

CSCI 5408 ASSIGNMENT 2

Prepared by: Sushank Saini
B00922727

Problem 1

Summary

The research paper[1] defines distributed database as a database that is physically scattered over multiple computer locations but is logically a single database. The designing of a distributed database is a three-phase process. The first phase is the initial design which involves fragmentation and allocation algorithms that minimize the total transaction cost of a set of given transactions. Several studies have been done to find better ways for fragmentation and allocation in distributed database systems that enhance the reliability, availability, and performance of such systems. For instance, one researcher's different approaches for vertical fragmentation resulted in minimal communication cost. Another researcher introduced an algorithm that used mixed fragmentation to reduce the number of disk accesses required to process the distributed transactions. The second phase is redesign. Redesigning requires generating new fragmentation and allocation schemes from the existing fragmentation and allocation schemes to account for the logical and physical modifications in the distributed database environment. The last and the third phase is materialization of the design in which some sets of sequential operations are performed to shape up the current design.

Fragmentation, which is a technique to divide a class or a relation of a database into two or more partitions without loss of information, has its advantages and disadvantages. The advantages include increased efficiency of the database system as well as security and privacy of the data. On the other hand, fragmentation leads to high speeds for data access, expensive reconstruction for recursive fragmentations and possibility of failure of site due to lack of back-up. There are different types of fragmentations namely horizontal, vertical, and mixed fragmentation. Horizontal fragmentation consists of partitioning a relation or class into disjoint tuples or instances stored at different nodes, such that each unique tuple has the same attributes. On the contrary, vertical fragmentation divides a relation into sets of columns with a common primary key attribute. A third type of fragmentation exists called mixed fragmentation which involves combination of both horizontal and vertical fragmentation. This fragmentation is the most complicated to manage. The research paper illustrates these types through the example of a Human Resources table. For fragmentation to be without errors, it needs to fulfill the correctness rules of fragmentation. These rules emphasize completeness, reconstruction, and disjointness to ensure that all data items are preserved and can be reconstructed.

Lastly, the paper presents a comparative analysis of different types of fragmentations which would be of great value to the researchers who would want to study the distributed databases.

Scope for improvements

Though the report provides a comparative analysis of the three types of fragmentation, it falls short of providing any technical details on the evaluation criteria and decision-making process for choosing an appropriate fragmentation strategy. Consequently, the report fails to make any recommendations for the best type of data fragmentation for distributed database design.

With respect to concepts, the report briefly mentions fragmentation algorithms and the concepts utilized in those algorithms. However, it does not provide a thorough explanation of how they work. For example, in the approach provided by Seung-Jin Lim (2016), the report does not explain what "maximal locality of query evaluation" is [1,p.725]. Besides this, the report does not give a detailed explanation using examples for the correctness rules of fragmentation.

Problem 2

Gitlab Link

https://git.cs.dal.ca/sushank/csci5408_s23_b00922727_sushank_saini.git

Overview

The DBMSLite is a lightweight prototype of a DBMS tool like MySQL. Once the user logs in, it provides them with the functionality to create a database and a table. In addition, it provides the execution of standard Data Manipulation Language statements such as insert, delete, select, and update.

Design principles used for program development

I have used **SOLID** design principles for developing my program(see **Figure 1**).

All the classes follow the **single responsibility principle(SRP)**. They do only one thing i.e., have a single responsibility. The SignUp class saves the user login details and user's security answer to persistence storage. The TwoFactorAuthentication class allows a user to login by checking their password and the security answer. The Helper class provides a utility method to hash the password of the user. The Logger class has a method that saves logs to persistence storage. The Query class has methods that check the type of query and accordingly creates a table or database, inserts into a table, updates a table, deletes from a table, and fetches records from a table. The Write class does the write operations on persistence storage. Since all classes follow SRP, consequently, they adhere to the **open and close principle** too.

To implement the **Liskov substitution, interface segregation** and **dependency inversion**, I have **coded to the interfaces**. All classes are dependent on the abstractions, not concretions. For instance, the Query class implements the IQuery interface.

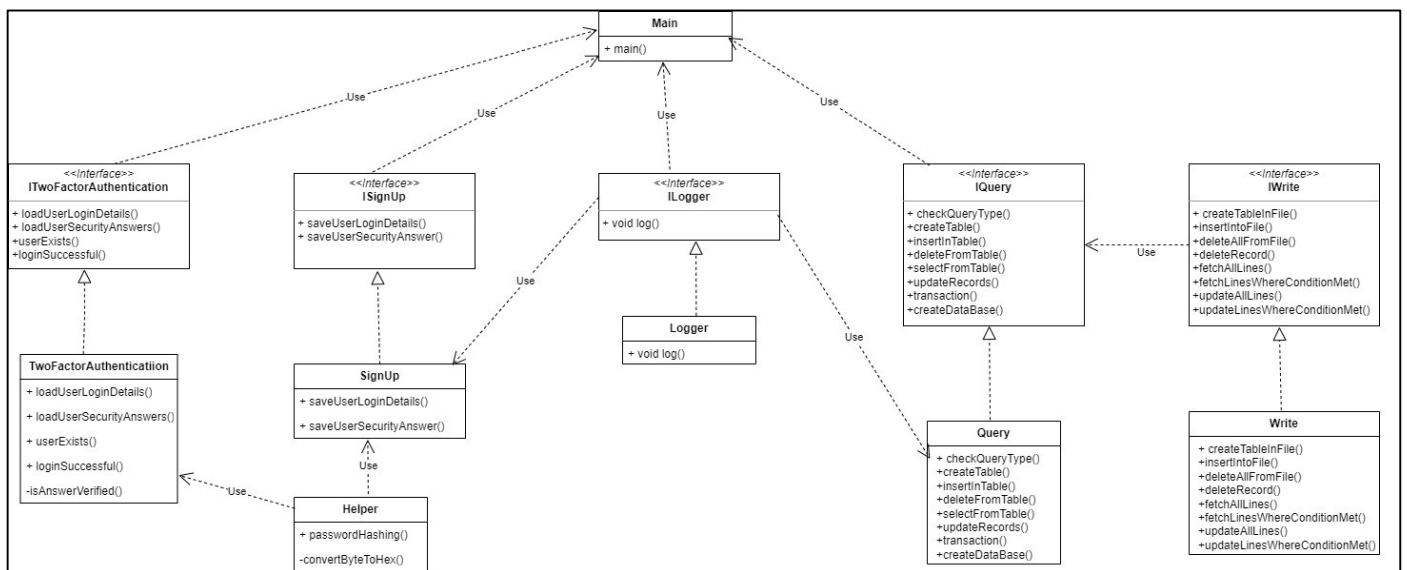
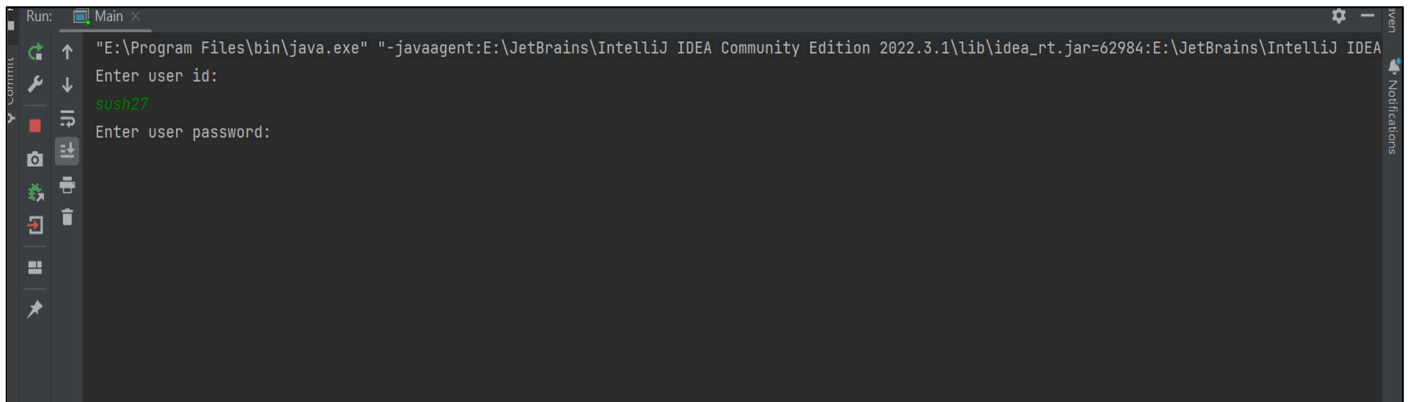


Figure 1: Class Design for lightweight DBMS software.

Explanation of the program execution

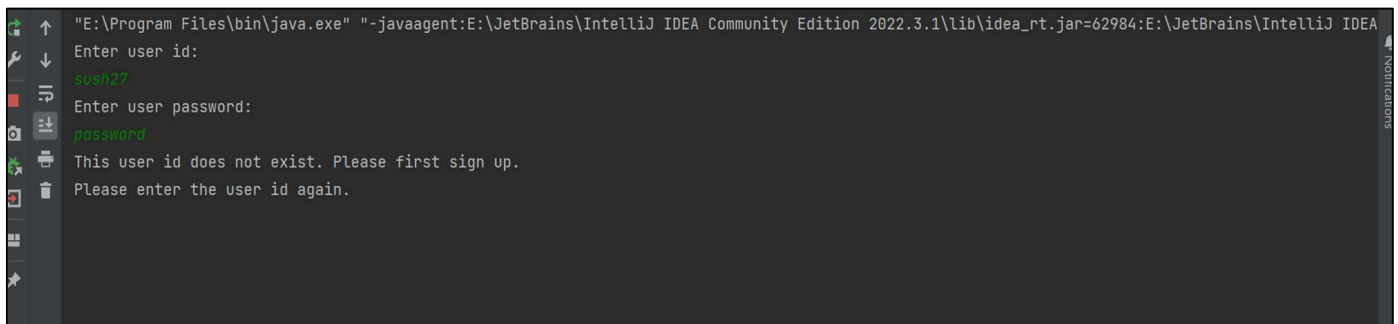
The program starts by asking the user for their user id and password(see **Figure 2**).



```
Run: "E:\Program Files\bin\java.exe" "-javaagent:E:\JetBrains\IntelliJ IDEA Community Edition 2022.3.1\lib\idea_rt.jar=62984:E:\JetBrains\IntelliJ IDEA
Enter user id:
sush27
Enter user password:
```

Figure 2: Prompting user for their credentials.

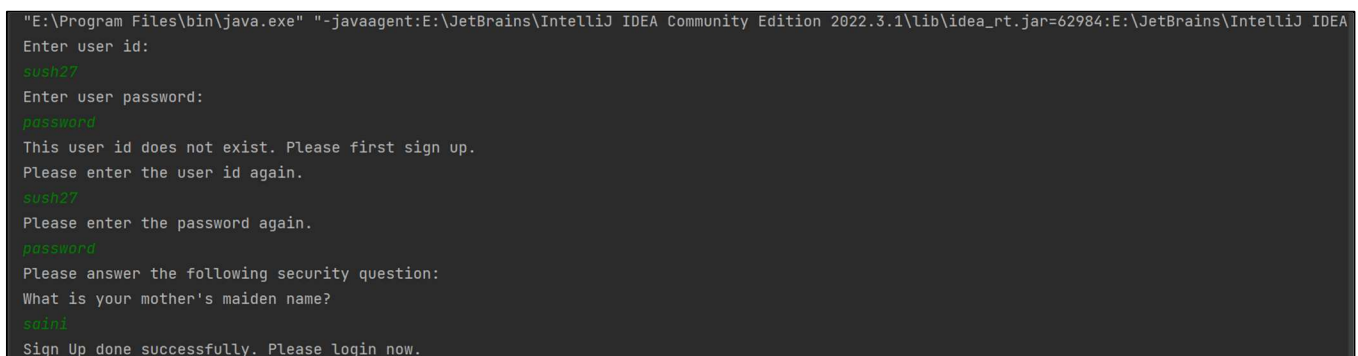
Since the user id does not exist, the user is asked to sign up first(**Figure 3**).



```
"E:\Program Files\bin\java.exe" "-javaagent:E:\JetBrains\IntelliJ IDEA Community Edition 2022.3.1\lib\idea_rt.jar=62984:E:\JetBrains\IntelliJ IDEA
Enter user id:
sush27
Enter user password:
password
This user id does not exist. Please first sign up.
Please enter the user id again.
```

Figure 3: Asking user to sign up.

For signing up, the user provides the user id, password, and answer to a security question(**Figure 4**). On successful sign up, the user login details are saved in the files userLoginDetails.txt and userSecurityAnswer.txt(**Figure 5**).



```
"E:\Program Files\bin\java.exe" "-javaagent:E:\JetBrains\IntelliJ IDEA Community Edition 2022.3.1\lib\idea_rt.jar=62984:E:\JetBrains\IntelliJ IDEA
Enter user id:
sush27
Enter user password:
password
This user id does not exist. Please first sign up.
Please enter the user id again.
sush27
Please enter the password again.
password
Please answer the following security question:
What is your mother's maiden name?
saini
Sign Up done successfully. Please login now.
```

Figure 4: Successful sign up

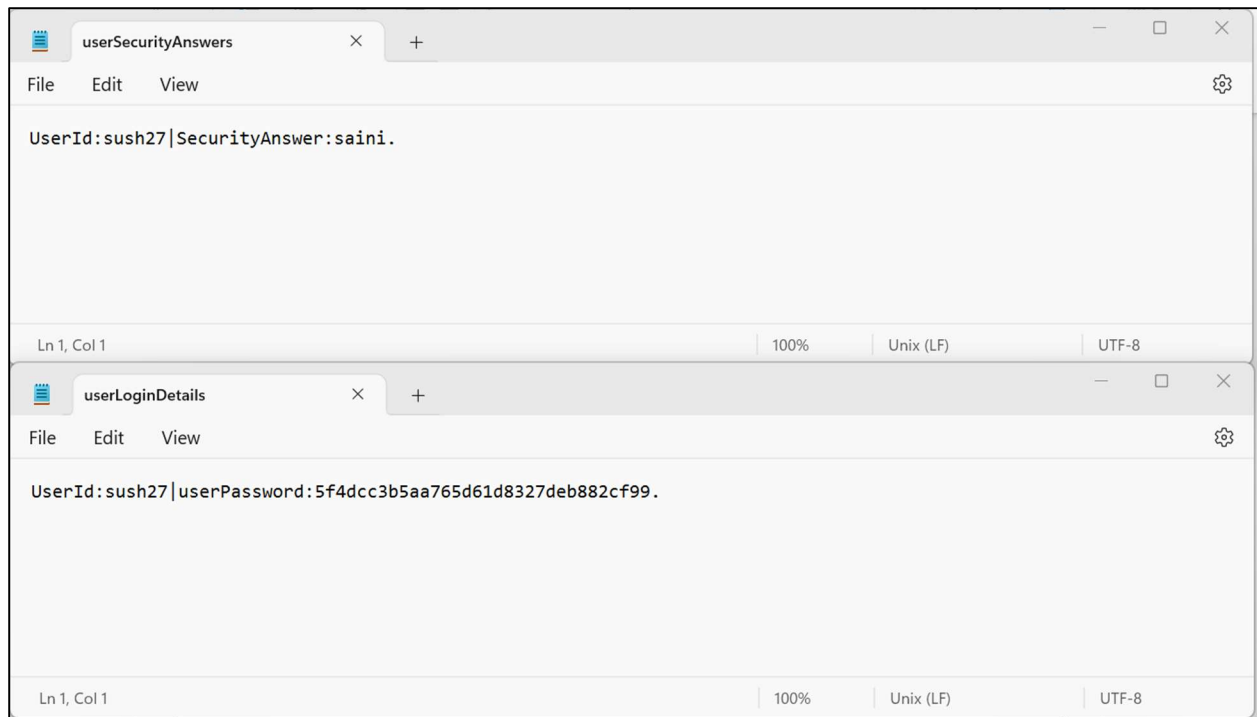


Figure 5: User details saved to a file.

To store the password securely, I have used the **MD5 hashing algorithm**[2] and to convert the hashed password into readable format I used **string formatter**[3].

Upon successful login, the user is asked to enter their SQL query(**Figure 6**).

```
Sign Up done successfully. Please login now.
Please enter the user id again:
sush27
Please enter the password again:
password
Answer to the Security Question:
What is your mother's maiden name?
saini
Enter your SQL query
```

Figure 6: Asking user to enter a SQL query.

For **parsing the SQL statements**, I have utilized **Regex**[4][5] and for matching the patterns, I have used **Pattern-Matcher**[6].

Creating a database

When the user provides a query to create a database(**Figure 7**), a directory with the name of the database given in the SQL query is created on the hard drive(**Figure 8**).

```
Enter your SQL query
CREATE DATABASE EmployeeDB;
Database created successfully!
Do you wish to continue? Y/N
```

Figure 7: SQL query to create a database.

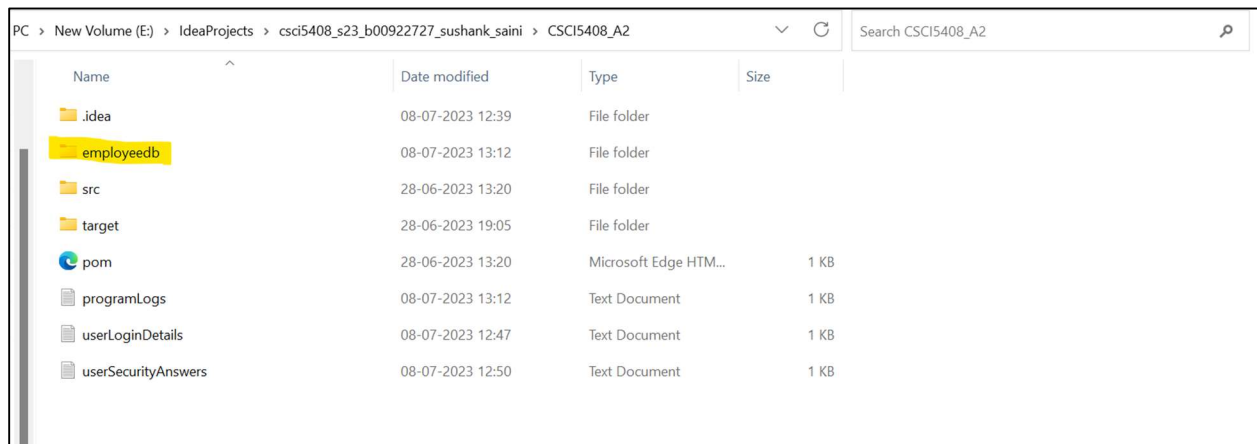


Figure 8: Directory created with the database name.

Creating a table

When user provides a SQL query to create a table(**Figure 9**), the file having the table name is created which consists of the attribute names(**Figure 10**). Along with this, another file gets created that stores the datatype information of the attributes(**Figure 11**).

```
Enter your SQL query
CREATE TABLE Employee (id int, name varchar(30));
Table created successfully!
Do you wish to continue? Y/N
```

Figure 9: SQL query to create table provided by the user.



Figure 10: Empty table with its attributes created.



Figure 11: File created that contains the datatype information of the attributes.

Inserting into a table

When a user provides an insert SQL query(**Figure 12**) , a new record is inserted into the table file(**Figure 13**).

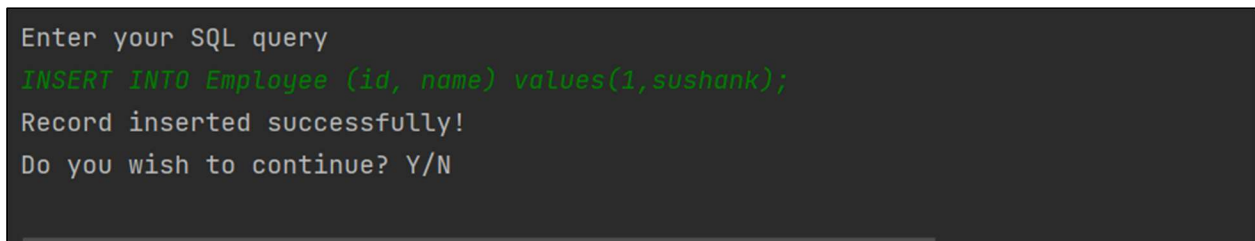


Figure 12: SQL insert query.



The screenshot shows a text editor window with the title 'employee'. The menu bar includes 'File', 'Edit', and 'View'. The main text area contains a table with two columns: 'id' and 'name'. The first row of the table has the values '1' and 'sushank'. The status bar at the bottom indicates 'Ln 1, Col 1', '100%', 'Unix (LF)', and 'UTF-8'.

id	name
1	sushank

Figure 13: Record inserted in the file.

Fetching records from a table

There are two cases:

- *Fetching all records:* The select statement will fetch all the records present in the table (**Figure 14**).

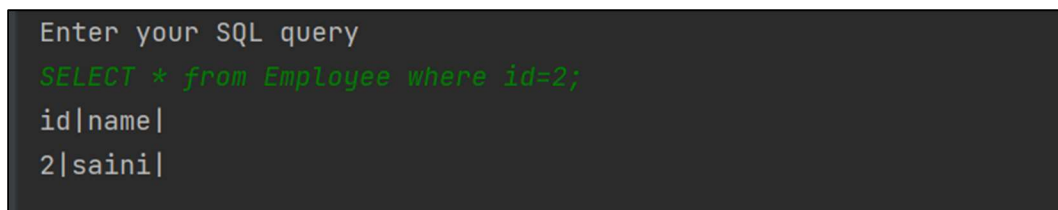


The screenshot shows a terminal window with the prompt 'Enter your SQL query'. The query entered is 'SELECT * from Employee;'. The output is a table with two columns: 'id' and 'name'. The first row has values '1' and 'sushank', the second row has '2' and 'saini', and the third row has '3' and 'sush'.

```
Enter your SQL query
SELECT * from Employee;
id|name|
1|sushank|
2|saini|
3|sush|
```

Figure 14: Fetching all records.

- *Fetching with a where clause:* The select statement has a where clause and therefore fetches only those records(**Figure 15**).



The screenshot shows a terminal window with the prompt 'Enter your SQL query'. The query entered is 'SELECT * from Employee where id=2;'. The output is a table with two columns: 'id' and 'name'. The first row has values '2' and 'saini'.

```
Enter your SQL query
SELECT * from Employee where id=2;
id|name|
2|saini|
```

Figure 15: Selecting records with a where clause.

Updating a record

There are two cases:

- *Updating with a where clause:* Only those records get updated which satisfy where condition(**Figure 16**).

```
Enter your SQL query
SELECT * from Employee;
id|name|
1|sushank|
2|saini|
3|sush|

Do you wish to continue? Y/N
y
Enter your SQL query
UPDATE Employee SET name=shooshank WHERE id=1;
Record(s) updated successfully!
Do you wish to continue? Y/N
y
Enter your SQL query
SELECT * from Employee;
id|name|
1|shooshank|
2|saini|
3|sush|
```

Figure 16: Updating with a where clause.

- *Updating without a where clause:* All the values of that attribute get updated(**Figure 17**).

```
Enter your SQL query
SELECT * from Employee;
id|name|
1|shooshank|
2|saini|
3|sush|

Do you wish to continue? Y/N
y
Enter your SQL query
UPDATE Employee SET name=shooshank;
Record(s) updated successfully!
Do you wish to continue? Y/N
y
Enter your SQL query
SELECT * from Employee;
id|name|
1|shooshank|
2|shooshank|
3|shooshank|

Do you wish to continue? Y/N
```

Figure 17: Updating without a where clause.

Deleting records

There are two cases:

- *Deleting a particular record*: The SQL query has a where clause and only that record gets deleted(**Figure 18**).

```
Enter your SQL query
SELECT * from Employee;
id|name|
1|shooshank|
2|shooshank|
3|shooshank|

Do you wish to continue? Y/N
y
Enter your SQL query
DELETE FROM Employee where id=2
Record(s) deleted successfully!
Do you wish to continue? Y/N
y
Enter your SQL query
SELECT * from Employee;
id|name|
1|shooshank|
3|shooshank|
```

Figure 18: Deleting a particular record.

- *Deleting all records:* The SQL query does not have a where clause and therefore, all the records are deleted(**Figure 19**).

```

Enter your SQL query
SELECT * from Employee;
id|name|
1|shooshank|
3|shooshank|

Do you wish to continue? Y/N
y
Enter your SQL query
DELETE FROM Employee;
Record(s) deleted successfully!
Do you wish to continue? Y/N
SELECT * from Employee;
Enter your SQL query
SELECT * from Employee;
id|name|

```

Figure 19: Deleting all records.

Transaction

In case of a transaction, if there is a commit, the queries inside the transaction block are executed(**Figure 20**) else there is a rollback, and no operations are executed(**Figure 21**).

```

Enter your SQL query
BEGIN TRANSACTION;INSERT INTO Employee (id, name) values(1,sushank);SELECT * from Employee;COMMIT;
Record inserted successfully!
id|name|
1|sushank|

Transaction completed successfully!
Do you wish to continue? Y/N

```

Figure 20: Transaction committed.

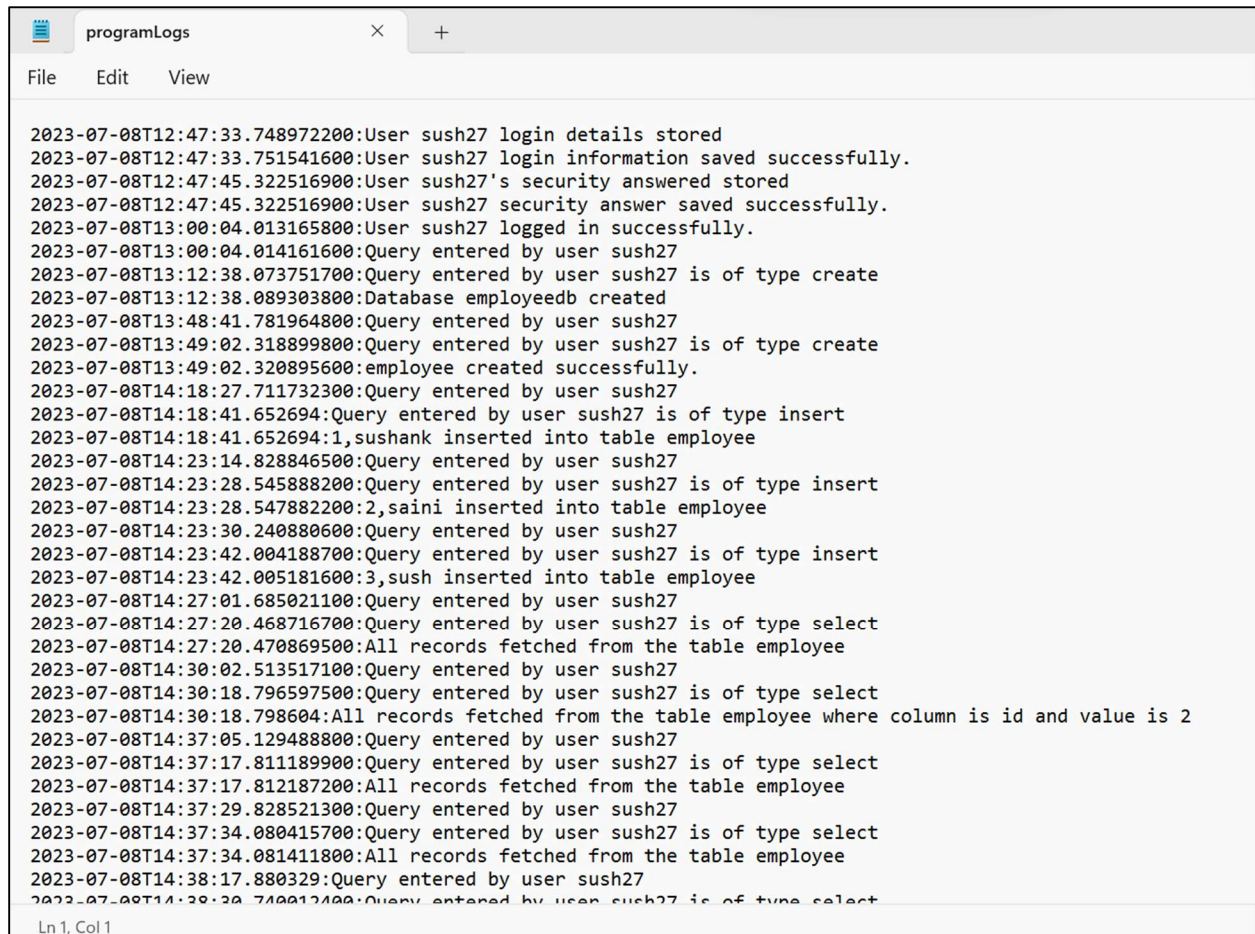
```

Enter your SQL query
BEGIN TRANSACTION;DELETE FROM Employee where name=sushank;SELECT * from Employee;ROLLBACK;
Transaction could not be completed successfully due to rollback!
Do you wish to continue? Y/N

```

Figure 21: Transaction rollbacked

For each execution in the program, everything is **logged in the programLogs.txt** file(Figure 22).



```
2023-07-08T12:47:33.748972200:User sush27 login details stored
2023-07-08T12:47:33.751541600:User sush27 login information saved successfully.
2023-07-08T12:47:45.322516900:User sush27's security answered stored
2023-07-08T12:47:45.322516900:User sush27 security answer saved successfully.
2023-07-08T13:00:04.013165800:User sush27 logged in successfully.
2023-07-08T13:00:04.014161600:Query entered by user sush27
2023-07-08T13:12:38.073751700:Query entered by user sush27 is of type create
2023-07-08T13:12:38.089303800:Database employeeedb created
2023-07-08T13:48:41.781964800:Query entered by user sush27
2023-07-08T13:49:02.318899800:Query entered by user sush27 is of type create
2023-07-08T13:49:02.320895600:employee created successfully.
2023-07-08T14:18:27.711732300:Query entered by user sush27
2023-07-08T14:18:41.652694:Query entered by user sush27 is of type insert
2023-07-08T14:18:41.652694:1,sushank inserted into table employee
2023-07-08T14:23:14.828846500:Query entered by user sush27
2023-07-08T14:23:28.545888200:Query entered by user sush27 is of type insert
2023-07-08T14:23:28.547882200:2,saini inserted into table employee
2023-07-08T14:23:30.240880600:Query entered by user sush27
2023-07-08T14:23:42.004188700:Query entered by user sush27 is of type insert
2023-07-08T14:23:42.005181600:3,sush inserted into table employee
2023-07-08T14:27:01.685021100:Query entered by user sush27
2023-07-08T14:27:20.468716700:Query entered by user sush27 is of type select
2023-07-08T14:27:20.470869500:All records fetched from the table employee
2023-07-08T14:30:02.513517100:Query entered by user sush27
2023-07-08T14:30:18.796597500:Query entered by user sush27 is of type select
2023-07-08T14:30:18.798604:All records fetched from the table employee where column is id and value is 2
2023-07-08T14:37:05.129488800:Query entered by user sush27
2023-07-08T14:37:17.811189900:Query entered by user sush27 is of type select
2023-07-08T14:37:17.812187200:All records fetched from the table employee
2023-07-08T14:37:29.828521300:Query entered by user sush27
2023-07-08T14:37:34.080415700:Query entered by user sush27 is of type select
2023-07-08T14:37:34.081411800:All records fetched from the table employee
2023-07-08T14:38:17.880329:Query entered by user sush27
2023-07-08T14:38:20.740012400:Query entered by user sush27 is of type select
```

Figure 22: Logs

Constraints

- Users can only use the same name for table once.
- Only one record can be inserted at a time.
- All the queries should be in a single line.

Limitations

- For create table command the program does not read and store the constraints of the column but only the column names and their datatypes.
- Insertion does not work with where clause and simply adds the record as the next record in the table.
- There is no check for key constraints(primary, foreign etc.). This will lead to redundancy.

References

- [1] A. H. Al-Sanhani, A. Hamdan, A. B. Al-Thaher and A. Al-Dahoud, "A comparative analysis of data fragmentation in distributed database," in *2017 8th International Conference on Information Technology (ICIT)*, Amman, Jordan, 2017, pp. 724-729, doi: 10.1109/ICITECH.2017.8079934.
- [2] Baeldung, "MD5 Hashing in Java", *Baeldung* [Online]. Available: <https://www.baeldung.com/java-md5>, January 09,2021.[Accessed: June 30,2023].
- [3] Baeldung, "Guide to java.util.Formatter", *Baeldung* [Online]. Available: <https://www.baeldung.com/java-string-formatter>, May 06,2023. [Accessed: July 01,2023].
- [4] GeeksforGeeks , "How to write Regular Expressions?", GeeksforGeeks [Online]. Available: <https://www.geeksforgeeks.org/write-regular-expressions/>. [Accessed: July 03,2023].
- [5] *Regular expressions 101* [Online]. Available: <https://regex101.com/>. [Accessed: July 03,2023].
- [6] "Class Pattern", *Java™ Platform Standard Ed. 8* [Online]. Available: <https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html>. [Accessed: July 03,2023].