Contents

PART 1

Summary

This paper gives a brief introduction about distributed systems. It introduces the concept by talking about how an organization will decide to store its databases when an organization is geographically dispersed. It may either store it on a central database server or decide to distribute them to local servers. This paper talks about distributing data which is spread physically across computers in multiple locations that are connected by a data communications network. It logically belongs to the same system but is spread over the sites of a computer network. The sites of the distributed database may be allocated in the same space and have the same network address but the communication between them is done over a network instead of sharing memory. These systems are mainly concerned with delivery of data to points of query. Then, it talks about how the database distribution is achieved with the help of data fragmentation and describes the process of distribution of data as a collection of fragmentation, replication and allocation.

Moreover, it talks about the fragmentation of data, its types, its importance and advantages as well as disadvantages.

The distributed database design process consists of three phrases – Initial Design, Redesign and materializing that redesign. Distributed database system – makes fragments of classes – in case of object-oriented database or relations – in case of relational database. This paper describes fragmentation as a technique to split a single class or relation of a database into two or more partitions, also; the combination of the partitions should support the original database without any loss of information. Fragmentation helps to improve reliability, performance, balanced storage capacity, communication costs, security.

Fragmentation is carried out on the basis of two parameters

1. Quantitative information: cardinality of relations, frequency of queries, site where query is run, selectivity of the queries, etc.
2. Qualitative information: predicates in queries, types of access of data, read/write, etc.

There are three types of fragmentations.

Horizontal

Horizontal fragmentation (HF) allows a relation or class to be partitioned into disjoint tuples which has unique rows but same columns and each fragment is stored at a different node. It stems from the intuition that every site should hold all information that is used to query at the site and the information at the site should be fragmented so the queries of the site run fast.

For example, Human Resources table is having 1000 records can be horizontally fragmented into ten fragments, each fragment having 100 unique records.

Vertical

A class or relation in VF will be partitioned into separate sets of columns or attributes except the primary key and these primary key attributes must be included in each set. It's useful when different sites are responsible for processing different functions involving an entity.

For example, in the Human Resources table; the attributes as EMP_ID (PRIMARY KEY), F_NAME, JOB, SALARY, and DEPT_ID could be Vertically fragmented into two sites such as site1 and site2.

Mixed

Mixed fragmentation refers to a combination of horizontal and vertical strategies. A table may be divided into several rows, each one having a subset of the columns. Mixed fragmentation needs two-steps procedures. First of all, horizontal fragmentation is introduced for each site. The horizontal fragmentation yields the subsets of Human Resources horizontal fragments that are located at each site. Vertical fragmentation is used within each horizontal fragment to divide the attributes.

Then, it talks about the correctness rules that should hold true for fragmentation. It talks about three types of correctness rules:

1. Completeness – It ensures no data is lost during the fragmentation process by the correctness property that every data item in the original relation R can be found in at least one of the fragments.

2. Reconstruction – It’s a process of combining the fragments into its original relation. Union is used to combine the fragments created in horizontal fragmentation. Join is used to combine the fragments used in vertical fragmentation. It will need expensive techniques in case of recursive fragmentations.
3. Disjointness - In horizontal fragmentation, where data items are tuples, each tuple should exist in only one fragment to maintain disjointness. In vertical fragmentation, where data items are attributes, each attribute should be present in only one fragment to ensure disjointness.

REFERENCES

[1] L. M. Tuitions, “Vertical fragmentation in Hindi | Distributed Database Tutorials,” 06-May-2018. [Online].Available:https://www.youtube.com/watch?v=0VXb8UN7_Xg&list=PL0s3O6GgLL5ftzY3smhxUmzbzIE7JIraU&index=6. [Accessed: 12-Jul-2023].

[2] L. M. Tuitions, “Fragmentation and correctness rules in Hindi | Distributed Database Tutorials,” 06-May-2018. [Online]. Available: https://www.youtube.com/watch?v=3SCHeqVQDUY&list=PL0s3O6GgLL5ftzY3smhxUmzbzIE7JIraU&index=4. [Accessed: 12-Jul-2023].

PART 2

GitHub Repository Link

https://git.cs.dal.ca/athaker/csci5408_s23_b00937694_abhisha_thaker/-/tree/main/A2

SOLID DESIGN

Query.java – which will hold the single responsibility principle – as it will methods related to all the queries only. It can have following methods

1. CreateDB(String query)
2. CreateTable(String query)
3. InsertTable(String query)
4. UpdateTable(String query)
5. DeleteRow(String query)

All the dependencies of query.java will be passed as dependency instead of initializing the object in the constructor – so it will also hold the dependency inversion.

The above file will depend on objects of regex, queryfile and maybe a helper.

Single responsibility - RegexPatternMatcher.java – which will separate the strings by matching the pattern passed in the pattern.compile() method. It also holds the single responsibility principle by just doing the operations related to regex.

Single responsibility - queryFile.java – This class handles all file operations related to the queries.

Single responsibility - User.java – which will be a getter and setter method for user object. It will also hold the single responsibility principle.

userAuthentication.java - It will add user, save user information to a file, and validates credentials and security answers.

Main.java – It’s a simple user authentication with the ability to perform switch-case based database like operations in file.

File and Folder Structure

For storing and retrieving user Information, have created a file

userInfo.txt: stores all the information related to user. It stores the Id, HashedPassword, SecurityQuestion, securityAnswer

It stores the Id, password, securityQuestion, securityAnswer per line in the order mentioned. Each field is separated by commas per line.

All database related information is stored in the folderPath//databaseName. Here, databaseName implies the name of the database in the query – CREATE DATABASE databaseName. Within this folder, files can be created with the tableName.

databaseName – Name of the folder

tableName – Name of the empty file within the databaseName folder.

alltableinfo.txt – Name of the txt file which will contain the information about the column values related to all the tables. It stores in the following format

e.g

Table_Name: users

ColumnName1: id INT

ColumnName2: name VARCHAR(50)

Table_Name2: student

ColumnName1: studentid INT

ColumnName2: studentname VARCHAR(50)

tableName.txt

When insert operation is performed --

(id,name)

(1,’Abhisha’)

(2,’Krishna’)

Main.java

It’s a simple user authentication with the ability to perform switch-case based database like operations in file.

All the userInformation is stored in the file **userInfo.txt.**

Methods

getUserFromConsole()

It prompts the user for an Id.

If the Id already exists in the file userInfo.txt, then it retrieves the values from the file and authenticates basis that the validateCredentials method. If the credentials are valid, then it asks for securityanswer basis the already retrieved securityQuestion basis the Id.

If the authentication is successful, the program prompts the user to enter a database query. The user is presented with a list of query options to choose from, including creating a database, creating a table, inserting values, selecting values, updating rows, and deleting rows. To create a database, enter the query and enter the corresponding number of the operation that you want to perform. If you want to perform

1. Create a database– then press 1.
2. Create a table - then press 2
3. Insert values – then press 3
4. Select values – then press 4.
5. Update rows – then press 5.
6. Delete rows – then press 6.

The selected option is passed to the Query class, where the corresponding query operation is performed based on user input.

If the Id doesn't exist in the file userInfo.txt, then it will ask for password, securityquestion, securityanswer. The information is used to create a new User object, which is added to the UserAuthentication instance and saved in the file using the saveUsersToFile method defined in the userAuthentication class.

isIdExists method

Checks whether the file exists or not.

LoadUsersFromFile

If the Id already exists, then it retrieves the userinformation from the file basis the Id and stores the details in the addUser() method of the user

hashPassword

It takes password as an input and uses the MD5 algorithm to hash it and returns the hashed password as a string when called upon.

User.java

This class is used with userAuthentication class and main class methods. It's like a model.

This class helps to create user objects with specific parameters.

1. This class has private instance variables – ‘Id’, ‘Password’,’securityQuesiton’,’securityAnswer’.
2. The getter Methods (‘getId()’,’getPassword()’,’getSecurityQuestion()’ and ‘getSecurityAnswer()’ returns the current value of the above written instance variables.
3. The setter Methods (‘getId()’,’getPassword()’,’getSecurityQuestion()’ and ‘getSecurityAnswer()’ sets the new value passed as parameters for the above written instance variables.

UserAuthentication.java

It adds users, saves user information to a file, and validates credentials and security answers.

Methods

Map users – key: Id, Value: User Object

addUser() method

Takes a the user object as a parameter and adds it to the users map. The user's ID is used as the key, and the user object itself is the value.

saveUsersToFile() method

saves the userinformation stored in the users map to userInfo.txt, which is specified by USER_FILE_PATH constant. Then, it write the userInformation per line. Every line has userId, password, securityQuestion, security Answer separated by commas.

validateCredentials

checks if the Id and password in the temp stored users map match the Id and password stored in the userInfo.txt File. Returns true – if match found.

validateSecurityAnswer

checks if security answer stored in the temp stored users map match the security Answer stored in the userInfo.txt File. Returns true – if match found.

getSecurityQuestion

fetches the securityquestion associated with the Id

Query.java

It’s constructor creates the instances of QueryFile, File, RegexPatternMatcher and Helper objects

public String createDB(String query). Query should be in the form CREATE DATABASE dbName;

Methods	**How ?**
	First, the query is trimmed to remove any additional whitespace.

public String createDB(String query) It extracts the database name from the query, sets the necessary variables, creates the folder, and returns the database name if the folder creation is successful.	Then, the tokenizer splits the query into individual tokens basis the white space. And it skips – create database And jumps to the next token and stores the value of this token to databaseName. Then, it replaces semicolon withwhitespace. Then, it sets the folderPath. If the folderPath exists, then create a directory with the name of the database in the folderPath.
Public void createTables(String query, String dbName)	First, the query is trimmed to remove any additional whitespace. Then, the tokenizer splits the query into individual tokens basis the white space. And it skips – create table And jumps to the next token and stores the value of this token to tableName. Then it creates an empty file inside the databaseName folder. Then it extracts the text between the opening and the closing braces and tokenizes it using the , delimiter. Then, it writes the columnvalues and tableName to alltableinfo.txt
Public void insertValues(String query, String dbName)	First, the query is trimmed to remove any additional whitespace. Then, the tokenizer splits the query into individual tokens basis the white space. And it skips – INSERT INTO And jumps to the next token and stores the value of this token to tableName. Then, 2 methods from regexPatternMatcher class is called to match the patterns and store the attributeNames and valueNames. Then, 2 methods from queryFile method is called to store the attributeName in the file named as tableName. And to append the values in the file named as tableName

Public void selectValues(String query, String dbName) **Public void updateValues(String query, String dbName)**	e.g. SELECT lastName from tableName where id = 1; tableName = is extracted using the method extractSelectTableNames in the regexPatternMatcher class ie. tableName columnName: is extracted using the method extractSelectColumnName in the regexPatternMatcher class . ie. lastName condition is extracted using the method extractSelectCondition in the regexPatternMatcher class. Ie. Id = 1 It's again matched using the regex. targetRowName ie. id operator ie. = targetValue ie. 1 tableName -- extracted table name from the query using the extractUpdateTableName method of the regexP object. columnNameValue variable - the extracted concatenated column name and update value from the query using the extractUpdateColumnNameValue method of the regexP object. The columnNameValue is split into two parts using a single quote (') as the delimiter. The first part is assigned to columnName, and the second part, wrapped in single quotes, is assigned to updateToValue. The targetRow variable - the extracted condition from the query using the extractUpdateCondition method of the regexP object. The target row name, operator, and target value are extracted from the targetRow using the regular expression pattern and stored in the respective variables.
Public void deleteValues(String query, String dbName)	The tableName variable - extracted table name from the query using the extractDeleteTableInfo method of the regexP object. The targetRow variable - the extracted condition from the query using the extractDeleteCondition method of the regexP object.

	The target row name, operator, and target value are extracted from the targetRow using the regular expression pattern and stored in the respective variables. linesFinal list is created to store the remaining lines of the file after deletion.

Queryfile.java

This class handles all file operations related to the queries.

createFileTableName() method

creates a new file with specified tablename inside the getFolderpath directory.

writeTableNamesToAllTableInfo() method

Appends the tablename to the 'alltableinfo.txt' file

writeAttributeNamesToAllTableInfo() method

appends the columnname with its datatypes to the alltableinfo.txt file.

saveToFileInsert1

appends just the columnames of the insert file to the file the name as the tableName. Internally, it checks if the columname values already exists in the file using checkLineexists method.

saveToFileInsert2

appends the values of the insert file to the file with name as the tableName

checkLineExists method

checks if the columnname value already exists in the file with the name as the tableName

Overall, this class provides methods to create files, write table and column names to specific files, and save values from insert queries to files.

RegexPatternMatcher.java

1. `**extractColumnInsertQuery(String query)**`: Extracts all the column names from an `INSERT` query. It uses a regular expression pattern to match the column names within the query and returns them as a string.

2. **`extractValuesInsertQuery(String query)`:** Extracts all the values section from an `INSERT` query. It uses a regular expression pattern to match the values section within the query and returns it as a string.

3. **extractSelectCondition(String query):** This method extracts the condition section after the `WHERE` keyword in a `SELECT` query.. If multiple conditions are present (separated by `AND`), it only returns the first condition.

4**. extractSelectColumnName(String query):** This matches the pattern between the `SELECT` and `FROM` keywords and returns the column names as a string. If multiple column names are present (separated by commas), it only returns the first column name.

5. extractSelectTableName(String query): This method extracts the table name from a `SELECT` query. It uses a regular expression pattern to match the table name between the `FROM` and `WHERE` keywords and returnds it as a string.

6. extractUpdateCondition(String query): This method extracts the condition section after the `WHERE` keyword in an `UPDATE` query. It uses a regular expression pattern to match the condition section and returns it as a string.

7. extractUpdateColumnNameValue(String query): It splits the values after 'SET' into individual column name and value. And returns the pair as a concatenated string

8. extractUpdateTableName(String query): Extracts the tableName. Uses regex expression pattern to match the tableName between UPDATE and SET and returns the matched value as the string.

9. extractDeleteCondition(String query): Extracts and returns the condition section after the `WHERE` keyword in a delete query.

10. extractDeleteTableInfo(String query): Extracts the tableName. Uses regex expression pattern to match the tableName between 'FROM' and 'WHERE' and returns the matched value as the string.

These methods provide a way to extract specific information from SQL queries using regular expressions, allowing for further processing and manipulation of the query components.

HELPER.JAVA

1. getColumnIndex(String columnName, String[] attributeName): This method takes a `columnName` and an array of `attributeName` as parameters. It compares the `columnName` with each element in the `attributeName` array (ignoring case) and returns the index of the matching column name. If no match is found, it returns -1.

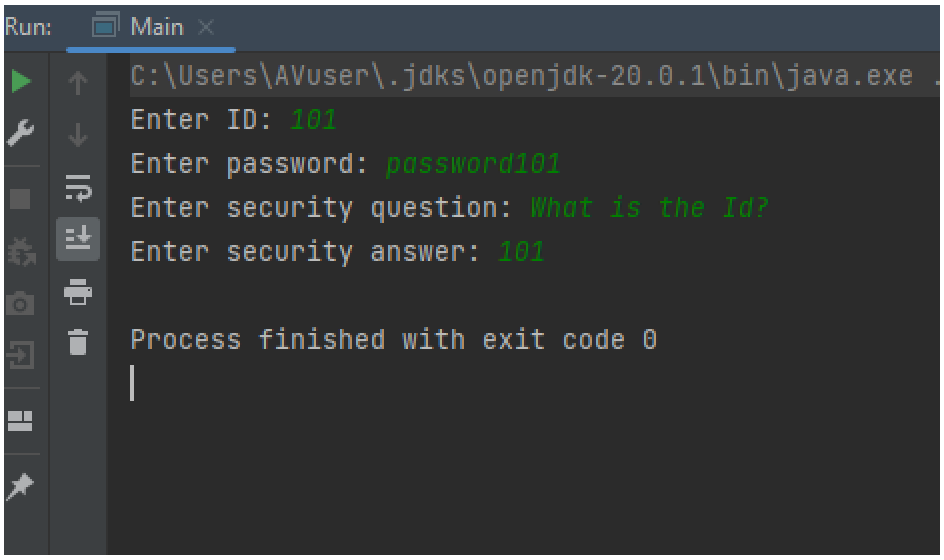
2. getTrimmedValue(String value): This method takes a `value` as a parameter and removes leading/trailing whitespace and single quotes from the value. It returns the trimmed value as a string.

The getColumnIndex method helps in finding the index of a column name from an array of attribute names, which can be useful for accessing specific columns in a table.

The getTrimmedValue method assists in removing unwanted characters from a value, such as whitespace or single quotes.

OUTPUT DATA

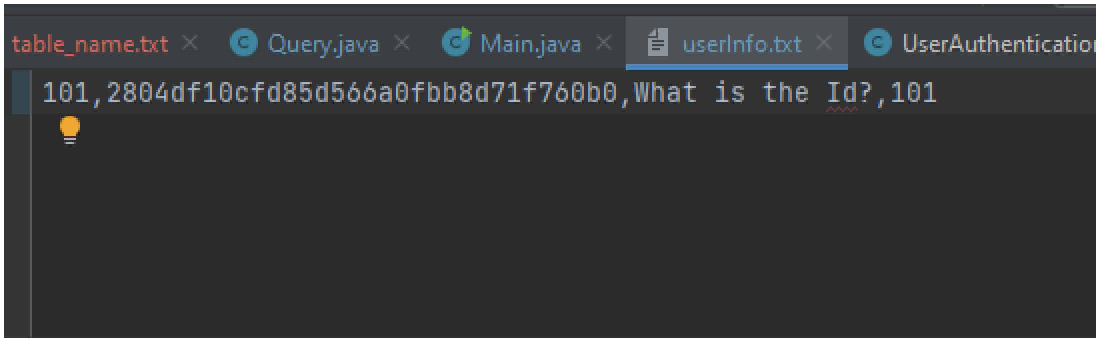
Output



```
C:\Users\AVuser\.jdks\openjdk-20.0.1\bin\java.exe
Enter ID: 101
Enter password: password101
Enter security question: What is the Id?
Enter security answer: 101

Process finished with exit code 0
```

saveUsersToFile image1

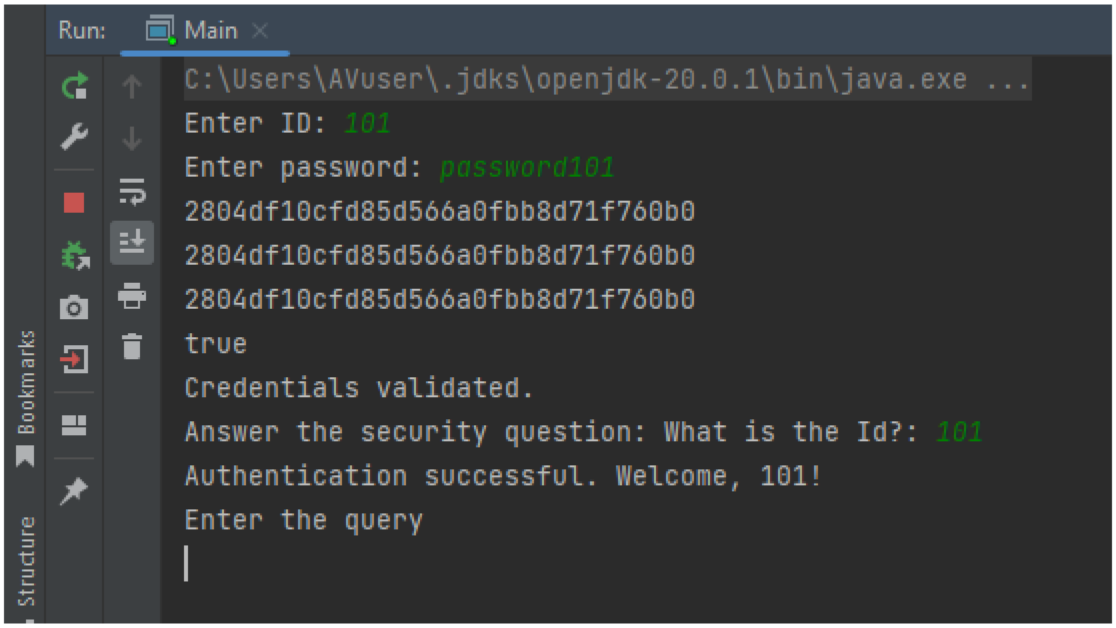
Here, Id – 101 doesn’t exist in the userInfo.txt file. Hence, it asks for the password, securityqeustion and security answer and it will store it in the file as shown below



```
101,2804df10cfd85d566a0fbb8d71f760b0,What is the Id?,101
```

saveUsersToFile image2

Here, Id – 101 is stored. And hashedPassword is stored, along with security question and securityAnswer



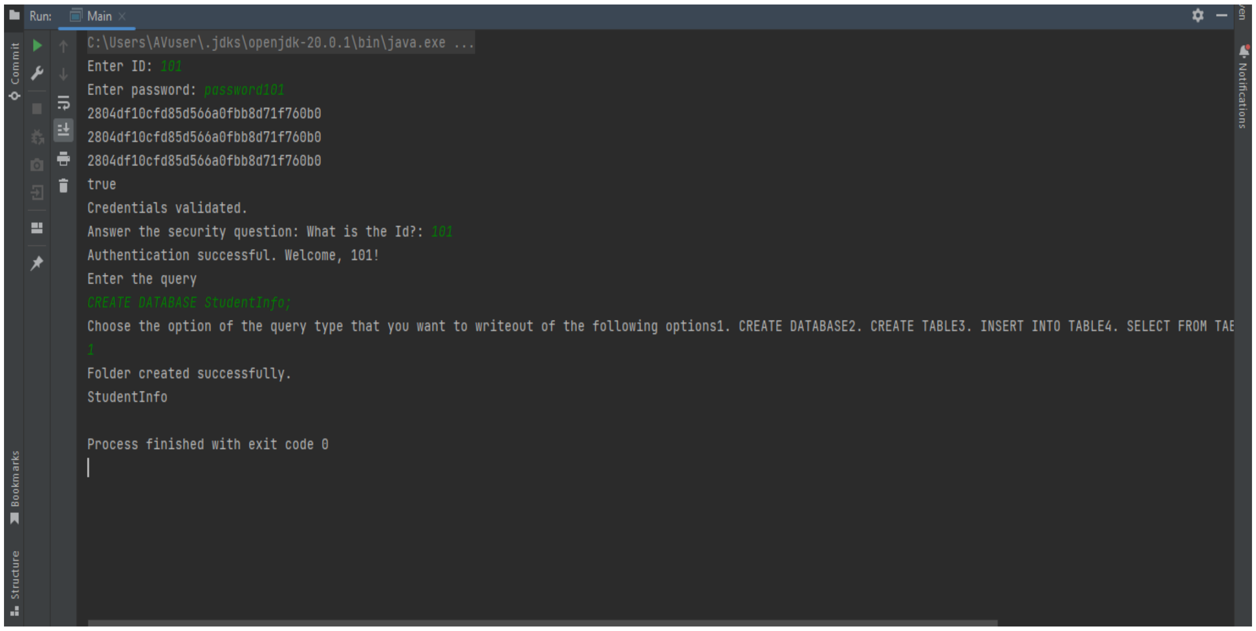
```
C:\Users\AVuser\.jdks\openjdk-20.0.1\bin\java.exe ...
Enter ID: 101
Enter password: password101
2804df10cfd85d566a0fbb8d71f760b0
2804df10cfd85d566a0fbb8d71f760b0
2804df10cfd85d566a0fbb8d71f760b0
true
Credentials validated.
Answer the security question: What is the Id?: 101
Authentication successful. Welcome, 101!
Enter the query
```

loadusersFromFile Image3

Here, Id – 101 and password are provided by the user. If it's matched against the file, then it asks for securityQuestion and prompts the user to enter securityAnswer.

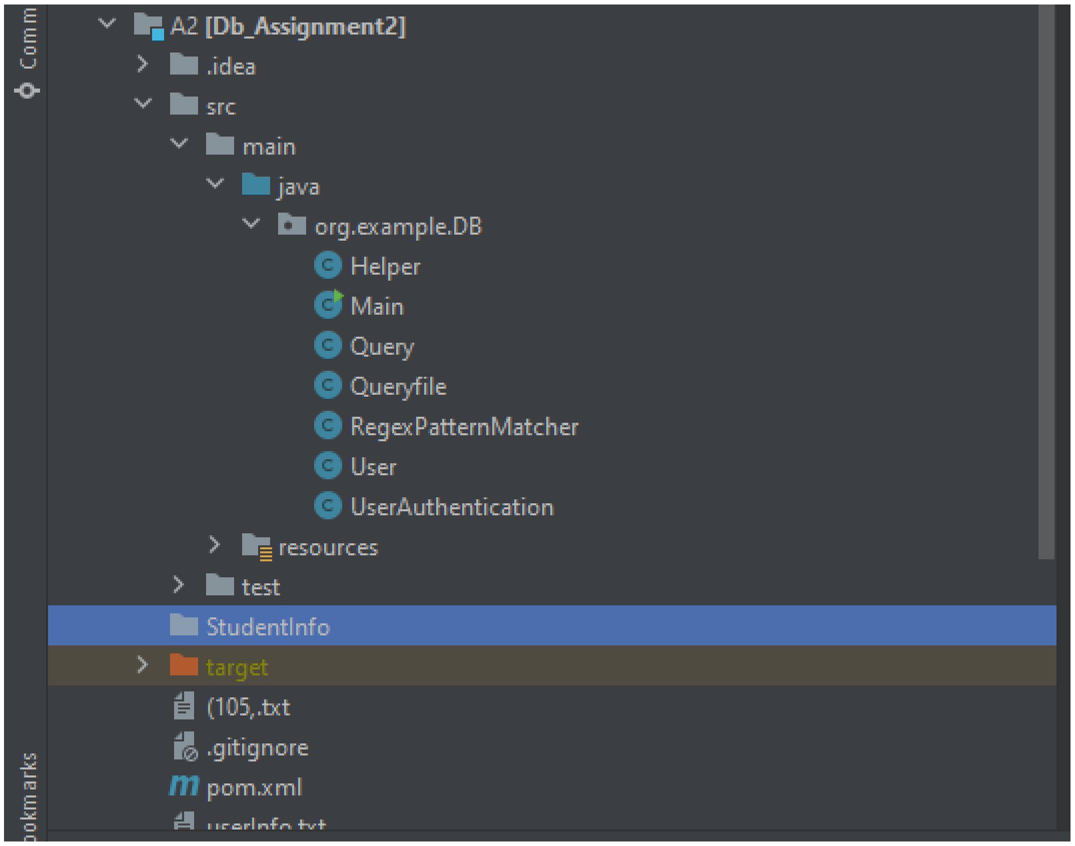
Enter the query

1 CREATE DATABASE StudentInfo;



```
C:\Users\AVuser\.jdks\openjdk-20.0.1\bin\java.exe ...
Enter ID: 101
Enter password: password101
2804df10cfd85d566a0fbb8d71f760b0
2804df10cfd85d566a0fbb8d71f760b0
2804df10cfd85d566a0fbb8d71f760b0
true
Credentials validated.
Answer the security question: What is the Id?: 101
Authentication successful. Welcome, 101!
Enter the query
CREATE DATABASE StudentInfo;
Choose the option of the query type that you want to writeout of the following options1. CREATE DATABASE2. CREATE TABLE3. INSERT INTO TABLE4. SELECT FROM TAE
1
Folder created successfully.
StudentInfo

Process finished with exit code 0
```

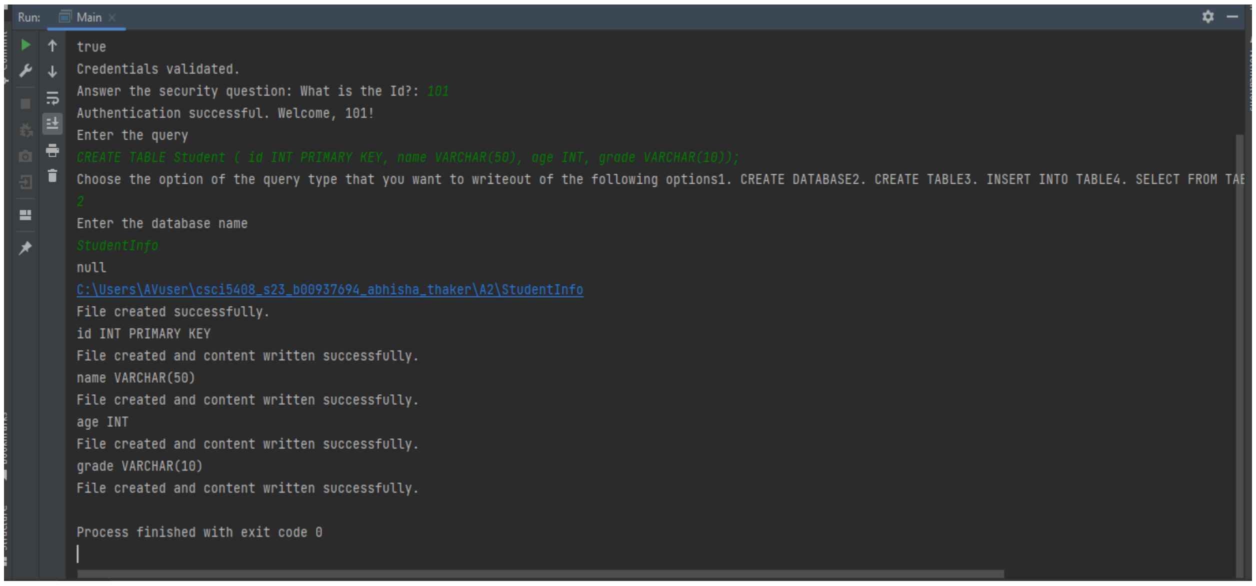
Databasecreation image4



```
A2 [Db_Assignment2]
  .idea
  src
    main
      java
        org.example.DB
          Helper
          Main
          Query
          Queryfile
          RegexPatternMatcher
          User
          UserAuthentication
      resources
    test
  StudentInfo
  target
  (105,.txt
  .gitignore
  pom.xml
  userInfo.txt
```

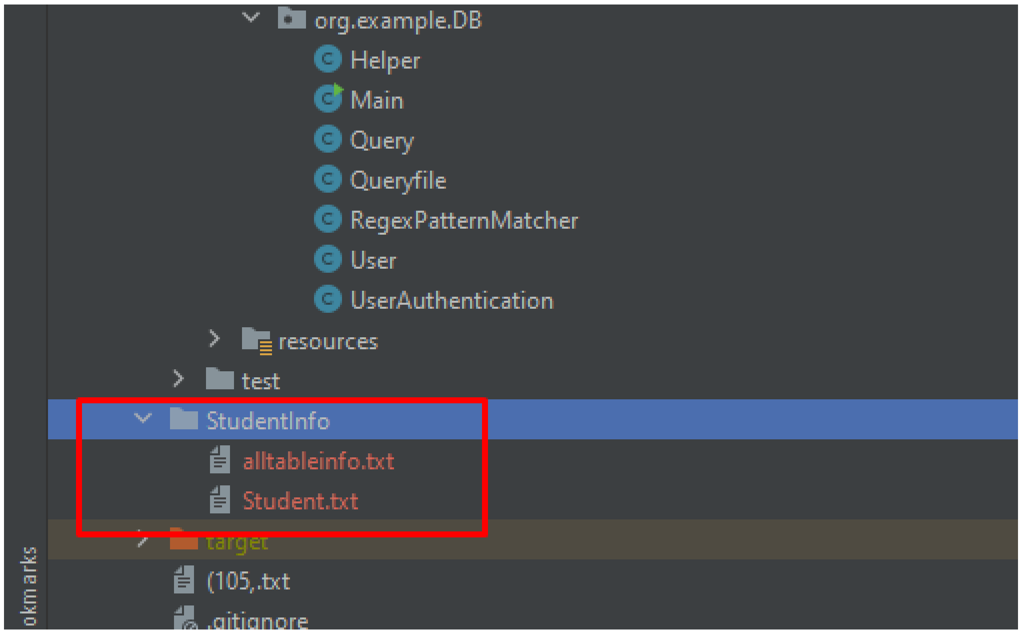
StudentInfo file

2 CREATE TABLE Student (id INT PRIMARY KEY, name VARCHAR(50), age INT, grade VARCHAR(10));

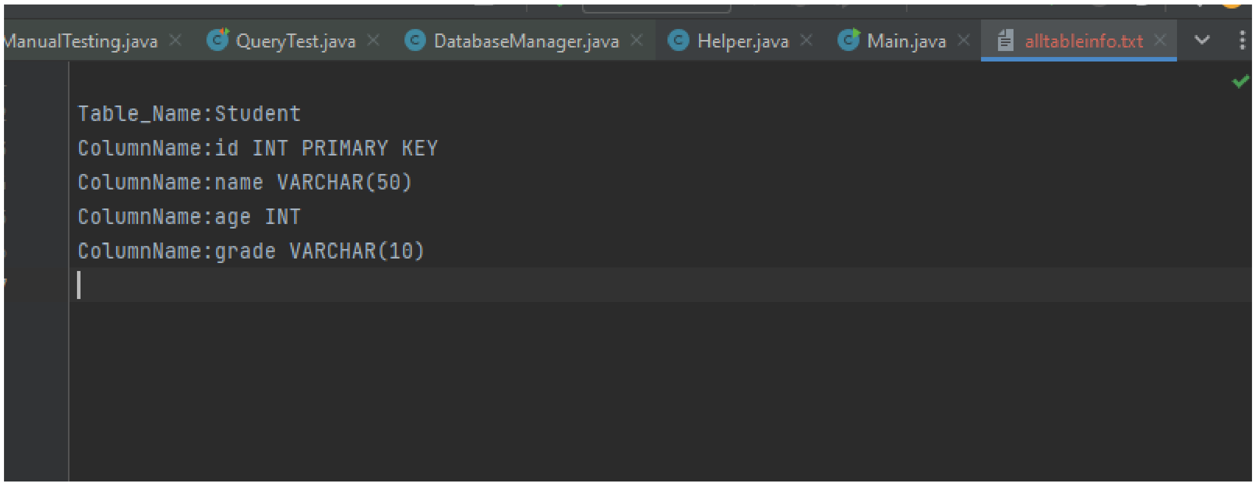


```
true
Credentials validated.
Answer the security question: What is the Id?: 101
Authentication successful. Welcome, 101!
Enter the query
CREATE TABLE Student ( id INT PRIMARY KEY, name VARCHAR(50), age INT, grade VARCHAR(10));
Choose the option of the query type that you want to writeout of the following options1. CREATE DATABASE2. CREATE TABLE3. INSERT INTO TABLE4. SELECT FROM TAE
2
Enter the database name
StudentInfo
null
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2\StudentInfo
File created successfully.
id INT PRIMARY KEY
File created and content written successfully.
name VARCHAR(50)
File created and content written successfully.
age INT
File created and content written successfully.
grade VARCHAR(10)
File created and content written successfully.

Process finished with exit code 0
```

createTable query screenshot

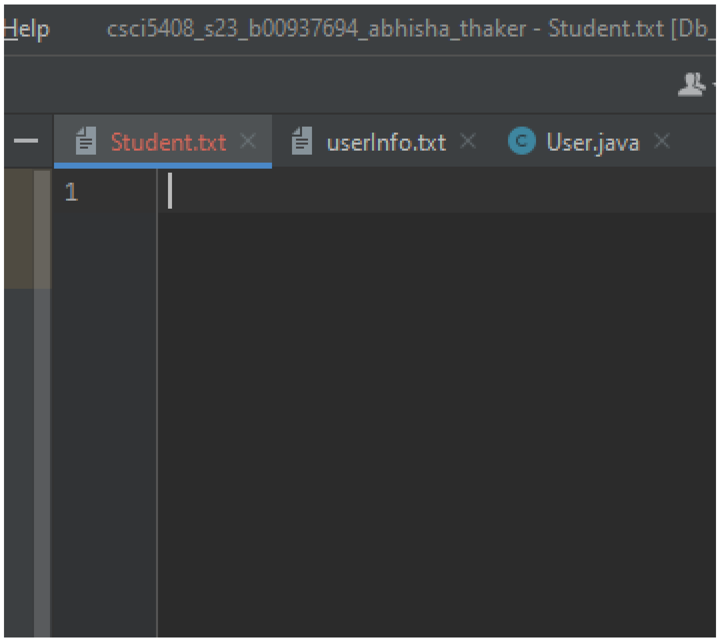


```
org.example.DB
    Helper
    Main
    Query
    Queryfile
    RegexPatternMatcher
    User
    UserAuthentication
  resources
 test
StudentInfo
    alltableinfo.txt
    Student.txt
target
(105,.txt
.gitignore
```



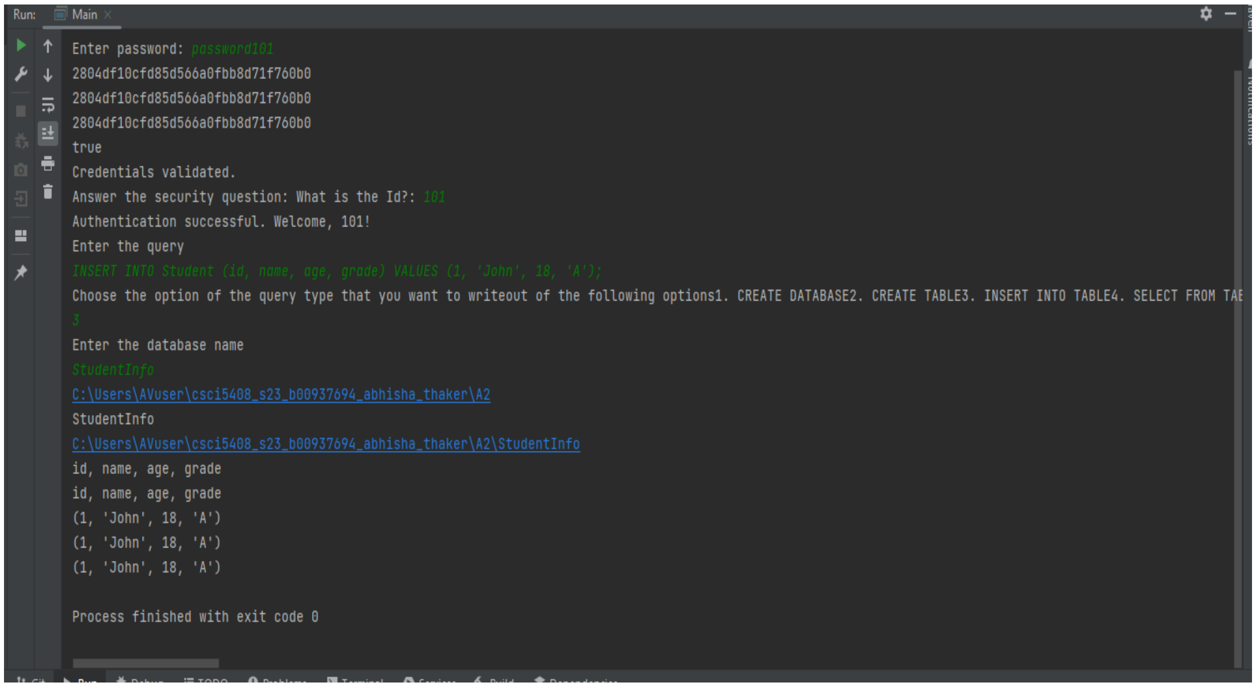
```
Table_Name:Student
ColumnName:id INT PRIMARY KEY
ColumnName:name VARCHAR(50)
ColumnName:age INT
ColumnName:grade VARCHAR(10)
```

alltableInfo stores



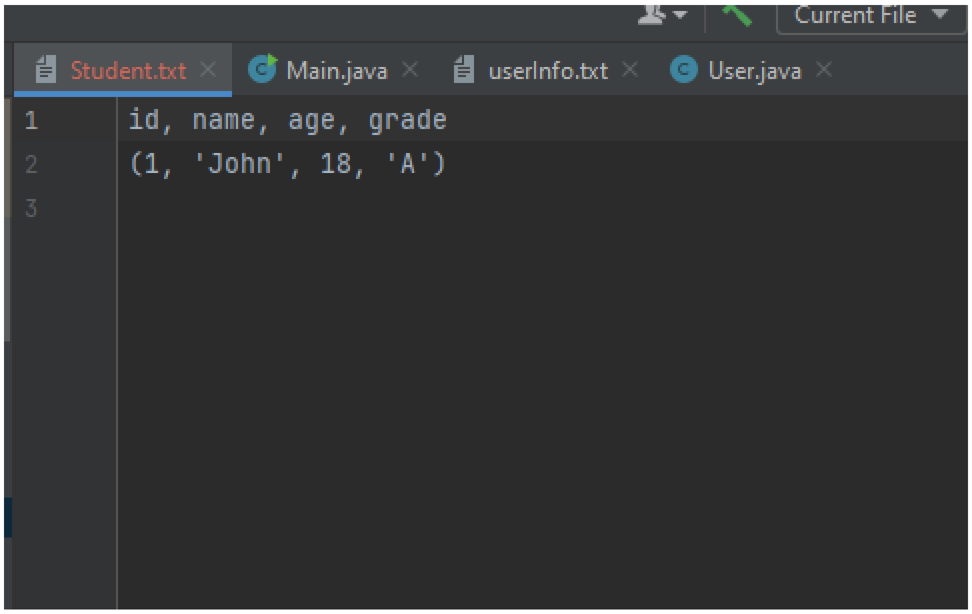
emptyStudent.txt file

3 INSERT INTO Student (id, name, age, grade) VALUES (1, 'John Doe', 18, 'A');



```
Enter password: password101
2804df10cfd85d566a0fbb8d71f760b0
2804df10cfd85d566a0fbb8d71f760b0
2804df10cfd85d566a0fbb8d71f760b0
true
Credentials validated.
Answer the security question: What is the Id?: 101
Authentication successful. Welcome, 101!
Enter the query
INSERT INTO Student (id, name, age, grade) VALUES (1, 'John', 18, 'A');
Choose the option of the query type that you want to writeout of the following options1. CREATE DATABASE2. CREATE TABLE3. INSERT INTO TABLE4. SELECT FROM TAE
3
Enter the database name
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2\StudentInfo
id, name, age, grade
id, name, age, grade
(1, 'John', 18, 'A')
(1, 'John', 18, 'A')
(1, 'John', 18, 'A')

Process finished with exit code 0
```

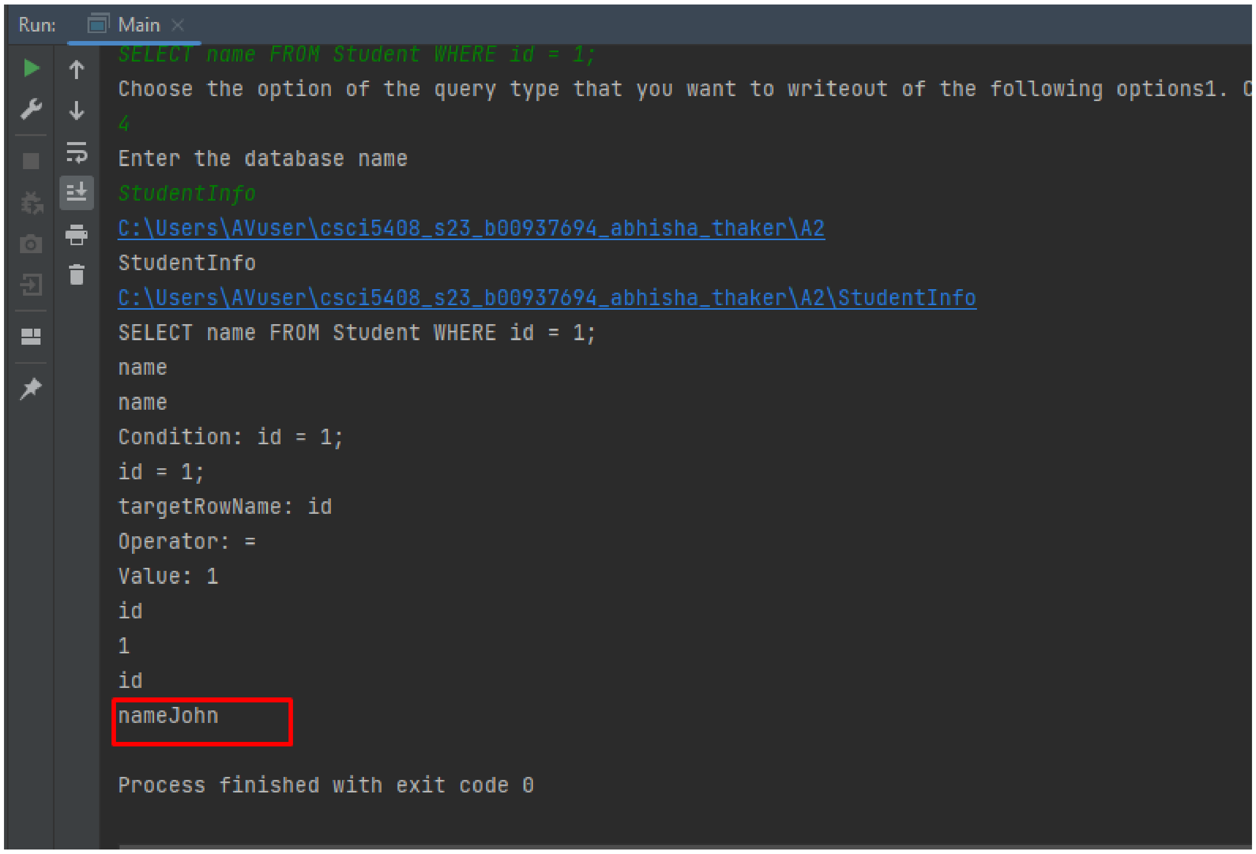
insertQuery image



```
id, name, age, grade
(1, 'John', 18, 'A')
```

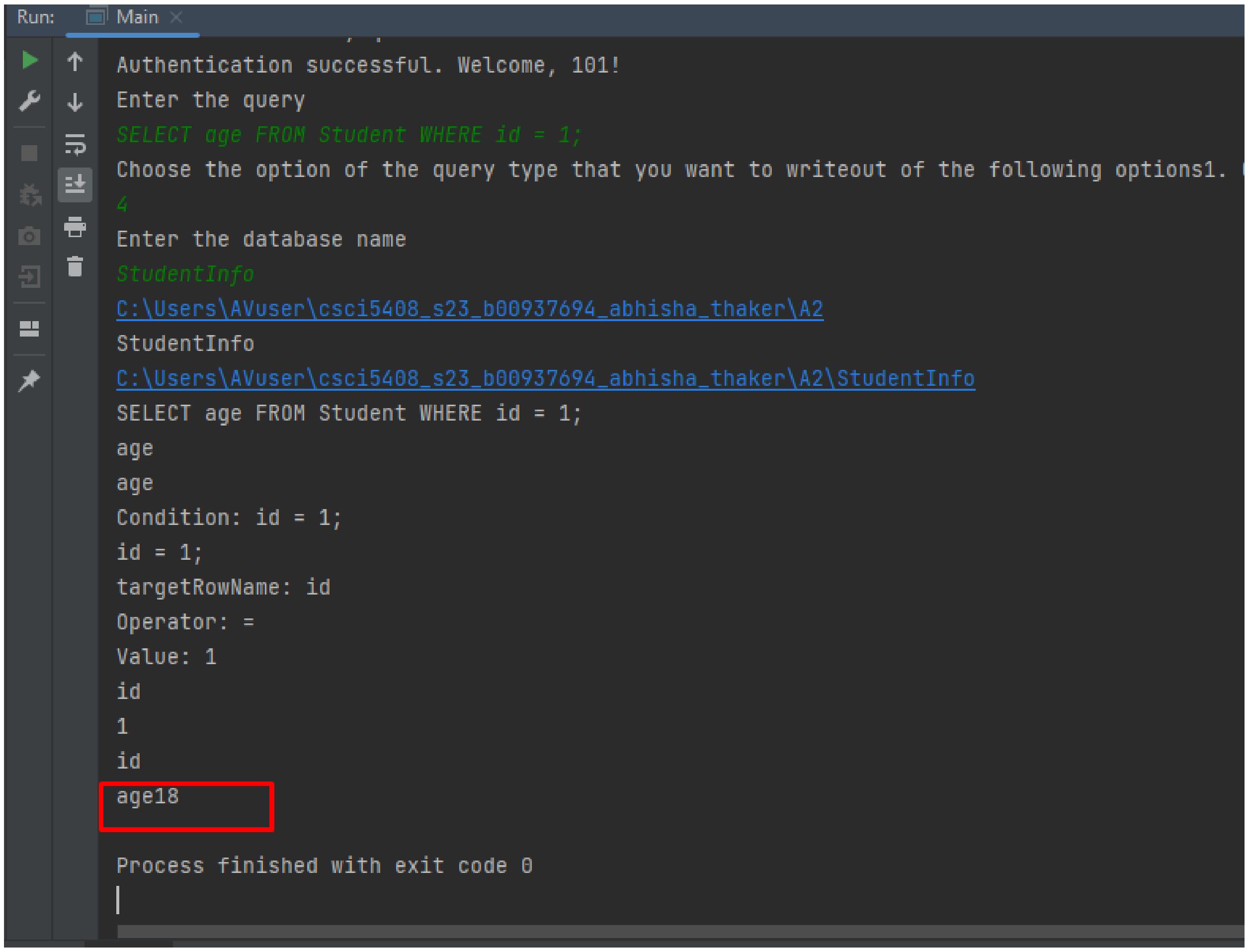
Student.txt image – written to the Student.txt

4.1 SELECT name FROM Student WHERE id = 1;



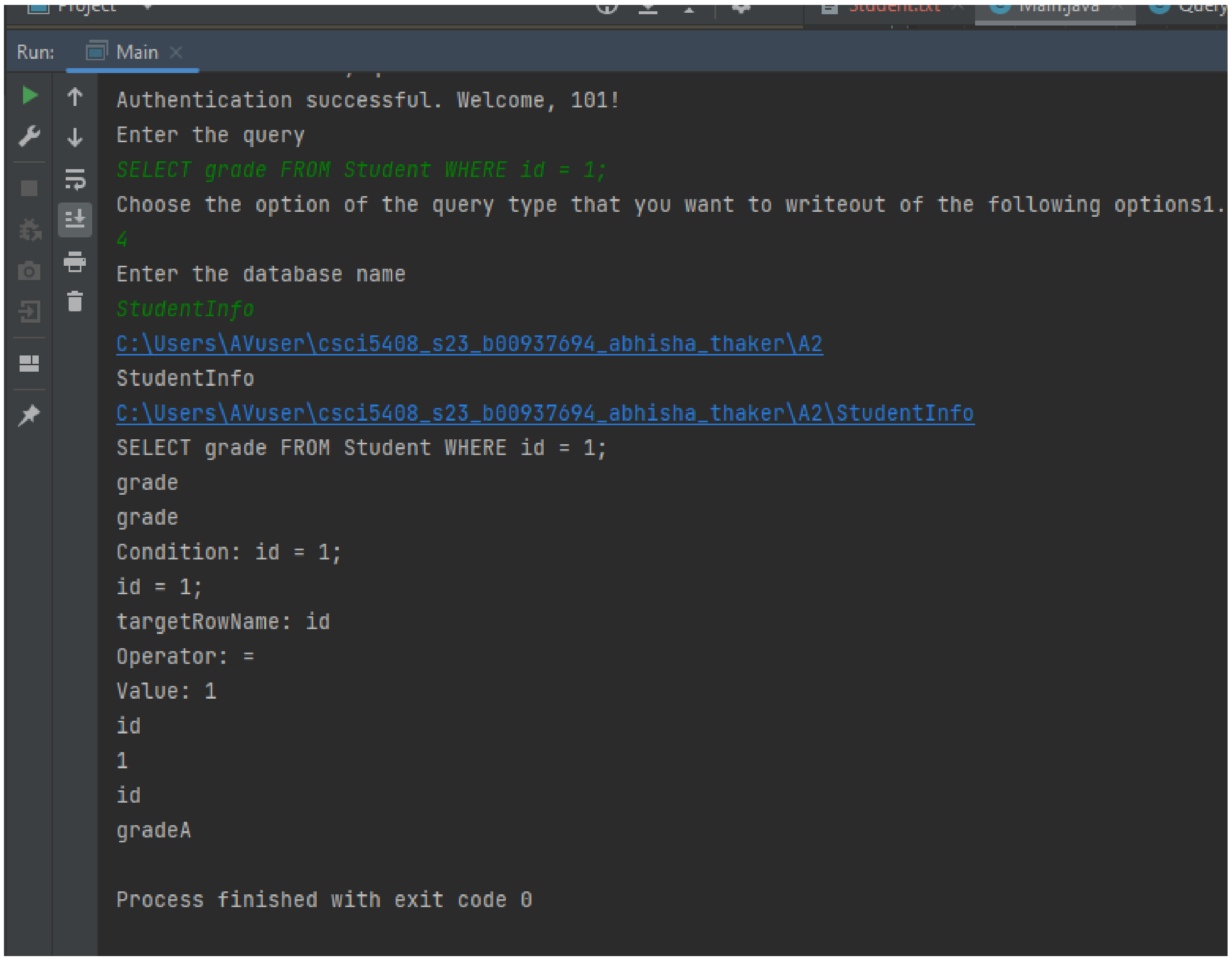
```
SELECT name FROM Student WHERE id = 1;
Choose the option of the query type that you want to writeout of the following options1. C
4
Enter the database name
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2\StudentInfo
SELECT name FROM Student WHERE id = 1;
name
name
Condition: id = 1;
id = 1;
targetRowName: id
Operator: =
Value: 1
id
1
id
nameJohn

Process finished with exit code 0
```

4.2 SELECT name FROM Student WHERE id = 1;



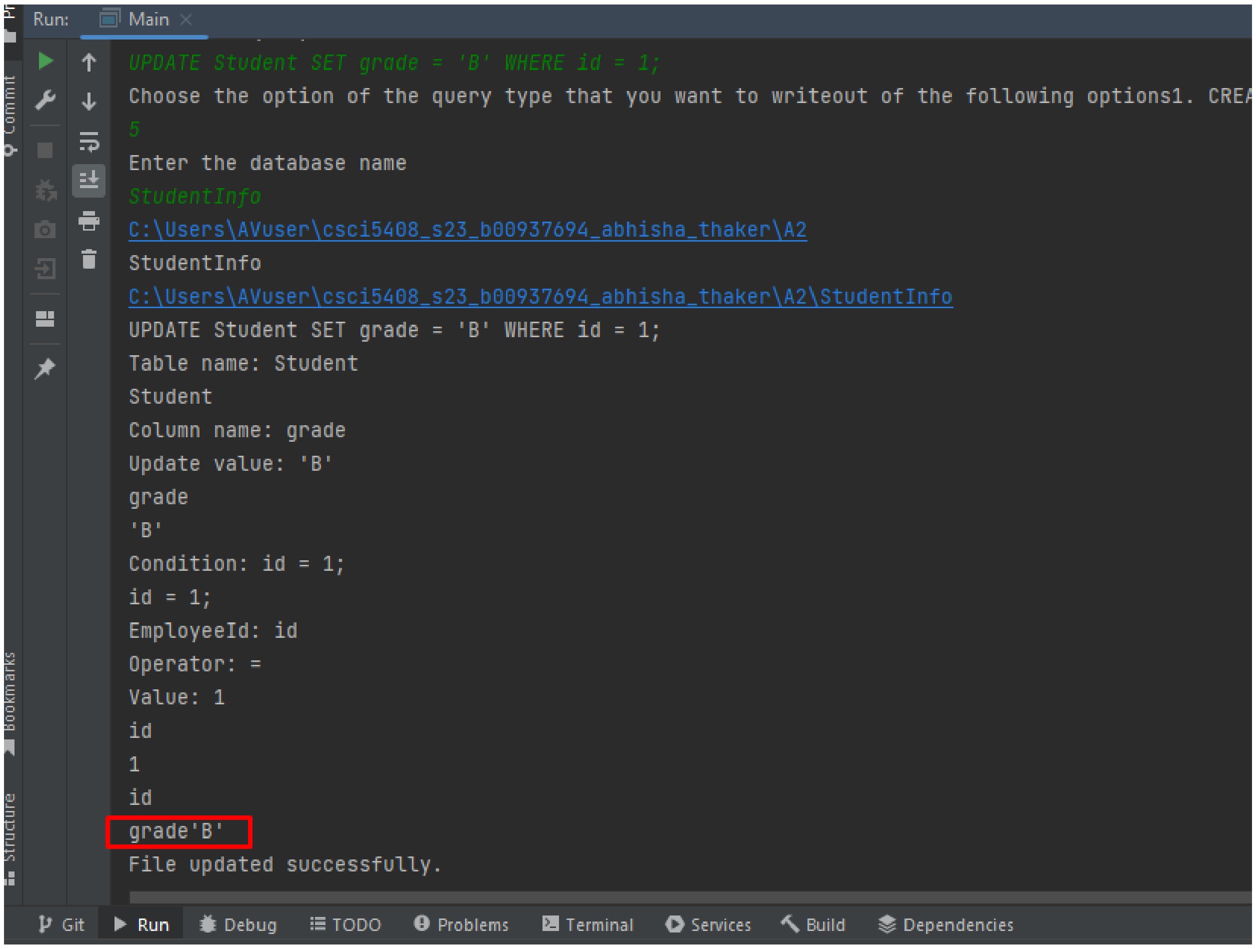
```
Authentication successful. Welcome, 101!
Enter the query
SELECT age FROM Student WHERE id = 1;
Choose the option of the query type that you want to writeout of the following options1.
4
Enter the database name
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2\StudentInfo
SELECT age FROM Student WHERE id = 1;
age
age
Condition: id = 1;
id = 1;
targetRowName: id
Operator: =
Value: 1
id
1
id
age18

Process finished with exit code 0
```

4.3 SELECT grade FROM Student WHERE id = 1;



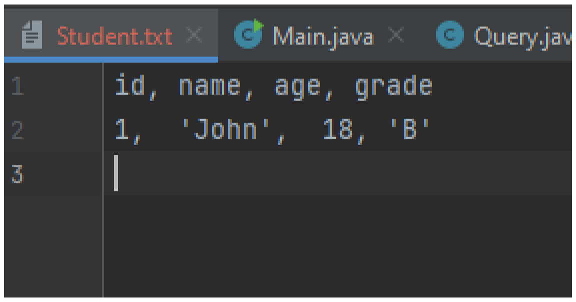
```
Authentication successful. Welcome, 101!
Enter the query
SELECT grade FROM Student WHERE id = 1;
Choose the option of the query type that you want to writeout of the following options1.
4
Enter the database name
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2\StudentInfo
SELECT grade FROM Student WHERE id = 1;
grade
grade
Condition: id = 1;
id = 1;
targetRowName: id
Operator: =
Value: 1
id
1
id
gradeA

Process finished with exit code 0
```

5 UPDATE Student SET grade = 'A' WHERE id = 1;


Run: Main
Git Run Debug TODO Problems Terminal Services Build Dependencies

```
UPDATE Student SET grade = 'B' WHERE id = 1;
Choose the option of the query type that you want to writeout of the following options1. CREA
5
Enter the database name
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2\StudentInfo
UPDATE Student SET grade = 'B' WHERE id = 1;
Table name: Student
Student
Column name: grade
Update value: 'B'
grade
'B'
Condition: id = 1;
id = 1;
EmployeeId: id
Operator: =
Value: 1
id
1
id
grade'B'
File updated successfully.
```

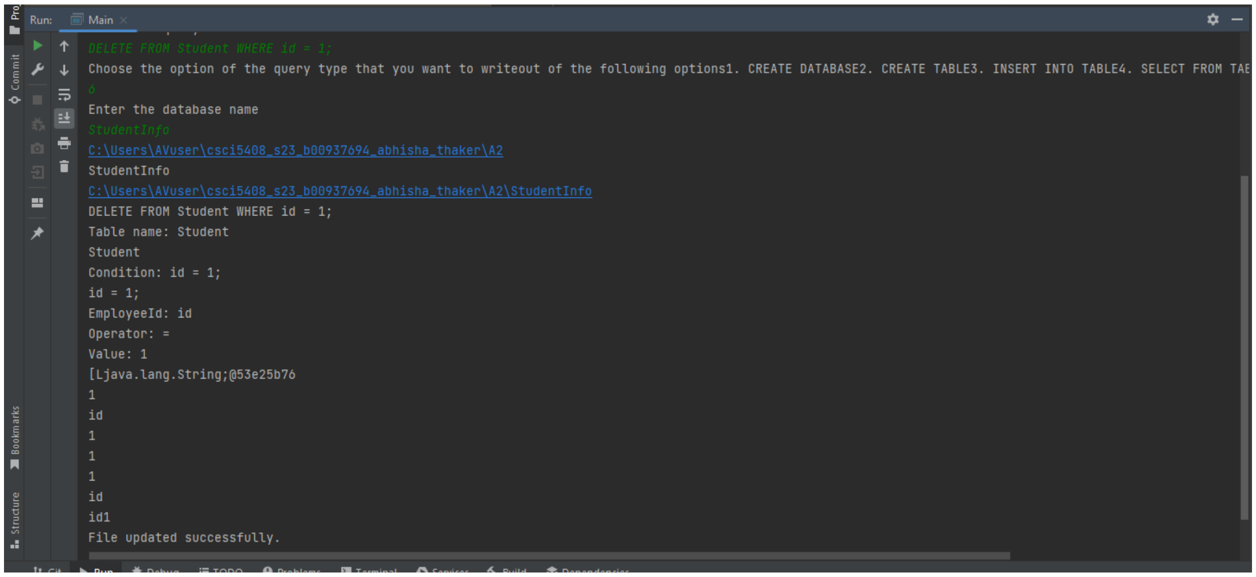
updateQuery image



```
id, name, age, grade
1,  'John',  18, 'B'
```

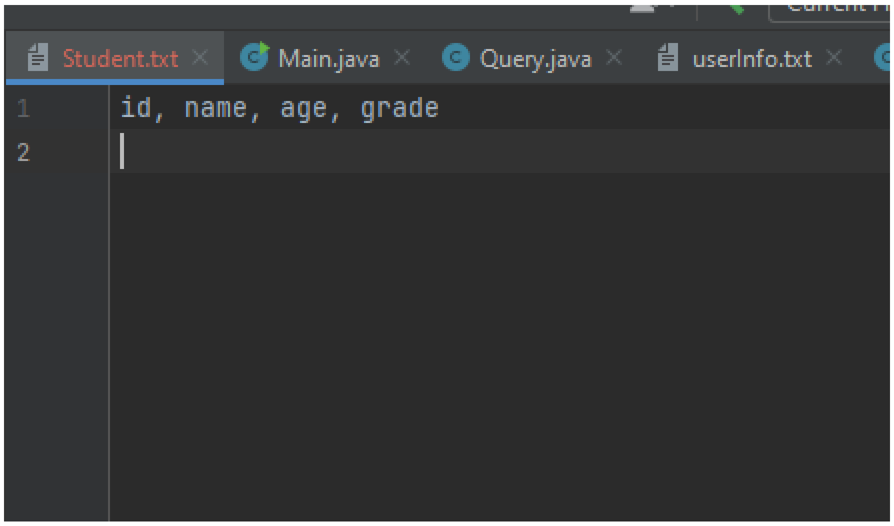
Student.txt – grade 'B' is updated.

6 DELETE FROM Student WHERE id = 4;



```
DELETE FROM Student WHERE id = 1;
Choose the option of the query type that you want to writeout of the following options1. CREATE DATABASE2. CREATE TABLE3. INSERT INTO TABLE4. SELECT FROM TAE
6
Enter the database name
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2
StudentInfo
C:\Users\AVuser\csci5408_s23_b00937694_abhisha_thaker\A2\StudentInfo
DELETE FROM Student WHERE id = 1;
Table name: Student
Student
Condition: id = 1;
id = 1;
EmployeeId: id
Operator: =
Value: 1
[Ljava.lang.String;@53e25b76
1
id
1
1
1
id
id1
File updated successfully.
```

deleteQuery File



```
id, name, age, grade

```

deletedRow in Student.txt

ASSUMPTIONS

1. Id in the user is assumed to be Integer
2. Database should be first created before executing any other query. It’s recommended to execute every query in the order from 1 to 6.

LIMITATIONS

1. For select operation, select * from tableName won't work for this version of the code. Please specify columnName, tableName and corresponding value - select columnName from tableName where columnName = value. In the condition only Id will work for now.
2. For creating database and creating table, don't use anything other than – CREATE DATABASE databaseName, CREATE TABLE tableName. Here, databaseName and tableName can be of your choice.
3. Insert query won't work for multiple values insertion. At a time, you will be able to insert just one set of values.
4. Don't pass values separated by space. For e.g. 'John Doe' as a value in the Insert query. Though, it will insert just fine, but would create trouble while retrieving the values.

REFERENCES

[1] F. Dib, "Regex101: Build, test, and debug regex," *regex101*. [Online]. Available: https://regex101.com/. [Accessed: 12-Jul-2023].

[2] "StringTokenizer (java platform SE 8)," Oracle.com, 05-Apr-2023. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/util/StringTokenizer.html. [Accessed: 12-Jul-2023].

[3] "StringTokenizer in java," *W3schools*, 28-Aug-2014. [Online]. Available: https://www.w3schools.blog/stringtokenizer-in-java. [Accessed: 12-Jul-2023].

[4] "Methods of the pattern class," *Oracle.com*. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/regex/pattern.html. [Accessed: 12-Jul-2023].

[5] "Boundary matchers," Oracle.com. [Online]. Available: https://docs.oracle.com/javase/tutorial/essential/regex/bounds.html. [Accessed: 12-Jul-2023].

[6] W. D. Simplified, "Learn Regular Expressions In 20 Minutes," 29-Oct-2019. [Online]. Available: https://www.youtube.com/watch?v=rhzKDrUiJVk. [Accessed: 12-Jul-2023].

[7] C. Schafer, "Regular expressions (regex) tutorial: How to match any pattern of text," 05-Oct-2017. [Online]. Available: https://www.youtube.com/watch?v=sa-TUpSx1JA. [Accessed: 12-Jul-2023].

[8] "Java FileWriter with append mode," *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/1225146/java-filewriter-with-append-mode. [Accessed: 12-Jul-2023].

[9] “Java: How to get keys and values from a map,” *Stack Abuse*, 24-Nov-2020. [Online]. Available: https://stackabuse.com/java-how-to-get-keys-and-values-from-a-map/. [Accessed: 12-Jul-2023].

[10] “How can I generate an MD5 hash in Java?,” *Stack Overflow*. [Online]. Available: https://stackoverflow.com/questions/415953/how-can-i-generate-an-md5-hash-in-java. [Accessed: 12-Jul-2023].

[11] “Java BufferedWriter (with examples),” *Programiz.com*. [Online]. Available: https://www.programiz.com/java-programming/bufferedwriter. [Accessed: 13-Jul-2023].

[12] “BufferedWriter (java platform SE 8),” *Oracle.com*, 05-Apr-2023. [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/io/BufferedWriter.html. [Accessed: 13-Jul-2023].