



**DALHOUSIE
UNIVERSITY**

CSCI 5408

Data Management, Warehousing, and Analytics

Assignment 2: Review paper and Light DBMS

Name : Keyur Pradipbhai Khant

Banner ID : B00935171

Email ID: keyur.khant@dal.ca

Overview

In this assignment, I have two tasks to complete. The first task requires you to write a summary or review of a research paper titled "A comparative analysis of data fragmentation in a distributed database." It is essential for me to thoroughly read the paper and provide a concise overview of its main points, findings, and contributions.

The second task involves creating my own lightweight database management system (DBMS) using Java. I am expected to design and implement a simple DBMS that can handle basic data storage and retrieval operations. This task will require me to apply your understanding of database concepts and programming skills in Java to develop a functional DBMS.

Self Declaration

I, Keyur Khant, hereby declare that in assignment 2 of the CSCI 5408 course, data scraping has not been performed programmatically or utilizing any online or offline tools. Instead, the webpages or domains mentioned in this document have been manually visited, and only relevant information has been gathered strictly for educational purposes. No personal contact numbers, email addresses, or individual names have been extracted.

Problem 1

Paper:

A. H. Al-Sanhani, A. Hamdan, A. B. Al-Thaher and A. Al-Dahoud, "A comparative analysis of data fragmentation in distributed database," 2017 8th International Conference on Information Technology (ICIT), Amman, Jordan, 2017, pp. 724-729, doi: 10.1109/ICITECH.2017.8079934.

Summary:

This research paper provides a comprehensive overview of data fragmentation in distributed database systems. The paper explores the different types of fragmentation, compares their characteristics, and suggests the most suitable approach. The study aims to contribute to the field of efficient distributed database design [1].

The paper highlights the significance of distributed systems in delivering expansive data to points of query. It introduces distributed databases as single logical databases physically dispersed across interconnected computers [1][2]. Furthermore, the integration of distributed databases and distributed database management systems (DDBMS) is discussed. The introduction outlines the three fundamental phases of distributed database design: initial design, redesign, and materialization of the redesign.

The "Related Works" section provides an overview of prior research endeavors in the realm of distributed database systems, with a particular focus on fragmentation and allocation schemes. Noteworthy studies, such as Seung-Jin Lim (2016) on vertical fragmentation for query optimization, Van Nghia Luong et al. (2015) employing clustering techniques for efficient fragmentation, and Akashkumar Patel et al. (2014) exploring diverse fragmentation methods, are examined. Other contributions by Bhuyar et al. (2012) addressing initial fragmentation issues, Nicoleta-Magdalena Iacob (2011) analyzing data allocation and cost, Adrian Runceanu (2007) proposing a heuristic algorithm for vertical fragmentation, HABABEH et al. (2003) developing a communication cost reduction strategy, and Navathe et al. (1995) presenting fragmentation algorithms and a design methodology, collectively enhance our understanding and facilitate the improvement of distributed database systems.

Fragmentation, the process of dividing a single object (e.g., database or table) into distinct fragments, is discussed in detail. Three primary fragmentation types are examined [2]-[4]: horizontal fragmentation, which divides data into non-overlapping tuples for site-specific query information; vertical fragmentation, which partitions data into sets of columns (excluding primary keys) to suit different site functions; and mixed fragmentation, which combines horizontal and vertical approaches, dividing a table into horizontal subsets with attribute divisions [1][4]. The careful management of these fragmentation strategies enables efficient distributed databases with optimized query execution and resource utilization, addressing concerns such as reliability, performance, storage capacity, communication costs, and security.

The "Correctness Rules of Fragmentation" section introduces three crucial rules for ensuring the effectiveness and reliability of fragmented distributed database systems[1]. The completeness rule

ensures that no data items are lost during fragmentation, while the reconstruction rule emphasizes the need for relational operators to maintain data integrity during the reconstruction of the original relation from fragments. Additionally, the disjointness rule prevents data duplication and ensures consistency across fragments [4].

In conclusion, this research paper has provided a comprehensive introduction to distributed databases and the associated design considerations. It has offered an in-depth explanation of data fragmentation, including a comparative analysis of different fragmentation strategies, their advantages, and disadvantages [3][4]. The findings of this study serve as a valuable resource for researchers, particularly those entering the field of distributed databases, providing insights and knowledge to guide future investigations and advancements in the domain.

In my opinion, this research paper offers a well-structured and informative exploration of distributed database systems, specifically focusing on the topic of data fragmentation and its associated strategies. The paper successfully provides a clear and comprehensive explanation of the fundamental concepts, supported by relevant examples. The inclusion of related works and the comparative analysis of fragmentation strategies add valuable insights to the paper's content. However, to enhance the overall contribution of the paper, it could benefit from a more thorough examination and evaluation of the presented fragmentation techniques. Additionally, addressing potential challenges and practical considerations related to the implementation and management of distributed databases would further strengthen the paper's scholarly merit. Nevertheless, this paper serves as an invaluable resource for researchers seeking to acquire a deep understanding of distributed databases and offers a solid foundation for further investigations in this field.

Scope of Improvement:

There exist several potential avenues for enhancing the research paper:

Comprehensive analysis of fragmentation strategies: The paper would greatly benefit from a thorough and meticulous analysis of the advantages, limitations, and pertinent considerations pertaining to each fragmentation strategy. By delving deeper into the implications and trade-offs associated with horizontal, vertical, and mixed fragmentations, the discussion can be enriched and rendered more nuanced.

Conducting experimental evaluations: In order to fortify the findings of the paper, it is advisable to undertake experimental evaluations or case studies that validate the efficacy and performance of the proposed fragmentation strategies. The inclusion of quantitative results and comparisons would furnish empirical evidence to substantiate the assertions put forth in the paper.

Incorporation of practical implementation considerations: Inclusion of deliberations concerning the practical implementation aspects of the fragmentation strategies would confer added value to the research. Addressing real-world challenges, such as data consistency, scalability, and managing updates within distributed environments, would render the research more pragmatic and relevant to practitioners.

By embracing these recommendations, the research paper can be elevated to a higher standard, characterized by a more exhaustive analysis, strengthened empirical underpinnings, and a heightened practical applicability, thereby augmenting its contribution to the field of study.

Problem 2

The goal of this assignment is to create a simple database management system (DBMS) using Java. The DBMS should be lightweight and able to handle different types of queries, like creating and modifying the **database structure (DDL)** and **manipulating data (DML)**. The data will be stored in TXT files, and there will be a security feature for authentication. An interesting addition is the **transaction [6]** processing feature, which ensures that data changes are consistent, isolated, durable, and atomic. This report will provide an overview of the implemented DBMS, highlighting its main features, design choices, and functionality.

Gitlab Repository:

https://git.cs.dal.ca/kpkhant/csci5408_s23_b00935171_keyur_khant/-/tree/main/A2

File structure:

```
.
├── LightDB/
│   ├── database/
│   │   ├── tables/
│   │   │   └── employee.txt
│   │   ├── tables-metadata/
│   │   │   └── employee_metadata.txt
│   │   └── users.txt
│   └── src/
│       ├── main/
│       │   └── java/
│       │       ├── database/
│       │       │   ├── auth/
│       │       │   │   ├── IAuth.java
│       │       │   │   └── Auth.java
│       │       │   ├── query/
│       │       │   │   ├── IQueryHandler.java
│       │       │   │   ├── IQueryManager.java
│       │       │   │   ├── QueryHandler.java
│       │       │   │   ├── QueryManager.java
│       │       │   │   └── QueryUtils.java
│       │       │   ├── user/
│       │       │   │   ├── User.java
│       │       │   │   └── UserManager.java
│       │       │   ├── utils/
│       │       │   │   └── Utils.java
│       │       │   └── Main.java
```

Here, I have followed the above structure in which code is segregated into various classes and all data is stored in a database directory in **LightDB**.

The **UserManager** class is a part of the lightweight database management system (DBMS) implementation in Java. It is responsible for managing user-related operations and authentication within the DBMS.

The **QueryManager** class is a key component of the lightweight database management system (DBMS) implementation in Java. It handles the execution and management of different types of queries within the DBMS.

The **QueryHandler** class is a part of the lightweight database management system (DBMS) implementation in Java. It is responsible for handling and executing individual queries received by the system. The **QueryHandler** class interacts with the **QueryManager** and other relevant components to process the query. It translates the query into appropriate database operations and ensures their successful execution.

The **Auth** class interacts with the user management module, such as the **UserManager**, to validate user credentials during the login process. It securely stores and compares user passwords to ensure authentication accuracy.

In the **database** directory, users.txt contains all user credentials. Furthermore, tables contain all database tables and table-metadata contains column fields of the specific tables.

Note: Throughout the application development, I have used Java W3 documentation pattern to add class and method commenting.

User Registration and Authentication:

User Registration:

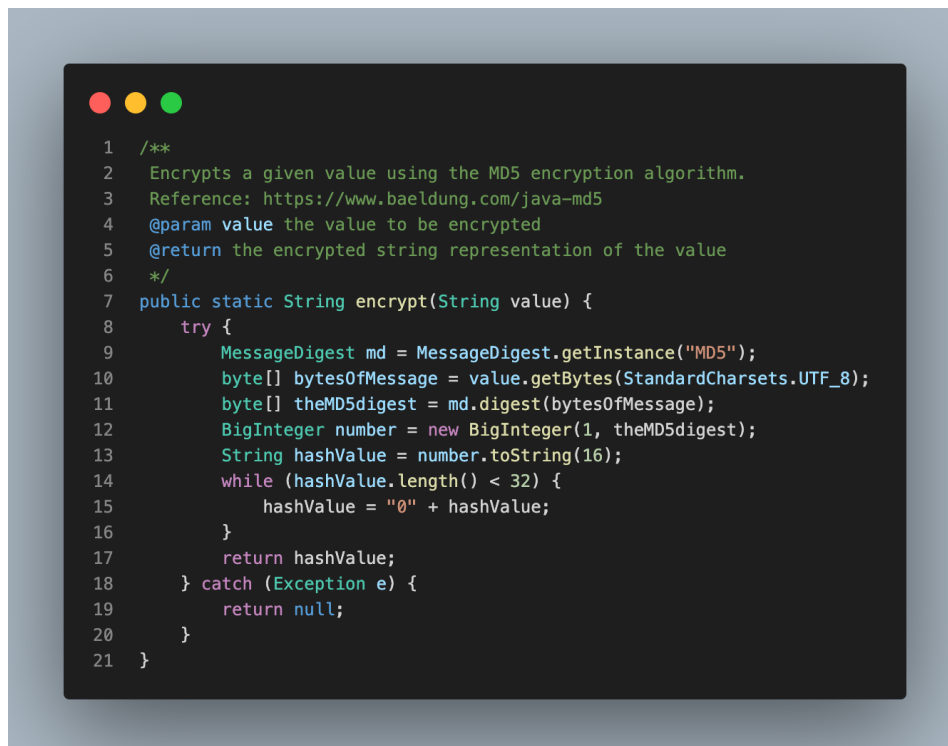
- The registration process allows users to create an account with the application.
- Users provide a username, password, and select security questions.
- The password and security answers are encrypted using the MD5 encryption algorithm to enhance security.
- The user's registration information is typically stored in a database for future reference.

User Authentication:

- The authentication process verifies the user's identity before granting access to their account.
- Users enter their username and password during the authentication process.
- The entered password is encrypted using the MD5 encryption algorithm [5] for comparison with the stored encrypted password.
- If the entered username and encrypted password match the stored information, the user is authenticated and granted access.

Security Questions and Answers:

- Security questions are additional authentication measures used to verify the user's identity.
- During registration, users select security questions from a predefined set or provide their own questions.
- Users also provide corresponding answers to these security questions.
- The security answers are encrypted using the MD5 encryption algorithm and stored securely.



```
1  /**
2   * Encrypts a given value using the MD5 encryption algorithm.
3   * Reference: https://www.baeldung.com/java-md5
4   * @param value the value to be encrypted
5   * @return the encrypted string representation of the value
6   */
7  public static String encrypt(String value) {
8      try {
9          MessageDigest md = MessageDigest.getInstance("MD5");
10         byte[] bytesOfMessage = value.getBytes(StandardCharsets.UTF_8);
11         byte[] theMD5digest = md.digest(bytesOfMessage);
12         BigInteger number = new BigInteger(1, theMD5digest);
13         String hashValue = number.toString(16);
14         while (hashValue.length() < 32) {
15             hashValue = "0" + hashValue;
16         }
17         return hashValue;
18     } catch (Exception e) {
19         return null;
20     }
21 }
```

Figure 1: MD5 encryption in Java [5]


```

1  /**
2   Private method used for user registration, which registers a new user.
3   @param user the User object containing the user's information
4   @return an integer value representing the registration status or result
5   */
6  private int register(User user) {
7      UserManager userManager = new UserManager();
8      PrintWriter usersFile = null;
9      try {
10         User existingUser = userManager.getUserByUsername(user.getUsername());
11         if (existingUser != null) return 0;
12
13         File file = new File("database/users.txt");
14         FileWriter fr = new FileWriter(file, true);
15         BufferedWriter br = new BufferedWriter(fr);
16         usersFile = new PrintWriter(br);
17         String userString = user.getUsername() + "," + Utils.encrypt(user.getPassword()) + "," + user.getQuestion() + "," + Utils.encrypt(user.getAnswer());
18         usersFile.println(userString);
19     } catch (Exception e) {
20         System.out.println(e);
21         return 2;
22     } finally {
23         if (usersFile != null) usersFile.close();
24     }
25     return 1;
26 }

```

Figure 2: User Registration

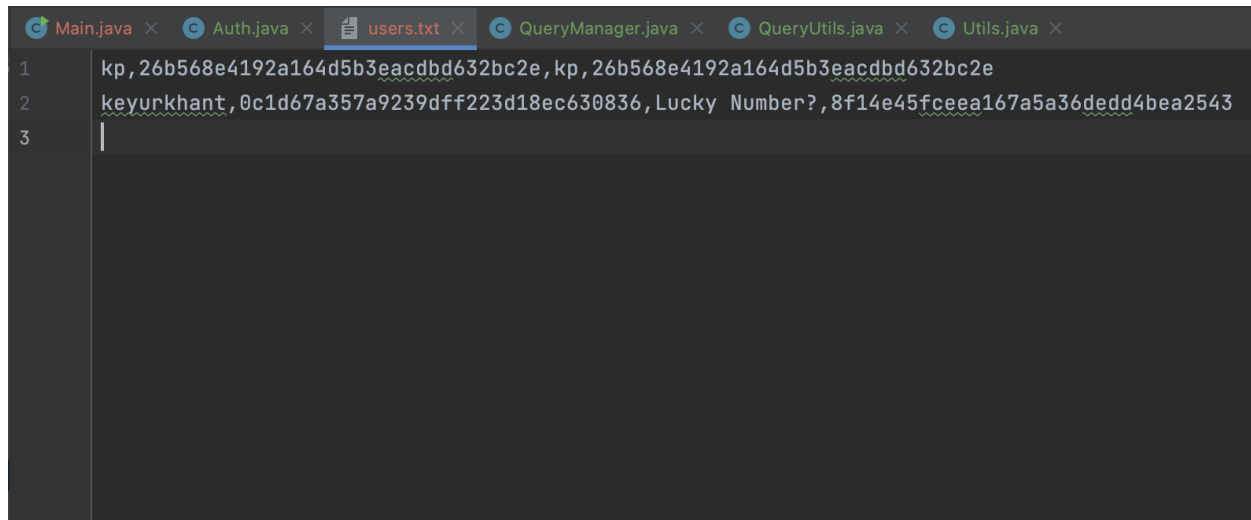
```

/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 2
Enter your username: keyurkhant
Enter your password: Keyur@123
Enter your security question: Lucky Number?
Enter your security answer: 7
User with username keyurkhant successfully registered!
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice:

```

Figure 3: Registration process

In this process, users can register with the required process. After successful registration, user details are getting stored in users.txt file.

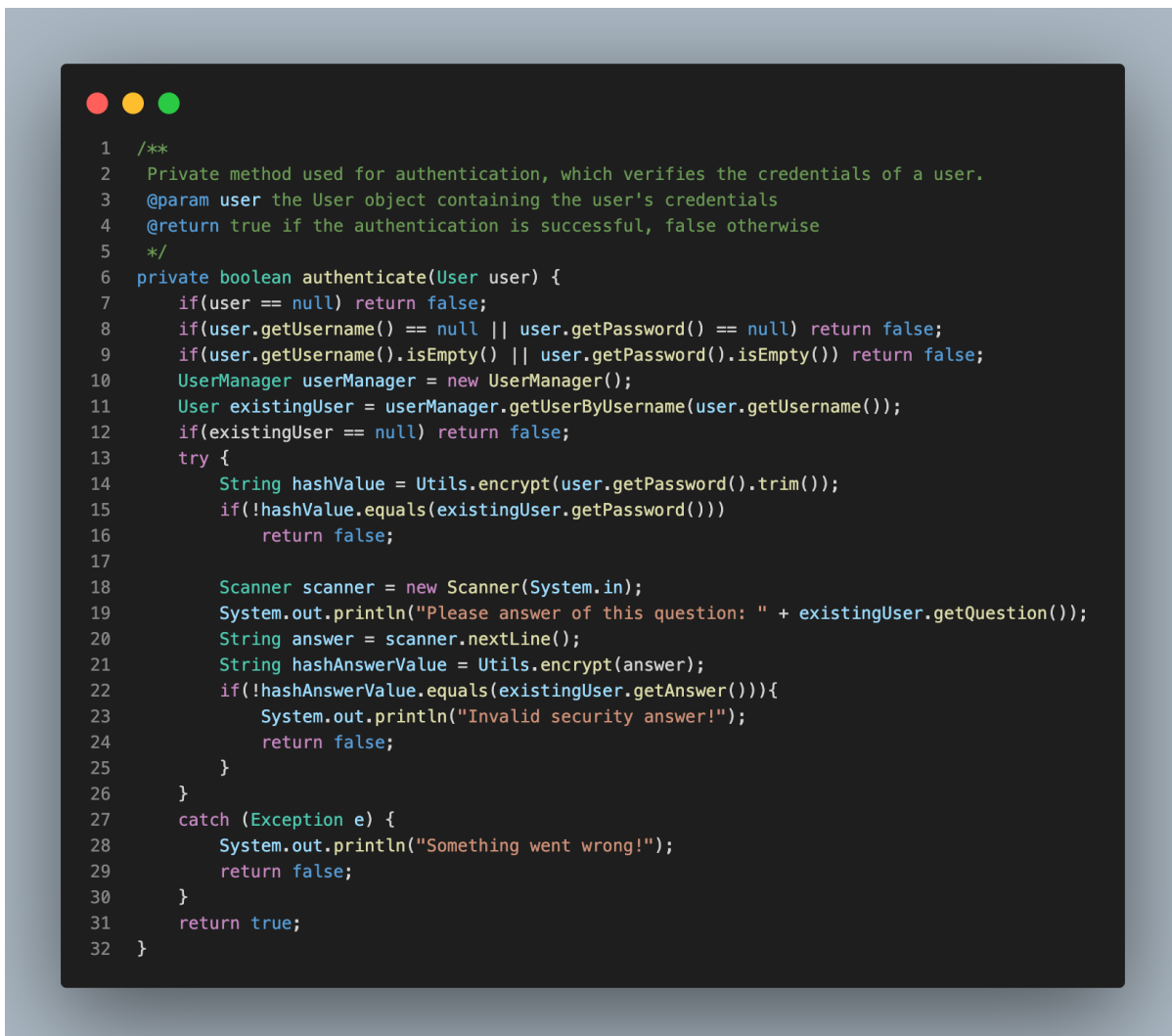


```

1 kp,26b568e4192a164d5b3eacdbd632bc2e,kp,26b568e4192a164d5b3eacdbd632bc2e
2 keyurkhant,0c1d67a357a9239dff223d18ec630836,Lucky Number?,8f14e45fceeaa167a5a36dedd4bea2543
3

```

Figure 4: Stored user data with required encrypted details



```

1  /**
2   Private method used for authentication, which verifies the credentials of a user.
3   @param user the User object containing the user's credentials
4   @return true if the authentication is successful, false otherwise
5   */
6  private boolean authenticate(User user) {
7      if(user == null) return false;
8      if(user.getUsername() == null || user.getPassword() == null) return false;
9      if(user.getUsername().isEmpty() || user.getPassword().isEmpty()) return false;
10     UserManager userManager = new UserManager();
11     User existingUser = userManager.getUserByUsername(user.getUsername());
12     if(existingUser == null) return false;
13     try {
14         String hashValue = Utils.encrypt(user.getPassword().trim());
15         if(!hashValue.equals(existingUser.getPassword()))
16             return false;
17
18         Scanner scanner = new Scanner(System.in);
19         System.out.println("Please answer of this question: " + existingUser.getQuestion());
20         String answer = scanner.nextLine();
21         String hashAnswerValue = Utils.encrypt(answer);
22         if(!hashAnswerValue.equals(existingUser.getAnswer())){
23             System.out.println("Invalid security answer!");
24             return false;
25         }
26     }
27     catch (Exception e) {
28         System.out.println("Something went wrong!");
29         return false;
30     }
31     return true;
32 }

```

Figure 5: User Authentication

```
/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@1234
Invalid credentials
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: |
```

Figure 6: Invalid Credentials error with invalid username or password

```
/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
9
Invalid security answer!
Invalid credentials
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: |
```

Figure 7: Invalid Credentials error with invalid security answer

In this process, in both cases, invalid credentials or security answers, the user will not be allowed to access the database and the prompt will be redirected to the login and registration page.

Here, below will be the successful login attempt.

```
/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
7
User logged in successfully
QUERY> |
```

Figure 8: Valid credentials allow to access database

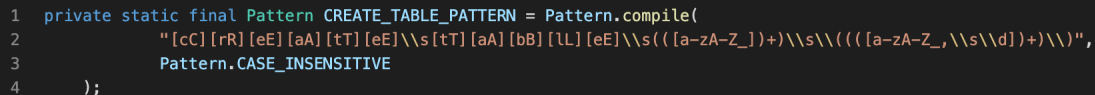
Table creation:

Table creation queries are used to define and create database tables. They specify the structure of the table, including the column names, data types, constraints, and other properties.

Syntax of create table:

```
CREATE TABLE <table_name> (id int,name varchar,amount double);
```

This will allow the user to create case insensitive query.



```
1 private static final Pattern CREATE_TABLE_PATTERN = Pattern.compile(
2     "[cC][rR][eE][aA][tT][eE]\\s\\s[aA][bB][lL][eE]\\s\\s([a-zA-Z_]+)\\s\\s((([a-zA-Z_][\\s\\d]+)\\s\\s)",
3     Pattern.CASE_INSENSITIVE
4 );
```

Figure 9: Pattern for table create



```
1 public boolean createQuery(String tableName, String typeColumns) {
2     File tableFile = new File(Utils.getFileName(tableName, FileTypes.TABLE));
3     if(tableFile.exists()) {
4         System.out.println("Table not exists");
5         return false;
6     }
7
8     try {
9         tableFile.createNewFile();
10
11         File tableMetaFile = new File(Utils.getFileName(tableName, FileTypes.META_TABLE));
12         tableMetaFile.createNewFile();
13
14         List<String> typeFields = List.of(typeColumns.split(","));
15         FileWriter fileWriter = new FileWriter(tableMetaFile);
16         PrintWriter printWriter = new PrintWriter(fileWriter);
17
18         for(String field: typeFields){
19             field = field.trim();
20             field = field.replace(' ', '|');
21             printWriter.println(field);
22         }
23         printWriter.close();
24         System.out.println("New table created: " + tableName);
25         return true;
26     } catch (Exception e) {
27         System.out.println("Error while creating table, try again!");
28         e.printStackTrace();
29         return false;
30     }
31 }
32 }
```

Figure 10: Create table query logic

```

/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
7
User logged in successfully
QUERY> Create table Compnay AA;
Invalid Query!
Please try again.
QUERY> CREATE table Company (id int primaryKey,name varchar,city varchar);
New table created: Company
QUERY> |

```

Figure 11: Invalid and valid create table queries

```

Main.java x Company.txt x IQueryManager.java x Auth.java x Company_metadata.txt x
1 | 1 id|int|primaryKey ✓ 1 ^ v
2 | 2 name|varchar
3 | 3 city|varchar
4 | 4

```

Figure 12: New entries of file in database with metadata file

Supported Data types:

- Integer
- VarChar (String)
- Double

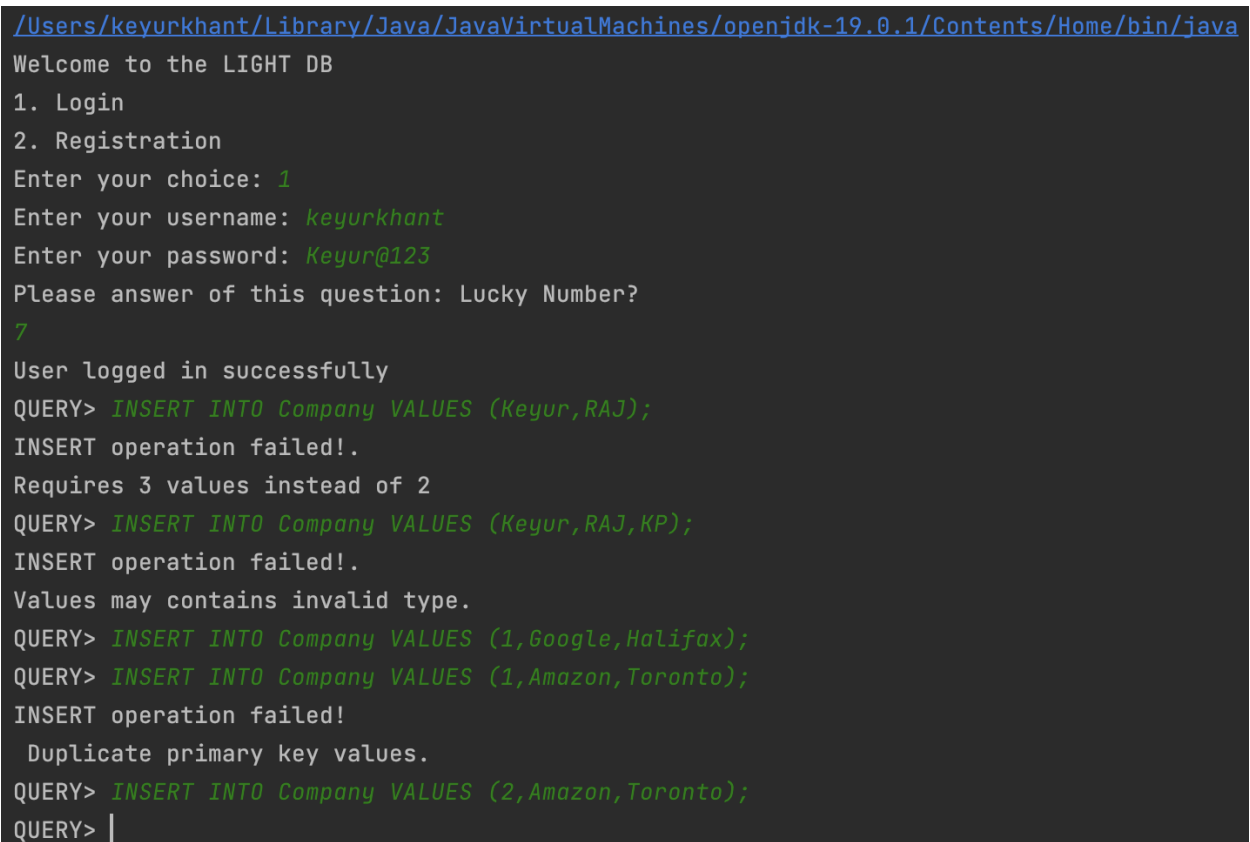
Insert data query:

Insert query will add data into the table (created previously) according to the data types and created schema. If invalid data is inserted, the processor will not allow it and the user gets a specific error message for that.

Syntax of insert table:

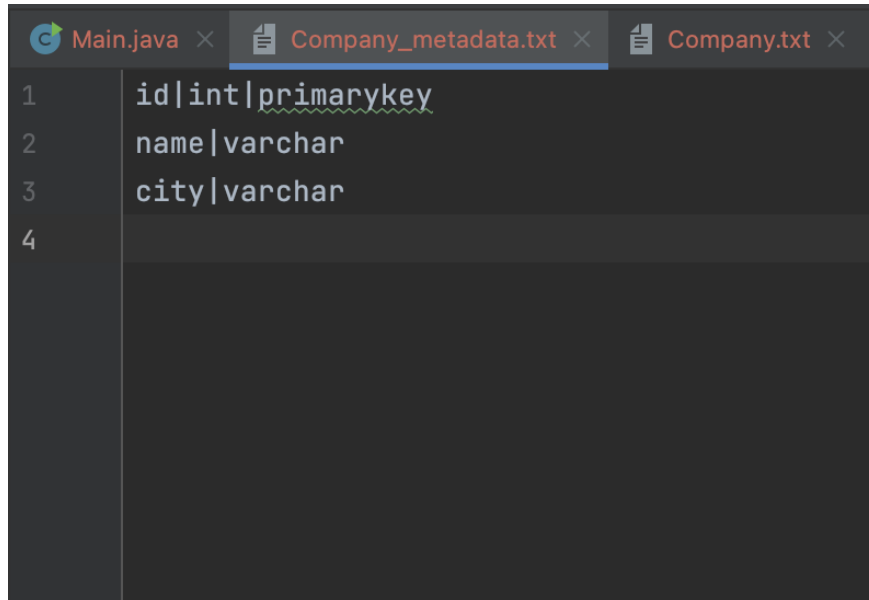
```
INSERT INTO <table_name> VALUES (value1,value2);
```

This will allow the user to insert value into a specific table.



```
/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
7
User logged in successfully
QUERY> INSERT INTO Company VALUES (Keyur,RAJ);
INSERT operation failed!.
Requires 3 values instead of 2
QUERY> INSERT INTO Company VALUES (Keyur,RAJ,KP);
INSERT operation failed!.
Values may contains invalid type.
QUERY> INSERT INTO Company VALUES (1,Google,Halifax);
QUERY> INSERT INTO Company VALUES (1,Amazon,Toronto);
INSERT operation failed!
Duplicate primary key values.
QUERY> INSERT INTO Company VALUES (2,Amazon,Toronto);
QUERY> |
```

Figure 13: INSERT query with all possible validations and success



```
1 id|int|primarykey
2 name|varchar
3 city|varchar
4
```

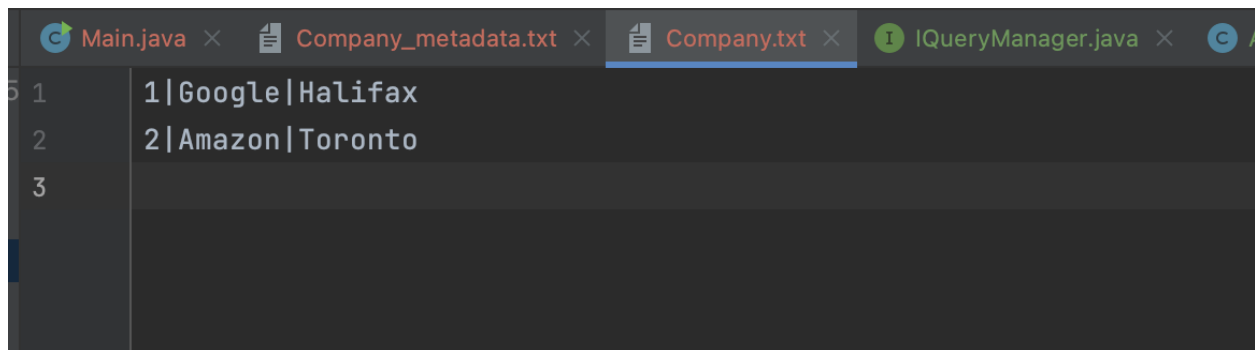
Figure 14: Company table schema with data type

Here, we created id as a primary key. Hence, it will not allow duplicate entries with the same id.

In Figure 13, when a user tries to add an Amazon company entry with the same id, it will cause a duplicate entry error.

Primary Key Constraint:

A primary key constraint is used to uniquely identify each record or row in a database table. It ensures that the values in the specified column or combination of columns are unique and not null.



```
1 1|Google|Halifax
2 2|Amazon|Toronto
3
```

Figure 15: Successful entries in Company table

Select data query:

Select query will allow user to select all or specific columns from the table along with the conditional AND and OR operations.

Syntax of select table:

```
SELECT * FROM <table_name>;
```

```
SELECT column1, column2 FROM <table_name>;
```

```
SELECT * FROM <table_name> WHERE column1=value1;
```

```
SELECT column2 FROM <table_name> WHERE column1=value1 or column2=value2;
```

```
SELECT column2 FROM <table_name> WHERE column1=value1 and column2=value2;
```

This will allow the user to select values from a specific table.

```
/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
7
User logged in successfully
QUERY> SELECT * FROM Company;
1 | Google | Halifax | 2
2 | Amazon | Toronto | 1
3 | Microsoft | Mumbai | 5
4 | OpenAI | NewYork | 1
QUERY> SELECT * FROM NO_NAME;
Table NO_NAME does not exist
QUERY> SELECT age from Company;
SELECT operation failed!
age not exist!
QUERY> SELECT name,rank FROM Company;
Google | 2
Amazon | 1
Microsoft | 5
OpenAI | 1
QUERY> |
```

Figure 16: Possible SELECT query operations

- If a user enters an invalid query, it will throw Invalid Query error.
- If the user enters an invalid table name, it will throw No Table Found error.
- If a user enters an invalid column name, it will throw NO Column Type found error.
- Here, below will be the AND/OR operation of the select query

```
/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
7
User logged in successfully
QUERY> SELECT * FROM Company WHERE rank=1 or city=Mumbai;
2 | Amazon | Toronto | 1
3 | Microsoft | Mumbai | 5
4 | OpenAI | NewYork | 1
QUERY> SELECT * FROM Company WHERE rank=1 and city=Mumbai;
QUERY> SELECT * FROM Company WHERE rank=1 and city=NewYork;
4 | OpenAI | NewYork | 1
QUERY> SELECT name,city FROM Company WHERE rank=1 or city=Toronto;
Amazon | Toronto
OpenAI | NewYork
QUERY> SELECT name,city FROM Company WHERE rank=1 or city=Halifax;
Google | Halifax
Amazon | Toronto
OpenAI | NewYork
QUERY> |
```

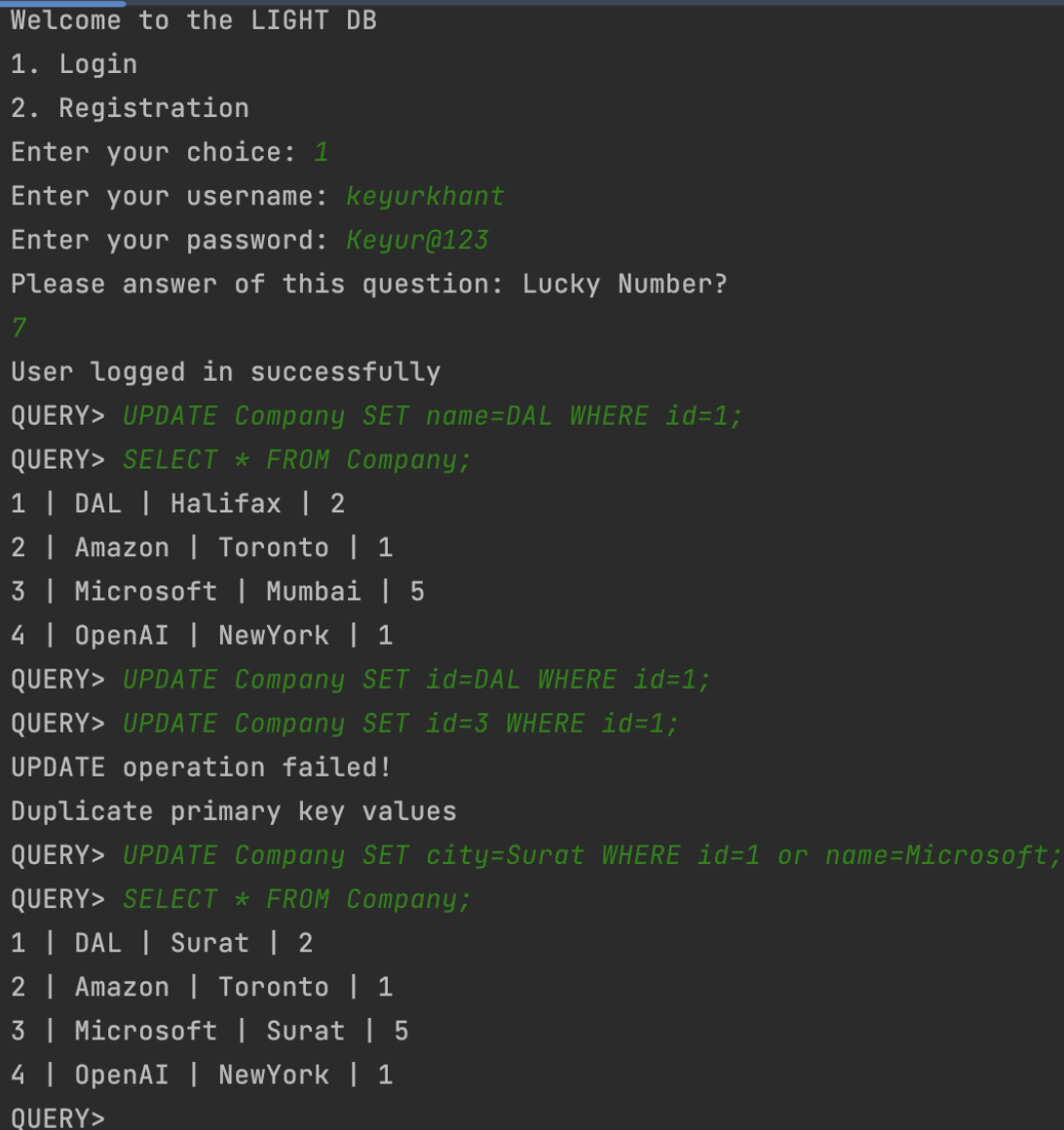
Figure 17: AND/OR operation in select query

Update data query:

Update query will allow users to update specific table entries. It allows the user to handle AND/OR operation as well.

Syntax of update table:

```
UPDATE <table_name> SET column1=value1 WHERE column2=value2;
```



```
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
7
User logged in successfully
QUERY> UPDATE Company SET name=DAL WHERE id=1;
QUERY> SELECT * FROM Company;
1 | DAL | Halifax | 2
2 | Amazon | Toronto | 1
3 | Microsoft | Mumbai | 5
4 | OpenAI | NewYork | 1
QUERY> UPDATE Company SET id=DAL WHERE id=1;
QUERY> UPDATE Company SET id=3 WHERE id=1;
UPDATE operation failed!
Duplicate primary key values
QUERY> UPDATE Company SET city=Surat WHERE id=1 or name=Microsoft;
QUERY> SELECT * FROM Company;
1 | DAL | Surat | 2
2 | Amazon | Toronto | 1
3 | Microsoft | Surat | 5
4 | OpenAI | NewYork | 1
QUERY>
```

Figure 18: UPDATE operation with all possible combination

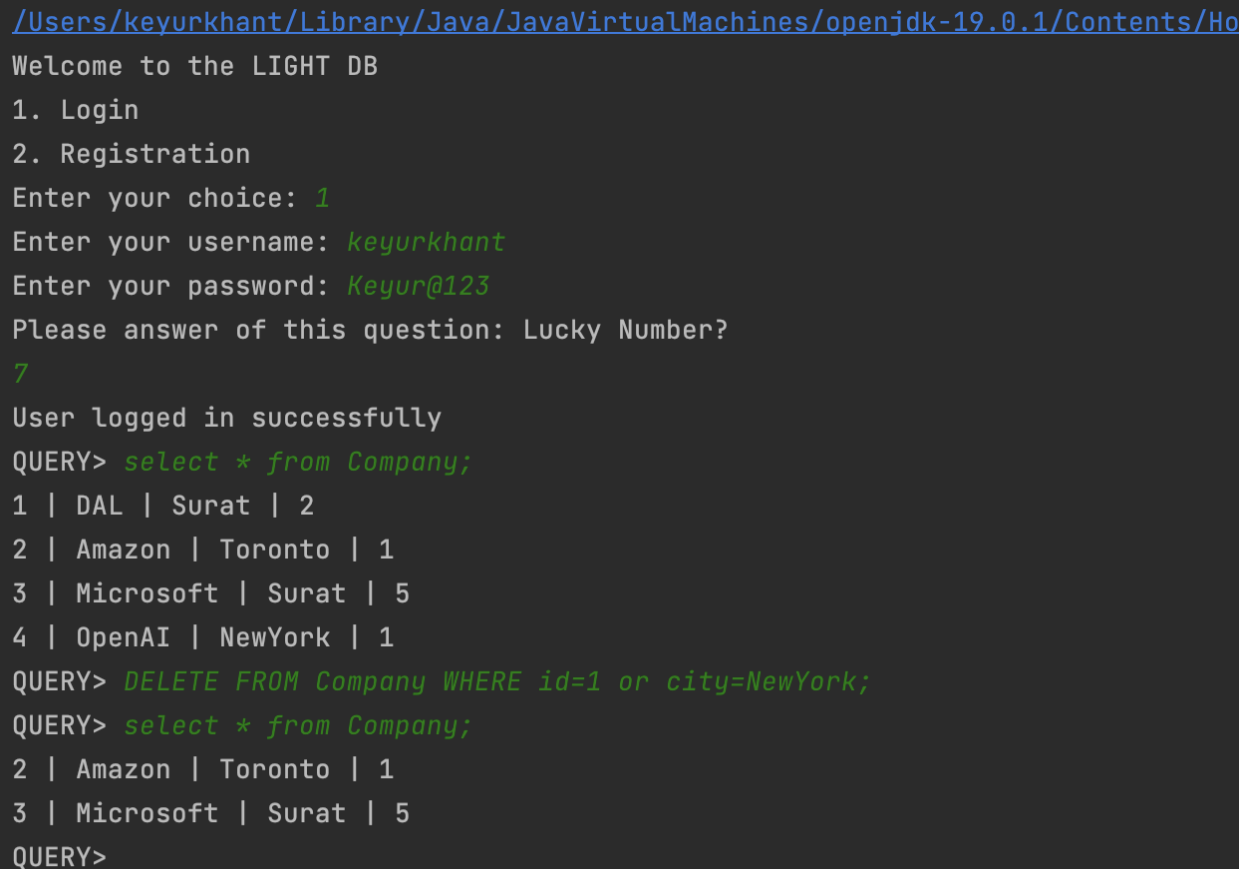
It will not allow updating tables as per the primary key and handle duplicate keys.

Delete data query:

Delete query will allow users to delete specific entries from the table along with AND/OR operations.

Syntax of delete table:

```
DELETE FROM <table_name> WHERE column1=value1;
```



```
/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
7
User logged in successfully
QUERY> select * from Company;
1 | DAL | Surat | 2
2 | Amazon | Toronto | 1
3 | Microsoft | Surat | 5
4 | OpenAI | NewYork | 1
QUERY> DELETE FROM Company WHERE id=1 or city=NewYork;
QUERY> select * from Company;
2 | Amazon | Toronto | 1
3 | Microsoft | Surat | 5
QUERY>
```

Figure 19: DELETE operation

Transaction in Database:

The task at hand is to implement a single transaction handling logic that follows the ACID (Atomicity, Consistency, Isolation, Durability) properties. The transaction is identified by specific user input commands such as "Begin Transaction," "End Transaction," "Commit," and "Rollback."

To achieve this, the system should process queries within a transaction without immediately writing the changes to the underlying database file. Instead, the queries should be stored in an intermediate data structure such as LinkedList, ArrayList, or any other suitable collection.

Here's an overview of the logic:

Begin Transaction: When the user inputs the "Begin Transaction" command, the system initializes a data structure (e.g., LinkedList) to hold the queries to be executed within the transaction.

Query Processing: As the user submits queries within the transaction, the system processes and executes them but does not immediately update the actual database file. Instead, it adds the queries to the intermediate data structure.

Commit: When the user inputs the "Commit" command, the system ensures that all queries within the transaction are valid and can be executed successfully. If the validation passes, the system applies the changes by executing the stored queries and updates the database file accordingly.

Rollback: If the user inputs the "Rollback" command, the system empties the intermediate data structure, discarding all queries within the transaction. This action reverts the system to the state before the transaction began, ensuring that no changes from the transaction are persisted.

```
/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Contents/Home/bin/java
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
7
User logged in successfully
QUERY> select * from Company;
2 | Amazon | Toronto | 1
3 | Microsoft | Surat | 5
QUERY> BEGIN TRANSACTION;
QUERY> select * from Company;
QUERY>
```

Figure 20: Begin transaction

Here, after initializing the transaction, the user can write the query but it will not run as the transaction should be committed to run the query.

```

/Users/keyurkhant/Library/Java/JavaVirtualMachines/openjdk-19.0.1/Cont
Welcome to the LIGHT DB
1. Login
2. Registration
Enter your choice: 1
Enter your username: keyurkhant
Enter your password: Keyur@123
Please answer of this question: Lucky Number?
7
User logged in successfully
QUERY> select * from Company;
2 | Amazon | Toronto | 1
3 | Microsoft | Surat | 5
QUERY> BEGIN TRANSACTION;
QUERY> select * from Company;
QUERY> INSERT INTO Company VALUES (4,Google,Mumbai,2);
QUERY> INSERT INTO Company VALUES (1,OpenAI,Halifax,1);
QUERY> UPDATE Company SET name=AMAZONINC where id=2;
QUERY> end transaction;
QUERY> select * from Company;
2 | Amazon | Toronto | 1
3 | Microsoft | Surat | 5
QUERY> |

```

Figure 21: End transaction but not committed

```

QUERY> end transaction;
QUERY> select * from Company;
2 | Amazon | Toronto | 1
3 | Microsoft | Surat | 5
QUERY> COMMIT;
2 | Amazon | Toronto | 1
3 | Microsoft | Surat | 5
QUERY> select * from Company;
2 | AMAZONINC | Toronto | 1
3 | Microsoft | Surat | 5
4 | Google | Mumbai | 2
1 | OpenAI | Halifax | 1
QUERY> |

```

Figure 22: After commit transaction applied

```
QUERY> BEGIN TRANSACTION;  
QUERY> select * from Company;  
QUERY> END TRANSACTION;  
QUERY> ROLLBACK;  
QUERY> COMMIT;  
QUERY> |
```

Figure 23: After rollback; commit will be cleaned

Summary:

LightDB application follow various design patterns such as -

Single Responsibility Principle

Application follows single responsibility for each module and functionality. Query will be handled by query handler and manager classes, whereas authentication will be handled separately.

Open Close Principle

Application is designed in a way that it will allow extending features whereas closed for modification in existing modules. It was achieved through the interface.

Interface Segregation

All required interfaces are segregated as per the modules.

Factory Pattern

QueryHandler and **QueryManager** will handle each factory pattern to follow in LightDB.

A Light Weight Database using Java is a minimalistic database system implemented in Java that provides basic database functionalities. It offers data storage, retrieval, and manipulation capabilities while maintaining a small footprint. It utilizes simple data structures and file handling mechanisms to store and manage data efficiently.

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)*, 2017, pp. 724–729.
- [2] *Researchgate.net*. [Online]. Available: https://www.researchgate.net/publication/320634432_A_comparative_analysis_of_data_fragmentation_in_distributed_database. [Accessed: 13-Jul-2023].
- [3] N. I. (Ciobanu), "Fragmentation and data allocation in the distributed environments," *An. Univ. Craiova Ser. Mat. Inform.*, vol. 38, no. 3, pp. 76–83, 2011.
- [4] A. Runceanu, "Fragmentation in distributed databases," in *Innovations and Advanced Techniques in Systems, Computing Sciences and Software Engineering*, Dordrecht: Springer Netherlands, 2008, pp. 57–62.
- [5] *Baeldung.com*. [Online]. Available: <https://www.baeldung.com/java-md5>. [Accessed: 13-Jul-2023].
- [6] "SQL transaction," *w3resource*. [Online]. Available: <https://www.w3resource.com/sql/controlling-transactions.php>. [Accessed: 13-Jul-2023].