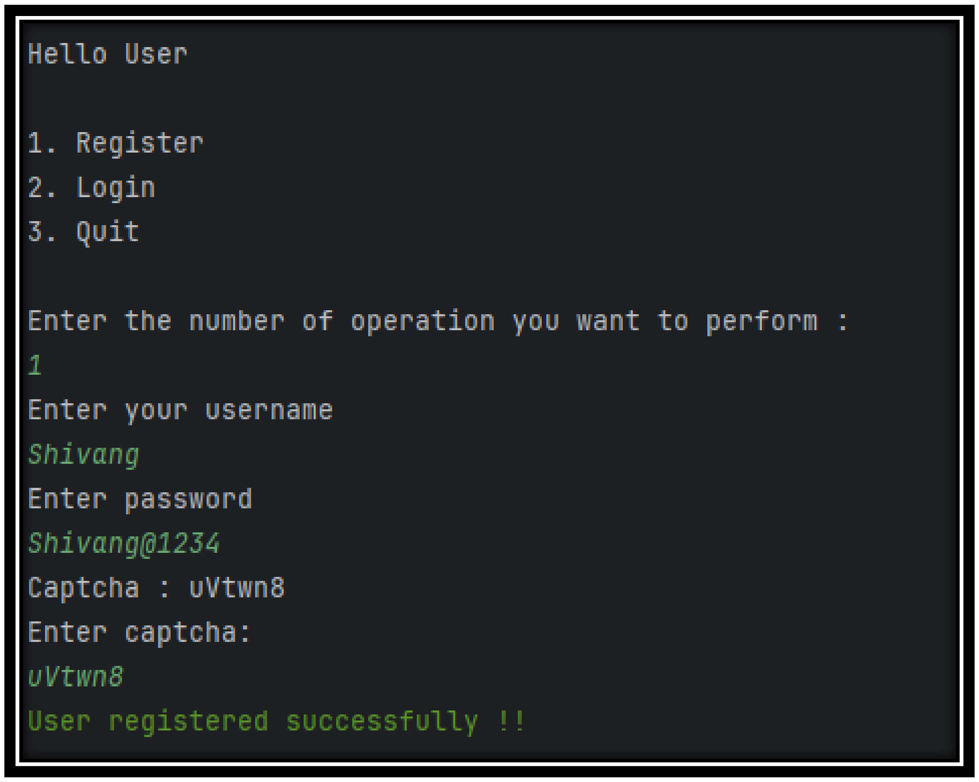
- Upon selection of registration option, username, password and captcha are asked to enter for registration purpose.
- After entering all the valid information, user is registered successfully in the database.
- Encryption algorithm called SHA256 is used to store passwords securely in the database in the form of 16 bits string.
- MD5[12] algorithm is used to encrypt the password.



```
Shivang;2ceb059b9a2b0a7204fedd17ea35f4d7;
```

Figure 2.2.2.1 : Encrypted Password

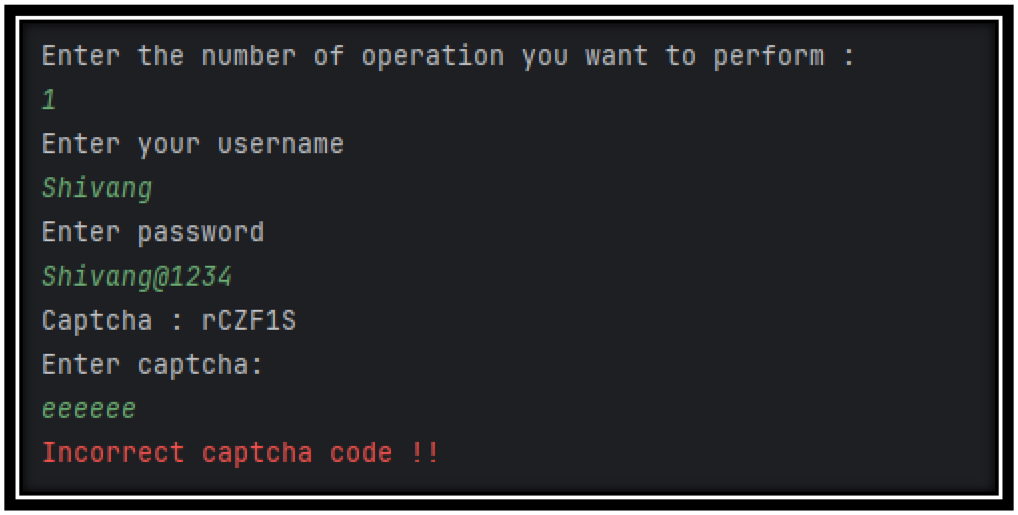
- Success message is shown if captcha is entered correctly.



```
Hello User

1. Register
2. Login
3. Quit

Enter the number of operation you want to perform :
1
Enter your username
Shivang
Enter password
Shivang@1234
Captcha : uVtwn8
Enter captcha:
uVtwn8
User registered successfully !!
```

Figure 2.2.2.2 : Successful registration

- If the captcha is incorrectly written, then the error message with error details is shown and the user is directed back to authentication options.

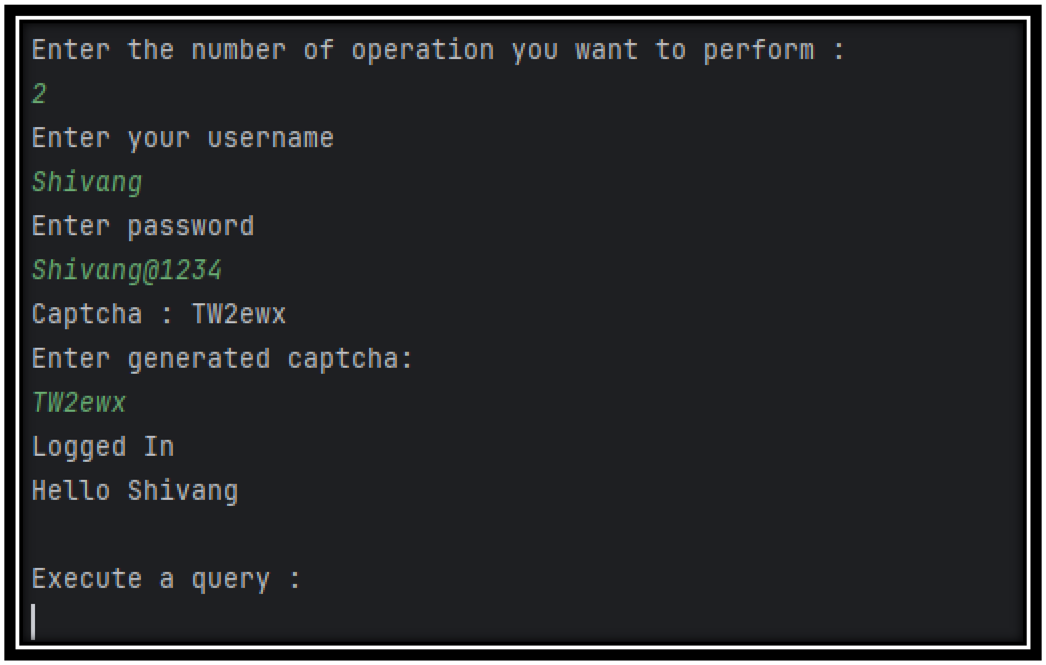


```
Enter the number of operation you want to perform :
1
Enter your username
Shivang
Enter password
Shivang@1234
Captcha : rCZF1S
Enter captcha:
eeeeee
Incorrect captcha code !!
```

Figure 2.2.2.3 : Incorrect captcha

3. Login:

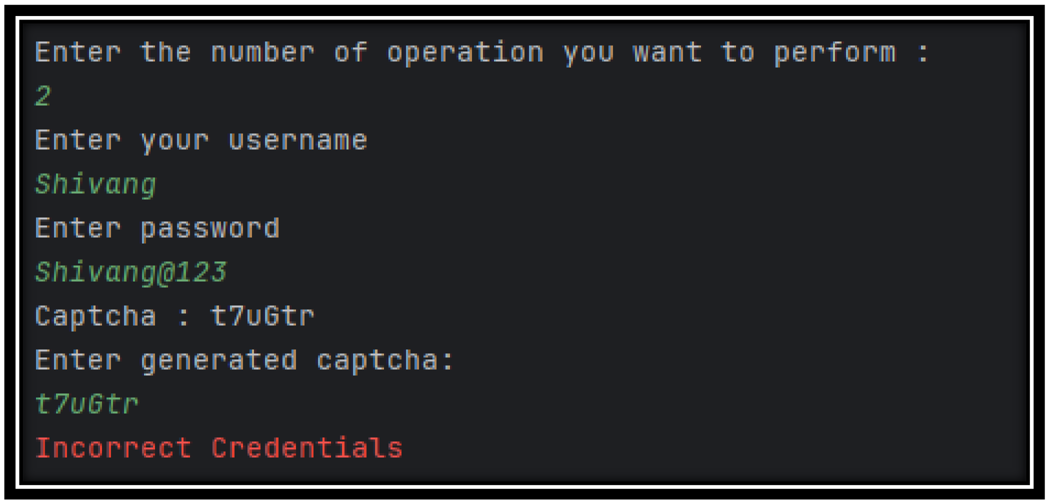
- After registering, we have to login with the correct username and password.
- We are also asked captcha while logging in and if provided incorrect is directed back to authentication page as shown in registration section.
- Passwords in this case are again hashed and compared with the one stored while registration process.



```
Enter the number of operation you want to perform :
2
Enter your username
Shivang
Enter password
Shivang@1234
Captcha : TW2ewx
Enter generated captcha:
TW2ewx
Logged In
Hello Shivang

Execute a query :
|
```

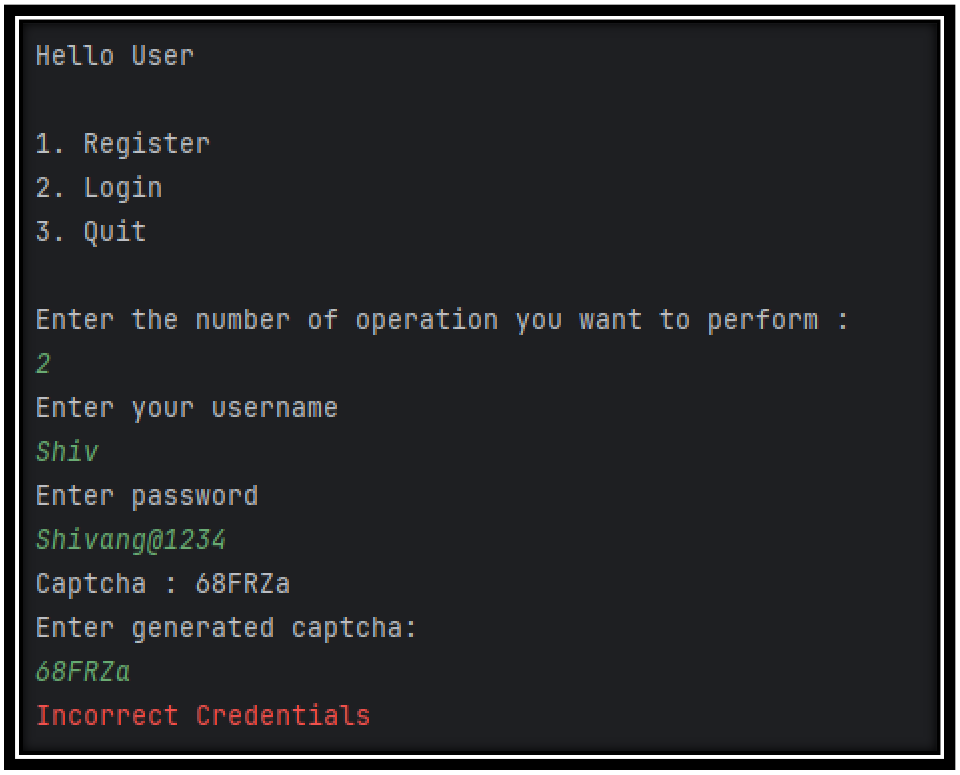
Figure 2.2.3.1 : Successful login

- If the entered username or password is incorrect then the error message with incorrect credentials is shown to the user and is asked back to login again.



```
Enter the number of operation you want to perform :
2
Enter your username
Shivang
Enter password
Shivang@123
Captcha : t7uGtr
Enter generated captcha:
t7uGtr
Incorrect Credentials
```

Figure 2.2.3.2 : Incorrect password in Login

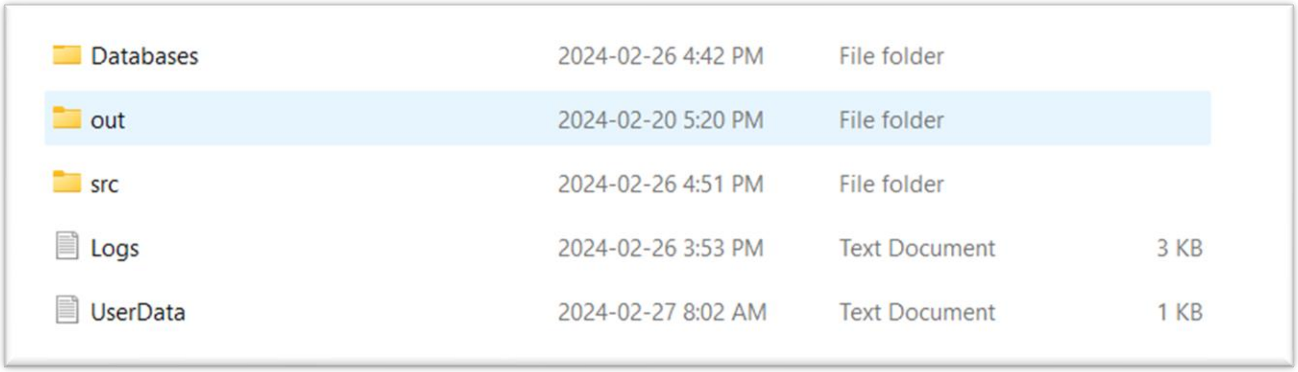


```
Hello User

1. Register
2. Login
3. Quit

Enter the number of operation you want to perform :
2
Enter your username
Shiv
Enter password
Shivang@1234
Captcha : 68FRZa
Enter generated captcha:
68FRZa
Incorrect Credentials
```

Figure 2.2.3.3 : Incorrect username in Login

Persistent Storage

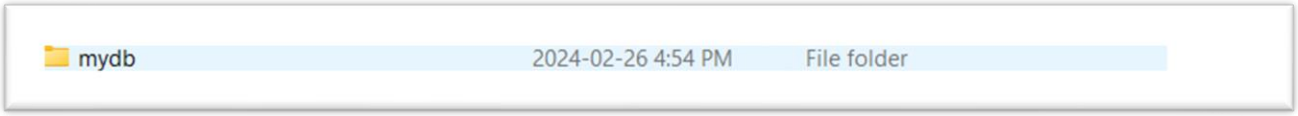
- To facilitate persistent storage, I have used a traditional method of storing data in folders and text files.
- Folders will be created for each database having database name as folder name.
- Each database has multiple tables and thus each folder would have files inside it with file name as table name and each file would contain data to tables.



Databases	2024-02-26 4:42 PM	File folder	
out	2024-02-20 5:20 PM	File folder	
src	2024-02-26 4:51 PM	File folder	
Logs	2024-02-26 3:53 PM	Text Document	3 KB
UserData	2024-02-27 8:02 AM	Text Document	1 KB

Figure 2.3.1 : Folder structure

- This is the folder structure where the information and credentials about username and password are stored in UserData.txt file and all the databases are stored in Database folder.



mydb	2024-02-26 4:54 PM	File folder

Figure 2.3.2 : Database as folder

- Inside the Database folder, there is a folder named mydb which is the name of database.
- There is only one database right now as our system provides functionality to create one database only.

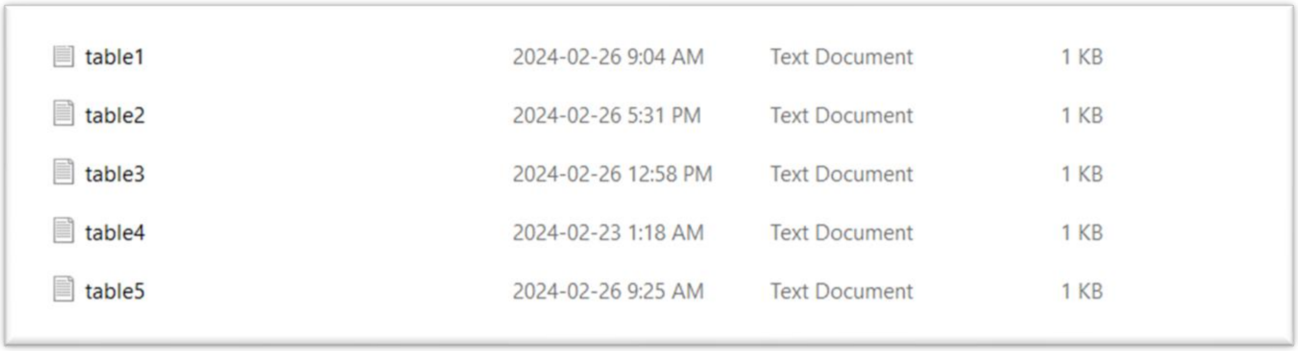
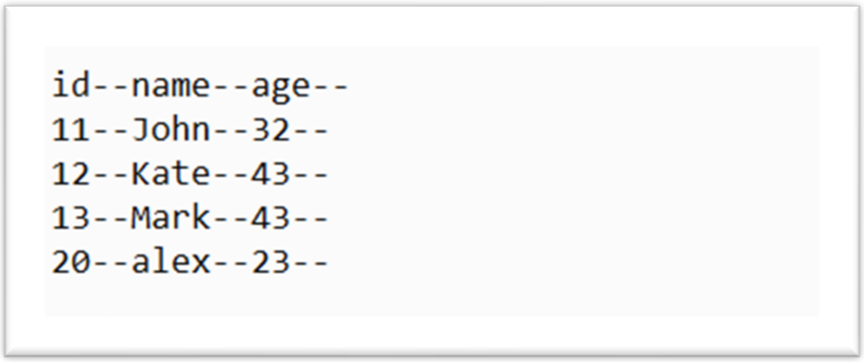


table1	2024-02-26 9:04 AM	Text Document	1 KB
table2	2024-02-26 5:31 PM	Text Document	1 KB
table3	2024-02-26 12:58 PM	Text Document	1 KB
table4	2024-02-23 1:18 AM	Text Document	1 KB
table5	2024-02-26 9:25 AM	Text Document	1 KB

Figure 2.3.3 : Tables as files

- ➢ Currently, this system has 5 tables, so all the tables are stored in text files as shown in the image.



```
id--name--age--
11--John--32--
12--Kate--43--
13--Mark--43--
20--alex--23--
```

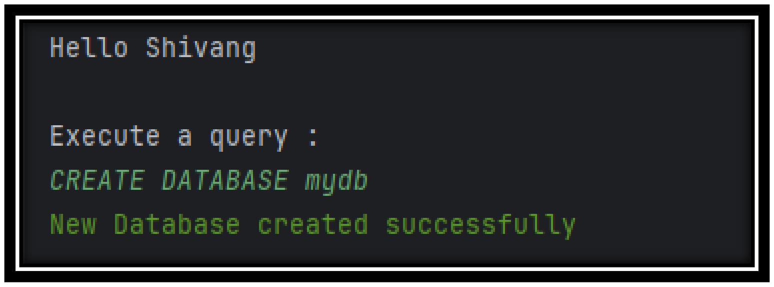
Figure 2.3.4 : Data stored in table

- ➢ All the data in files is stored in the format given above, I have used – as a delimiter to separate all the values.

Implementation of Queries (DDL & DML)

1. CREATE

- Create SQL command is a DDL command[15] used for creating databases and tables.

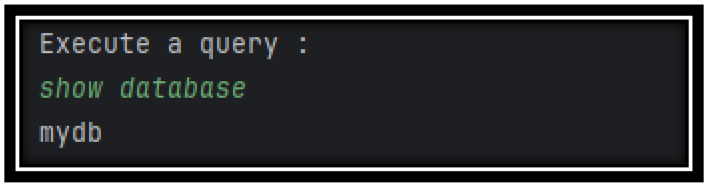


```
Hello Shivang

Execute a query :
CREATE DATABASE mydb
New Database created successfully
```

Figure 2.4.1.1 : Creating new database

- Currently there is no database in the system, so after logging in, we need to create a database first.
- So I have used create command to create a database named “mydb” in the application.
- A success message is shown which indicates that database has been created successfully.

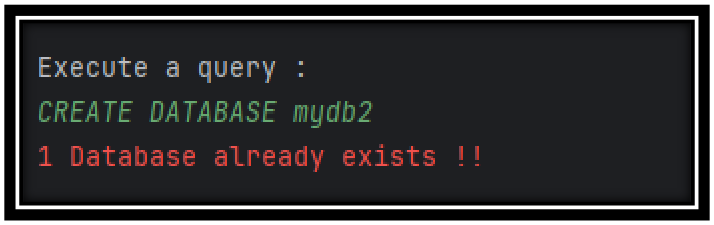
2. SHOW



```
Execute a query :
show database
mydb
```

Figure 2.4.2.1 : Show database

- Show database command is used to see all existing databases in the system.
- Since there is only one database named “mydb” which we created right now, it is named below.

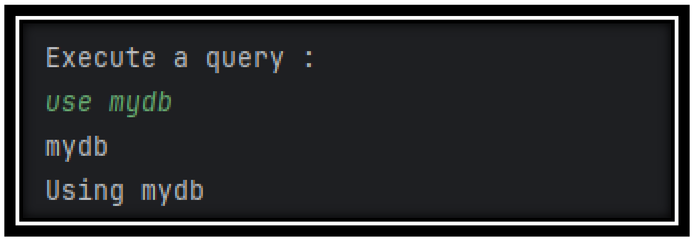


```
Execute a query :
CREATE DATABASE mydb2
1 Database already exists !!
```

Figure 2.4.2.2: Show database

- As mentioned, this system should only support creation of one database only so upon creation of other database, the application would show error message.

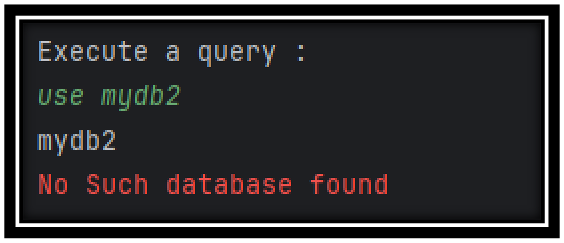
3. USE



```
Execute a query :
use mydb
mydb
Using mydb
```

Figure 2.4.3.1 : Use database correct

- After the creation of database, we need to select that particular database to perform further operations on it.
- The command for selecting the database is “use”.

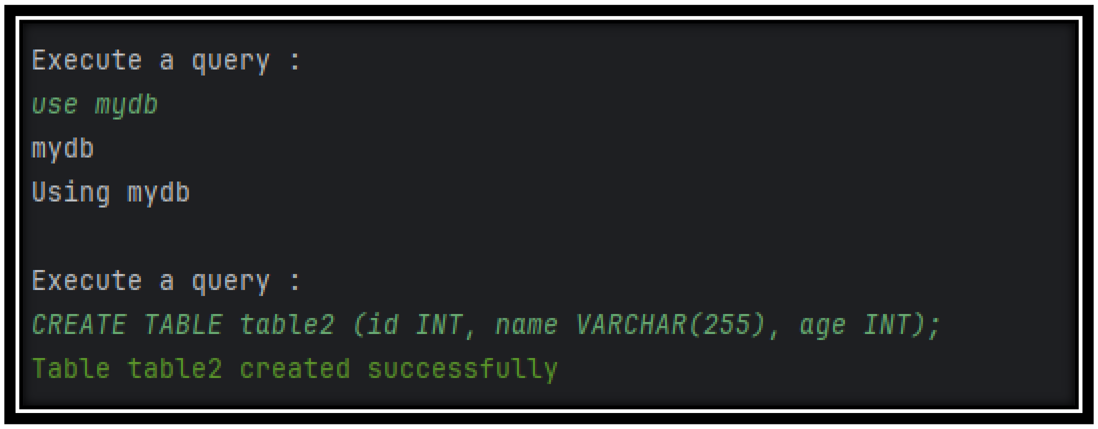


```
Execute a query :
use mydb2
mydb2
No Such database found
```

Figure 2.4.3.2 : Use database incorrect

- If database name is misspelled or database which doesn’t exist is passed than application would display error “No such databasefound”.

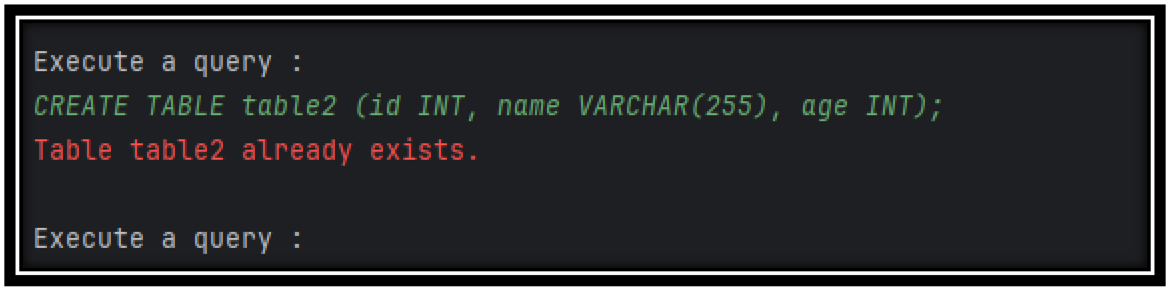
4. CREATE TABLE



```
Execute a query :
use mydb
mydb
Using mydb

Execute a query :
CREATE TABLE table2 (id INT, name VARCHAR(255), age INT);
Table table2 created successfully
```

Figure 2.4.4.1 : Create table query

- Create table query is used for creating tables is the database.
- Upon executing the query, table with table name and columns passed in the query will be created.
- On successful creation of table, success message will be displayed in the console as shown in the figure.

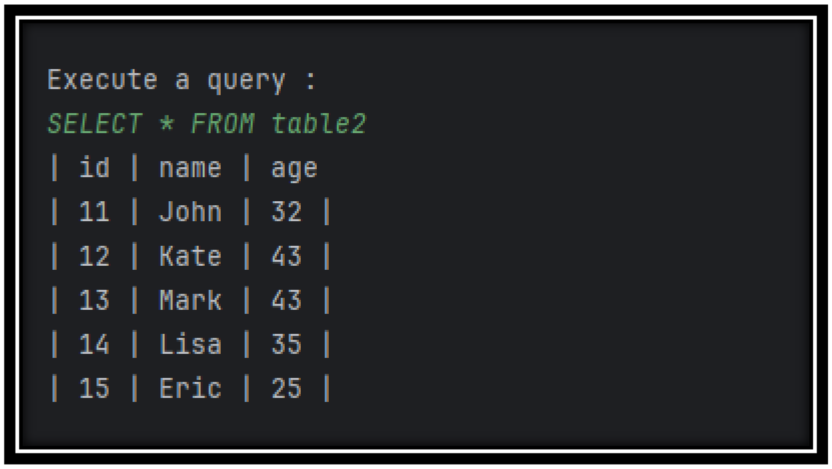


```
Execute a query :
CREATE TABLE table2 (id INT, name VARCHAR(255), age INT);
Table table2 already exists.

Execute a query :
```

Figure 2.4.4.2 : Create table incorrect

- If the table already exists in the database the application will show error message that table already exists.

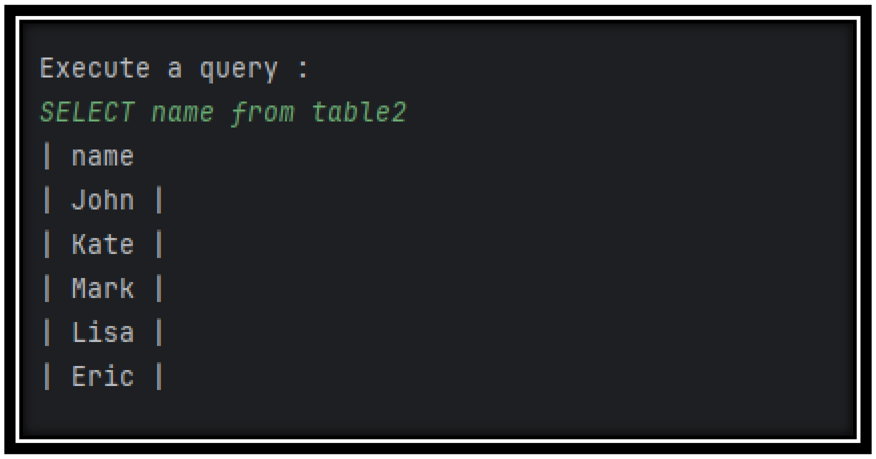
5. SELECT



```
Execute a query :
SELECT * FROM table2
| id | name | age
| 11 | John | 32 |
| 12 | Kate | 43 |
| 13 | Mark | 43 |
| 14 | Lisa | 35 |
| 15 | Eric | 25 |
```

Figure 2.4.5.1 : Select all query

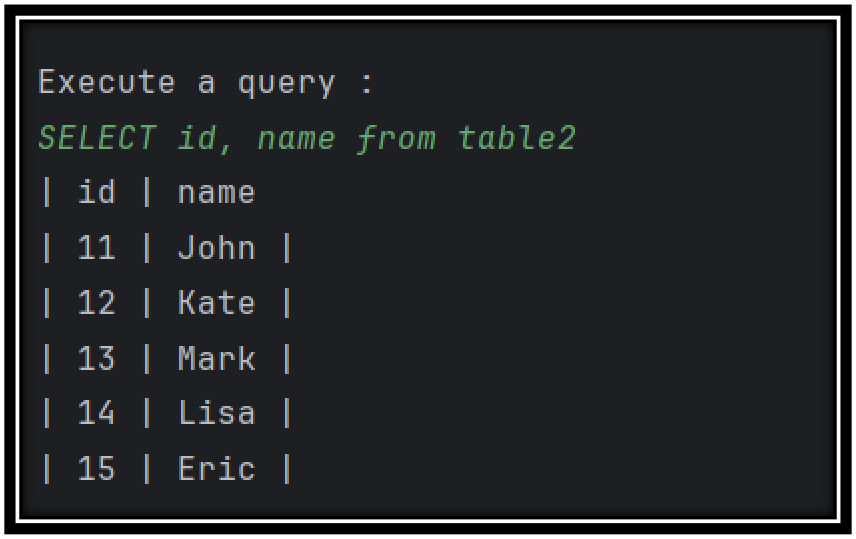
- Select query in SQL is used to select all data from the tables.
- It can also be used to select specific columns or cells from the table by providing conditions.



```
Execute a query :
SELECT name from table2
| name
| John |
| Kate |
| Mark |
| Lisa |
| Eric |
```

Figure 2.4.5.2 : Select specific column query

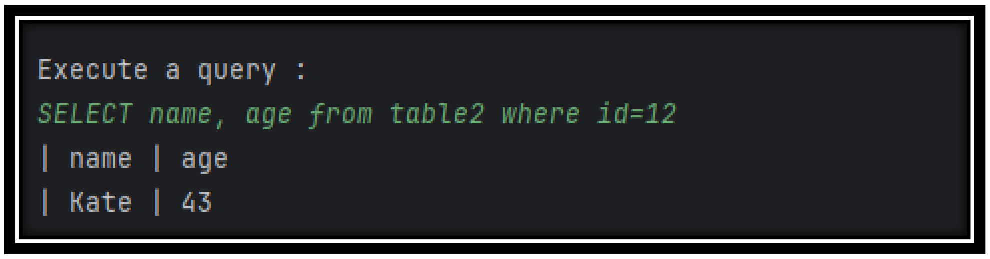
- ➢ The above query fetches the name column from table2.



```
Execute a query :
SELECT id, name from table2
| id | name
| 11 | John |
| 12 | Kate |
| 13 | Mark |
| 14 | Lisa |
| 15 | Eric |
```

Figure 2.4.5.3 : Select multiple columns from table

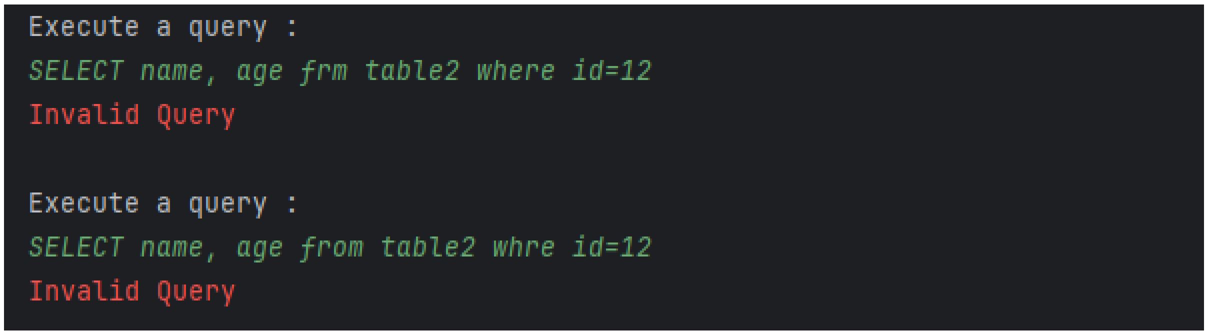
- ➢ The above query fetches both id and name column from table2.



```
Execute a query :
SELECT name, age from table2 where id=12
| name | age
| Kate | 43
```

Figure 2.4.5.4 : Select with where clause

➢ This select query fetches name and age columns where id = 2.

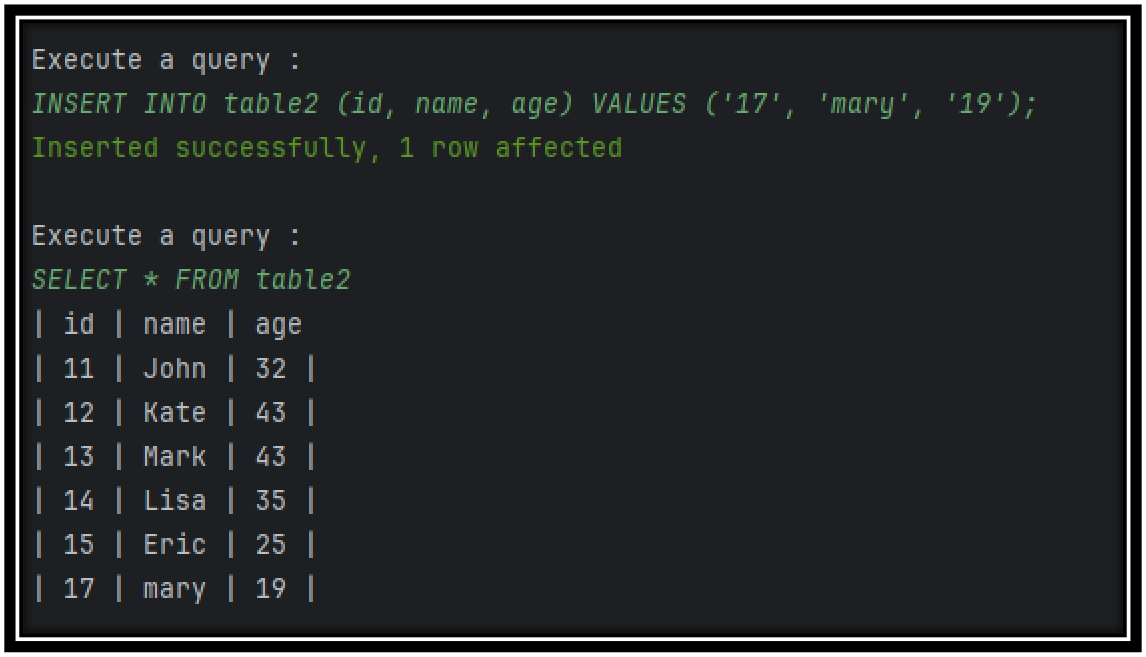


```
Execute a query :
SELECT name, age frm table2 where id=12
Invalid Query

Execute a query :
SELECT name, age from table2 whre id=12
Invalid Query
```

Figure 2.4.5.5 : Invalid select query

➢ There are few incorrect queries where spellings of where and from are misspelled so the application shows errors of Invalid query.

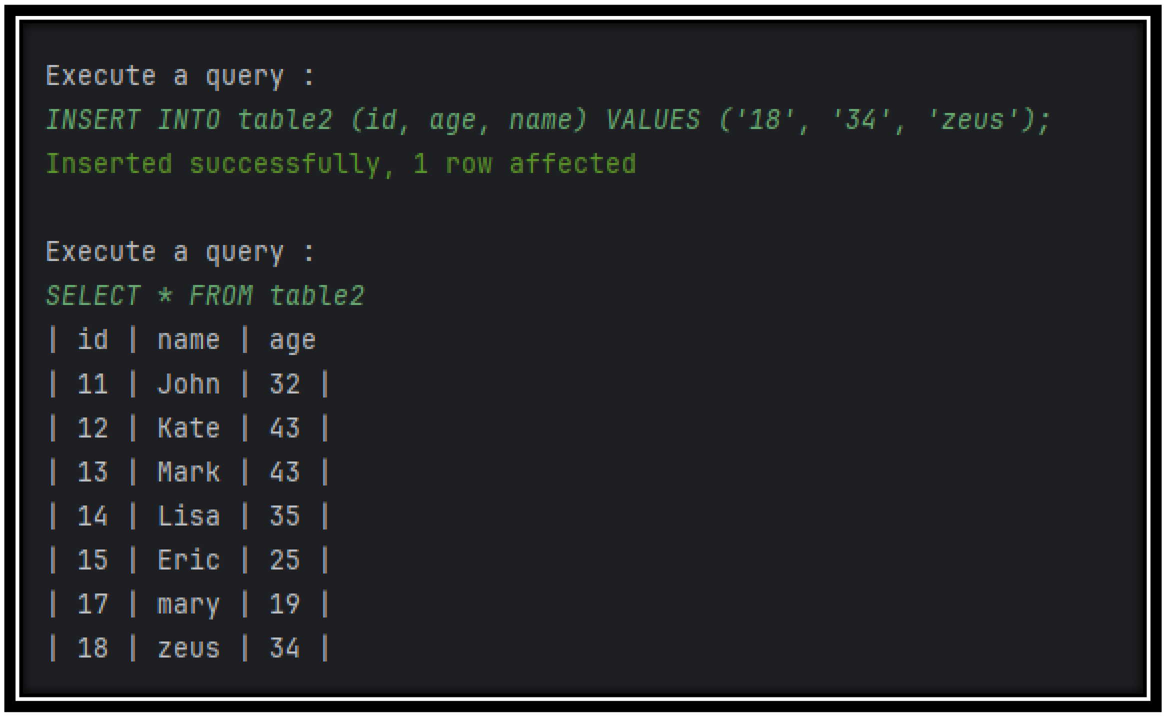
6. INSERT



```
Execute a query :
INSERT INTO table2 (id, name, age) VALUES ('17', 'mary', '19');
Inserted successfully, 1 row affected

Execute a query :
SELECT * FROM table2
| id | name | age
| 11 | John | 32 |
| 12 | Kate | 43 |
| 13 | Mark | 43 |
| 14 | Lisa | 35 |
| 15 | Eric | 25 |
| 17 | mary | 19 |
```

Figure 2.4.6.1 : Insert query

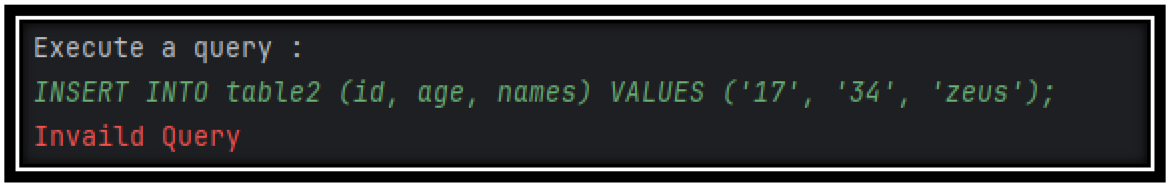
➢ Insert query in SQL is used to store data into the tables.
➢ The above shown SQL query inserts a new row in the table named table2.
➢ Following that, a select operation is performed whose output shows that a new row is added into the table.



```
Execute a query :
INSERT INTO table2 (id, age, name) VALUES ('18', '34', 'zeus');
Inserted successfully, 1 row affected

Execute a query :
SELECT * FROM table2
| id | name | age
| 11 | John | 32 |
| 12 | Kate | 43 |
| 13 | Mark | 43 |
| 14 | Lisa | 35 |
| 15 | Eric | 25 |
| 17 | mary | 19 |
| 18 | zeus | 34 |
```

Figure 2.4.6.2 : Insert query with shuffled column names

- Insert query even works if the column names are not in the order as specified during the table creation.
- Query parser will automatically map and store values in its correct place.



```
Execute a query :
INSERT INTO table2 (id, age, names) VALUES ('17', '34', 'zeus');
Invaild Query
```

Figure 2.4.6.3 : Incorrect insert query

- If the table columns are not found in the table then error message will be displayed to the user.

Implementation of Transaction

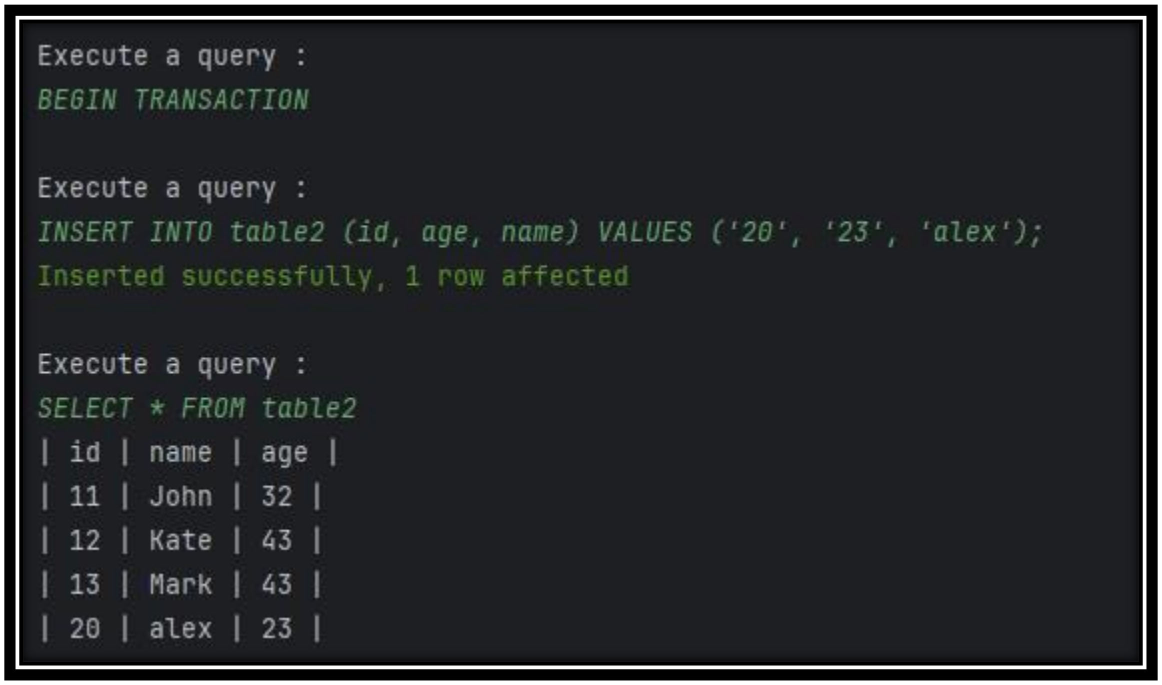
- ➢ Transaction[17] is a sequence of steps executed on database and are in the logical order.
- ➢ Transactions have four standard properties, generally called ACID properties.

  - ❖ **Atomicity**: makes ensures that every task carried out inside the work unit is finished successfully. If not, all prior operations are rolled back to their initial state and the transaction is aborted at the point of failure.
  - ❖ **Consistency**: guarantees that, following a successfully committed transaction, the database changes states appropriately.
  - ❖ **Isolation**: permits transactions to function independently of one another and in a transparent manner.
  - ❖ **Durability**: guarantees that, in the event of a system failure, the outcome or impact of a committed transaction remains intact.

- ➢ In our application, transactions have been implemented using a map of String and ArrayList called "buffer"[16] which works as a buffer and stores data during the transaction.

- ➢ Transactional Control Commands

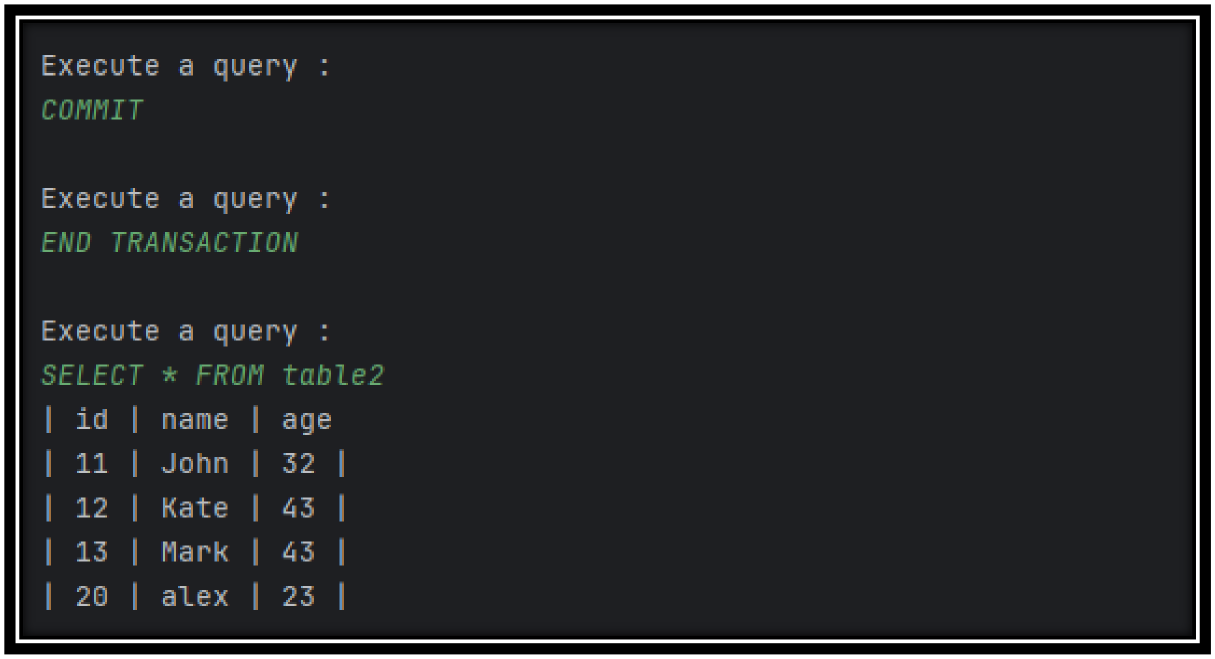
  - ❖ COMMIT
  - ❖ ROLLBACK
  - ❖ SAVEPOINT



```
Execute a query :
BEGIN TRANSACTION

Execute a query :
INSERT INTO table2 (id, age, name) VALUES ('20', '23', 'alex');
Inserted successfully, 1 row affected

Execute a query :
SELECT * FROM table2
| id | name | age |
| 11 | John | 32 |
| 12 | Kate | 43 |
| 13 | Mark | 43 |
| 20 | alex | 23 |
```

Figure 2.5.1: Starting transaction

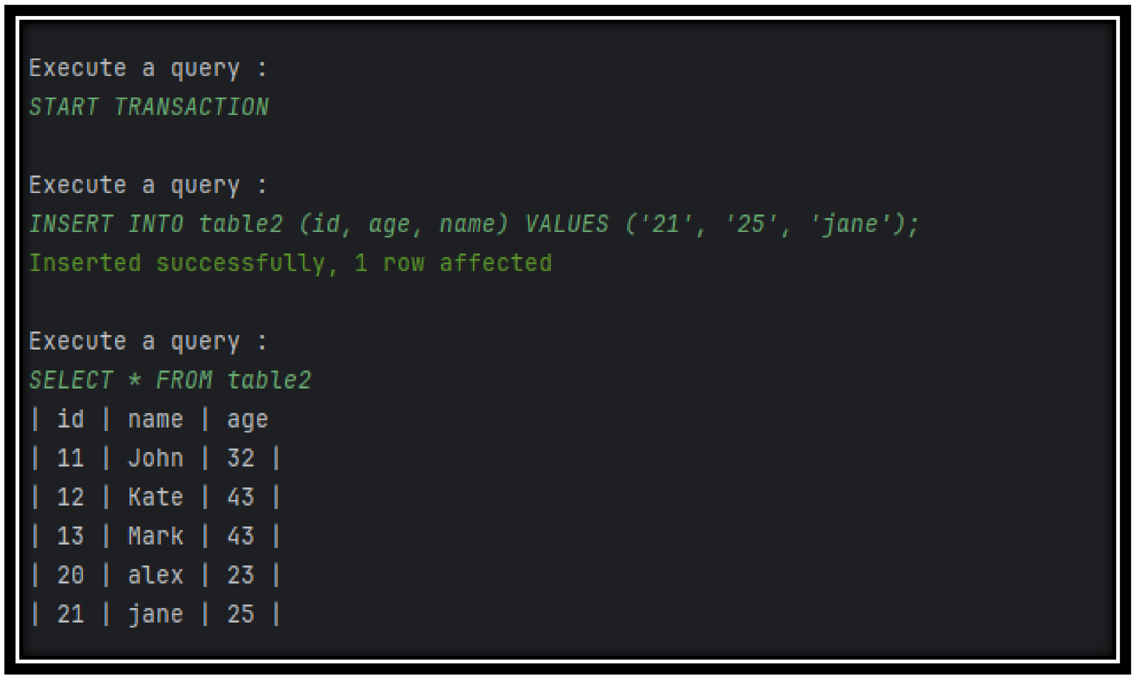
- Begin transaction command is used to indicate that a transaction is about to happen.
- Following that an insert query in performed on the table2 and a new row is inserted into the table.
- After that a select operation is performed to check if the data has been successfully inserted or not.
- Right now, the data is inserted into buffer and not in the actual database.



```
Execute a query :
COMMIT

Execute a query :
END TRANSACTION

Execute a query :
SELECT * FROM table2
| id | name | age
| 11 | John | 32 |
| 12 | Kate | 43 |
| 13 | Mark | 43 |
| 20 | alex | 23 |
```

Figure 2.5.2: Commit transaction

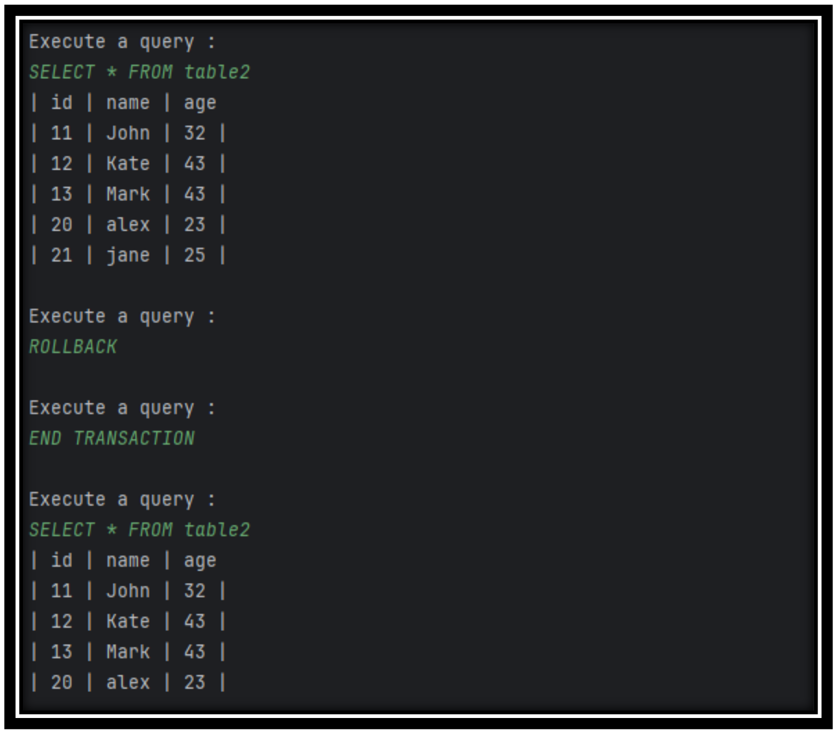
- Now commit is executed which indicated that a transaction is to be committed and the changes are to be reflected into the database.
- Then after, END TRANSACTION is performed to indicate that transaction has ended.



```
Execute a query :
START TRANSACTION

Execute a query :
INSERT INTO table2 (id, age, name) VALUES ('21', '25', 'jane');
Inserted successfully, 1 row affected

Execute a query :
SELECT * FROM table2
| id | name | age
| 11 | John | 32 |
| 12 | Kate | 43 |
| 13 | Mark | 43 |
| 20 | alex | 23 |
| 21 | jane | 25 |
```

Figure 2.5.3 : Starting transaction

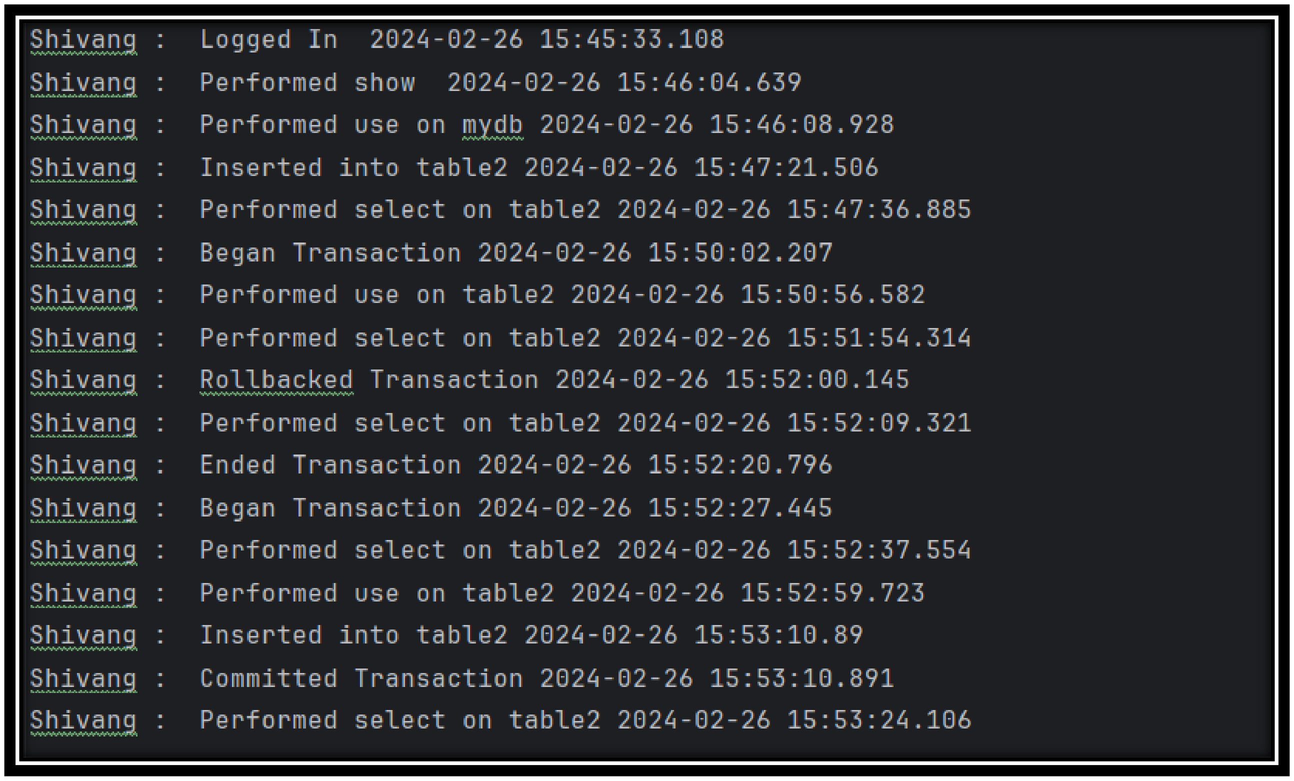
- The above image shows a new transaction has stared and a new row is added into the table2.
- Still the data is in buffer as transaction has not been committed.



```
Execute a query :
SELECT * FROM table2
| id | name | age
| 11 | John | 32 |
| 12 | Kate | 43 |
| 13 | Mark | 43 |
| 20 | alex | 23 |
| 21 | jane | 25 |

Execute a query :
ROLLBACK

Execute a query :
END TRANSACTION

Execute a query :
SELECT * FROM table2
| id | name | age
| 11 | John | 32 |
| 12 | Kate | 43 |
| 13 | Mark | 43 |
| 20 | alex | 23 |
```

Figure 2.5.4 : Transaction rollback

- ➢ Now instead of commit, rollback is performed to indicate that transaction has to be rolledback and no changes should be reflected in the database.
- ➢ On analyzing the select output of table2, it can be clearly seen that transaction has been rolled back and a new row is not appended into the database.



```
Shivang :  Logged In  2024-02-26 15:45:33.108
Shivang :  Performed show  2024-02-26 15:46:04.639
Shivang :  Performed use on mydb 2024-02-26 15:46:08.928
Shivang :  Inserted into table2 2024-02-26 15:47:21.506
Shivang :  Performed select on table2 2024-02-26 15:47:36.885
Shivang :  Began Transaction 2024-02-26 15:50:02.207
Shivang :  Performed use on table2 2024-02-26 15:50:56.582
Shivang :  Performed select on table2 2024-02-26 15:51:54.314
Shivang :  Rollbacked Transaction 2024-02-26 15:52:00.145
Shivang :  Performed select on table2 2024-02-26 15:52:09.321
Shivang :  Ended Transaction 2024-02-26 15:52:20.796
Shivang :  Began Transaction 2024-02-26 15:52:27.445
Shivang :  Performed select on table2 2024-02-26 15:52:37.554
Shivang :  Performed use on table2 2024-02-26 15:52:59.723
Shivang :  Inserted into table2 2024-02-26 15:53:10.89
Shivang :  Committed Transaction 2024-02-26 15:53:10.891
Shivang :  Performed select on table2 2024-02-26 15:53:24.106
```

Figure 2.5.5 : Logs

- ➢ All the transactions are recorded in the logs file.
- ➢ Whenever each transaction or operation in performed on the database it is entered in the database logs.

References

[1] Dalhousie University. (Online). Available: https://www.dal.ca/

[2] Saint Mary's University. (Online). Available: https://www.smu.ca/

[3] Mount Saint Vincent University. (Online). Available: https://www.msvu.ca/

[4] Gleek.io. (Online). Available: https://www.gleek.io/blog/erd-database-design

[5] Sisense Documentation. (Online). Available:
https://docs.sisense.com/main/SisenseLinux/chasm-and-fan-traps.htm

[6] GeeksforGeeks. (Online). Available: https://www.geeksforgeeks.org/weak-entity-set-in-er-diagrams/

[7] GeeksforGeeks. (Online). Available: https://www.geeksforgeeks.org/enhanced-er-model/

[8] Stackify. (Online). Available: https://stackify.com/solid-design-principles/

[9] Stackify. (Online). Available: https://stackify.com/solid-design-open-closed-principle/

[10] Oracle. (Online). Available:
https://docs.oracle.com/javase/tutorial/java/concepts/interface.html

[11] Stackify. (Online). Available: https://stackify.com/solid-design-liskov-substitution-principle/

[12] My Java Code. (Online). Available: https://www.myjavacode.com/how-to-generate-md5-hash-in-java/

[13] Stackify. (Online). Available: https://stackify.com/interface-segregation-principle/

[14] Stackify. (Online). Available: https://stackify.com/dependency-inversion-principle/
[15] Scaler. (Online). Available: https://www.scaler.com/topics/dbms/sql-commands/

[16] SQL Shack. (Online). Available: https://www.sqlshack.com/insight-into-the-sql-server-buffer-cache/

[17] Tutorialspoint. (Online). Available: https://www.tutorialspoint.com/sql/sql-transactions.htm