CSCI 5408
Data Management and Warehousing

Group-04

Project: Tiny DB

SPRINT 1 REPORT

Group Members:

Ramya Kommalapati (B00982851)

Parth Madhvani (B00981268)

Shifa Mirza(B00981843)

GitLab:

https://git.cs.dal.ca/mirza/csci_5408_s24_4

Table of Contents:

Table of Figures:

Background research:

In developing a database management system using Java file handling, we integrated several key technologies and methodologies from core Java. Our exploration and investigation primarily focused on optimizing the use of core Java functionalities. Prominent resources such as Stack Overflow, W3Schools, and JavaPoint were our go-to platforms for learning and effectively utilizing these features.

The system employed regex for robust query parsing, ensuring efficient handling and updating of table operations stored in text files. Each record was separated using the "###" delimiter for clear separation within the files.

Central to our implementation were data structures such as ArrayLists and LinkedLists, chosen to optimize data storage and retrieval. These structures facilitated efficient management of data, supporting functionalities ranging from data insertion to complex searching operations. By leveraging Java's buffered read and write mechanisms, we ensured streamlined performance in handling data operations, enhancing both reliability and speed within our system.

ArrayList

- **Purpose:** To handle query and data processing in-memory efficiently for our TinyDB project, allowing quick access, addition, and deletion of records.
- **Justification:**
    - Efficient Data Retrieval: ArrayLists allow for quick access to specific records using their index. This is essential for queries that require scanning through a large dataset, such as finding a customer within many orders.
    - Dynamic Resizing: As our TinyDB grows, handling increasing data volumes is crucial. ArrayLists provide dynamic resizing, expanding or contracting to accommodate the changing data size.
    - Code Integration: Since we are working in Java, ArrayLists integrate well with our existing codebase. Built-in methods for adding, accessing, and removing elements simplify development.

LinkedList

- **Purpose:** To support operations where frequent insertions and deletions are required, ensuring efficient data modification for our TinyDB project.
- **Justification:**
    - Efficient Modifications: LinkedLists excel when frequent insertions and deletions are needed. If we need to constantly add or remove records from the middle of a list, a LinkedList becomes advantageous. Its insertion and deletion operations are efficient for such scenarios.
    - Specialized Use Cases: While ArrayLists are useful, LinkedLists become valuable in specific situations. Managing a list of active transactions or

temporary data buffers is a good example where a LinkedList's strengths are beneficial.

Custom File Format

Custom Format: Delimiter-Separated Text File (.txt)

- **Purpose:** We opted for a custom file format to ensure data integrity and easy retrieval. It’s a delimiter-separated text file, using "###" as a unique separator to distinguish each field within a record.
- **Justification:**
  - Clear Delineation of Fields: Our custom format uses the "###" delimiter to clearly distinguish between data fields. This simplifies parsing the data and maintains its integrity. Parsing becomes straightforward, and data remains consistent and reliable.
  - Simplicity: Our custom format prioritizes ease of use, both for reading and writing data programmatically. It fulfills TinyDB’s persistence needs without introducing unnecessary complexity that might come with standard formats like JSON, XML, CSV, or serialized/binary formats.

Architecture diagram:

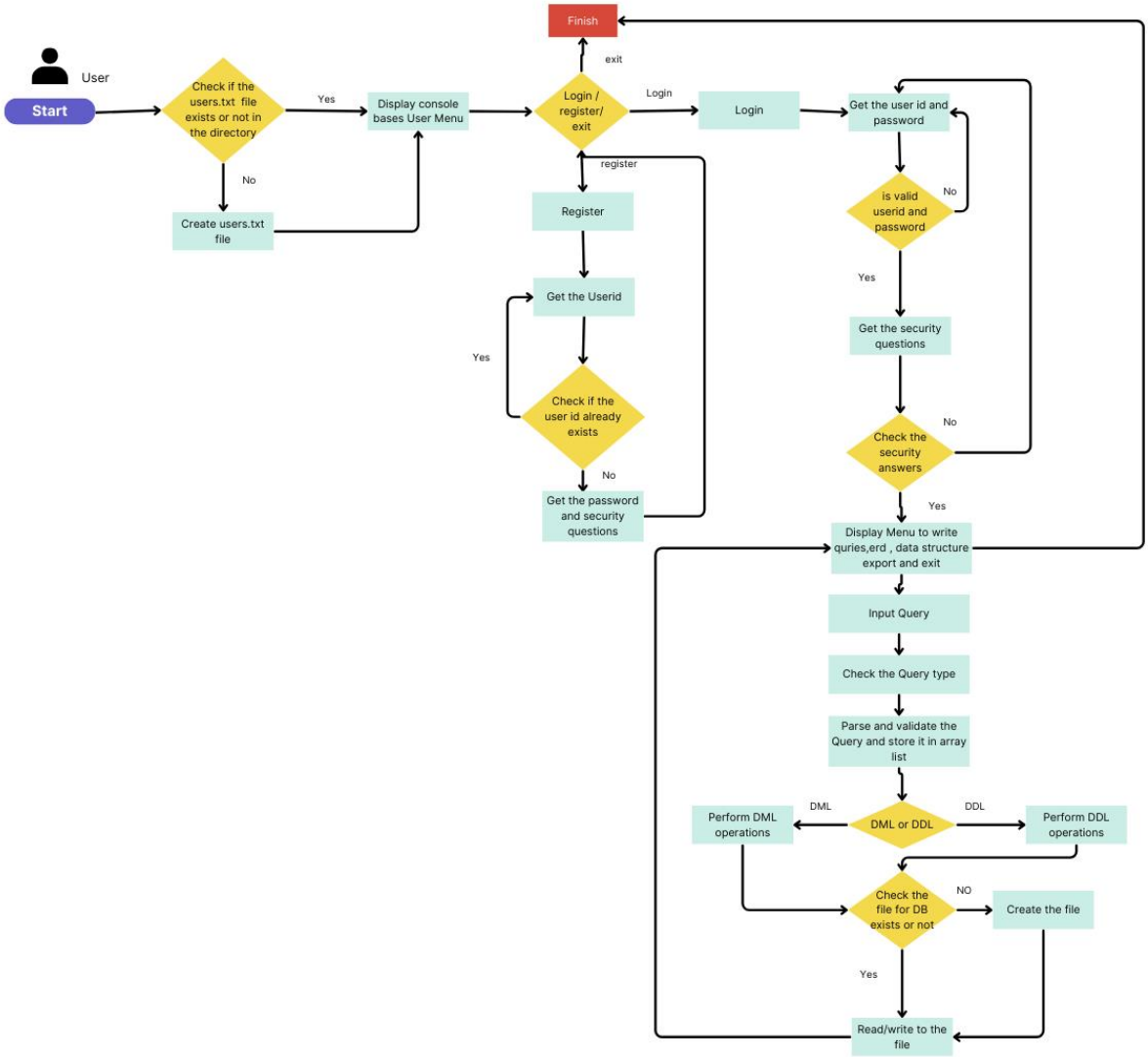

Figure 1 Work Flow Diagram TinyDBGroup04

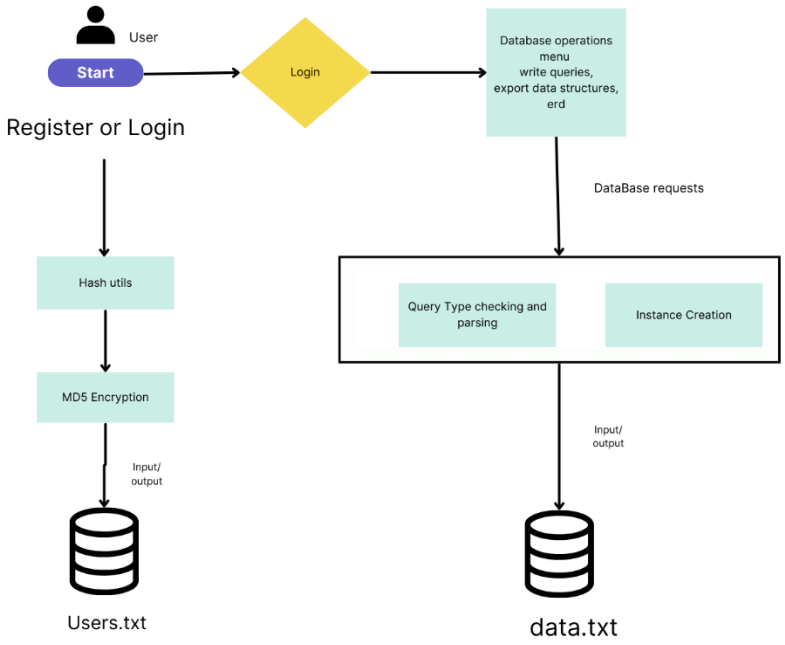

Figure 2: Diagram TinyDBGroup04

Pseudocode:

```
class TinyDBGroup04:
    CONSTANT scanner = new Scanner(System.in)
    CONSTANT DATA_FILE = "src/main/resources/users.txt"
    CONSTANT EMAIL_REGEX = "^[A-Za-z0-9+_.-]+@(.+)$"
    CONSTANT dbms = new DBMS()
    method main(args):
        ensureDataDirectoryExists()
        ensureFilesExist()
        while true:
            print "Main Menu"
            print "1. Register"
            print "2. Login"
            print "3. Exit"
            print "Enter your choice: "
            choice = scanner.nextLine()
            switch (choice):
                case "1":
                    registerUser()
                case "2":
                    loginUser()
                case "3":
                    print "Exiting program."
                    scanner.close()
                    return
                default:
                    print "Invalid choice. Please try again."
    method ensureDataDirectoryExists():
        try:
            create directories "src/main/resources"
        catch (IOException e):
            print error and exit
    method ensureFilesExist():
        try:
            if file does not exist DATA_FILE:
                create file DATA_FILE
            if file size is 0:
                write "userID###password###securityQuestions###securityAnswers\n" to DATA_FILE
        catch (IOException e):
            print error and exit

    method registerUser():
        print "--- User Registration ---"
```

```
while true:
    print "Enter UserID (email): "
    userID = scanner.nextLine()
    if isValidEmail(userID):
        break
    else:
        print "Invalid email format. Please try again."
users = UserUtils.loadUsers(DATA_FILE)
for each user in users:
    if user.getUserID().equals(userID):
        print "UserID already exists. Please try again."
        return
print "Enter Password: "
password = scanner.nextLine()
hashedPassword = HashUtils.hashMD5(password)
securityQuestions = ["What is your pet's name?", "What city were you born in?", "What is your favorite movie?"]
securityAnswers = []
for each question in securityQuestions:
    print question + ": "
    answer = scanner.nextLine()
    securityAnswers.add(answer)
newUser = new User(userID, hashedPassword, securityQuestions, securityAnswers)
UserUtils.saveUser(newUser, DATA_FILE)
print "Registration successful."
```

method loginUser():

```
print "--- User Login ---"
print "Enter UserID: "
userID = scanner.nextLine()
print "Enter Password: "
password = scanner.nextLine()
hashedPassword = HashUtils.hashMD5(password)
users = UserUtils.loadUsers(DATA_FILE)
currentUser = null
for each user in users:
    if user.getUserID().equals(userID) and user.getPassword().equals(hashedPassword):
        currentUser = user
        break
if currentUser is not null:
    if verifySecurityQuestions(currentUser):
        print "Login successful."
        userMenu(currentUser)
    else:
```

```
            print "Login failed. Incorrect answers to security questions."
        else:
            print "Login failed. Invalid UserID or Password."

    method verifySecurityQuestions(user):
        securityQuestions = user.getSecurityQuestions()
        securityAnswers = user.getSecurityAnswers()

        for i from 0 to size of securityQuestions:
            print securityQuestions.get(i) + ": "
            answer = scanner.nextLine()
            if answer not equals securityAnswers.get(i):
                return false
        return true

    method userMenu(user):
        while true:
            print "User Menu"
            print "1. Write Queries"
            print "2. Export Data and Structure"
            print "3. ERD"
            print "4. Exit"
            print "Enter your choice: "
            userChoice = scanner.nextLine()

            switch (userChoice):
                case "1":
                    writeAndExecuteQuery()
                case "2":
                    print "Export Data and Structure"
                case "3":
                    print "ERD or reverse engineer"
                case "4":
                    print "Logging out."
                    return
                default:
                    print "Invalid choice. Please try again."



method writeAndExecuteQuery():
        print "Enter your query: "
        query = scanner.nextLine()
        success = dbms.executeQuery(query)
```

```
        if success:
            print "Query executed successfully."
        else:
            print "Failed to execute query."

    method isValidEmail(email):
        return match EMAIL_REGEX with email


class User:
    method User(userID, password, securityQuestions, securityAnswers):
        this.userID = userID
        this.password = password
        this.securityQuestions = securityQuestions
        this.securityAnswers = securityAnswers
    method getUserID():
        return this.userID
    method getPassword():
        return this.password
    method getSecurityQuestions():
        return this.securityQuestions
    method getSecurityAnswers():
        return this.securityAnswers
    method toString():
        return userID + "###" + password + "###" + join securityQuestions with ";" + "###" + join
securityAnswers with ";"
    static method fromString(data):
        parts = split data by "###"
        userID = parts[0]
        password = parts[1]
        securityQuestions = split parts[2] by ";"
        securityAnswers = split parts[3] by ";"
        return new User(userID, password, securityQuestions, securityAnswers)




class UserUtils:
    static method saveUser(user, dataFile):
        try:
            append user.toString() + new line to dataFile
        catch (IOException e):
            print error
```

```
static method loadUsers(dataFile):
    users = new LinkedList<User>()
    try:
        lines = read all lines from dataFile
        for each line in lines:
            users.add(User.fromString(line))
    catch (IOException e):
        print error
    return users

class HashUtils:
    private HashUtils():
        raise AssertionError("HashUtils class cannot be instantiated.")

    static method hashMD5(password):
        return md5Hex(password)
```

UseDatabase:

```
function useDatabase(query, regex)
    print the query

    match query with regex.useDatabase pattern
    if query does not match then
        print "Invalid use database query"
        return false
    end if
    extract database name from matched group

    check if database is present with isDataBasePresent(databaseName)
    set isDatabasePresent to the result

    return isDatabasePresent
end method
```

CreateTable:

```
function createTable(tableName, columns)
    get the currently in-use database (db)

    if db is null then
        print "No database selected"
```

```
    return false
end if

set currentDir to the current working directory
set directoryPath to path for the table directory
set metadataFileName to "metadata.txt"
set dataFileName to "data.txt"

try
    create directory object with directoryPath
    create metadataFile object with metadata file path
    create dataFile object with data file path

    create directory
    create metadata file
    create data file
catch IOException e
    throw new RuntimeException(e)
end try

if dataFile, directory, and metadataFile are successfully created then
    create new Table object
    set filteredColumns to an empty list
    set metadataDirectoryPath to path for metadata file

    try
        open BufferedWriter with metadataDirectoryPath in append mode
        for each column in columns
            write column to file
            write new line
            add column name (without datatype) to filteredColumns
        end for
        write new line
    catch IOException e
        print e.getMessage()
    end try

    call addColumns(tableName, filteredColumns)
    add table to db
end if
```

```
    return true
end method
```

InsertData:

```
function insertData(query)
    Initialize tableName
    Initialize columns list
    Initialize values list

    Try
        Extract table name from query
        Extract columns part from query
        Extract values part from query

        Split columns string into array and add to columns list
        Split values string into array and add to values list
    Catch Exception e
        Print "Invalid insert query format."
        Return false
    End Try

    Check if table is present with isTablePresent(tableName)
    If dataFile is null then
        Return false
    End If

    Try
        Open BufferedWriter with dataFile in append mode
        Initialize row to be inserted
        For each value in values list
            If not the first value then
                Append separator " ### " to row
            End If
            Append value to row
        End For
        Write row to file
        Write new line
        Print "Data inserted successfully into table: " + tableName
        Return true
```

```
    Catch IOException e
        Print "Error writing data to table: " + e.getMessage()
        Return false
    End Try
End Method
```

SelectQuery:

```
Method selectQuery(query, regex)
    Initialize matcher
    Initialize hasWhereClause to false

    Match query with regex.selectPattern pattern
    Set hasWhereClause to true if query matches

    If query does not match selectPattern then
        Match query with regex.selectAllTablePattern pattern
        If query does not match then
            Print "Invalid SELECT query"
            Return false
        End If
    End If

    Extract columns string and table name from matched groups
    Split columns string into array; set to null if columns string is "*"
    Initialize whereField, whereValue, whereColumnIndex to null, null, -1
    If hasWhereClause then
        Extract whereField and whereValue from matched groups
    End If

    Check if table is present with isTablePresent(tableName)
    If dataFile is null then
        Return false
    End If



Try
        Open BufferedReader with dataFile
        Initialize line
        Initialize records list
        Initialize headers to null

        While line is not null do
```

```
        Split line into parts
        If headers are not initialized then
            Set headers to parts
            Trim headers if columns are specified

            If hasWhereClause then
                Find index of whereField in headers
                Print "WHERE field '" + whereField + "' not found" and available fields if not found
                Return false if not found
            End If
            Continue
        End If
        Add parts to records
    End While

    Initialize filteredRecords list
    For each record in records
        If no WHERE clause or record matches whereField and whereValue then
            Add record to filteredRecords
        End If
    End For

    If filteredRecords is empty then
        Print "No records found"
        Return true
    End If

    Initialize columnIndexes to null
    If specific columns are selected then
        Find indexes of columns in headers
        Print "Column " + column + " not found" if not found
        Return false if not found
    End If

    Print headers if all columns are selected; otherwise print selected columns

    For each record in filteredRecords
        Print all columns if columns is null; otherwise print selected columns
    End For
Catch IOException e
    Print "Error reading data file: " + e.getMessage()
    Return false
End Try
```

```
   Return true
End Method
```

UpdateQuery:

```
  function updateQuery(query, regex):
      if not matchQueryWithRegex(query, regex.update): return false
      tableName = extractTableName()
      setClause = extractSetClause()
      whereField = extractWhereField()
      whereValue = extractWhereValue()
      columnPair = splitSetClauseIntoPairs()
      valuePair = extractValuesFromPairs()
      filePath = buildDataFilePath(tableName)

      While data.txt has a line:
         Parts[] = get values line by line
         WhereColumIndex = get the index of column name to update
         WhereUpdateIndex = get the index of value to update

           for i from 0 to valuePair.size():
               Check if column name to update exist.
               Check if column value to update exist.
               If both is present then update usin whereColumnIndex and whereUpdateIndex the value and
               rewrite the text file.

return true
End method
```

createDatabse:

```
  function createDatabase(query, regex):

    //check if the regex is valid or not
    if not matchQueryWithRegex(query, regex.delete) return false

    //get the table name from the table
    tableName = extractTableName()

    //check if the database with same name already present or not
    checkIsDatabaseAlreadyPresent()

    //create the folder with name with the database name
    filePath = constructDataFilePath(tableName)

    //check if the directory is created or not
    IsDirectoryCreated = Makedir(filePath)
```

```
    if not isDirectoryCreated: return false
    else return true
```

DeleteQuery:

```
function deleteQuery(query, regex):

        //check if the regex is valid or not
        if not matchQueryWithRegex(query, regex.delete) return false

        //check if the database is selected
        if not isDatabaseSelected() return false

        //get table name , name of column in where clause , and value to compare in where clause
        tableName = extractTableName()
        whereField = extractWhereField()
        whereValue = extractWhereValue()

        //check if the table with the given name present
        isTableExist()

        //check if the field given in where cluse present in the table or not

        filePath = constructDataFilePath(tableName)
        While data.txt has a line:
            Get the index of the column name you want to delete from.
            Rewrite the content of the file except the line we want to delete.
```

DropQuery:

```
function dropQuery(query, regex):

        //check if the regex is valid or not
        i if not matchQueryWithRegex(query, regex.update): return false

        //get the table name from the query
        tableName = extractTableName()
        filePath = buildTableDirectoryPath(tableName)

         //check if the table is present in the database
```

```
    file = createFileObject(filePath)
    if not fileExists(file):
        print("Table does not exist")
        return false
  else delete the table folder along with it's subsequent data
```

UpdateQuery:

```
function updateQuery(query, regex):
    //check if the regex is valid or not
    i if not matchQueryWithRegex(query, regex.update): return false
    //check if the database is selected
    if not isDatabaseSelected() return false
    tableName = extractTableName()
    setClause = extractSetClause()
    whereField = extractWhereField()
    whereValue = extractWhereValue()
    columnPair = splitSetClauseIntoPairs()
    valuePair = extractValuesFromPairs()
    filePath = buildDataFilePath(tableName)

    //check if the table with the given name present
    isTableExist()
  While data.txt has a line:
  Parts[] = get values line by line
  WhereColumIndex = get the index of column name to update
  WhereUpdateIndex = get the index of value to update

        for i from 0 to valuePair.size():
          //Check if column name to update exist.
          isColumnPresent();

          //Check if column value in the where clause exist.
          isValuePresent()

          If both is present then update value using whereColumnIndex and whereUpdateIndex the
          value and rewrite the text file.

  return true;
```

Test cases and evidence of testing:

Description: Input: Existing UserID, Output: UserID already exists. Please try again

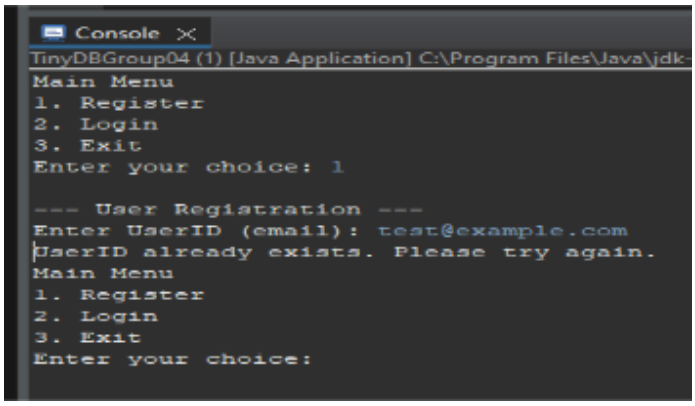

Figure 3 Menu to Login or Register into the System

Description: Input: UserId, Password and Security questions, Output: Registration successful

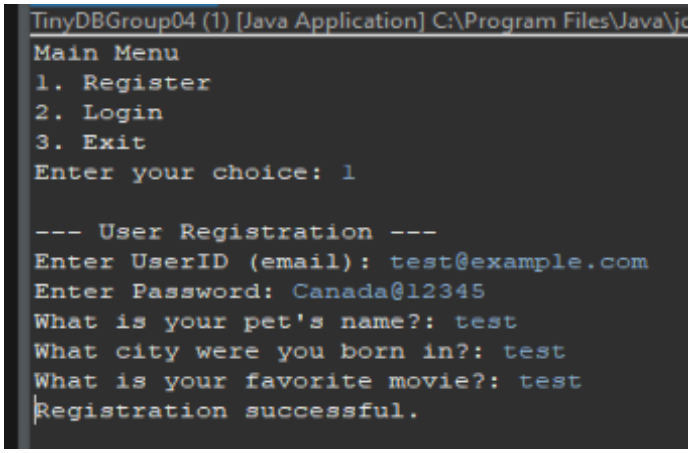

Figure 4 Successful User Registration

Description: Input: Login with registered UserId and Password, Output: Login Successful

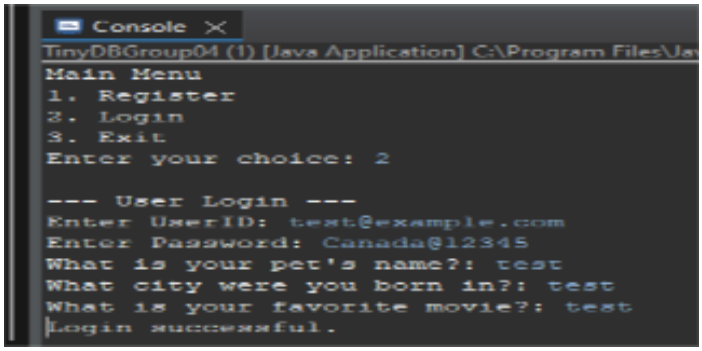

Figure 5 Successful Login

Description: Input: Login with incorrect UserID, Output: UserID incorrect

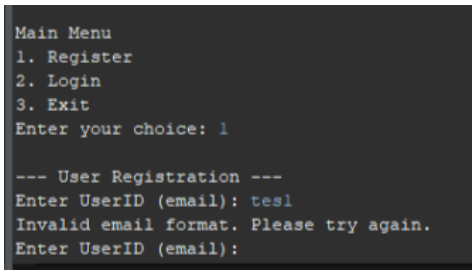

Figure 6 Registering with invalid UserID

Description: Input: Invalid UserID, Output: Login Failed. Invalid UserID

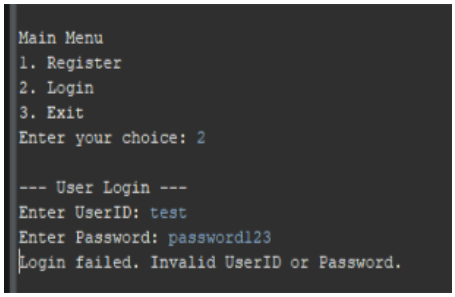

Figure 7 Login with Invalid UserID

Description: Input: Invalid Password, Output: Login Failed. Invalid Password

```
Main Menu
1. Register
2. Login
3. Exit
Enter your choice: 2

--- User Login ---
Enter UserID: test@example.com
Enter Password: password123
Login failed. Invalid UserID or Password.

Main Menu
1. Register
2. Login
3. Exit
Enter your choice:
```

Figure 8 Login with Invalid Password

Description: Input: Incorrect security answer, Output: Login Failed. Incorrect answer to security question

```
TinyDBGroup04 (1) [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (Jun
Main Menu
1. Register
2. Login
3. Exit
Enter your choice: 2

--- User Login ---
Enter UserID: test@example.com
Enter Password: Canada@12345
What is your pet's name?: test1
Login failed. Incorrect answers to security questions.

Main Menu
1. Register
2. Login
3. Exit
Enter your choice:
```

Figure 9 Login Failed due to Incorrect security Question

Description: Input: Create a valid database, Output: Query executed successfully

```
Enter your query: create database test
Query executed successfully.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice:
```

Figure 10 Create database successfully

Figure 11 Folder created for the database

Description: Input: Existing database creation, Output: Failed to execute query.

```
Enter your choice: 1
Enter your query: create database test
Failed to execute query.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice:
```

Figure 12 Create database failed because database already present

Description: Input: Invalid create database query, Output: Failed to execute query.

```
Enter your query: create test2
Unsupported query type
Failed to execute query.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
Enter your choice:
```

Figure 13 Create database failed because of invalid query

Description: Input: Create table query , Output: Query executed successfully

```
Enter your query: create table student (id int,name varchar)
Query executed successfully.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 14 Create table successful

```
1    id ### name
2
```

Figure 15 Table created

Description: Input: Insert query, Output: Query executed successfully

```
Enter your query: insert into student (1,'parth')
Data inserted successfully into table: student
Query executed successfully.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 16 Data inserted successfully

```
1    id ### name
2    1 ### parth
```

Figure 17 data.txt file after insertion

Description: Input: Delete query, Output: Query executed successfully

```
Enter your query: delete from student where id='1'
Query executed successfully.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 18 Delete from table successful

```
1    id ### name
2
```

Figure 19 Valued deleted successfully

Description: Input: Invalid Delete Query, Output: Failed to execute query

```
Enter your query: delete from student where address='2'
Failed to execute query.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 20 Delete unsuccessful because of invalid field name

```
Enter your query: delete from student2 where id='1'
Table does not exist
Failed to execute query.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 21 Delete unsuccessful because of invalid table name

```
Enter your query: drop table student
Query executed successfully.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 22 Drop table successful

```
Enter your choice: 1
Enter your query: drop table student2
Table does not exist
Failed to execute query.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 23 Drop table unsuccessful because of invalid table name

```
Enter your query: update student set name='ALEX' where id='1'
Query executed successfully.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 24 Update table successful

```
1    id ### name
2    1 ### ALEX
```

Figure 25 Data updated successfully in data.txt file

```
Enter your query: update student set address='halifax' where id='1'
address is not a valid field
Failed to execute query.
User Menu
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 26 Update invalid because of invalid field name

Team Meeting details:

Date	Time	Agenda	Attendees	Meeting Type	Meeting Recording Link
May 30	2:00 PM	Introduction and gathering requirements	Parth Madhvani, Shifa Mirza, Ramya Kommalapati	Offline	
June 07	9:30 PM	To discuss what formats and data structures to use. And who will do what?	Parth Madhvani, Shifa Mirza, Ramya Kommalapati	Online	**Link 1**

References

[1] "Which Data Structure to choose for frequent insert and delete operation?," [Online]. Available: https://www.geeksforgeeks.org/which-data-structure-should-we-choose-if-we-have-to-insert-and-delete-the-data-frequently/. [Accessed 29 June 2024].

[2] "https://www.scaler.com/topics/java/arraylist-in-java/," [Online]. Available: https://www.scaler.com/topics/java/arraylist-in-java/. [Accessed 29 June 2024].

[3] "What is Retrieval Operation in ArrayList Java?," JavaTPoint, [Online]. Available: https://www.javatpoint.com/what-is-retrieval-operation-in-arraylist-java. [Accessed 29 June 2024].

[4] "When to use LinkedList over ArrayList in Java?," [Online]. Available: https://stackoverflow.com/questions/322715/when-to-use-linkedlist-over-arraylist-in-java. [Accessed 29 June 2024].