CSCI 5408

Data Management, Warehousing, And Analytics

Assignment 2 - Problem 1

Performing a systematic literature review and providing summary.

Prepared By

Bhavisha Oza (B00935827)

Problem-1: Perform a systematic literature review and provide summary.

"A comparative Analysis of Data Fragmentation in Distributed Database" [1].

1. Summary:

A comparative Analysis of Data Fragmentation in Distributed Database [1] paper proposes a comparative analysis of data fragmentation in distributed database systems. It begins by highlighting the significance of distributed systems in modern computing technology and explains that distributed databases consist of data partitions or fragments that are distributed or replicated across multiple physical locations. The paper emphasizes the importance of the distributed database design process, which includes initial design, redesign, and materialization phases.

The introduction of the paper emphasizes the significance of distributed systems in modern computing technology, particularly in handling large amounts of data. Distributed databases are logical databases that are physically distributed across multiple locations connected by a network. The distribution of data involves fragmentation, replication, and allocation processes. The authors highlight the advancements in communication technology, software, and hardware, making distributed database systems more feasible and efficient.

The initial design, redesign, and materialisation of the redesign phases of the distributed database design process are covered. Algorithms for allocation and fragmentation are used in the early design to reduce the expense of transaction processing. New fragmentation and allocation methods are created throughout the redesign process based on modifications to the distributed database environment. The new fragmentation and allocation mechanism is put into place during the materialisation phase.

Related works in the field of distributed database systems are presented, including studies on fragmentation schemes and allocation schemes. Various researchers have proposed different approaches for vertical and horizontal fragmentation in distributed databases, aiming to improve query performance and minimize communication costs. They emphasise the benefits and drawbacks of fragmentation and discuss several strategies put forth by earlier researchers.

The paper then introduces three types of fragmentations: horizontal fragmentation (HF), vertical fragmentation (VF), and mixed fragmentation (MF). A relation or class is divided into disjoint tuples or instances using horizontal fragmentation, with each fragment being stored at a separate node. Vertical fragmentation divides a relation into sets of columns or characteristics, each set containing one or more of the table's main key properties. Combining horizontal and vertical fragmentations, known as mixed fragmentation, enables the creation of more complex fragmentation schemes.

Data fragmentation is explained as the process of breaking a single object into two or more fragments or segments. The three types of fragmentations are illustrated using the example of a Human Resources table, showing how the table can be horizontally fragmented into multiple fragments, vertically fragmented into separate sets of columns, or mixed fragmented with both horizontal and vertical partitions.

Finally, the research discusses fragmentation accuracy rules such as completeness, reconstruction, and disjointness. Each data item in the original relation must be discovered in at least one fragment to be complete. The existence of a relational operator capable of reconstructing the original relation from its fragments are required for reconstruction. Disjointness assures that the fragments do not overlap or have redundancy.

In conclusion, the paper provides an overview of data fragmentation in distributed database systems. It compares horizontal, vertical, and mixed fragmentations and discusses the advantages and disadvantages of each. In the end, the authors suggest that the choice of data fragmentation for distributed database depends on the specific requirements and objectives of the system. Understanding the different types of fragmentation and applying the correctness rules can help in designing efficient distributed databases.

2. Scope of Improvements:

There are few areas in the paper where some improvements can be made. As the paper focuses on the fragmentation types and its advantages, disadvantages, it should shed some lights on the methodologies as well. The paper does not mention the methodology used for the comparative analysis. It is important to describe the criteria and metrics used to evaluate the different types of fragmentation and how the analysis was conducted. Providing a clear methodology will enhance the credibility of the study and allow readers to understand the basis of the comparisons made.

It would be beneficial to provide insights into future research directions or potential areas for improvement in data fragmentation techniques in distributed databases. The paper does not address potential challenges or limitations of data fragmentation in distributed databases just like "*When applied to more complex applications such as CAD/CAM, software design, office information systems, and expert systems, the relational data model exhibits limitations in terms of complex object support, type system, and rule management*" [2]. It would be valuable to discuss issues and the issues which still requires a more work such as Sensitivity, Complexity, Transaction, Replication, Interface, Design, Interconnection [2].

References:

[1] A. Al-Sanhani, A. Hamdan, A. Al-Dahoud and A. Al-Dahoud "A comparative Analysis of Data Fragmentation in Distributed Database," *2017 8th International Conference on Information Technology (ICIT)*, Amman (11733), Jordan, 2017, pp. 724-729. DOI: 10.1109/ICITECH.2017.8079934.

[2] M. T. Ozsu and P. Valduriez, "Distributed database systems: where are we now?," *in Computer*, vol. 24, no. 8, pp. 72-78, Aug. 1991. DOI: 10.1109/2.84879.

CSCI 5408

Data Management, Warehousing, And Analytics

Assignment 2 - Problem 2

Prototype of a light-weight DBMS using Java programming language.

Prepared By

Bhavisha Oza (B00935827)

Problem-2: *Prototype of a light-weight DBMS using Java programming language (no 3rd party libraries allowed).*

1. Summary:

This document outlines the requirements for developing a console-based Java application using a standard Java IDE [1]. The application is designed to accept user input in the form of SQL queries after successful login authentication. The Java code do follow JavaDocs specification for commenting styles, including annotations such as "@param" and "@return" [2]. The design principles used in the application's development is utilising the SOLID design principles [3].

Two main functionalities are there in the application. First, a two-factor authentication module which is necessary for user authentication, which involves using ID, password, and question/answer authentication [4]. The application supports multiple users, and it is designed in a separate class to handle the authentication process. Secondly, the application implements a custom-designed persistent storage system to store data, user information, logs, etc. A custom file format is used, and standard formats like JSON, XML, and CSV are not used. Custom delimiters are designed for storing and accessing data within a text file.

In addition to the required functionalities, the application also implements various SQL queries such as CREATE, SELECT, INSERT, UPDATE, and DELETE on any number of tables. Separate methods have been created for each query. Furthermore, a transaction handling logic is implemented, allowing users to initiate and end transactions. To ensure the ACID (Atomicity, Consistency, Isolation, Durability) properties, the processed queries do not immediately update the custom-made database text file. Instead, they are stored in intermediate data structures such as Lists. Only when a "Commit" operation is performed, the changes are applied to the text file.

Overall, the application's focus is on developing a console-based Java program that supports user authentication, custom storage, SQL query execution, and transaction handling following the ACID properties.

Requirements & Tasks fulfilled:

1. Use a standard **Java IDE** to develop your application. Any JDK version is acceptable [1]:

   IntelliJ IDEA Community Edition is used for the entire code.

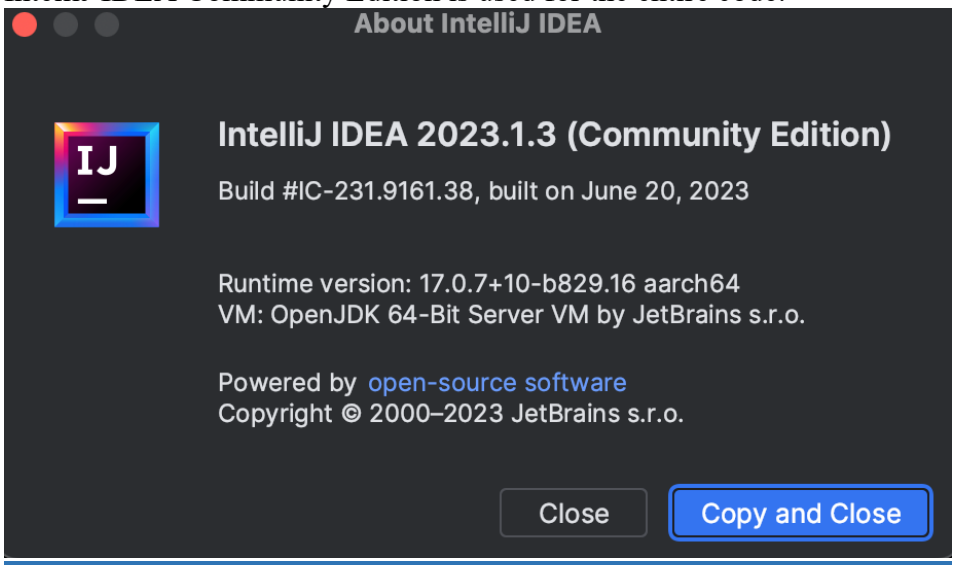

Figure 1: Java IDE setup.

2. While writing the Java code, followed the **JavaDocs specification** for commenting styles, such as @param, @return [2].

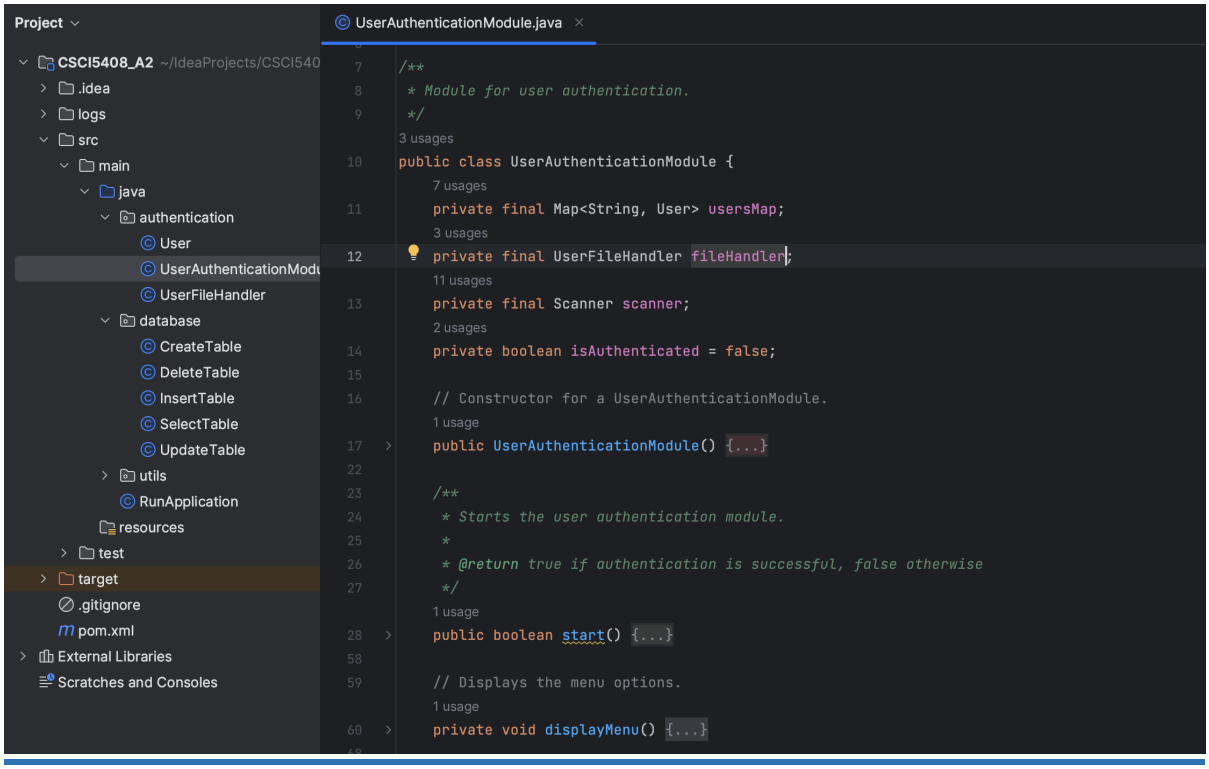

Figure 2: User Authentication module class- JavaDocs specification for commenting styles.

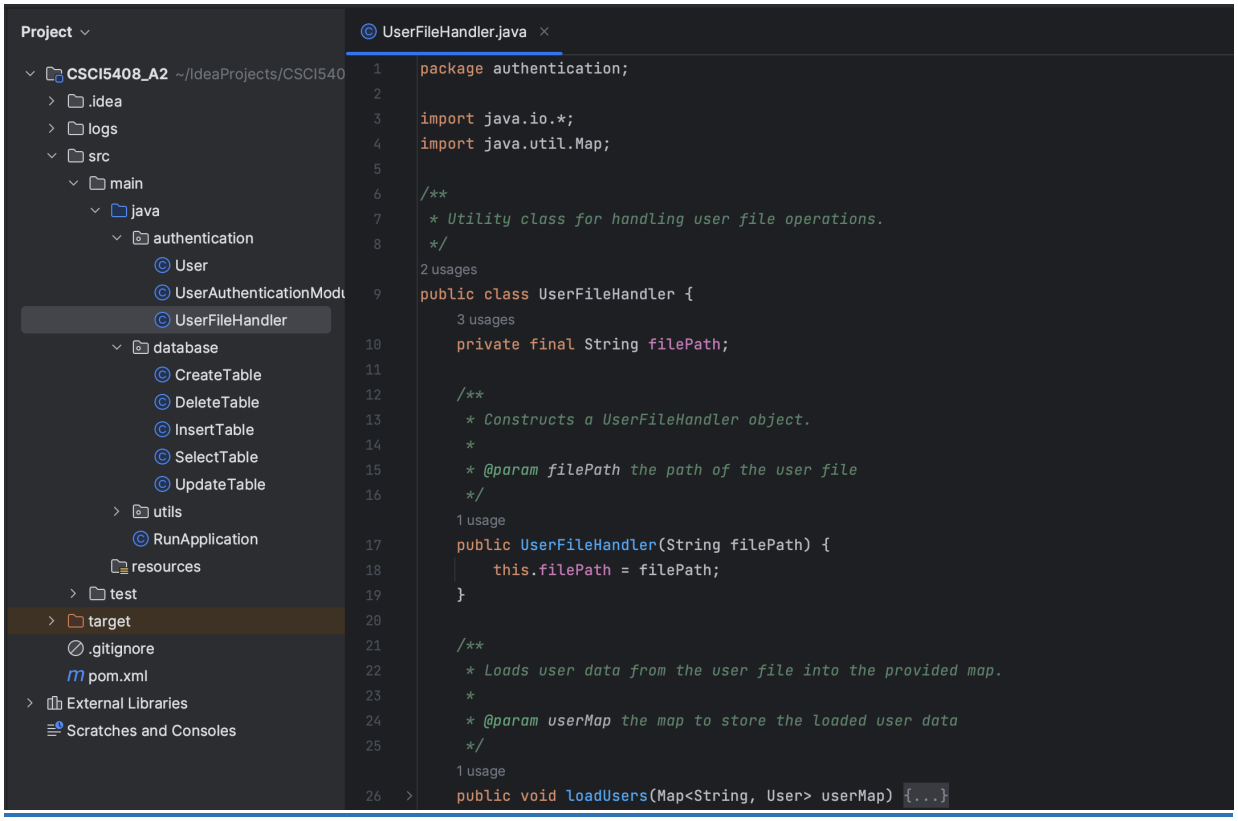

Figure 3: User File Handler class- JavaDocs specification for commenting styles.

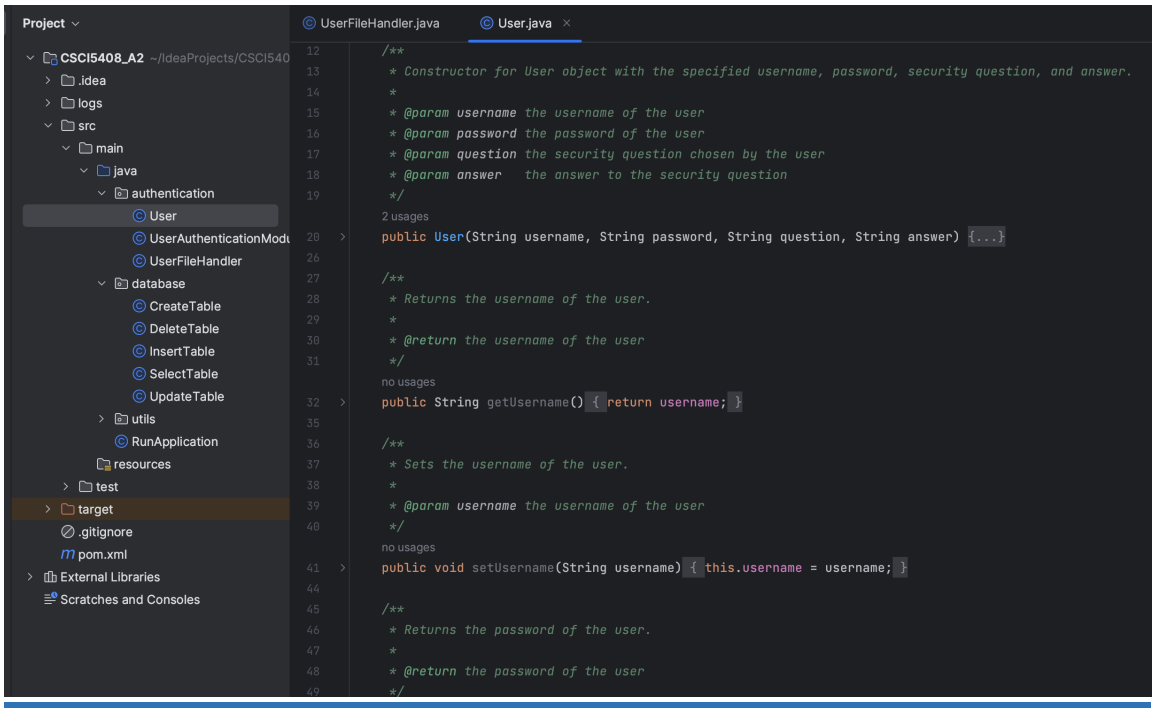

Figure 4: User class- JavaDocs specification for commenting styles.

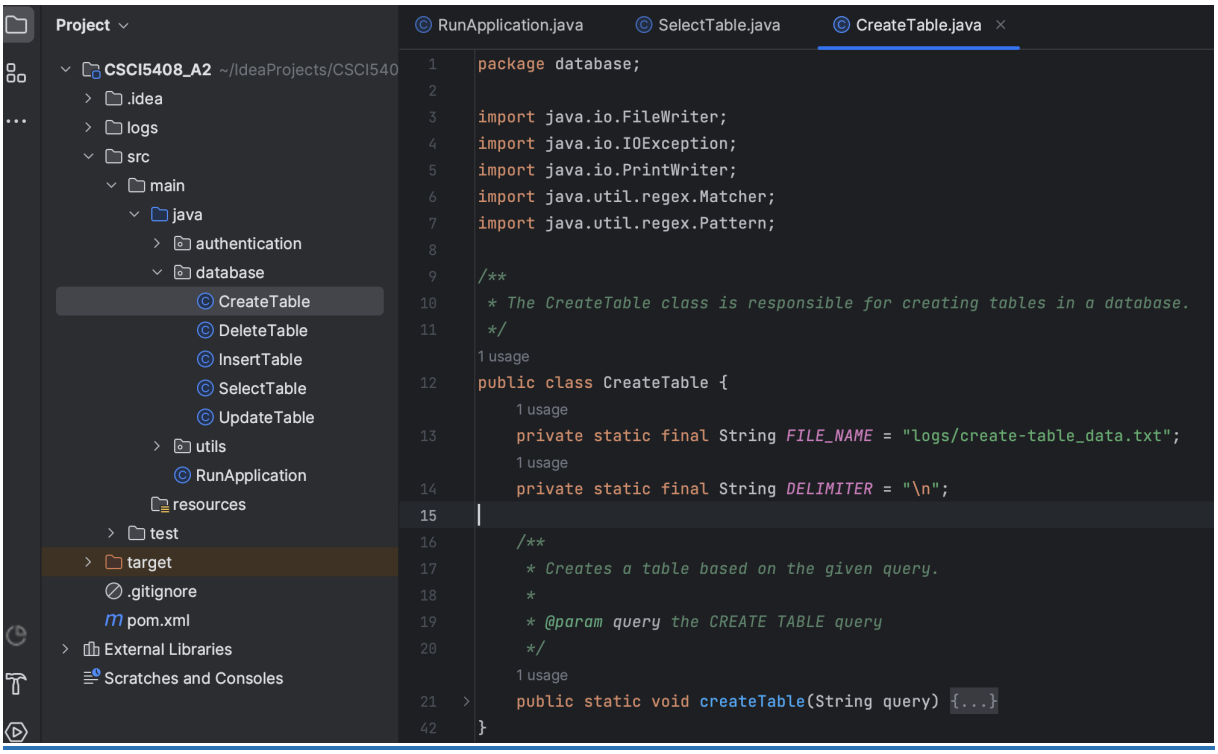

Figure 5: CreateTable class- JavaDocs specification for commenting styles.

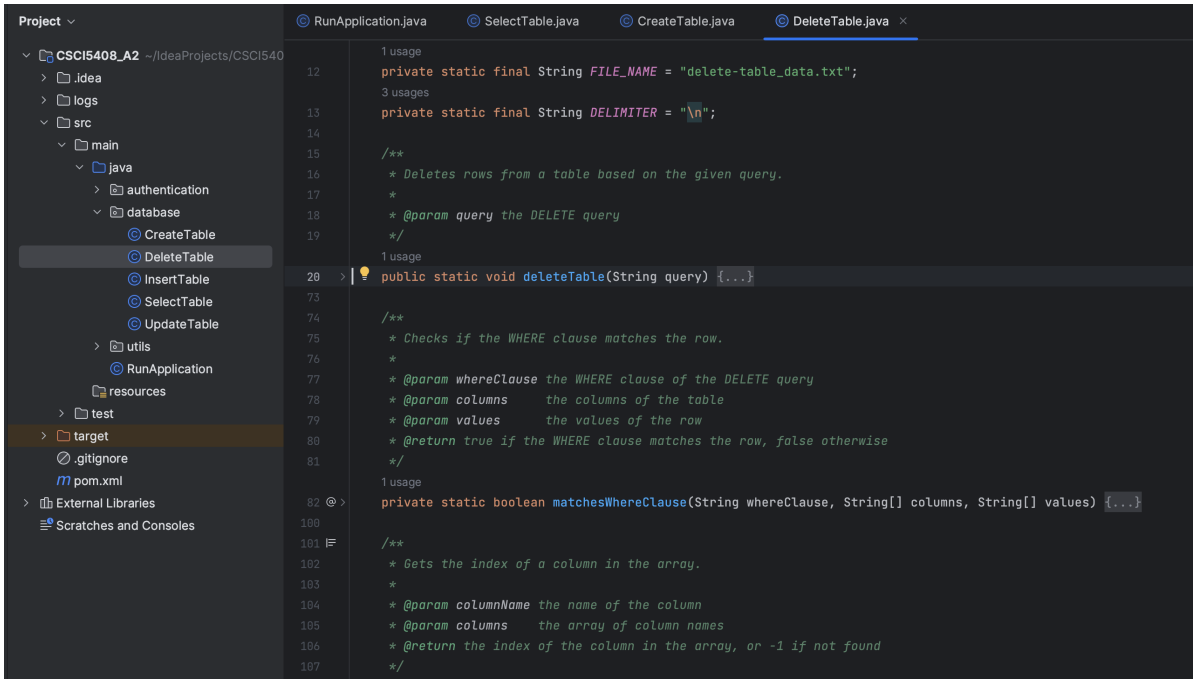

Figure 6: Delete Table class- JavaDocs specification for commenting styles.

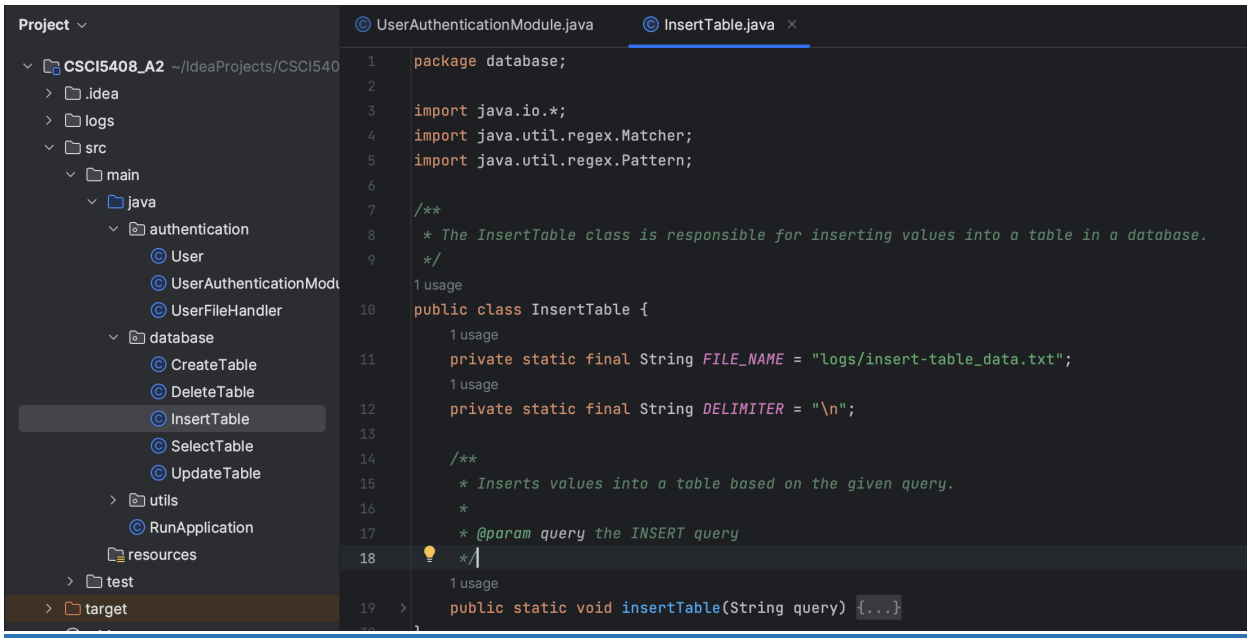

Figure 7: Insert Table class- JavaDocs specification for commenting styles.

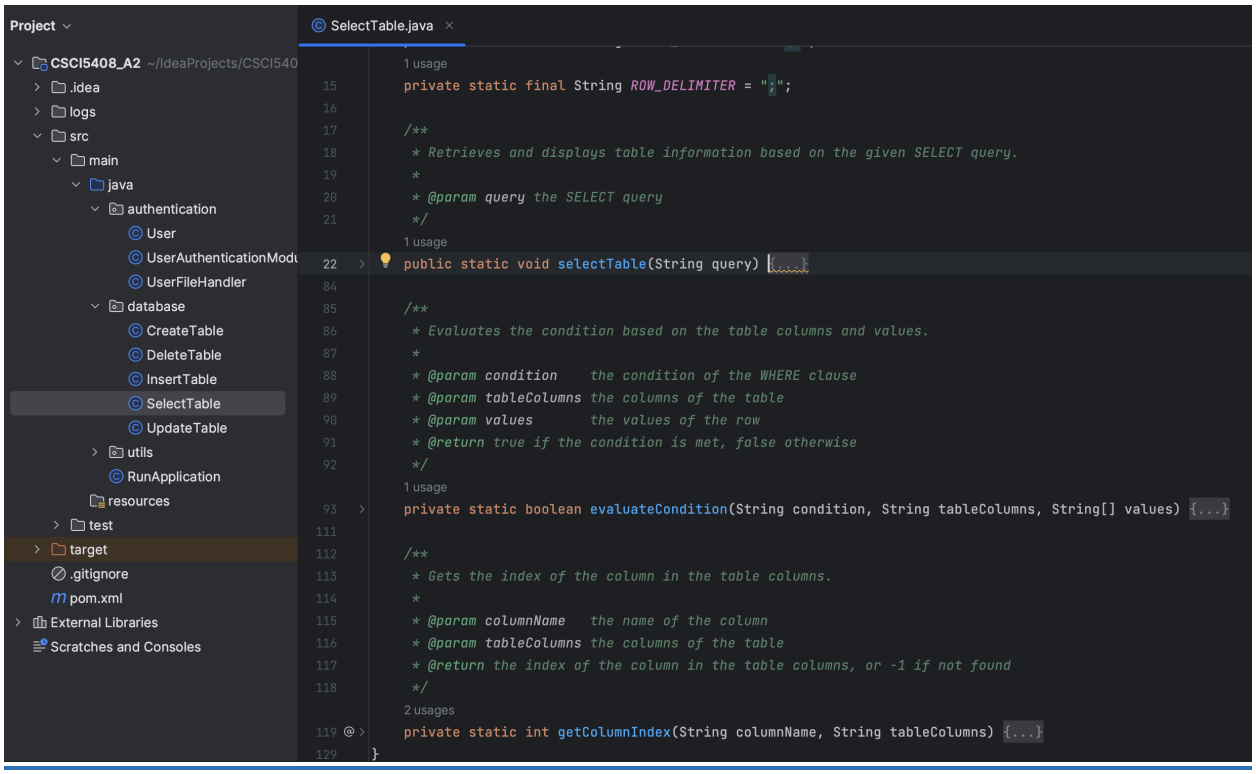

Figure 8: Select Table class- JavaDocs specification for commenting styles.

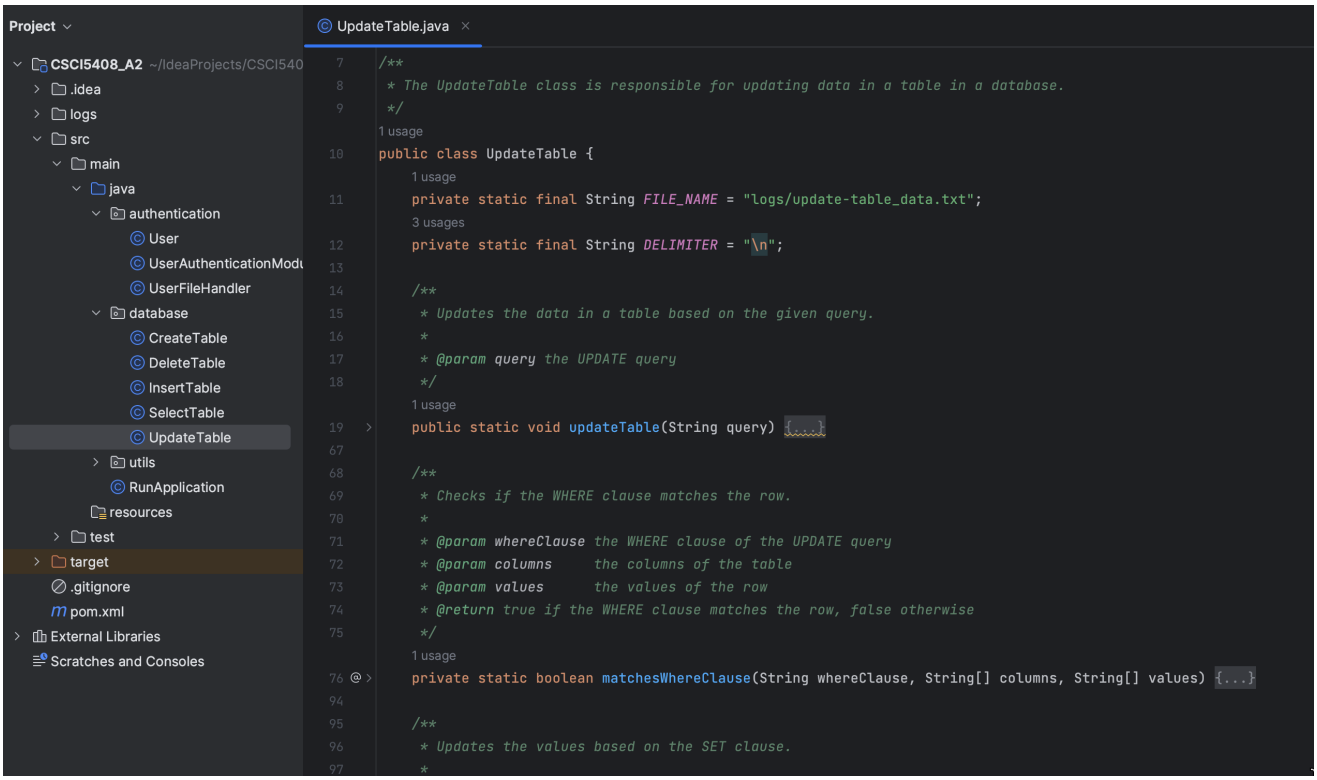

Figure 9: Update Table class- JavaDocs specification for commenting styles.

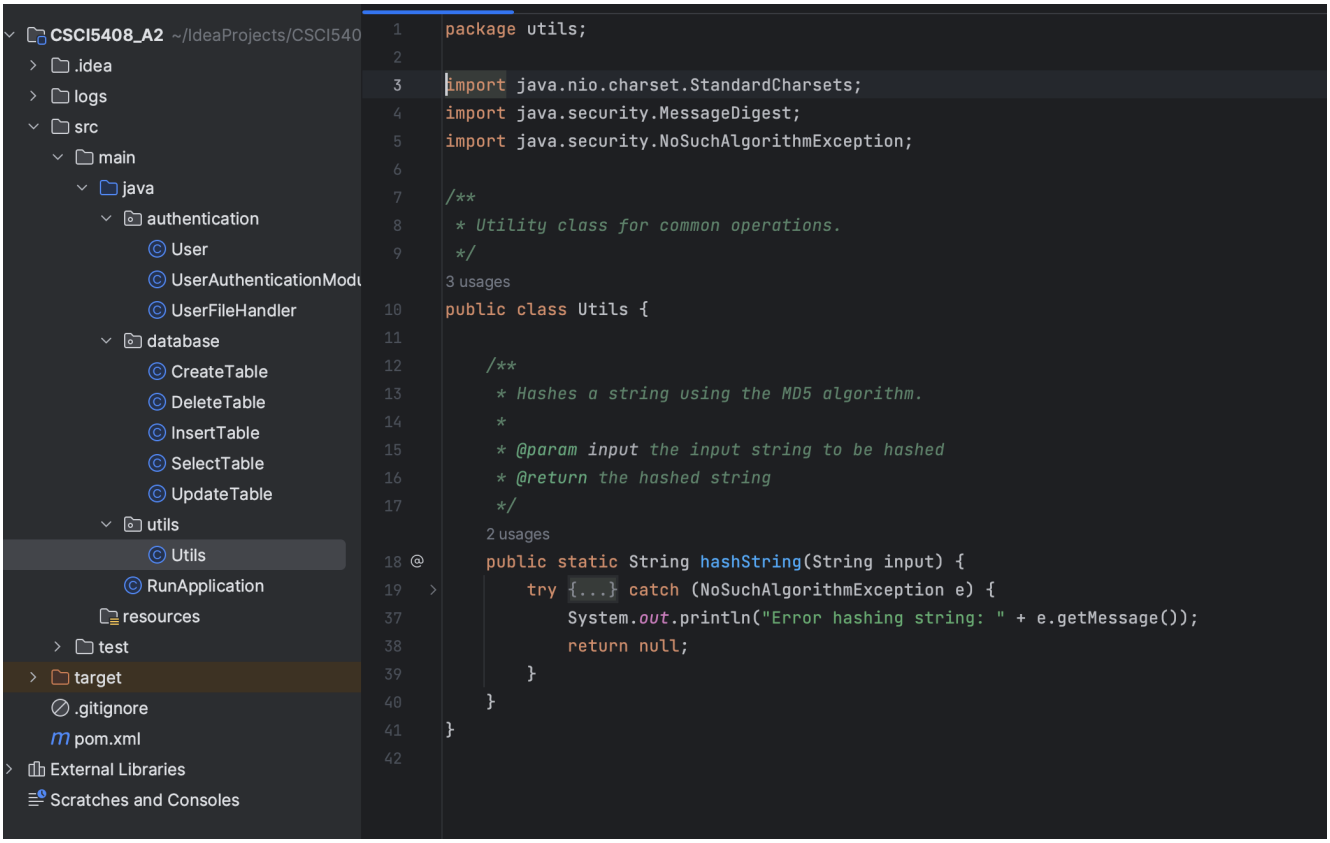

Figure 10: Utils class- JavaDocs specification for commenting styles.

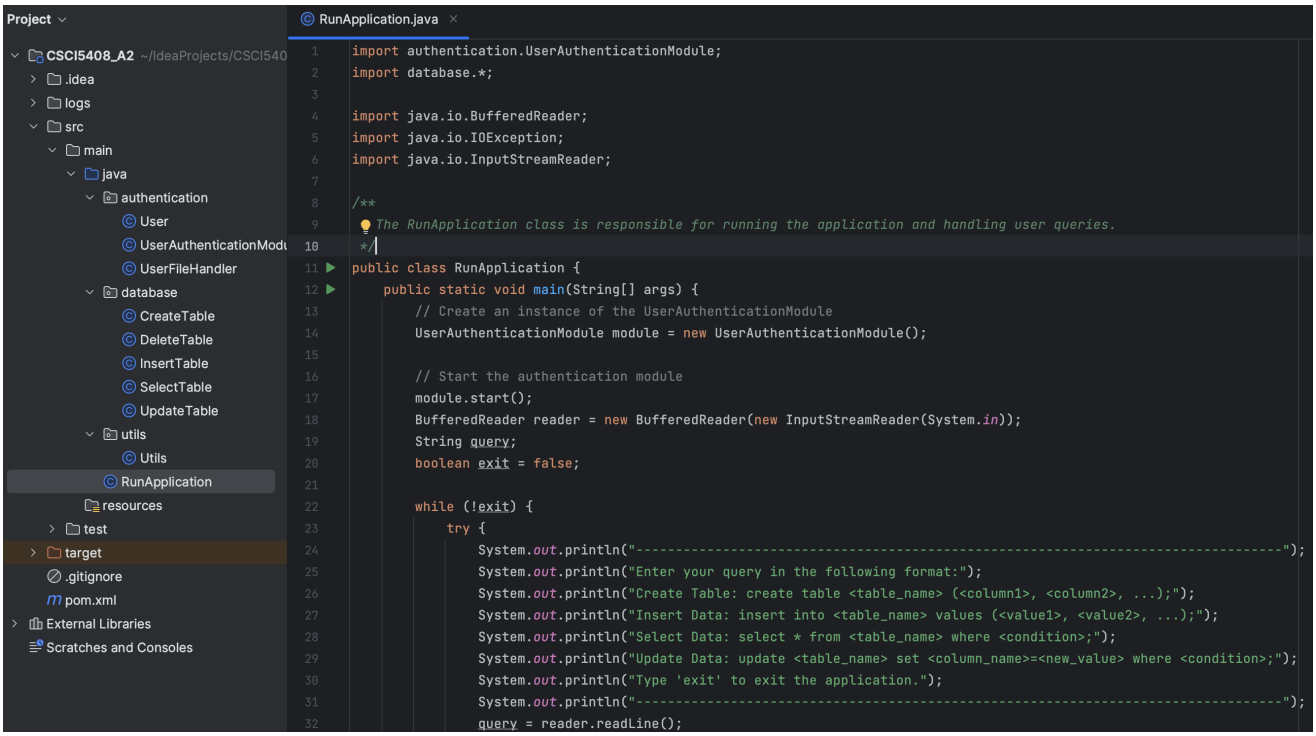

Figure 11: RunApplication class- JavaDocs specification for commenting styles.

3. The application code demonstrates some of the **SOLID principles** [3]. These design principles help promote code organization, maintainability, extensibility, and reusability, making the application more robust and easier to understand and maintain. The design principles used in the application program development/execution are:

   a. Single Responsibility Principle (SRP):
   - The UserAuthenticationModule class is responsible for handling user authentication.
   - The User class represents a user with authentication information.
   - The UserFileHandler class is responsible for handling user file operations.
   - The CreateTable, InsertTable, SelectTable, UpdateTable, and DeleteTable classes are responsible for their respective database operations.

   b. Open/Closed Principle (OCP):
   - The UserAuthenticationModule class is open for extension but closed for modification. It can be extended to support additional authentication features without modifying its existing code.

   c. Liskov Substitution Principle (LSP):
   - The UserAuthenticationModule class uses the UserFileHandler class as a dependency, following the LSP. It can use any subclass of UserFileHandler without issues.

   d. Dependency Inversion Principle (DIP):
   - The UserAuthenticationModule class depends on abstractions (interfaces) like Map and Scanner, instead of concrete implementations.
   - The UserAuthenticationModule class depends on the UserFileHandler class, which is an abstraction, allowing for flexibility in choosing different file handling methods.

   Additionally:

- The code separates different concerns into separate classes and modules. For example, the authentication functionality is encapsulated in the UserAuthenticationModule, and the database operations are handled by separate database-related classes.
- The code uses encapsulation to hide internal details and expose only necessary information through getters and setters. For example, the User class encapsulates the user's authentication information, and the UserAuthenticationModule provides methods for creating users, logging in, and checking authentication status.

4. The application is totally **console-based** (no GUI added) and it accept user input in the form of SQL query, once the user has successfully logged in.

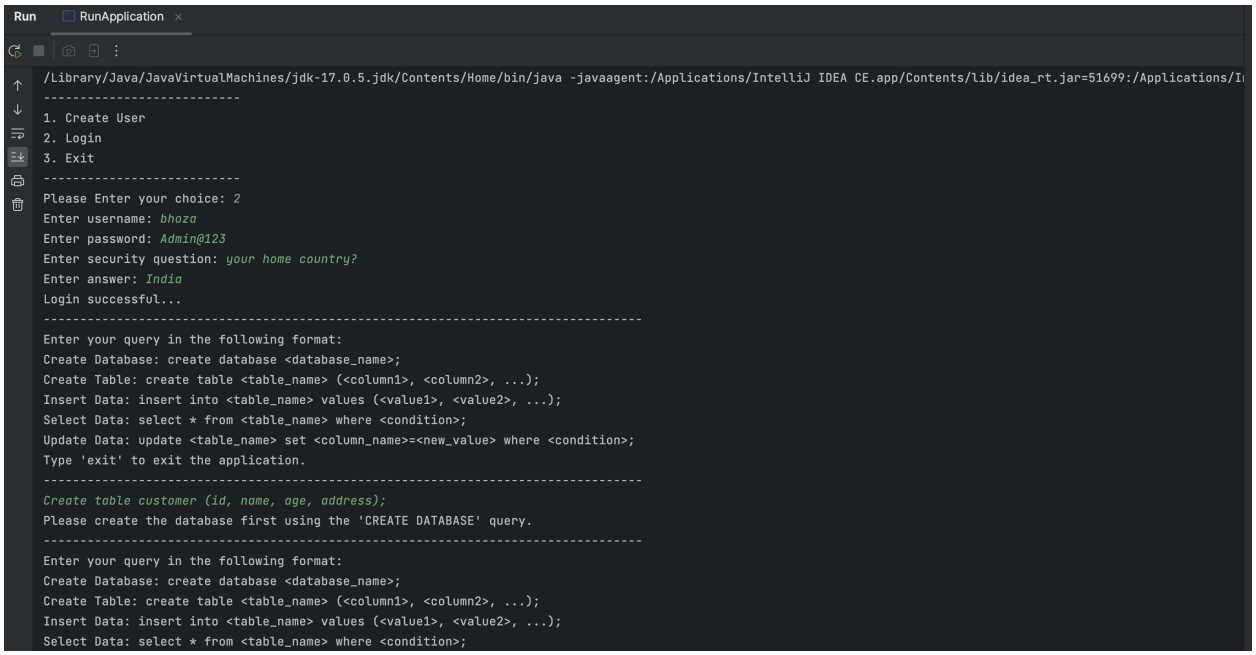

Figure 12: Console based User authentication program.

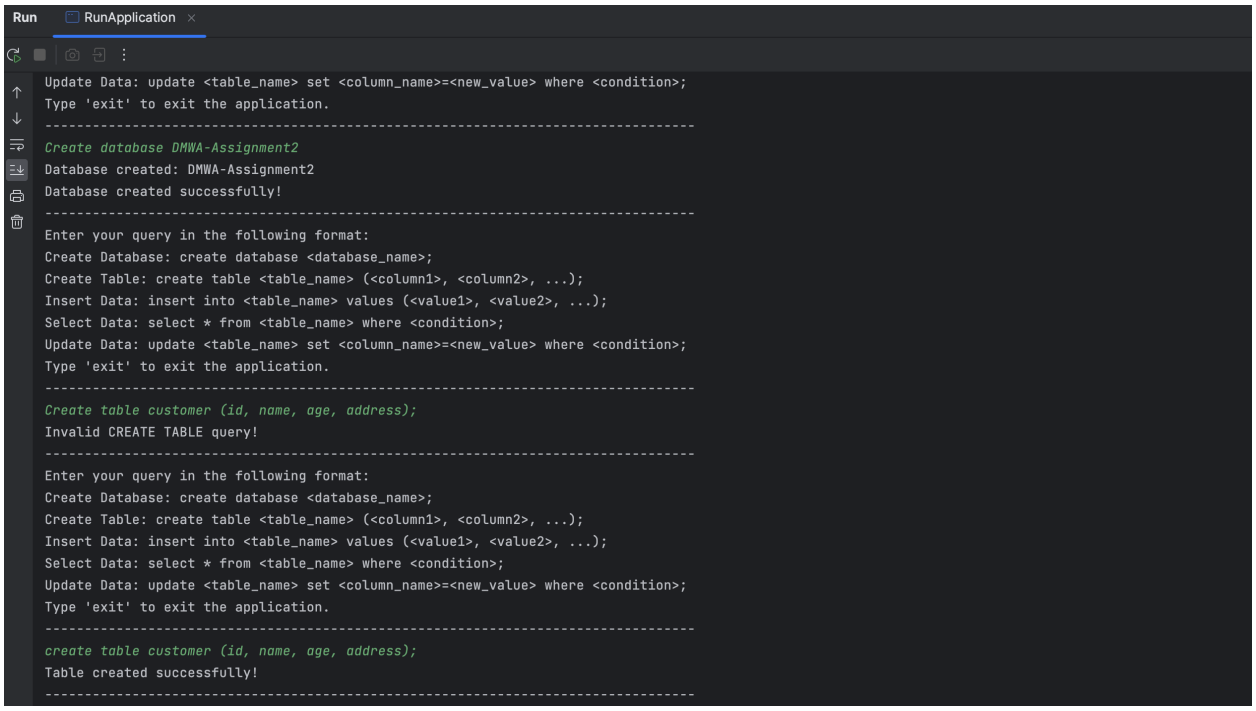

Figure 13: Console-based program which accept user input in the form of SQL query, after the user login.

5. **Two factor user authentication**:

The application supports multi-user authentication, which is critical for protecting sensitive data. When users create accounts, their passwords are hashed and securely stored in the database. This ensures that even if the database is compromised, the passwords remain secure.

To gain access to the database, users must enter their login credentials and answer to the security questions, which are then validated against the hashed passwords stored in the database using MD5 algorithm [4]. This adds another layer of security by preventing unauthorised access to sensitive data.

Overall, the code provides a console-based interface for user authentication and data management, ensuring the security and persistence of user information.

a. The **UserAuthenticationModule** class handles user authentication by allowing users to create accounts, log in with their credentials, and perform various operations based on their authentication status.
b. The **User** class represents a user with authentication information such as username, password, security question, and answer.
c. The **UserFileHandler** class is responsible for loading and saving user data to a file.
d. The **Utils** class provides utility methods, including hashing strings using the MD5 algorithm.
e. user_details.txt file captures the user login details including the hashing for the password.

Working flow of User Authentication module:

- The application starts by executing the main method in the RunApplication class, which displays the menu options for the user: "1. Create User", "2. Login", and "3. Exit"

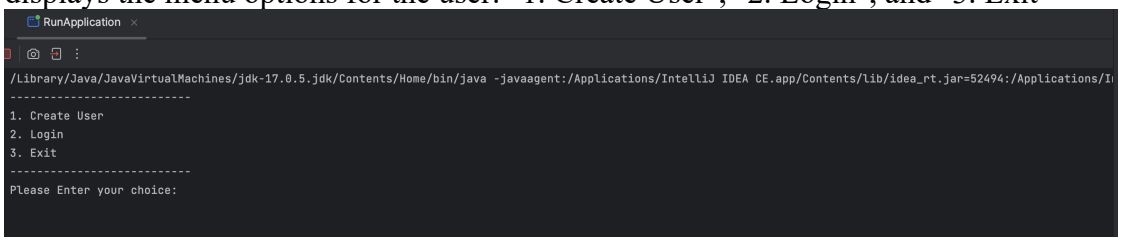

Figure 14: Menu options for User Authentication

- Let's say the user enters "1" to create a new user. They are prompted to enter a username.

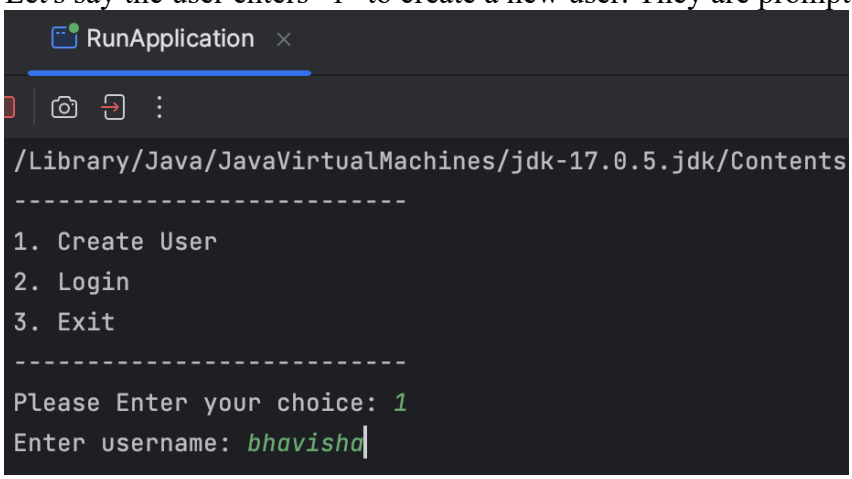

Figure 15: Starting of Register User process.

- Since it's a new user, the username is valid. The user is then prompted to enter a password. The application hashes the entered password using the Utils.hashString method for security.

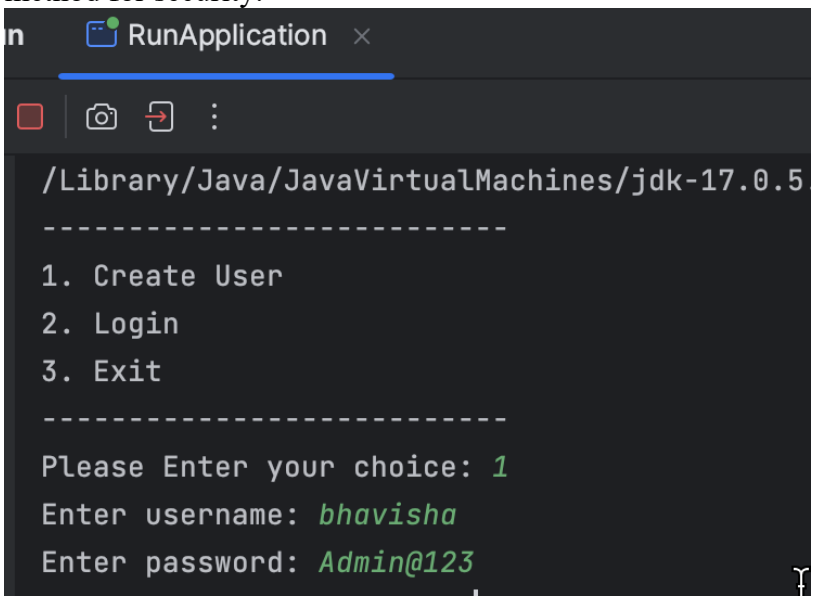

Figure 16: Username and Password entering for registration.

- The user is prompted to choose a security question after that.

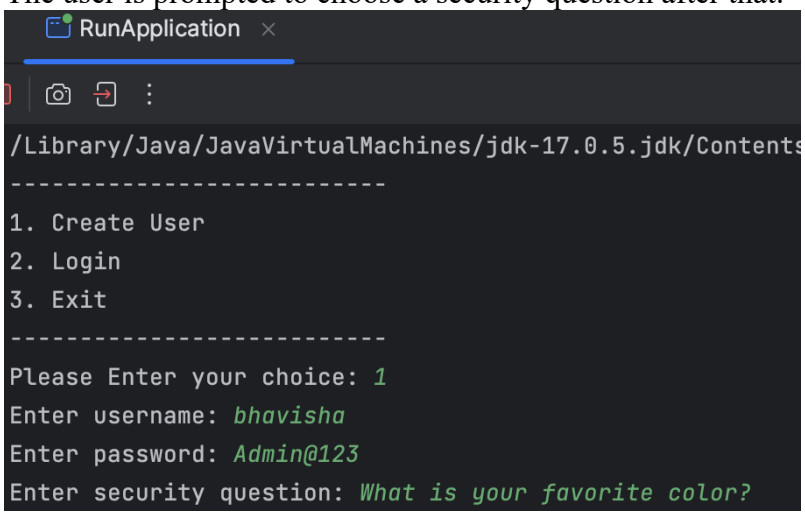

Figure 17: Security question to enter for registration.

- Once the answer is entered, it displays a success message, indicating that the user has been created. and it will again prompt the menu options.

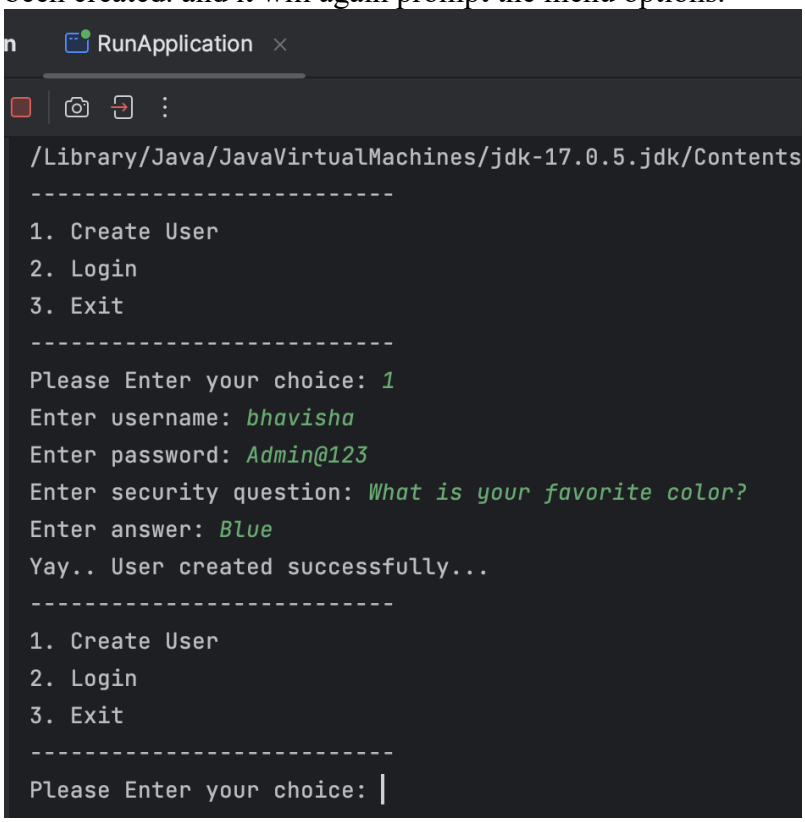

Figure 18: User registration completion

- The entered details are stored in text file named “user_details.txt” under logs directory. It supports the multiuser authentication, so it can add all data, when it is registered.

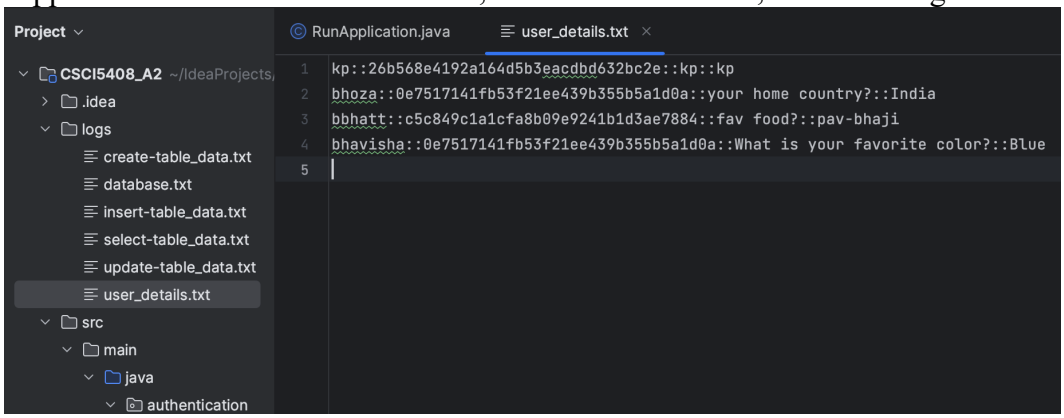

Figure 19: User registration data stored in text file.

- Now let’s say user wants to login and enters “2” on console, they are prompted to enter their username.

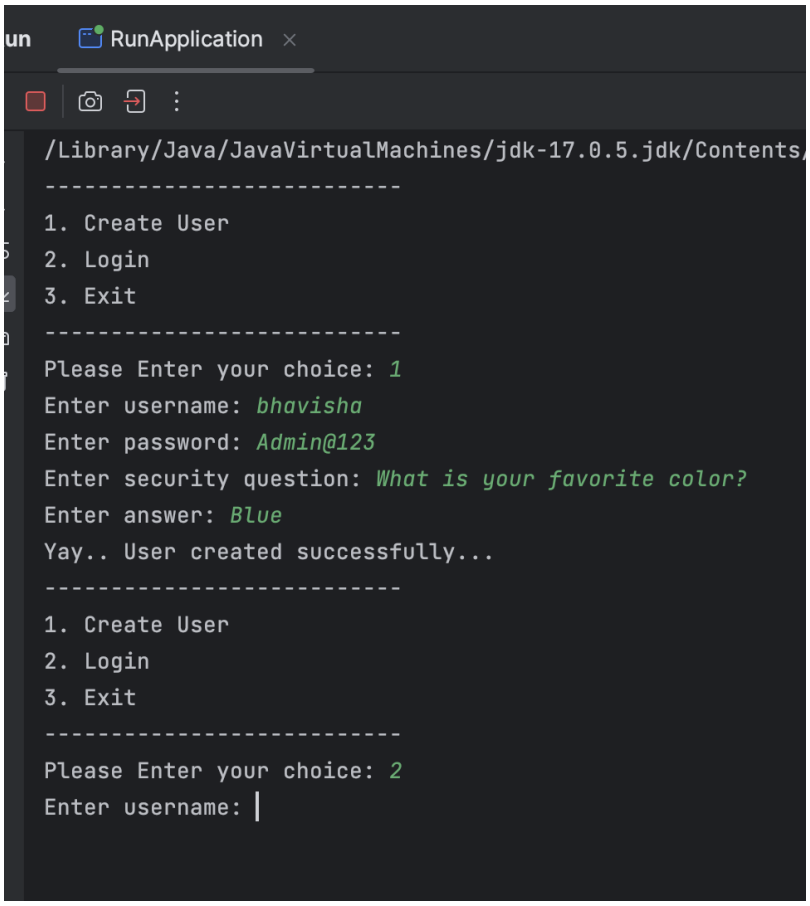

Figure 20: User Login screen.

- The application checks if the entered username exists, otherwise it will show the message of “Invalid Username!”. Since the username exists, the login process continues, and will happen same for password, security question and answer. If the answer is correct, the login process is considered successful, and a login success message is

displayed.

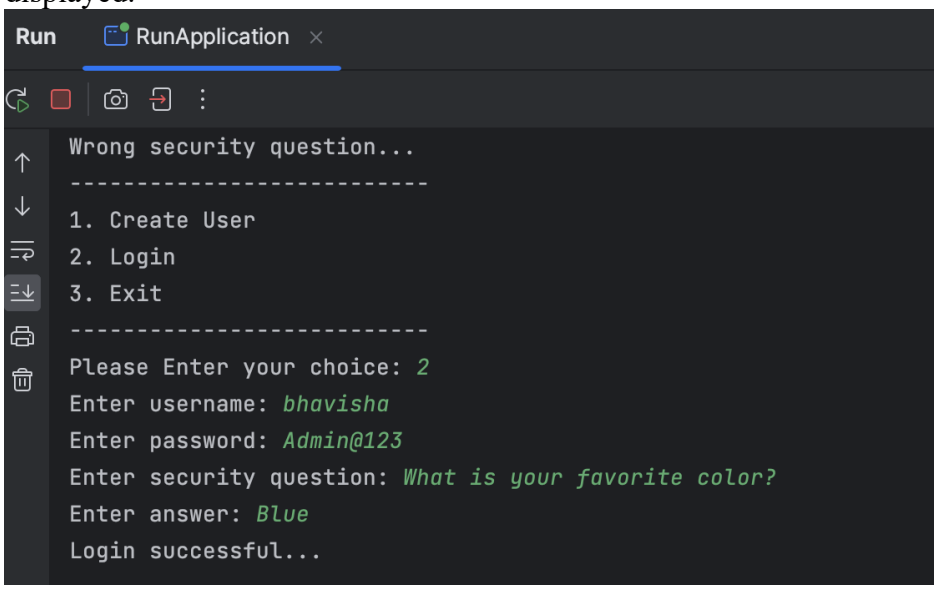

Figure 21: User login completion

If the username is not found it will show the message of "Username not found!"

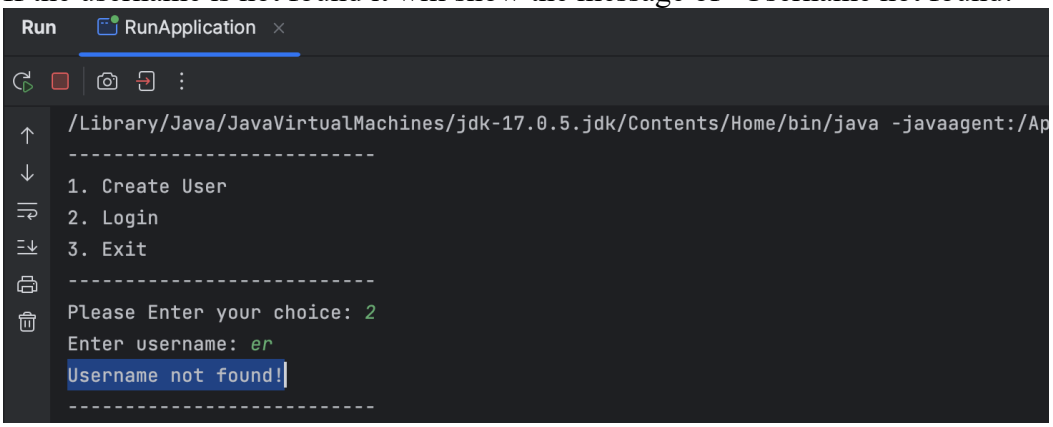

Figure 22: Username mismatch

If the password is wrong, then it will show the message "Wrong password..."

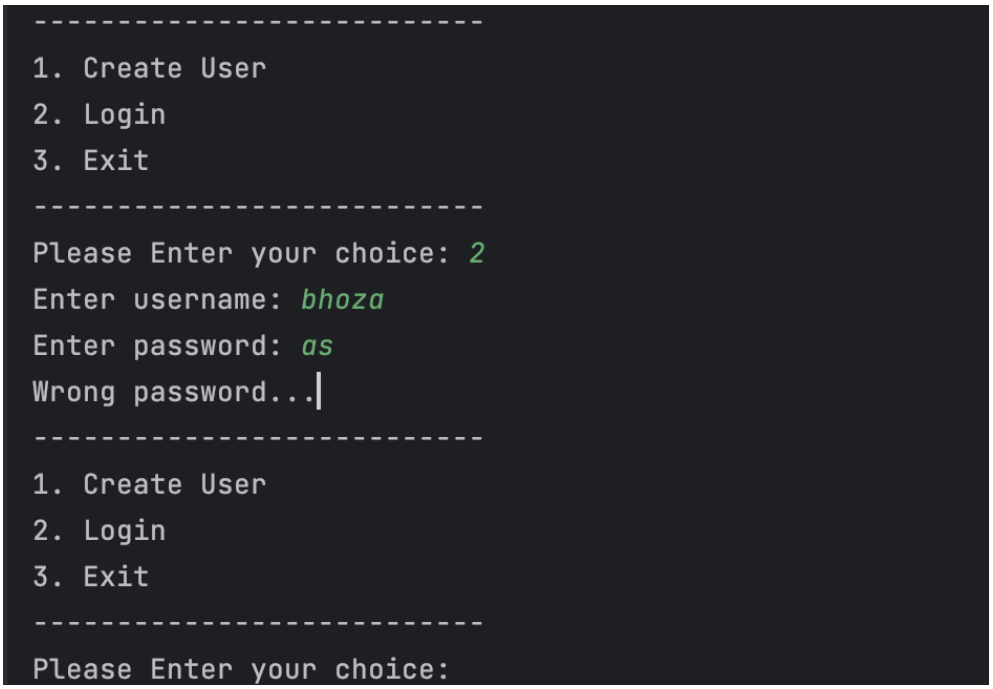

Figure 23: Password mismatch

If the security question/answer mismatches, it will show the error message as "Wrong answer."

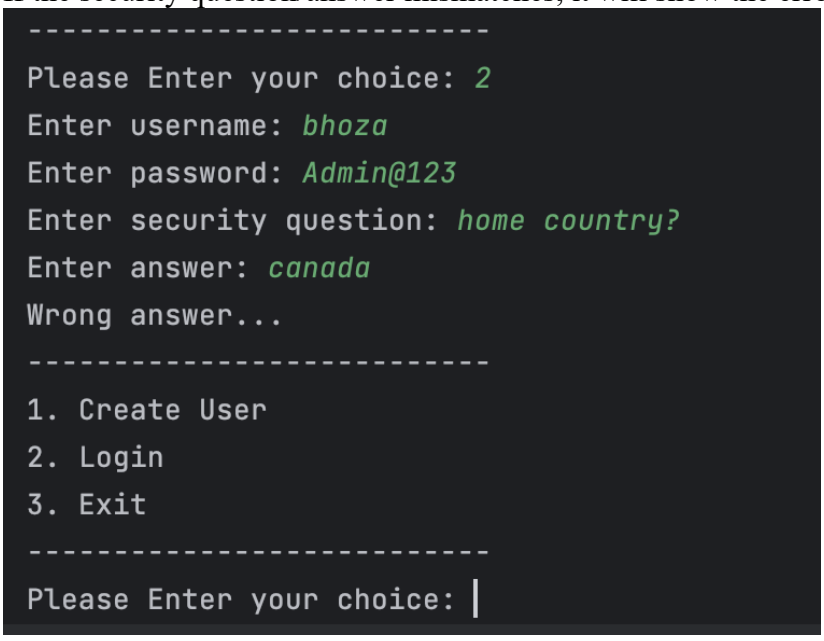

Figure 24: Security question/answer mismatch

6. Implementation of Queries (DDL & DML):

The application also implements various SQL queries such as CREATE, SELECT, INSERT, UPDATE, and DELETE on any number of tables. Separate methods have been created for each query.

The application parses queries using **regular expressions (regex)**, providing users with a powerful and flexible way to query their database. The application's regex engine is highly efficient and can easily handle complex questions.

The application supports a diverse set of regex patterns, allowing users to query the database in a flexible and efficient manner. Users can, for example, search for records based on specific keywords or retrieve data from specific columns. This allows users to find the information quickly and efficiently they require. The concept of regex and checking if the queries are rightly written or not, have used the regex101 site which highlights the syntax which is written correctly [5].

Working flow of queries implementation (CREATE, SELECT, INSERT, UPDATE, and DELETE):

- The application starts by executing the main method in the RunApplication class, which displays the menu options for the user authentication. Once the user is logged in, it will prompt the user to enter a query in a specific format.

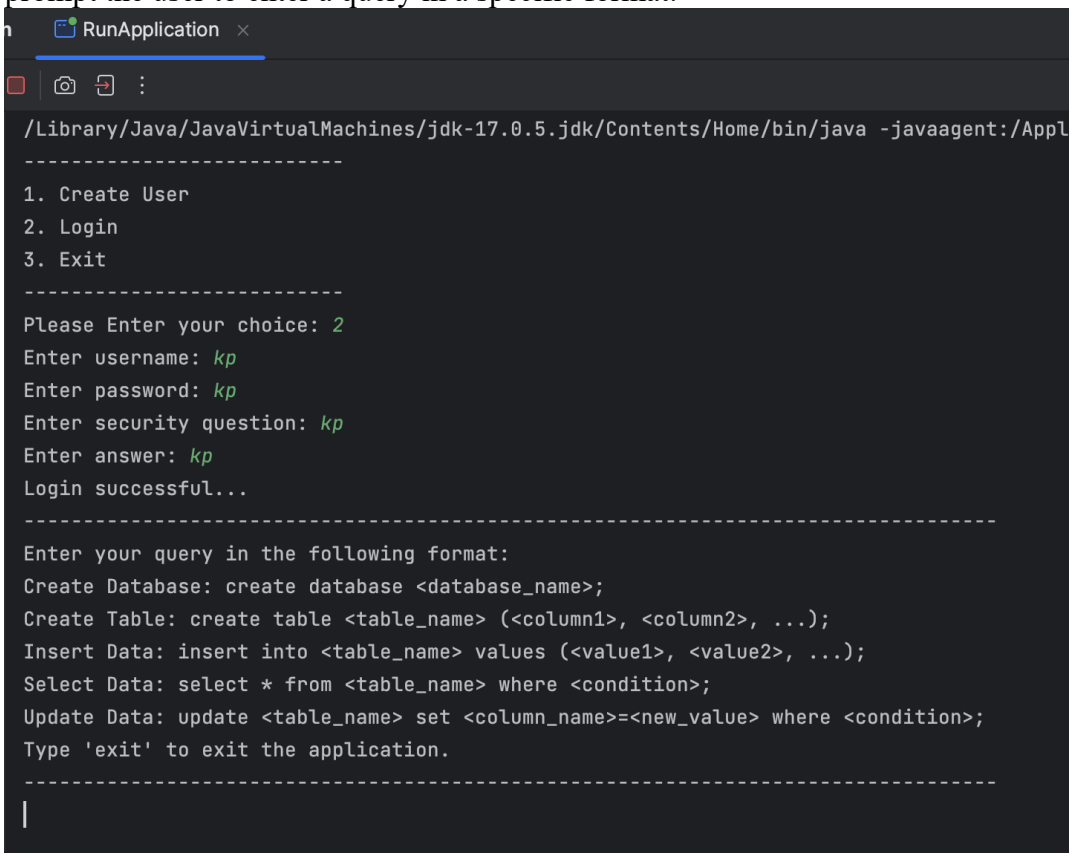

Figure 25: Query insertion menu after login completion

- If the query is not in the given format, it will show the error message “Please create the database first using the ‘**CREATE DATABASE**’ query.”

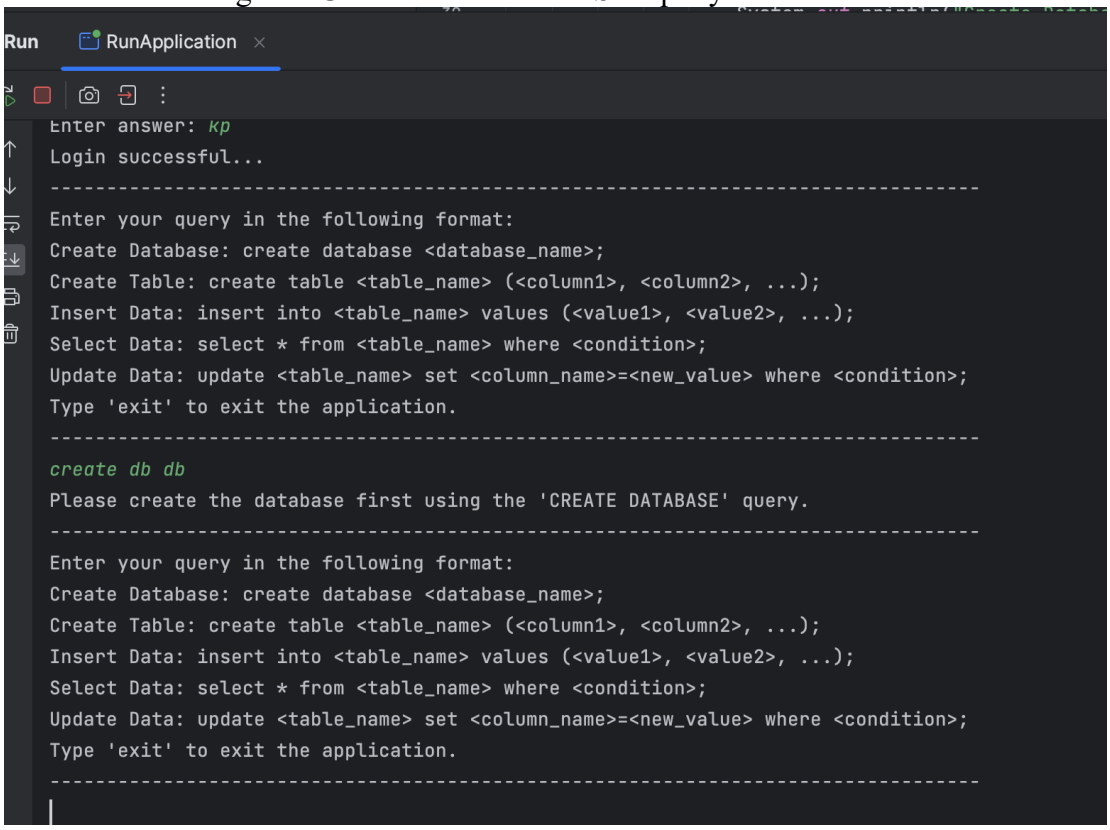

Figure 26: Error message for invalid query insertion

- Let's say the user wants to create a database and enters the query “create database my_dmwaA2_db;”. The application checks if the query starts with create database (case-insensitive). Since it matches, it continues processing and, a success message is displayed: “Database created: my_dmwaA2_db.”

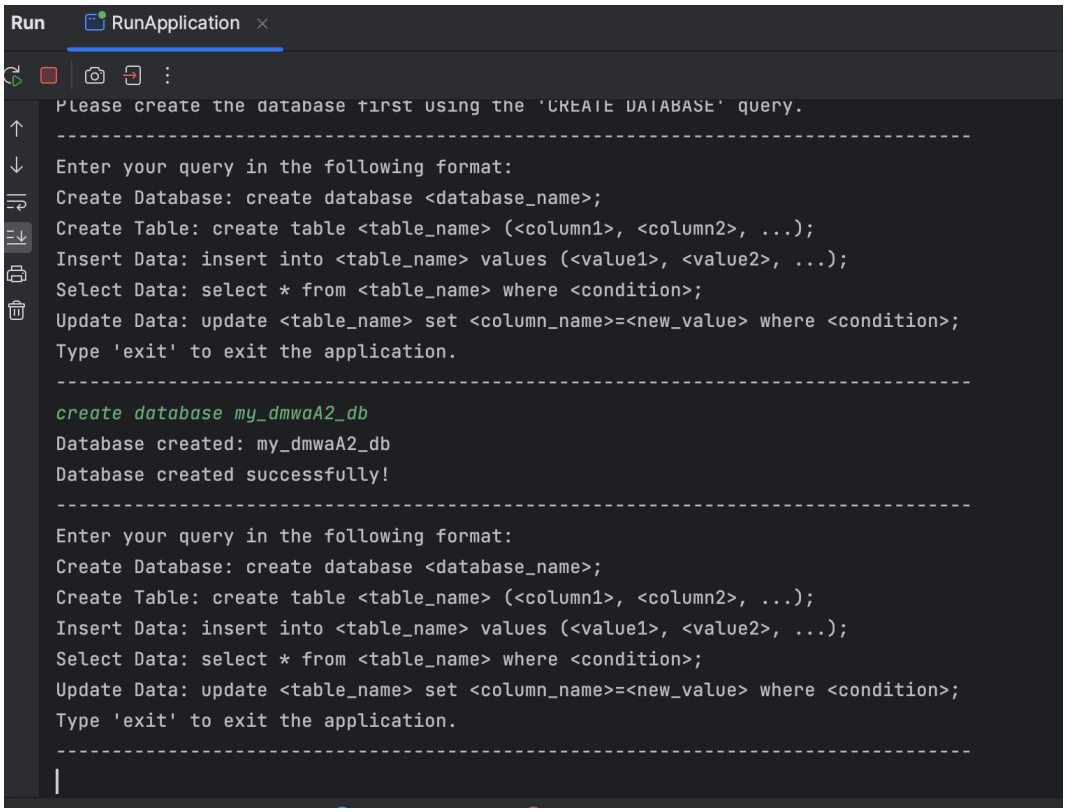

Figure 27: Successful Database creation

- If the database name is not null (indicating a valid query), the method proceeds to save the database name to a text file.

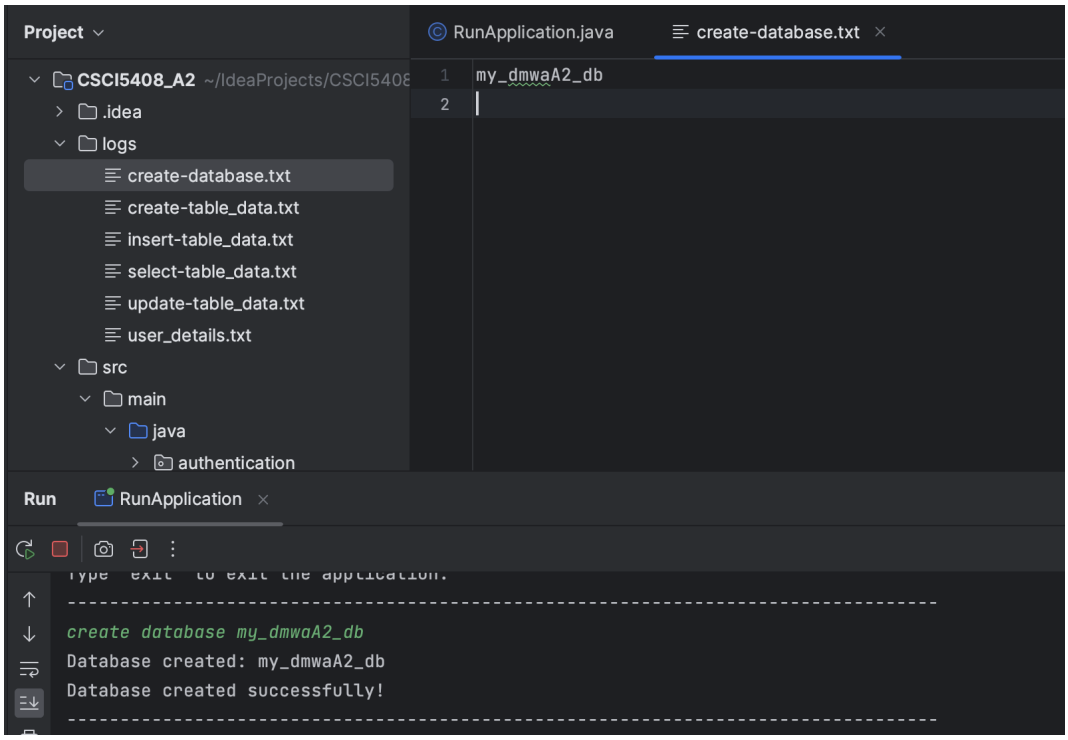

Figure 28: Create Database saved in text file.

- After processing the “create database” query, the application continues to display the menu options and wait for the next user input. If the user wants to exit, they can enter "exit" to terminate the application.

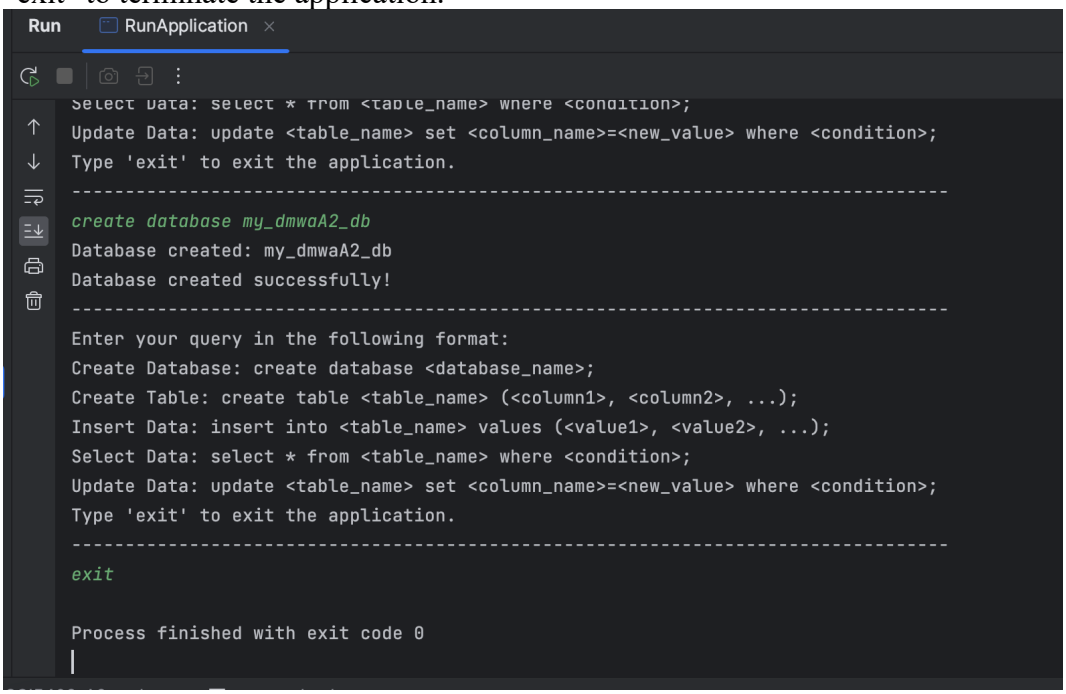

Figure 29: Exit to terminate the application.

- Now let's say the user wants to create a table and enters the query “create table students (banner_id, name, course);”. The application checks if the query starts with "create table" (case-insensitive). Since it matches, it continues processing. Inside the CreateTable.createTable method, the query is parsed and validated using regular expressions (regex). The regex pattern "create\s+table\s+(\w+)\s+\((.+)\);" is used to extract the table name and the column definitions. Once the write operation is successful, a success message is displayed: "Table created successfully!"

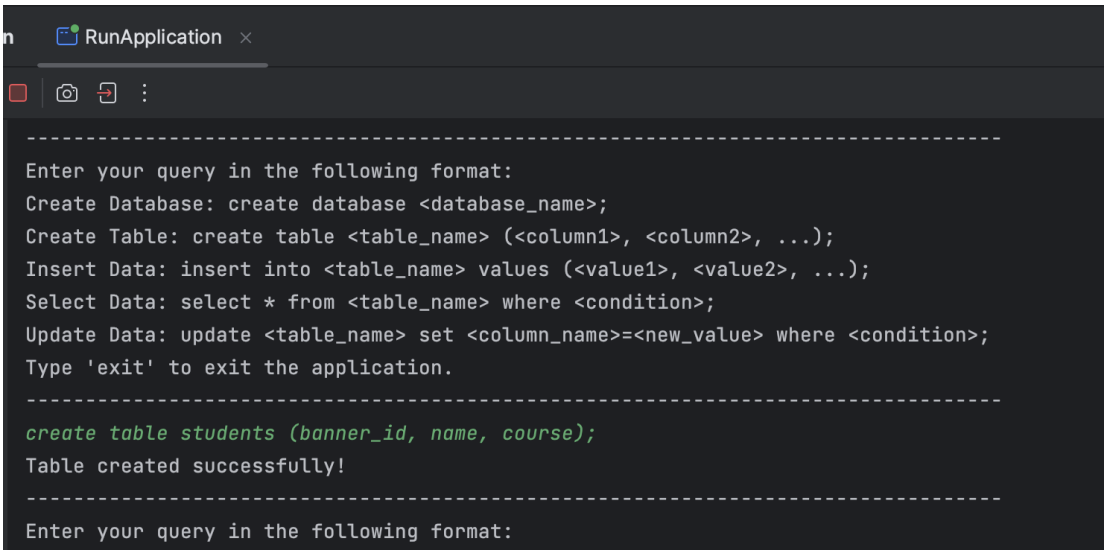

Figure 30: Successful Student table creation

- If the query matches the pattern and the table name and column definitions are successfully extracted, the method proceeds to store the table information in a custom file “create-table_data.txt”.

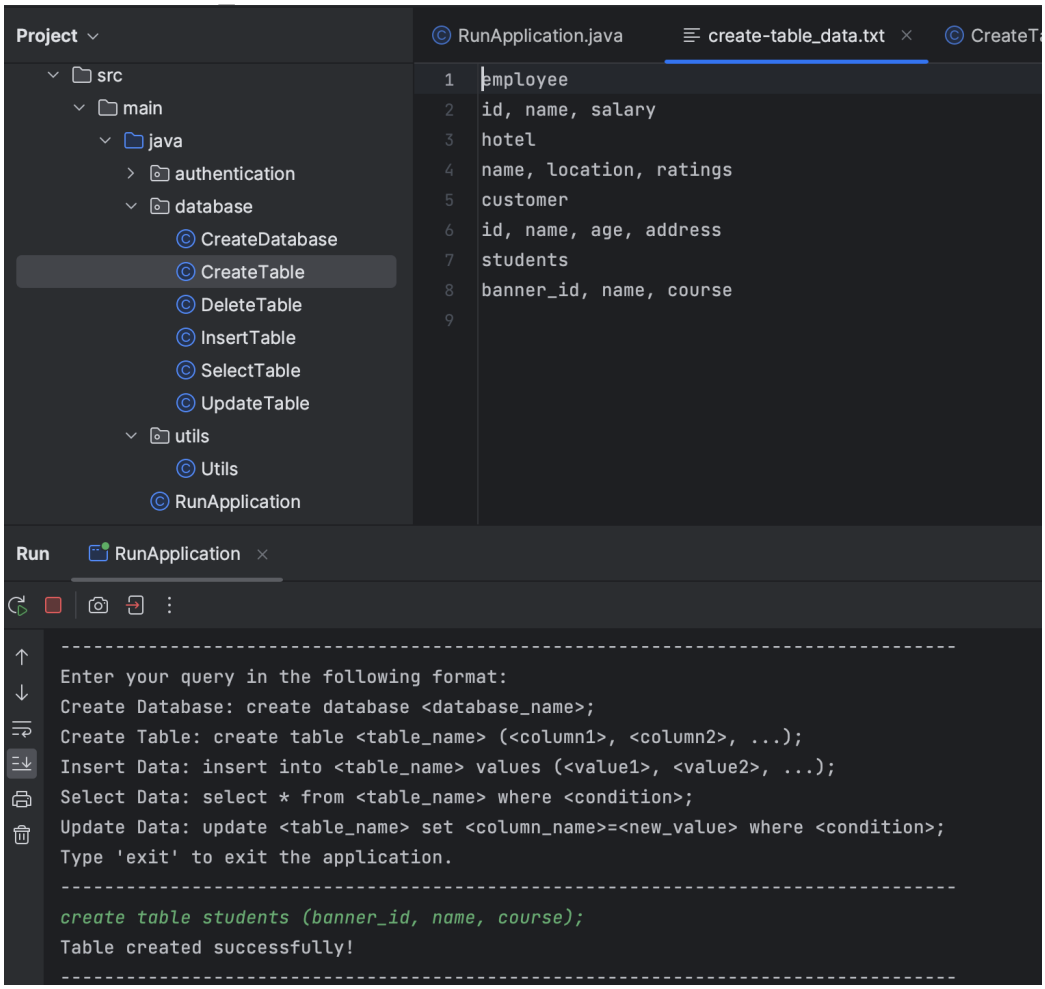

Figure 31: Create table saved in text file.

- After processing the “create table” query, the application continues to display the menu options and waits for the next user input. Let's say the user wants to insert values into a students table and enters the query “insert into students values (B00935827, Bhavisha_Oza, MACS);”. The application checks if the query starts with “insert into” (case-insensitive). Since it matches, it continues processing. Inside the InsertTable.insertTable method, the query is parsed and validated using regular expressions (regex). The regex pattern "insert\s+into\s+(\w+)\s+values\s*\((.+)\)\s*;\s*" is used to extract the table name and the values to be inserted. Once the write operation is successful, a success message is displayed: “Data inserted successfully!”

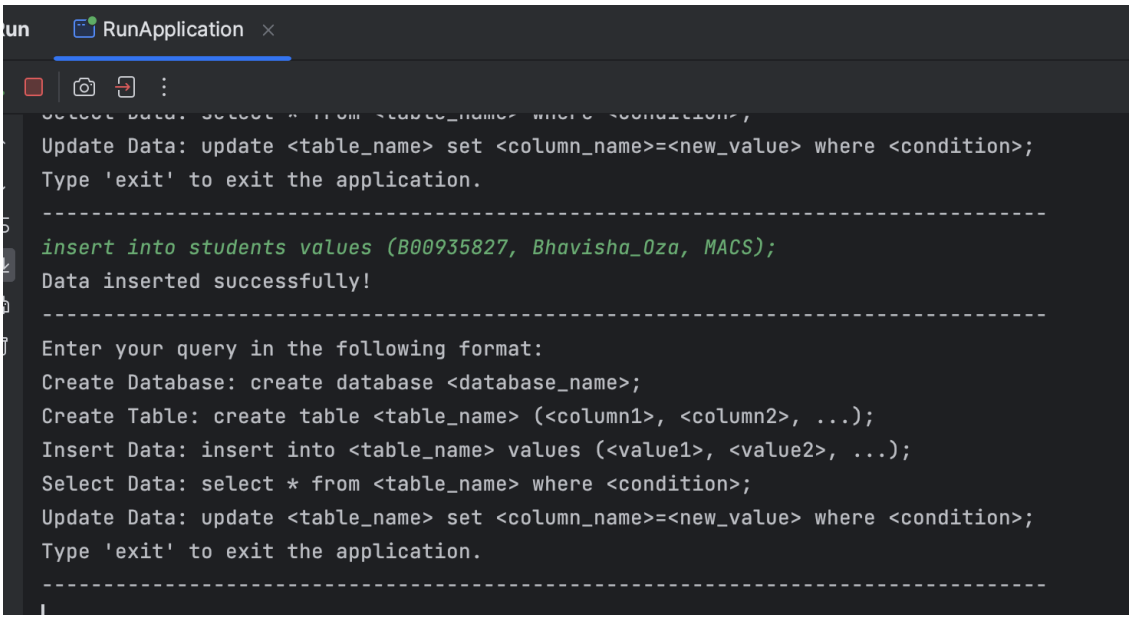

Figure 32: Successful data insertion in Student table.

- If the query matches the pattern and the table name and values are successfully extracted, the method proceeds to store the inserted values in a custom file “insert-table_data.txt”.

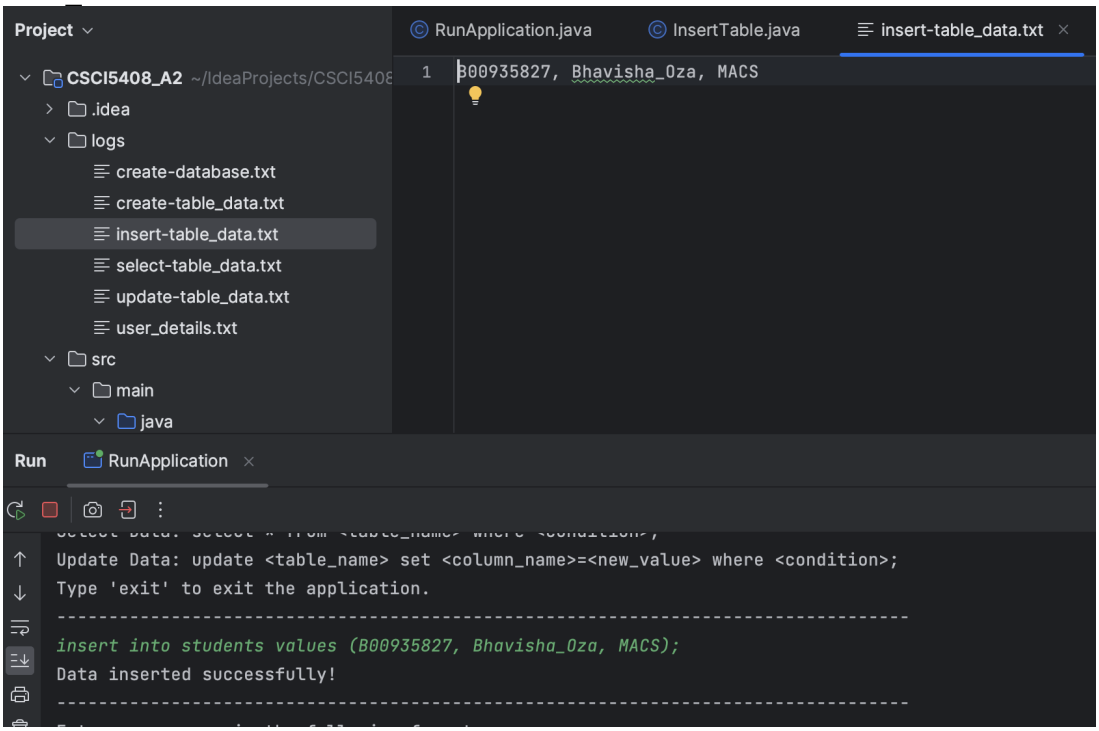

Figure 33: Insert table data saved in text file.

- After processing the “insert table” query, the application continues to display the menu options and waits for the next user input. Let's say the user wants to select values into a students table and enters the query “select * from students where banner_id = 1;”. The application checks if the query starts with "select" (case-insensitive). Since it matches, it

continues processing. Inside the SelectTable.selectTable method, the query is parsed and validated using regular expressions (regex). The regex pattern "select\s+(\|[a-zA-Z_,]+)\s+from\s+([a-zA-Z_]+)(?:\s+where\s+([a-zA-Z\d_]+\s=\s*\d+));?" is used to extract the columns, table name, and condition.
It iterates through the lines of the file and checks if the table name matches the requested table. If found, it splits the line using the TABLE_DELIMITER constant and extracts the table columns and rows.
Once a condition is provided, it evaluates the condition by comparing the column value with the given condition. If the condition is met, it either displays the entire row or selectively displays the requested columns based on the user's query.
If the query matches the pattern and the columns, table name, and condition are successfully extracted, the method proceeds to retrieve and display the table information.

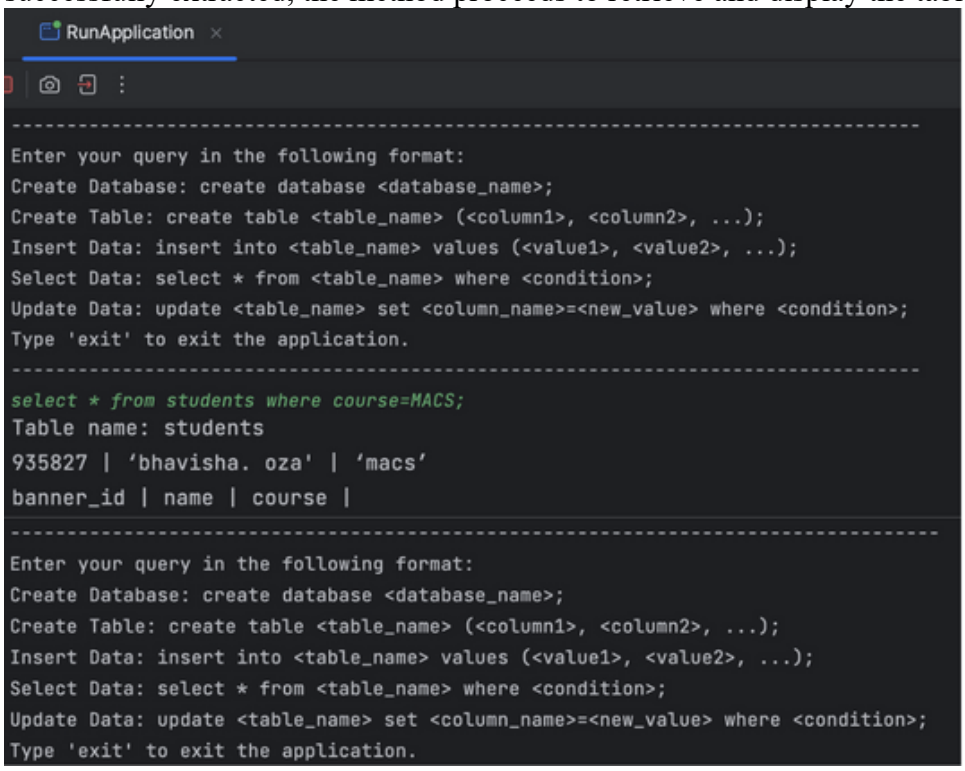

Figure 34: Select query for the condition course = MACS.

- After processing the table data, the application continues to display the menu options and waits for the next user input. Let's say the user wants to update data in a table and enters the query “update students set name=Kairavi where course=MACS;”.
  The application checks if the query starts with "update" (case-insensitive). Since it matches, it continues processing.
  Inside the UpdateTable.updateTable method, the query is parsed and validated using regular expressions (regex). The regex pattern "update\s+(\w+)\s+set\s+(\w+)\s*=\s*(\w+)\s+where\s+(\w+)\s*=\s*(\w+)\s*;" is used to extract the table name, set clause, and where clause.
  Once the query matches the pattern and the table name, set clause, and where clause are successfully extracted, the method proceeds to update the data.
  It checks if the row matches the condition specified in the where clause. If the condition is met, it updates the values based on the set clause.

The application displays the message "Data updated successfully!" to indicate that the update was successful.

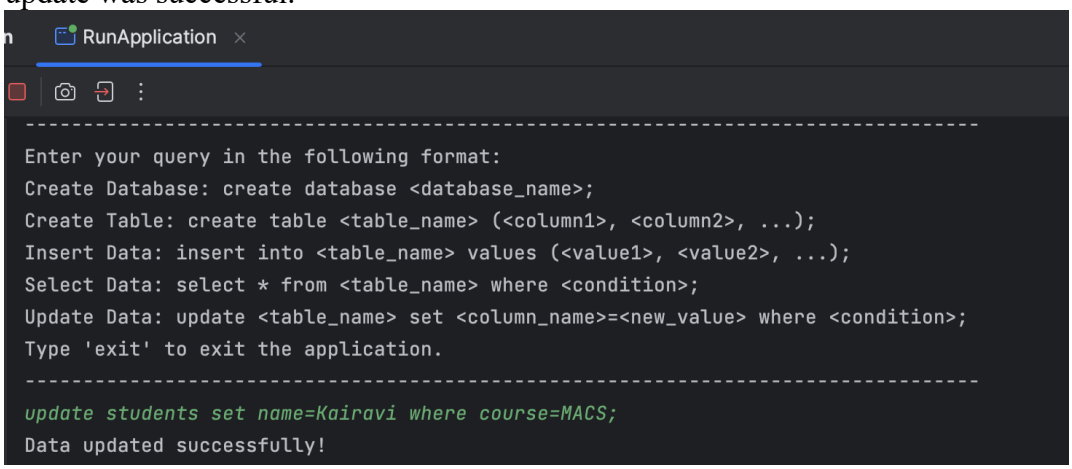

Figure 35: Update query for the condition course = MACS.

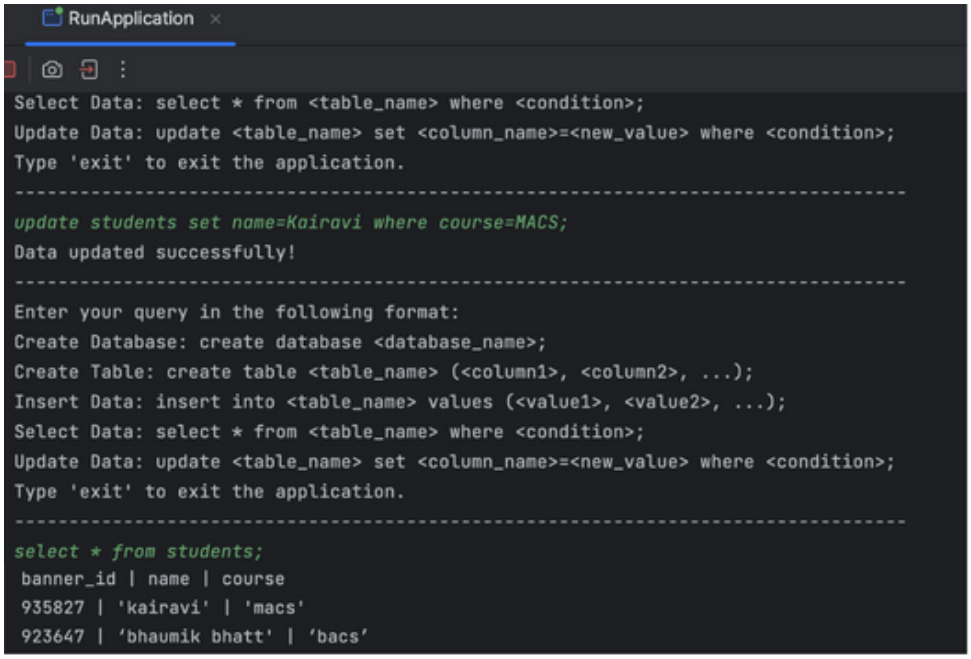

Figure 36: Select query for the updated value check name = Kairavi.

- If the query matches the pattern and the table name and values are successfully extracted, the method proceeds to update the inserted values in a custom file “update-table_data.txt”.

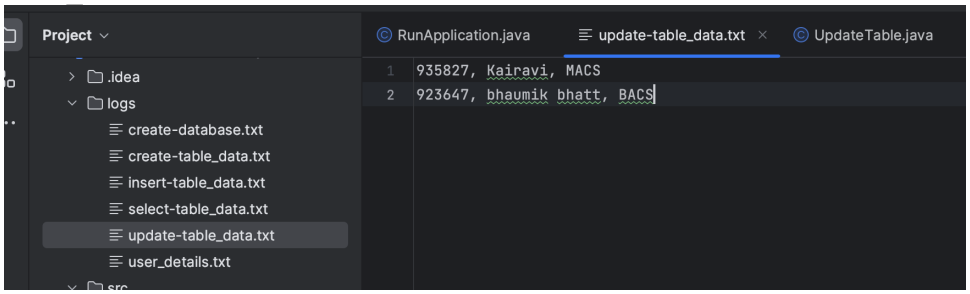

Figure 37: Update table data saved in text file.

- The application continues to display the menu options and waits for the next user input. Let's say the user wants to delete rows from a table and enters the query “DELETE FROM students where banner_id=923647;”.
  The application checks if the query starts with “delete from” (case-insensitive). Since it matches, it continues processing.
  Inside the DeleteTable.deleteTable method, the query is parsed and validated using regular expressions (regex). The regex pattern "DELETE FROM ([a-zA-Z0-9_]+) WHERE (.+)" is used to extract the table name and the WHERE clause.
  If the query matches the pattern and the table name and WHERE clause is successfully extracted, the method proceeds to delete the rows. If the file replacement is successful, it displays the message "Data deleted successfully!" to indicate that the deletion was successful.

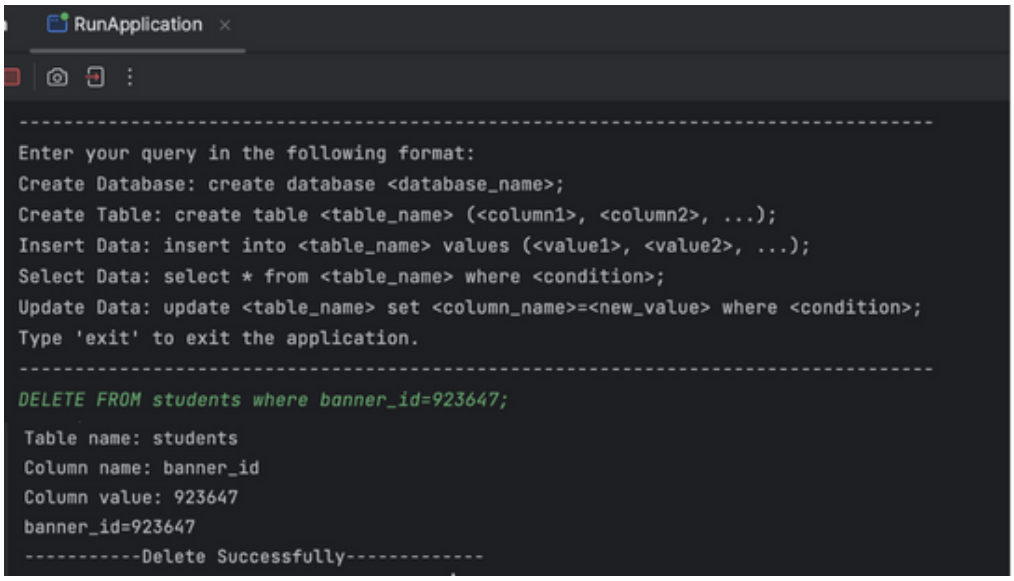

Figure 38: Delete query for the condition banner_id = 923647.

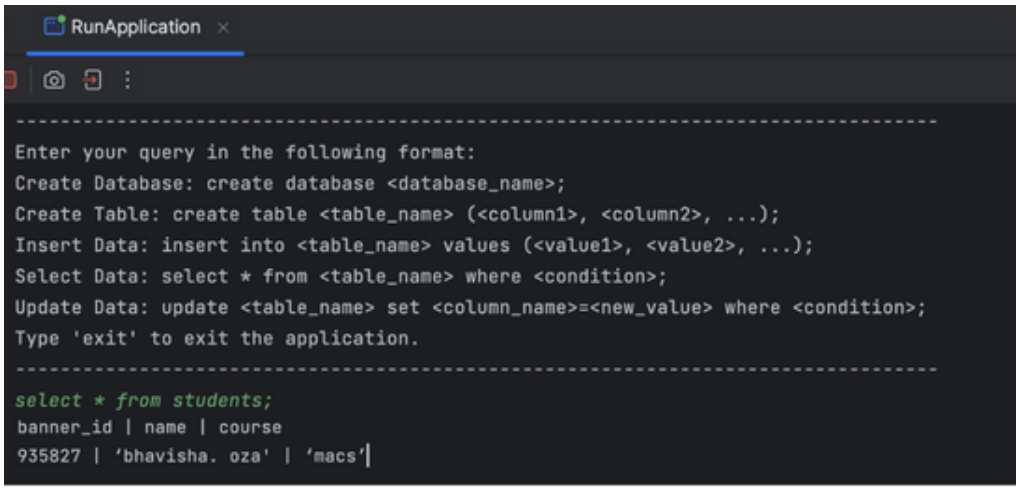

Figure 39: Select query for the deleted value banner_id = 923647.

7. Implementation of Transaction:

The implemented code introduces a single transaction handling logic that follows the ACID (Atomicity, Consistency, Isolation, Durability) properties. The system identifies a transaction based on user input, such as "Begin Transaction", "End Transaction", etc.

To ensure the ACID properties, the processed queries are not immediately written to the custom-made database text file. Instead, they are stored in an intermediate data structure, such as Lists. On the other hand, if the user inputs "Rollback", the data structure is emptied, and no changes are made to the text file.

This approach allows for the execution of multiple queries within a single transaction, ensuring that the changes are consistent and durable. By separating the execution from the actual updates to the database file, the code provides a mechanism to handle transactions and maintain data integrity.

Working flow of Transaction:

- The user is prompted to enter a query in a specific format. If the user enters a query that starts a transaction ("begin transaction"), the transaction handler's handleTransaction. method is called. The transaction handler displays a transaction menu and waits for the user to enter a query or command.
  The user can enter one of the following commands within the transaction: "begin transaction", "end transaction", "commit", "rollback", or a regular query.

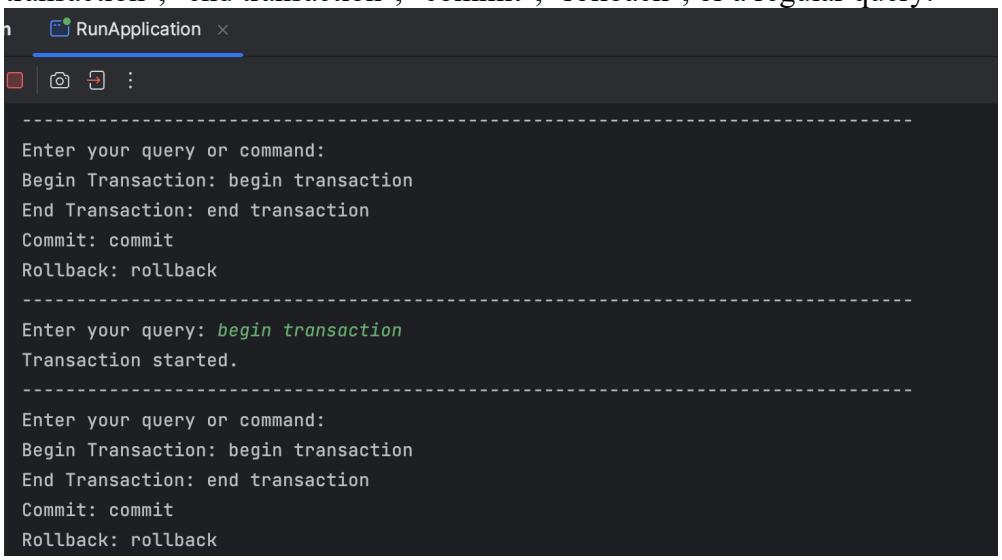

Figure 40: Begin Transaction query.

- If the user enters "begin transaction" when not in a transaction, an error message is displayed.

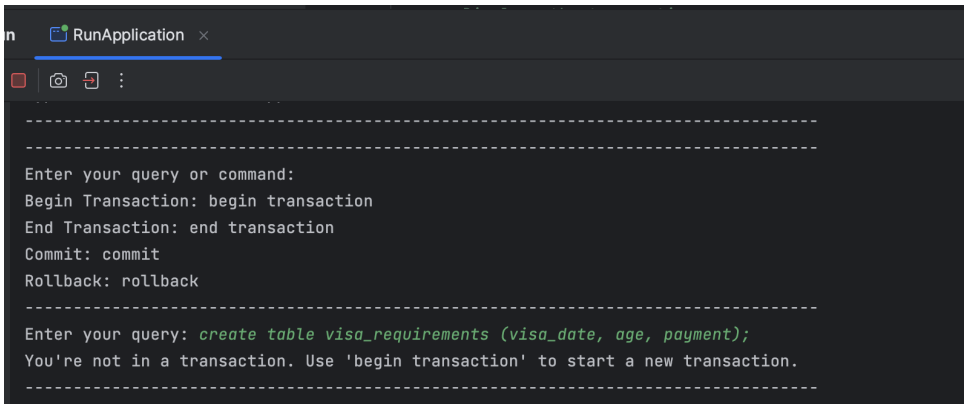

Figure 41: Error message for Begin Transaction when not in a transaction.

- If the user enters "begin transaction" while already in a transaction, an error message is displayed.

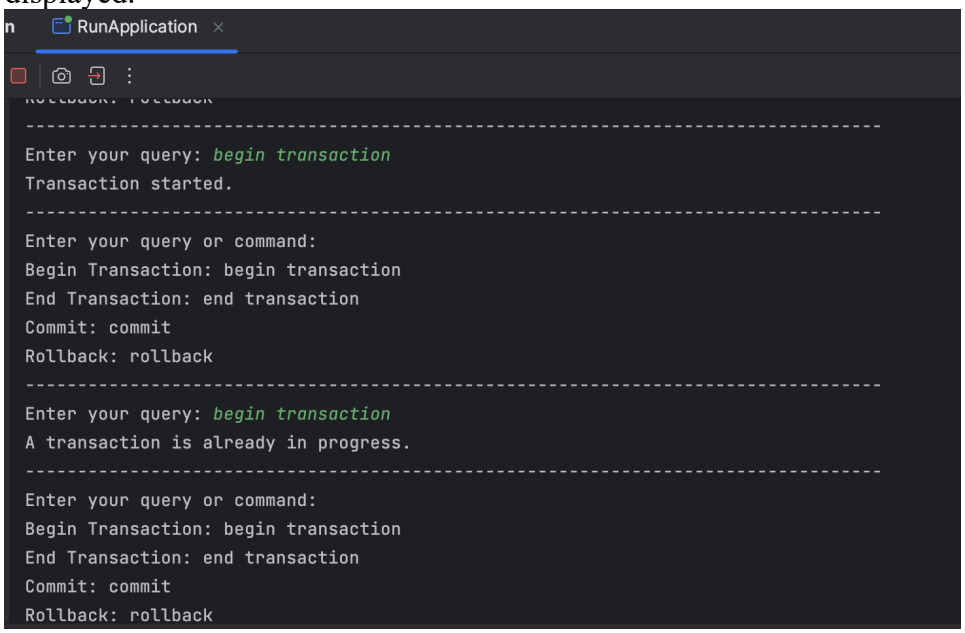

Figure 42: Error message for Begin Transaction while already in a transaction.

- Once the user enters “end transaction”, the transaction handler checks if any queries were added to the transaction. If queries were added, it calls the transaction handler's executeQueries method to process and execute the queries.
  The executeQueries method iterates over the queries and processes each query. In this example, it simply writes a response for each query to the data file.
  If the queries are executed successfully, the transaction is considered committed. If the queries list is empty, the variable exitTransaction is set to true, indicating that the transaction should be exited.

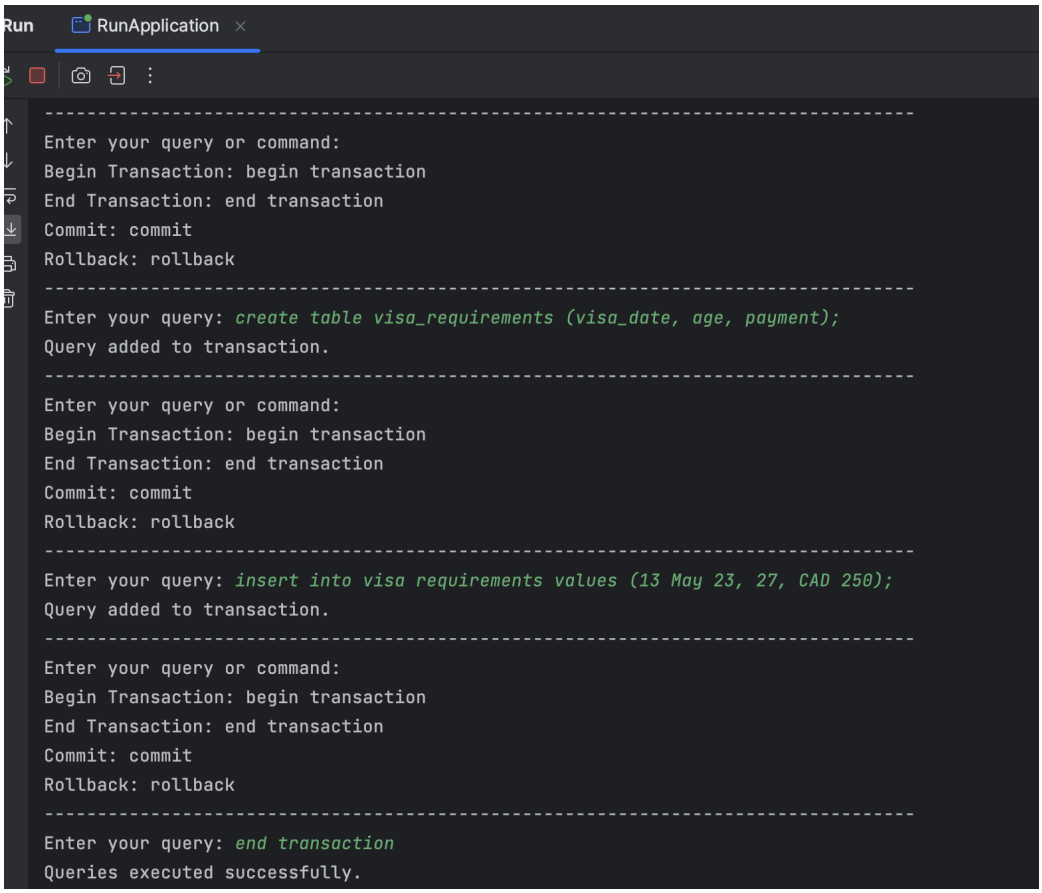

Figure 43: Transaction completion (Commit)

- If the user enters "rollback" while in a transaction, the transaction handler clears the queries list.

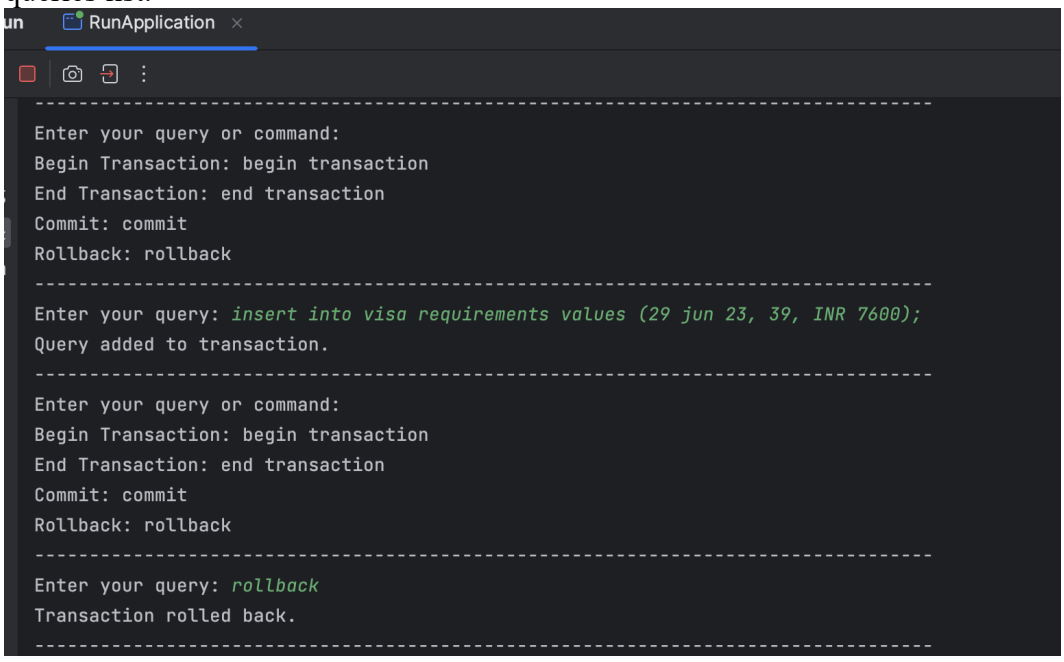

Figure 44: Transaction Rollback.

8. Implementation of ERD*:

The task is to implement an ERD (Entity-Relationship Diagram) generation class that can scan through the data and structure files of a database and generate a console-based representation of the ERD. Primary keys and foreign keys are denoted as "Pk" and "Fk", respectively. The ERD can be displayed as textual output on the screen using symbols like "|" and "-" to represent entity representations and relationships.

I have attempted to generate the ERD by having some custom designed text files. But it was not working correctly so have commented the code into the main method for the same. To accomplish this, the following steps has been taken:

- Created an ERDGenerator class that takes the paths of the data file and structure file as input.
- The generateERD() method is called to generate the Entity-Relationship Diagram (ERD) based on the data and structure files.
- Inside the generateERD() method:
  1. The method readTableColumns() is called to read the table columns from the structure file and store them in a map called tableColumns.
  2. The method readTableData() is called to read the table data from the data file and store it in a map called tableData.
  3. For each table, the entity representation is printed by retrieving the column names, primary keys, and data rows (if available) from the tableColumns and tableData maps. The entity representation is displayed in the console.
  4. The method readRelationships() is called to read the relationships between tables from the structure file and store them in a list called relationships.
  5. The entity representation and relationships are printed in the console. If there are relationships, they are displayed under the "Relationships:" section. If there are no relationships, the message "No relationships found." is displayed.
- The execution of the ERDGenerator class is completed, and the ERD is displayed in the console.

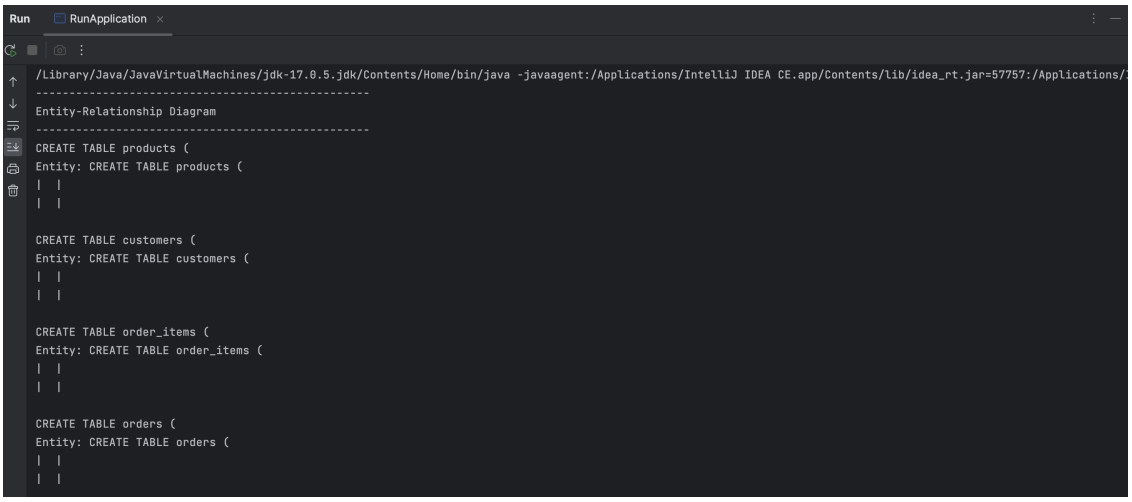

Figure 45: Attempt to create ERD.

Full code can be found in the given repository: https://git.cs.dal.ca/boza/csci-5408/-/tree/main/A2.

References:

[1] "Download intellij idea – the leading Java and Kotlin IDE," *JetBrains* [Online]. Available: https://www.jetbrains.com/idea/download/?section=mac [Accessed: 01 July 2023].

[2] "How to Write Doc Comments for the Javadoc Tool," *Oracle Canada* [Online]. Available: https://www.oracle.com/ca-en/technical-resources/articles/java/javadoc-tool.html#styleguide [Accessed: 06 July 2023].

[3] S. Desai, "All you Need to Know About Solid Principles in Java," *edureka* [Online]. Available: https://www.edureka.co/blog/solid-principles-in-java, 2021. [Accessed: July 08, 2023].

[4] "MD5 hash in java," *GeeksforGeeks* [Online]. Available: https://www.geeksforgeeks.org/md5-hash-in-java/ [Accessed: 05 July 2023].

[5] "Build, test, and debug regex," *regex101* [Online]. Available: https://regex101.com/ [Accessed: 06 July 2023].