

CSCI 5408

DATA MANAGEMENT AND WAREHOUSING

Group-9
Project: Tiny DB

SPRINT 1 Report

Group Members:

Naqib Ali(B00981531)

Shrey Nimeshkumar Patel(B00960433)

Syeda Misbah Hussain(B00982800)

Gitlab: https://git.cs.dal.ca/naqib/csci_5408_s24_9

Table of Contents

1. Background research	3
2. Architecture diagram	4
3. Pseudocode	5
4. Link to git code repository	7
5. Test cases and evidence of testing.....	8
6. References	15

1. Background research

In developing a database management system using Java file handling, our approach integrated several key technologies and methodologies that are part of core java.

Much of our exploration and investigation focused on optimising the utilisation of core Java functionalities. Prominent resources such as Stack Overflow, W3Schools, and JavaPoint were among the go to platforms for learning and efficiently using them.

The system employed regex[1] for robust query parsing, ensuring efficient handling and updating of table operations stored in text files. Each record was separated using a pipe symbol for clear separation within the files.

Central to our implementation were data structures such as arrays[2] and hash maps[3], chosen to Optimise storing conditions and values. These structures facilitated effective management of key-value pairs, supporting diverse functionalities ranging from data insertion to complex searching operations. By leveraging Java's buffered read and write mechanisms[4], we ensured streamlined performance in handling data operations, enhancing both reliability and speed within our system.

2. Architecture diagram

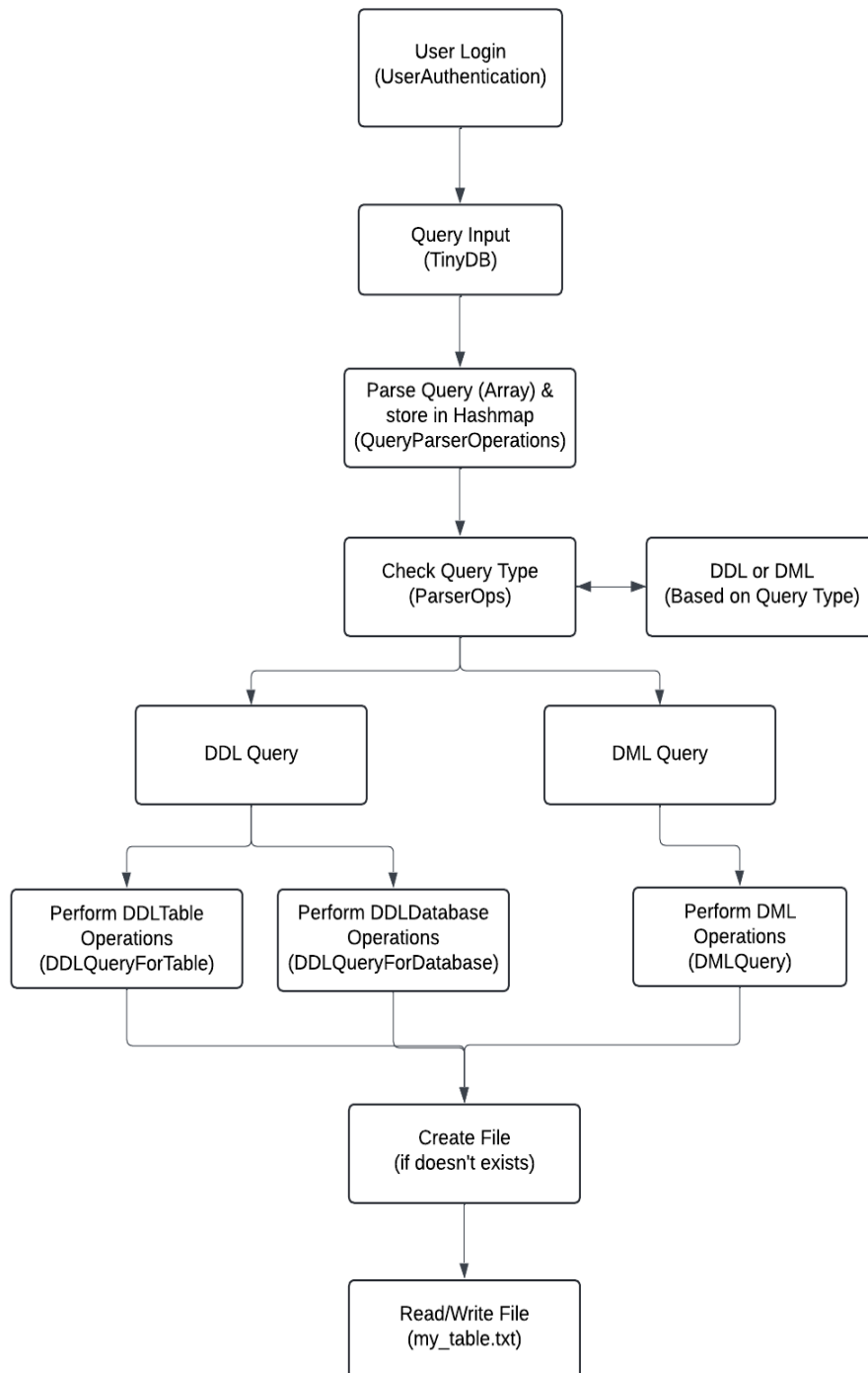


Figure 1: Architecture Diagram

3. Pseudocode

Below is the pseudocode for all the functionalities of database queries and user authentication that have been implemented in our Tiny DB java program:

Pseudocode for User Authentication:

- Step 1: Take input from the user for user name, password and security question and answer.
- Step 2: Check whether the username already exists or not, if yes then show a message.
- Step 3: After all 4 details are fetched, call the authentication function
- Step 4: That function will first use another function which will convert user id and password to hash values using SHA-256 hashing method.
- Step 5: The hash value will be converted into string.
- Step 6: Store the hashed string of user id and password and security question, answer to txt file of User_details.

Pseudocode for Create Database:

- Step 1: Call function queryparser(userInput)
- Step 2: Match with the regex expression
- Step 3: Call Database operations repository via interface
- Step 4: Call DDLQueryForDatabase via interface passing the string name
- Step 5: In Database folder of project create a new folder with given input name if it doesnot exists.
- Step 6: Show proper message in the CLI.

Pseudocode for Use Database:

- Step 1: Call function queryparser(userInput)
- Step 2: Match with the regex expression
- Step 3: Call Database operations repository via interface
- Step 4: Call DDLQueryForDatabase via interface passing the string name
- Step 5: Initialised a global path name that can be used in every module
- Step 6: Set the global path name to that of the name of the database provided which will be a folder.
- Step 7: Show proper message in the CLI.

Pseudocode for Create table:

- Step 1: Call function queryparser(userInput)
- Step 2: Match with the regex expression
- Step 3: Separate the table name and column details and pass to Table operations.
- Step 4: Call Table operations repository via interface
- Step 5: Separate the column details and store them as a key value pair in hashmap
- Step 6: Send the updated map to the DDLQueryTable interface.
- Step 7: Call DDLQueryForTable
- Step 8: Using the global path name setted in use database, create a new .txt file with column name separated by a pipe("|") if the table does not exist.

Step 9: Show proper message in the CLI.

Pseudocode for Insert table:

Step 1: Call function `queryparser(userInput)`

Step 2: Match with the regex expression

Step 3: Separate the table name, column details and its values and pass to Table operations.

Step 4: Call Table operations repository via interface

Step 5: Separate the column details and values of it and store them as a key value pair in hash map where key is the column name and value is the row value.

Step 6: Send the updated map and table name to the `DDLQueryTable` interface.

Step 7: Call `DMLQueryForTable`

Step 8: Using the global path name setted in use database, open the file and write the data record separated by a pipe below the column name or old data if any.

Step 9: Show proper message in the CLI.

Pseudocode for Select:

Step 1: Call function `queryparser(userInput)`

Step 2: Match with the regex expression

Step 3: Separate the table name, column details and condition passed (if any) to Table operations.

Step 4: Check if there exist any condition and store condition column and its value to match later

Step 5: Loop till the last line of the file

Step 6: Check if the line satisfies the condition and print matching lines and required columns. If no condition exists matching columns will be directly printed.

Step 7: Result count is printed out

Pseudocode for Update:

Step 1: Call function `queryparser(userInput)`

Step 2: Match with the regex expression

Step 3: Separate the table name, column to be modified, condition and pass to Table operations.

Step 4: Store column to be modified and the value in an array and store condition in another array

Step 5: Loop till the last line of the file

Step 6: Check if the line satisfies the condition if so update the column with the value specified.

Step 7: Number of rows modified is printed out.

Pseudocode for Delete table:

Step 1: Call function `queryparser(userInput)`

Step 2: Match with the regex expression

Step 3: Separate the table name and condition from the query

Step 4: Call Table operations repository via interface
Step 5: Call DMLQuery
Step 6: Read the table data from the file into memory
Step 7: Parse the condition from the WHERE clause
Step 8: Iterate over the rows and remove the rows that match the condition
Step 9: Write the updated data back to the table file
Step 10: Show proper message in the CLI.

Pseudocode for Drop table:

Step 1: Call function queryparser(userInput)
Step 2: Match with the regex expression
Step 3: Separate the table name from the query
Step 4: Call Database operations repository via interface
Step 5: Verify if the table exists
Step 6: Call DDLQueryForDatabase
Step 7: Delete the table file from the system if exists
Step 8: Show proper message in the CLI.

4. Link to git code repository

https://git.cs.dal.ca/naqib/csci_5408_s24_9

5. Test cases and evidence of testing

We have rigorously tested every module of the JAVA implementation that we have done of user authentication and queries for database and tables. We have made sure that all the edge cases get covered and the Tiny DB works the same as the way a standard SQL editor like Workbench works. For this we have taken the image of the CLI based application that user can see in the console as well as the database that gets updated in form of a .txt file separated by a pipe “|” symbol. Below are all the images of the testing done both of CLI and database level of the .txt file.

Description: Input: Enter invalid choice, Output: Please enter a valid input

```
Welcome to TinyDB
1. Register
2. Login
3. Exit
Choose an option: one
Please enter valid input between 1 to 3!
```

Figure 2: Invalid input

Description: Input: UserId and password, Output: Registration successful

```
Run TinyDB x
C:\Users\User\jdk\openjdk-21.0.2\bin\java.exe --enable-preview "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.2
Welcome to TinyDB
1. Register
2. Login
3. Exit
Choose an option: 1
Enter UserID: misbah
Enter Password: Misbah@12
Enter Security Question: your first name?
Enter Security Answer: Syeda Misbah
Registration successful!
```

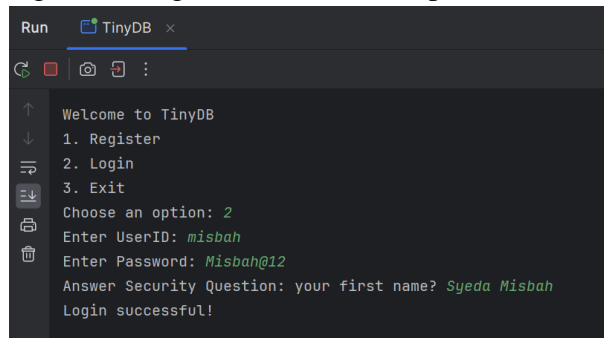
Figure 3: Registration - Registration successful

Description: Input: Existing UserId, Output: UserId already exists

```
Run TinyDB x
C:\Users\User\jdk\openjdk-21.0.2\bin\java.exe --enable-preview "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.2
Welcome to TinyDB
1. Register
2. Login
3. Exit
Choose an option: 1
Enter UserID: misbah
UserID already exists. Please choose a different UserID.
```

Figure 4: Registration - User already exists

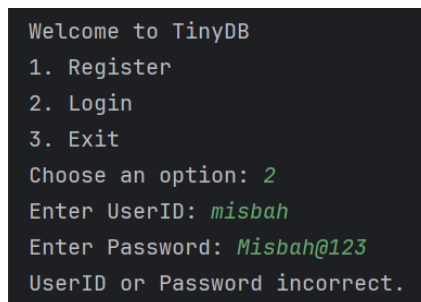
Description: Input: Login with registered UserId and password, Output: Login successful

A terminal window titled 'Run TinyDB' showing the following text: 'Welcome to TinyDB', '1. Register', '2. Login', '3. Exit', 'Choose an option: 2', 'Enter UserID: misbah', 'Enter Password: Misbah@12', 'Answer Security Question: your first name? Syeda Misbah', and 'Login successful!'.

```
Run TinyDB
Welcome to TinyDB
1. Register
2. Login
3. Exit
Choose an option: 2
Enter UserID: misbah
Enter Password: Misbah@12
Answer Security Question: your first name? Syeda Misbah
Login successful!
```

Figure 5: Login - Login successful

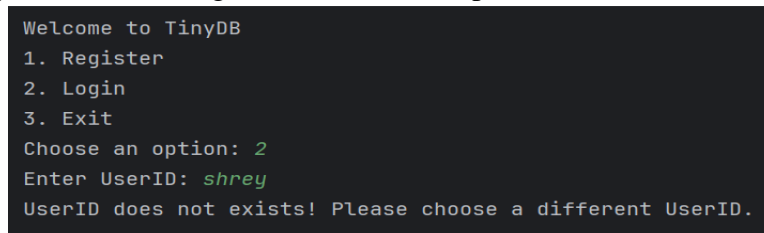
Description: Input: Login with incorrect UserId or password, Output: UserId or password incorrect

A terminal window showing the following text: 'Welcome to TinyDB', '1. Register', '2. Login', '3. Exit', 'Choose an option: 2', 'Enter UserID: misbah', 'Enter Password: Misbah@123', and 'UserID or Password incorrect.'.

```
Welcome to TinyDB
1. Register
2. Login
3. Exit
Choose an option: 2
Enter UserID: misbah
Enter Password: Misbah@123
UserID or Password incorrect.
```

Figure 6: Login - Incorrect UserID or Password

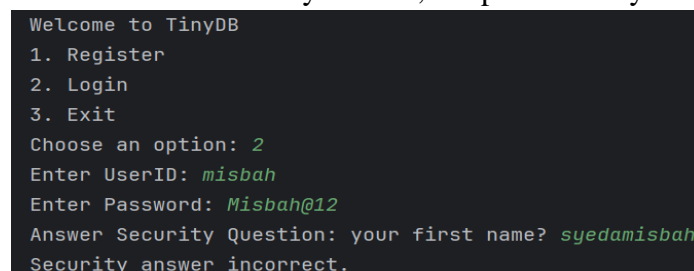
Description: Input: Enter non-registered UserId, Output: UserId does not exist

A terminal window showing the following text: 'Welcome to TinyDB', '1. Register', '2. Login', '3. Exit', 'Choose an option: 2', 'Enter UserID: shrey', and 'UserID does not exists! Please choose a different UserID.'.

```
Welcome to TinyDB
1. Register
2. Login
3. Exit
Choose an option: 2
Enter UserID: shrey
UserID does not exists! Please choose a different UserID.
```

Figure 7: Login - UserID does not exist

Description: Input: Enter incorrect security answer, Output: Security answer incorrect

A terminal window showing the following text: 'Welcome to TinyDB', '1. Register', '2. Login', '3. Exit', 'Choose an option: 2', 'Enter UserID: misbah', 'Enter Password: Misbah@12', 'Answer Security Question: your first name? syedamisbah', and 'Security answer incorrect.'.

```
Welcome to TinyDB
1. Register
2. Login
3. Exit
Choose an option: 2
Enter UserID: misbah
Enter Password: Misbah@12
Answer Security Question: your first name? syedamisbah
Security answer incorrect.
```

Figure 8: Login - Security answer incorrect

Description: Input: Create a valid database, Output: Database created

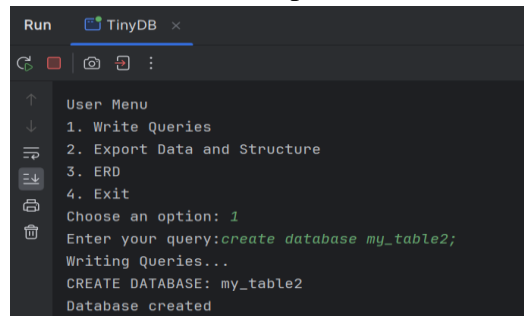
A screenshot of the TinyDB application interface. The title bar shows 'Run' and 'TinyDB x'. The interface has a dark theme. On the left is a sidebar with icons for navigation. The main area displays a 'User Menu' with four options: '1. Write Queries', '2. Export Data and Structure', '3. ERD', and '4. Exit'. Below the menu, it says 'Choose an option: 1'. Then, it prompts 'Enter your query: create database my_table2;'. The next line says 'Writing Queries...'. Then, it shows 'CREATE DATABASE: my_table2' and finally 'Database created'.

Figure 9: Create Database - Database created

Description: Valid database creation in folder structure

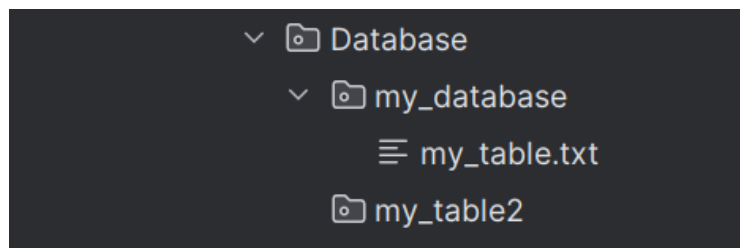


Figure 10: Database created in folder structure

Description: Input: Create existing database, Output: Database already exists

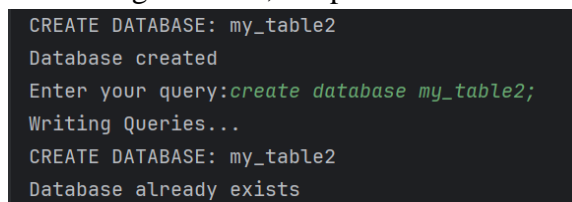
A screenshot of the TinyDB application interface showing the output of a failed database creation. It displays 'CREATE DATABASE: my_table2' followed by 'Database created'. Then, it prompts 'Enter your query: create database my_table2;'. The next line says 'Writing Queries...'. Then, it shows 'CREATE DATABASE: my_table2' and finally 'Database already exists'.

Figure 11: Create Database - Database already exists

Description: Input: Use existing Database, Output: Uses Database and moves forward to the next query

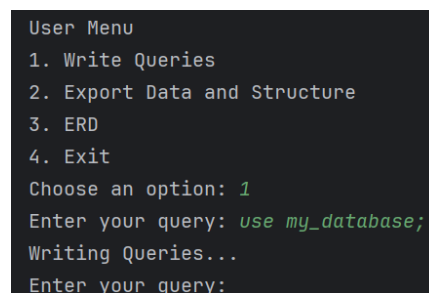
A screenshot of the TinyDB application interface showing the output of using an existing database. It displays the 'User Menu' with four options: '1. Write Queries', '2. Export Data and Structure', '3. ERD', and '4. Exit'. Below the menu, it says 'Choose an option: 1'. Then, it prompts 'Enter your query: use my_database;'. The next line says 'Writing Queries...'. Then, it prompts 'Enter your query:'.

Figure 12: Use - Database in use

Description: Input: Use non-existent database, Output: Database does not exist

```
Enter your query:use database;  
Writing Queries...  
USE DATABASE: database  
Database does not exist
```

Figure 13: Use - Database does not exist

Description: Input: Create a valid table, Output: Table created with the name

```
Enter your query:create table my_table2 (id int, name varchar(255), age int);  
Writing Queries...  
CREATE TABLE: my_table2 Columns: id int, name varchar(255), age int  
[id int, name varchar(255), age int]  
Table created with name: my_table2
```

Figure 14: Create Table - Table created

Description: Input: Create table with existing table name, Output: Table already exists

```
Enter your query:create table my_table2 (id int, name varchar(255), age int);  
Writing Queries...  
CREATE TABLE: my_table2 Columns: id int, name varchar(255), age int  
[id int, name varchar(255), age int]  
Table already exists!
```

Figure 15: Create Table - Table already exists

Description: Input: Insert row with valid columns and values, Output: Row added to the table

```
Enter your query:insert into my_table2 (id, name, age) values (1, 'Alice', 23);  
Writing Queries...  
INSERT INTO TABLE: my_table2 Columns: id, name, age Values: 1, 'Alice', 23  
{name=Alice, id=1, age=23}  
Row added to table 'my_table2': Alice | 1 | 23
```

Figure 16: Insert - Row added

Description: Inserted row

DMLQuery.java © UserAuthentication.java © TinyDB.java my_table2.txt ×			
1	name	id	age
2	Alice	1	23
3			

Figure 17: Insert - Row added in txt file

Description: Input: Select existing column from the table, Output: Shows the column and row selected

```
Enter your query:select * from my_table2 where id = 1;
Writing Queries...
SELECT FROM TABLE: my_table2 Columns: * WHERE Condition: id = 1
name | id | age|
Alice | 1 | 23
```

Figure 18: Select Query

Description: Input: Select not existent column from the table, Output: Unknown column

```
Enter your query: select * from my_table where id = 3;
Writing Queries...
Unknown column *
```

Figure 19: Select Query - Unknown column

Description: Input: Update existing row in the table, Output: Updates the row successfully

```
Enter your query:update my_table2 set name = 'Daisy' where id = 1;
Writing Queries...
UPDATE TABLE: my_table2 Set: name = 'Daisy' WHERE Condition: id = 1
Enter your query:select * from my_table2 where id = 1;
Writing Queries...
SELECT FROM TABLE: my_table2 Columns: * WHERE Condition: id = 1
name | id | age|
Daisy | 1 | 23
```

Figure 20: Update Successful

Description: Update row in txt file

base.java		© DMLQuery.java	© UserAuthentication.java	© TinyDB.java	☰ my_table2.txt ×
1	name id age				
2	Daisy 1 23				
3					

Figure 21: Row updated in txt file

Description: Input: Update row in a table that does not exist, Output: Table file does not exist

```
Enter your query:update my_table3 set name = 'Mike' where id = 1;
Writing Queries...
UPDATE TABLE: my_table3 Set: name = 'Mike' WHERE Condition: id = 1
Table file does not exist: csci_5408_s24_9\src\main\java\org\example\Database\my_table2\my_table3.txt
```

Figure 22: Update - Table not found

Description: Input: Update column that does not exist in the table, Output: Unknown column

```
Enter your query: update my_table2 set department = 'Uni' where id = 1;
Writing Queries...
UPDATE TABLE: my_table2 Set: department = 'Uni' WHERE Condition: id = 1
Unkown column department
```

Figure 23: Update - Unknown column

Description: Input: Update row where id does not exist, Output: No modifications made

```
Enter your query: use my_table2;
Writing Queries...
USE DATABASE: my_table2
Enter your query: UPDATE my_table2 SET name = 'Mike' WHERE id = 1;
Writing Queries...
UPDATE TABLE: my_table2 Set: name = 'Mike' WHERE Condition: id = 1
0 rows modified
```

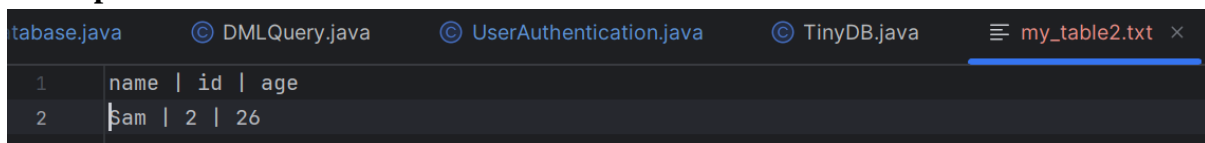
Figure 24: Update - no modifications

Description: Input: Delete existing row, Output: Deleted row successfully

```
Enter your query: delete from my_table2 where id = 1;
Writing Queries...
DELETE FROM TABLE: my_table2 WHERE Condition: id = 1
Deleted row: Daisy | 1 | 23
```

Figure 25: Delete - Delete successful

Description: Deleted row in txt file



	name	id	age
2	Sam	2	26

Figure 26: Delete - Row deleted in txt file

Description: Input: Delete row from a non-existent table, Output: Table file does not exist

```
Enter your query: delete from my_table3 where id = 1;
Writing Queries...
DELETE FROM TABLE: my_table3 WHERE Condition: id = 1
Table file does not exist: csci_5408_s24_9\src\main\java\org\example\Database\my_table2\my_table3.txt
```

Figure 27: Delete - Table not found

Description: Input: Delete a row that is not present, Output: Row either deleted or does not exist

```
Enter your query: delete from my_table2 where id = 1;
Writing Queries...
DELETE FROM TABLE: my_table2 WHERE Condition: id = 1
Row either deleted or does not exist
```

Figure 28: Delete - Row either deleted or does not exist

Description: Input: Drop existing table, Output: Table dropped successfully

```
Enter your query: drop table my_table2;
Writing Queries...
Table dropped successfully: my_table2
```

Figure 29: Drop - Dropped table successfully

Description: Drop table in database

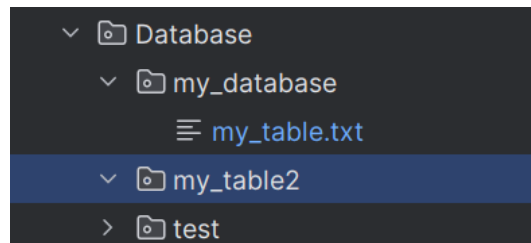


Figure 30: Dropped table in folder structure

Description: Input: Drop a table that is not present, Output: Table file does not exist

```
Enter your query: drop table my_table3;
Writing Queries...
Table file does not exist: csci_5408_s24_9\src\main\java\org\example\Database\my_table2\my_table3.txt
```

Figure 31: Drop - Table does not exist to drop

Description: Input: Exit from TinyDB, Output: Exits

```
Welcome to TinyDB
1. Register
2. Login
3. Exit
Choose an option: 3
```

Figure 32: Exit

6. References

- [1] “Java Regex” Available: <https://www.javatpoint.com/java-regex> [Accessed: June 10, 2024]
- [2] “Java Arrays” Available: https://www.w3schools.com/java/java_hashmap.asp [Accessed: June 10, 2024]
- [3] “Java HashMap” Available: https://www.w3schools.com/java/java_hashmap.asp [Accessed: June 10, 2024]
- [4] “Java Bufferedwriter class” Available: <https://www.javatpoint.com/java-bufferedwriter-class> [Accessed: June 10, 2024]