CSCI 5408
Data Management, Warehousing Analytics

Project – Final – Report

Submitted to

Dr. Saurabh Dey

Department of Computer Science

Dalhousie University.

Submitted by Group 5

Alishan Ali (B00754062)
Kenil (B00981263)
Harshil Makwana (B00985236)

Table of Contents

Gitlab Repository Link: 3

https://git.cs.dal.ca/alishan/csci_5408_s24_group5Overview 3

Background Research 3

Architecture Diagram: 4

Evidence of testing: 5

Module 1& 2: Query Implementation 5

Module 3: Transaction 10

Module 4: Log Management 28

Module 5: Data Modelling – Reverse Engineering 30

Module 6: Export Structure 34

Pseudocode: 36

Module 7: User Authentication 38

Meeting Log: 42

References: 43

Gitlab Repository Link:
https://git.cs.dal.ca/alishan/csci_5408_s24_group5

Overview

This report document will take you through all the first sprint work that was planned and completed in a descriptive manner. Moreover, highlighting the background research done and choices made while completing this sprint one to function as the base layer of the overall tiny dB project.

Background Research

In computer science, we focus more on problem solving, where the planning of any solution design plays a crucial role, and it is more important than the implementation part. Keeping that in mind, in sprint one, first we need to select the three modules. Although, we could have selected any modules, but we decided to go with module 1, 2 and 7 because of the following reasons:

1. Module 1 was about doing our research and selecting appropriate data structure which would help in keeping the project structure flexible to further extend new modules and reuse the existing code.
2. Module 2 was selected because its core requirement of the project and all the modules are related to this module, so it was essential to implement this in sprint 1 to get a clear view of how the project can progress.
3. Lastly, Module 7 work was implemented at the very end, its implementation helped us to understand how effectively we can integrate any new module with second module and how the starting point of the project would look like.

   As part of our background research, we first identified the best suited data structure out of diverse types of list data structure i.e., lists, arrays, Array List, HashMap List. We choose LinkedHashMap implemented using HashMap List. The main reason was that it maintains the order to insertion that we can get benefit later during the implementation of transaction module.

   Moreover, for the second module implementation we identified different entities based on the overall query execution design implementation and then created different classes for that. This also included the research of how we would store the data into the files and how we retrieve that from a file. This was done by keeping individual files for each database and using the database object containing columns and datatype as attribute we used concept of

serialization to keep all the data in the form of objects in the buffer.

Lastly, for the module 7, we reused most of the data read and write logic from the 2nd module. However, we researched the hashing algorithm that we are going to implement for storing user Id and password in hashed version for security purpose. We finalized the MD5 algorithm and successfully implemented that [1].

Further details of the technical implementation of each module can be found in the individual module sections along with screenshots in the functional testing section.

Architecture Diagram:

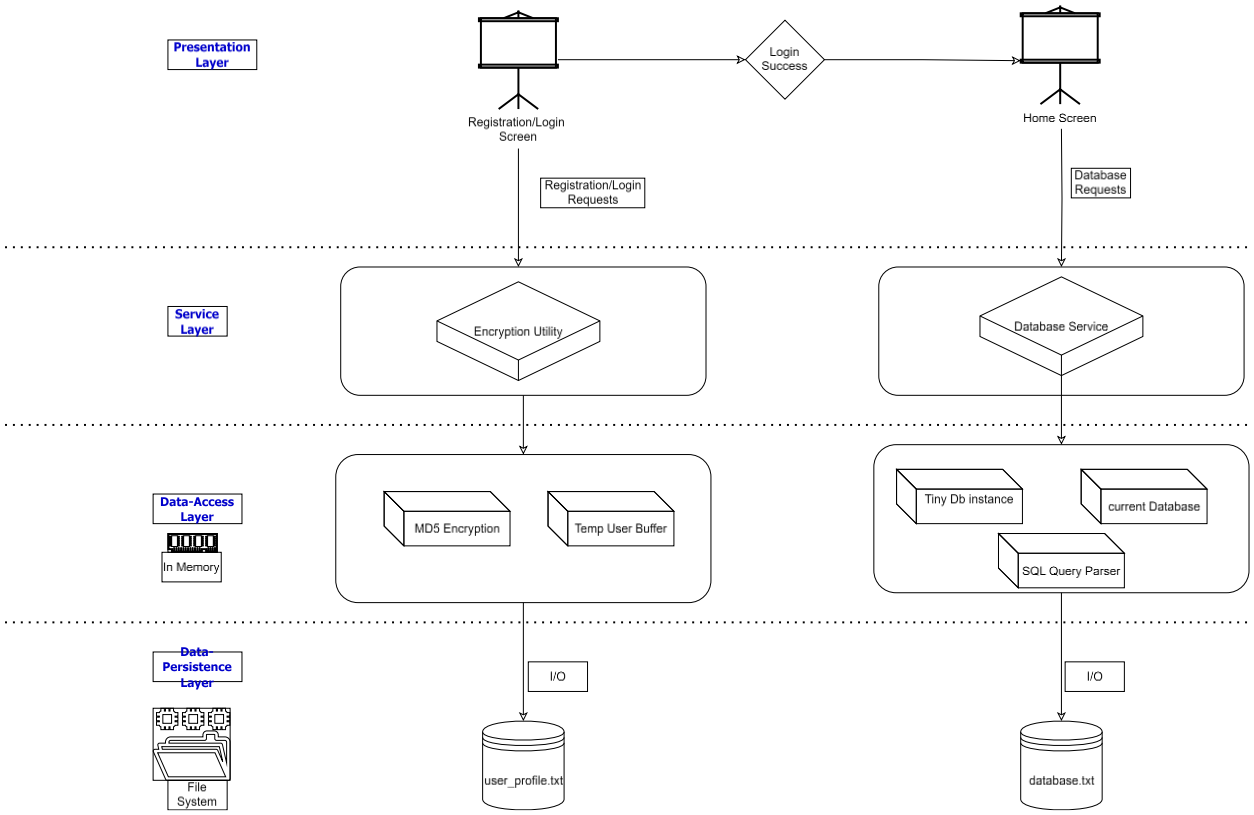

Figure 0: architectural diagram of the application.

Evidence of testing:

Module 1& 2: Query Implementation

- Creating the Database: -

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
CREATE DATABASE students;
The query entered is:CREATE DATABASE students;
```

Figure 1:database creation.

- Using the created Database: -

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
USE students;
The query entered is:USE students;
```

Figure 2: use database query execution.

- Creating the table in the selected database: -

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
CREATE TABLE student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50),
    age INT
);
Successfully added the table
The query entered is:CREATE TABLE student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50),
    age INT
);
```

Figure 3: create table query execution

- Inserting the data in the Table: -

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
INSERT INTO student (id, name, email, age) VALUES (1, 'Student1', 'student1@mail.com', 20);
Successfully added entry with Id:1
The query entered is:INSERT INTO student (id, name, email, age) VALUES (1, 'Student1', 'student1@mail.com', 20);
```

Figure 4: insert data into the table query execution.

- Selecting everything from the table: -

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
SELECT * FROM student;
{id=1, name='Student1', email='student1@mail.com', age=20}
{id=2, name='Student2', email='student2@mail.com', age=25}
The query entered is:SELECT * FROM student;
```

Figure 5: selecting all the data from the table query execution

- Selecting specific data from the table: -

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
SELECT name FROM student WHERE age = 20;
{name='Student1'}
The query entered is:SELECT name FROM student WHERE age = 20;
```

Figure 6: selecting the specific data from the table query execution

- Updating the data in the table: -

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
UPDATE student SET email = 'studentUpdate@mail.com' WHERE id = 1;
The existing data is:{id=1, name='Student1', email='student1@mail.com', age=20}
The current Column Name:id
The current value of the columns is:1
The current Column Name:name
The current value of the columns is:'Student1'
The current Column Name:email
The current value of the columns is:'student1@mail.com'
The index of the value in the Values is:0
The new Value to be setted is:'studentUpdate@mail.com'
The current Column Name:age
The current value of the columns is:20
The updated Map is:{id=1, name='Student1', email='studentUpdate@mail.com', age=20}
Successfully updated the entry:Updated the data successfully
The query entered is:UPDATE student SET email = 'studentUpdate@mail.com' WHERE id = 1;
```

Figure 7: updating the data in the table query execution

- Deleting the data from the table: -

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
DELETE FROM student WHERE id = 2;
The query entered is:DELETE FROM student WHERE id = 2;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
SELECT * FROM student;
{id=1, name='Student1', email='studentUpdate@mail.com', age=20}
The query entered is:SELECT * FROM student;
```

Figure 8: deleting the data in the table query execution

- Dropping the table from the database: -

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
DROP TABLE student;
Successfully dropped the table with name:student
The query entered is:DROP TABLE student;
```

Figure 9: dropping the table query execution

Module 3: Transaction

Insert Query: -

- The case when the user enters the transaction and performs an insert followed by the commit.

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=2, name='Rohn', email='rohn@mail.com', age=25}
The query entered is:select * from users;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
start transaction;
The query entered is:start transaction;
```

Figure 10:initial state be fore insert transaction operation

```
Run    Main ×
4. Exit
1
Please enter the query
start transaction;
The query entered is:start transaction;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
INSERT INTO users (id, name, email, age) VALUES (3, 'Tron', 'tron@mail.com', 29);
Successfully added entry with Id:3
The query entered is:INSERT INTO users (id, name, email, age) VALUES (3, 'Tron', 'tron@mail.com', 29);

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
commit;
The query entered is:commit;
```

Figure 11: The insert query while in the transaction state followed by commit

```
Run    Main
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
commit;
The query entered is:commit;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=2, name='Rohn', email='rohn@mail.com', age=25}
{id=3, name='Tron', email='tron@mail.com', age=29}
The query entered is:select * from users;
```

Figure 12: The result after the transaction insert operation after the commit

- The case when the user enters the transaction and performs an insert followed by the rollback.

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=2, name='Rohn', email='rohn@mail.com', age=25}
{id=3, name='Tron', email='tron@mail.com', age=29}
The query entered is:select * from users;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
start  transaction;
```

Figure 13: initial state before insert transaction operation

```
Run    Main ×

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
start transaction;
The query entered is:start transaction;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
INSERT INTO users (id, name, email, age) VALUES (4, 'insertwithrollback', 'rollback@mail.com', 29);
Successfully added entry with Id:4
The query entered is:INSERT INTO users (id, name, email, age) VALUES (4, 'insertwithrollback', 'rollback@mail.com', 29)

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
rollback;
```

Figure 14: The insert query while in the transaction state followed by rollback

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
rollback;
The query entered is:rollback;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=2, name='Rohn', email='rohn@mail.com', age=25}
{id=3, name='Tron', email='tron@mail.com', age=29}
The query entered is:select * from users;
```

Figure 15: The result after the transaction insert operation after the rollback

Update Query: -

- The case where the user enters the transaction and performs an update query followed by a commit.

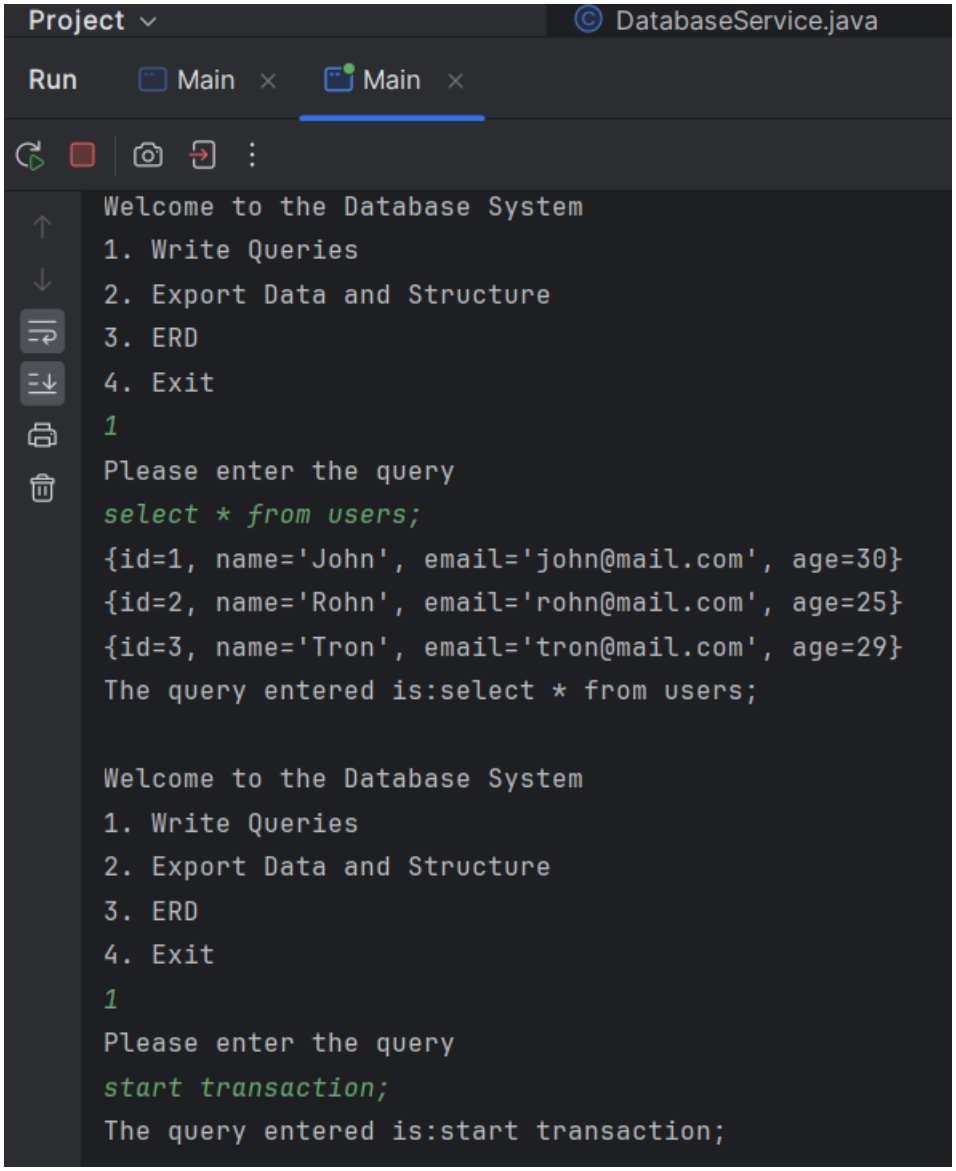

Figure 16: initial state of the db before transaction with update query

```
Run    Main    Main
1
Please enter the query
start transaction;
The query entered is:start transaction;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
update users set name='Doe' where id=2;
Successfully updated the entry:Updated the data successfully
The query entered is:update users set name='Doe' where id=2;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
commit;
The query entered is:commit;
```

Figure 17: update query while in transaction state with commit

```
Run    Main    Main
4. Exit
1
Please enter the query
update users set name='Doe' where id=2;
Successfully updated the entry:Updated the data successfully
The query entered is:update users set name='Doe' where id=2;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
commit;
The query entered is:commit;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=2, name='Doe', email='rohn@mail.com', age=25}
{id=3, name='Tron', email='tron@mail.com', age=29}
The query entered is:select * from users;
```

Figure 18: result after the transaction update query with commit

- The case where the user enters the transaction and performs an update query followed by a commit.

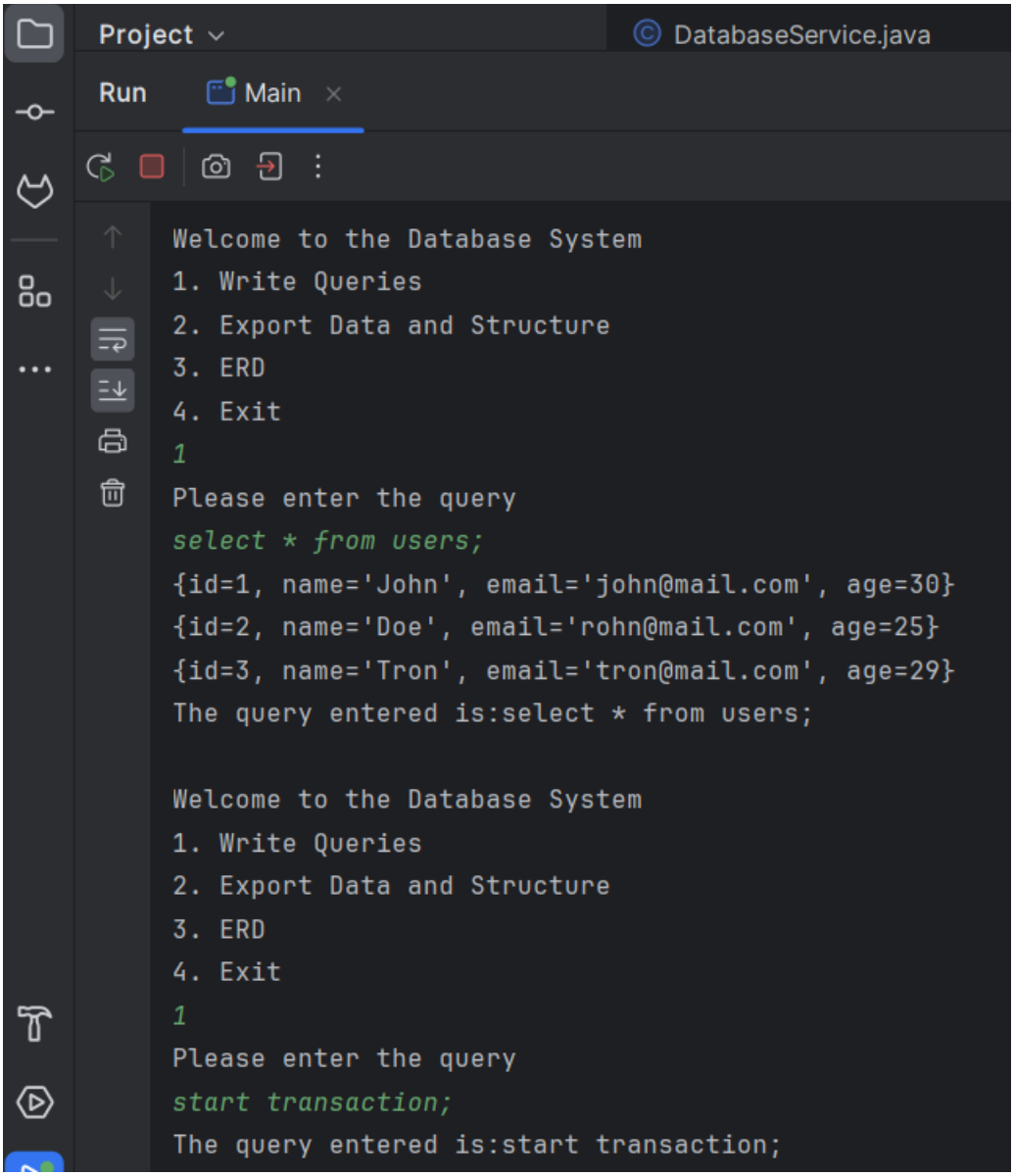

Figure 19: the db records before the transaction update query followed by rollback

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
start transaction;
The query entered is:start transaction;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
update users set name='Vron' where id=3;
Successfully updated the entry:Updated the data successfully
The query entered is:update users set name='Vron' where id=3;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
rollback;
```

Figure 20: the update transaction query followed by a rollback

```
Run    Main
1
Please enter the query
update users set name='Vron' where id=3;
Successfully updated the entry:Updated the data successfully
The query entered is:update users set name='Vron' where id=3;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
rollback;
The query entered is:rollback;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=2, name='Doe', email='rohn@mail.com', age=25}
{id=3, name='Tron', email='tron@mail.com', age=29}
The query entered is:select * from users;
```

Figure 21: the result after the update transaction with rollback

Delete Query: -

- The case where the user enters the transaction and performs a delete query followed by a commit.

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=2, name='Doe', email='rohn@mail.com', age=25}
{id=3, name='Tron', email='tron@mail.com', age=29}
The query entered is:select * from users;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
start transaction;
The query entered is:start transaction;
```

Figure 22: db state before the transaction with delete query followed by commit

```
Run    Main ×

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
start transaction;
The query entered is:start transaction;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
delete from users where id=2;
The query entered is:delete from users where id=2;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
commit;
The query entered is:commit;
```

Figure 23: the delete transaction query performed with the commit

```
Run    Main ×

3. ERD
4. Exit
1
Please enter the query
delete from users where id=2;
The query entered is:delete from users where id=2;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
commit;
The query entered is:commit;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=3, name='Tron', email='tron@mail.com', age=29}
The query entered is:select * from users;
```

Figure 24: the result after the transaction delete query followed by the commit

- The case where the user enters the transaction and performs a delete query followed by a rollback.

```
Run    Main

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=3, name='Tron', email='tron@mail.com', age=29}
The query entered is:select * from users;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
start transaction;
The query entered is:start transaction;
```

Figure 25: the initial db state before the transaction delete query with rollback

```
Run    Main

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
start transaction;
The query entered is:start transaction;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
delete from users where id=3;
The query entered is:delete from users where id=3;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
rollback;
```

Figure 26: the transaction delete query followed by the rollback

```
Run    Main
3. ERD
4. Exit
1
Please enter the query
delete from users where id=3;
The query entered is:delete from users where id=3;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
rollback;
The query entered is:rollback;

Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
select * from users;
{id=1, name='John', email='john@mail.com', age=30}
{id=3, name='Tron', email='tron@mail.com', age=29}
The query entered is:select * from users;
```

Figure 27: the result of the transaction delete query followed by rollback

Module 4: Log Management

General Logs:

1. Provides query execution time.
2. Provides database state after each DDL/DML query by showing total tables and total records in each table.

Within general log there are three types of logs:

1. Authentication
2. Execution time
3. Database State

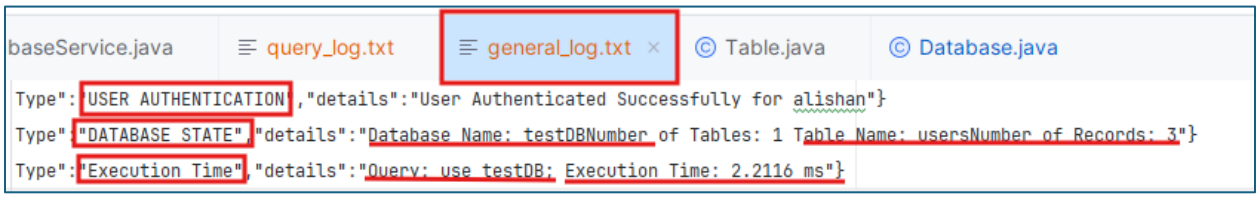

Figure 28: Shows typical general log example.

The execution time type log shows the query that was executed, and time taken.

Let's try to add a new table and insert new records to check if the logs are updating correctly or not.

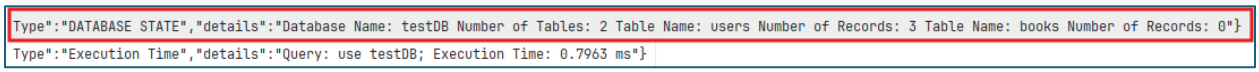

Figure 29: Shows newly added books table on the logs with 0 records.

Figure 30: Shows successful updating of database state, adding log for new record in books table.

Event Logs:

1. Shows transaction detection logs, commit or rollback of transaction based on user context.
2. Shows any runtime error if caused during query processing.

Typical event logs for error message are shown below:

Figure 31: Shows the error message in the event log.

Below figure 23 and 24 shows the event logs related to transactions, it shows on which table the lock was applied due to transaction, and when it was commit or rollbacked.

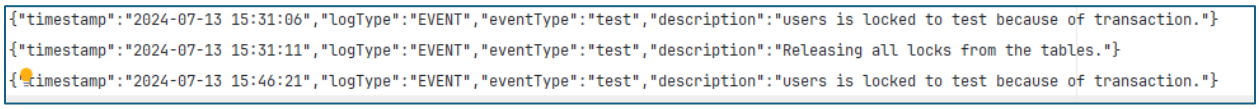

Figure 32: Shows the transaction lock applied and released on tables.

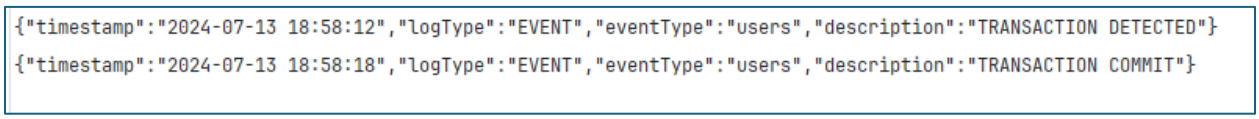

Figure 33: Shows the transaction detection and commit statement log.

Query Logs:

1. Shows query entered for execution based on user context.
2. Shows type of query that is entered with timestamp.

The query log generally shows the logs for query that are processed for execution, it shows the userId who has entered the query, timestamp and the query with its type for example it show if insert query was entered or create table.

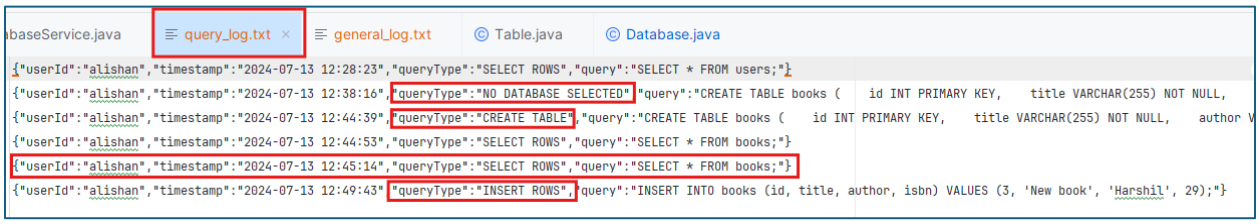

Figure 34: Shows the output of query logs.

Module 5: Data Modelling – Reverse Engineering

- Use database query to use “db” database

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
use db;
The query entered is:use db;
```

Figure 35: Use db query

- Create table query for user table

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
CREATE TABLE user (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(50),
    age INT
);
Successfully added the table
```

Figure 36: User table created successfully

- Create table query for account table with foreign key constraints

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
CREATE TABLE account (
    id INT PRIMARY KEY,
    accName VARCHAR(50),
    FOREIGN KEY id REFERENCES user(id)
);
Successfully added the table
```

Figure 37: Account table created successfully

- Create table query for course table with foreign key constraints

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
CREATE TABLE course (
    id INT PRIMARY KEY,
    courseName VARCHAR(50),
    userId VARCHAR(50),
    FOREIGN KEY userId REFERENCES user(id)
);
Successfully added the table
```

Figure 38: Course table created successfully

- Generation of ERD for “db” database

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
3
Generating ERD...
Successfully generated ERD in file: db_erd.txt
```

Figure 39: ERD generated successfully for “db” database (db_erd.txt)

- Generated ERD file for “db” database

```
eateTableQuery.java    SQLQueryParser.java    database.txt    db_erd.txt
1   user ( id) is related to account (id) [1 to 1]
2   user ( id) is related to course (userId) [1 to N]
```

Figure 40:db_erd.txt

- Here, in create table query post table is referencing department table that does not exist in database.

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
CREATE TABLE post (
    id INT PRIMARY KEY,
    accName VARCHAR(50),
    userId VARCHAR(50),
    FOREIGN KEY id REFERENCES department(id)
);
Table 'department' referenced in foreign key constraint does not exist in the database.
```

Figure 4: Referencing table which does not exist

- Here, in create table query job table is referencing email attribute of account table that does not exist in database.

```
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
1
Please enter the query
CREATE TABLE job (
    id INT PRIMARY KEY,
    userId VARCHAR(50),
    FOREIGN KEY id REFERENCES account(email)
);
Referenced column ' email' does not exist in table 'account'.
```

Figure 42: Referencing attribute of table which does not exist

Module 6: Export Structure

In this module, we were supposed to generate SQL dump based on the current state of the database. This file shows query structure for table creation and insert the records in each table. Figure 28 shows a successful SQL dump file generated using the option shown in Figure 27.

In the structure, all the tables that exist are present with all of the columns and datatypes, in case if a column is primary key, then it is shown in the structure and if it's a VARCHAR type then size of VARCHAR is also shown.

Note: User needs to be successfully logged into the tinyDB to use this feature.

```
Login successful!
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 43: Shows Option to export the current database structure into SQL dump files.

testDB_dump.sql × d1_dump.sql

No data sources are configured to run this SQL and provide advanced

SQL dialect is not configured.

```sql
DROP TABLE IF EXISTS users;
CREATE TABLE users (
    id INT PRIMARY KEY,
    name VARCHAR(100),
    age INT
);
INSERT INTO users VALUES
    (1, 'Alice', 30),
    (3, 'Tron', 29),
    (10, 'John', 25),
    (11, 'Alishan', 25),
    (2, 'Rohn', 25)
;
DROP TABLE IF EXISTS books;
CREATE TABLE books (
    id INT PRIMARY KEY,
    title VARCHAR(255),
    author VARCHAR(255),
    isbn VARCHAR(20)
);
INSERT INTO books VALUES
    (3, 'New book', 'Harshil', '29')
;
```

Figure 44: Shows SQL dump of TestDB containing 2 tables.

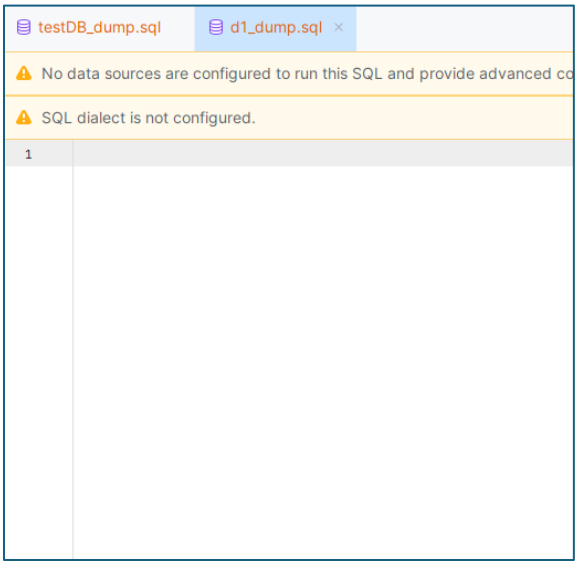

Figure 45: Shows SQL dump of empty database.

Pseudocode:

```
Function DumpDB() Returns boolean
    Set sqlDumpGenerated to false
    Get the list of databases from tinyDb

    For each database in databases
        Get the database name
        Initialize an empty StringBuilder called sqlDump

        For each table in the database
            Get the table name
            Append "DROP TABLE IF EXISTS" and "CREATE TABLE" SQL statements to sqlDump
            Get the list of columns for the table
            For each column in columns
```

```
            Append the column definition to sqlDump
            If the column data type is VARCHAR
                Append the size to sqlDump
            End If
        End For
        Append ");\n" to sqlDump

        Get the list of rows for the table
        If there are rows in the table
            Append "INSERT INTO" SQL statement to sqlDump
            For each row in rows
                Append the row data to sqlDump
            End For
            Append ";\n" to sqlDump
        End If
    End For
    Try to write the sqlDump to a file named after the database
        Set sqlDumpGenerated to true
    Catch IOException
        Set sqlDumpGenerated to false
        Throw RuntimeException with the error message
End For
Return sqlDumpGenerated

End Function
```

Module 7: User Authentication

This module focuses on the user authentication part and to complete its implementation this module was divided into following tasks:

1. Identify and create all classes that would be responsible for this module implementation and integration with Module 2.
2. Think of the logic that will be used for storing the user credentials data along with the security answers.
3. How to load existing user credentials in the buffer to be validated when a user tries to login.
4. How to redirect to the main menu after successful user login.
5. Format for storing the user data.

For this module, we identified the need of buffer class which would load all the user's data from User Profile file and then converting all the data into HashMap for efficiently authenticating the user credentials whenever a user tries to login.

The other approach would be to use List of User type objects but that would compare all objects until a match was found. This could be easy in terms of implementation but could have increased the space complexity.

Then we implemented the presentation layer for the user for registration and login. Which could be referred in the figure 30 and figure 31.

```
******* TINY DATABASE - MAIN MENU *******

1. User Login
2. Registration
0. Exit

Select an option:
```

Figure 46: Main menu screen when tiny db program executes.

```
******* LOGIN - PAGE *******

Enter your username:
alishan
UserId not exists. Do you want to try again? (yes/no)
alishan143
Invalid input. Please enter 'yes/no'.
yes


******* LOGIN - PAGE *******

Enter your username:
alishan143
Enter your password:
```

Figure 47: Illustrating the login page screen.

```
******* TINY DATABASE - REGISTRATION PAGE *******

Enter your username:
User2
Enter your password:
Password2
What is your mother's maiden name?
Mother2
What was the name of your first pet?
Pet2
What was the name of your elementary school?
School2
User Registered Successfully.
```

Figure 48: Illustrating the registration page.

Whenever the user gets registered a record has been entered into the User Profile which can be referred to figure 33.

```
Main.java    User_Profile.txt    PrintScreen.java
527bd5b5d689e2c32ae974c6229ff785$34819d7beeabb9260a5c854bc85b3e44$1=1cb251ec0d568de6a929b520c4aed8d1,2=1cb251ec0d568de6a929b520c4aed8d1,3=1cb251ec0d568de6a929b520c4aed8d1;
2829fc16ad8ca5a79da932f910afad1c$5f4dcc3b5aa765d61d8327deb882cf99$1=098f6bcd4621d373cade4e832627b4f6,2=098f6bcd4621d373cade4e832627b4f6,3=098f6bcd4621d373cade4e832627b4f6;
```

Figure 49: User details stored in the User Profile after hashing.

```
******* LOGIN - PAGE *******


Enter your username:
alishan1
UserId not exists. Do you want to try again? (yes/no)
```

Figure 50: Shows login failure when userId not exists.

```
******* LOGIN - PAGE *******


Enter your username:
alishan
Enter your password:
asdasd
Incorrect password. Do you want to try again? (yes/no)
```

Figure 51: Shows login failure when the password doesnt match.

All the login details such as user Id, Password and answers to all security questions are stored in the hash version and separated using delimiters to easily convert into objects while loading into the buffer.

For login, whenever user enters correct user Id and password its hashed on runtime and matched with data that is loading into buffer from User Profile file, then a random security question appears from the three security questions. If the user is successfully logged in, then the main menu appears which can refer to figure 36.

```
******* LOGIN - PAGE *******

Enter your username:
User2
Enter your password:
Password2
What is your mother's maiden name?
Mother2
Login successful!
Welcome to the Database System
1. Write Queries
2. Export Data and Structure
3. ERD
4. Exit
```

Figure 52: Main menu after successfully logging inside the database system.

For hashing, we have used the MD5 algorithm which is implemented in the Encryption Algorithm class and this algorithm works with the concept of converting all the characters into bytes and then to hexadecimal string.

Meeting Log:

Table 1: All meeting logs for group 5

Date	Time	Attendees	Agenda	Meeting Type	Meeting Recording Link
24/05/2024	10:30 PM - 11:00 PM	Harshil, Kenil, Alishan	First Project discussion meet	Online	https://dalu-my.sharepoint.com/:v:/r/personal/hr767612_dal_ca/Documents/Recordings/Team%20Meet%20-%200-20240524_223251-Meeting%20Recording.mp4?csf=1&web=1&e=Wgh9Cu
16/06/2024	11:15 AM - 12:00 PM	Harshil, Kenil, Alishan	Discuss Module 1 and allocate other work	Online	https://dalu-my.sharepoint.com/:v:/g/personal/al459703_dal_ca/EXS33xlYmadGk3MJcwsEFL8B_EaRbSSvyttrujDGmloZDw?referrer=Teams.TEAMS-ELECTRON&referrerScenario=MeetingChicletExpiration.view.view
25/06/2024	02:00 PM - 02:30 PM	Harshil, Kenil, Alishan	Functional Testing Sprint 1	Online	https://dalu-my.sharepoint.com/personal/kn486501_dal_ca/_layouts/15/stream.aspx?id=%2Fpersonal%2Fkn486501%5Fdal%5Fca%2FDocuments%2FRecordings%2FTeam%20Meet%20%2D%200%2D20240625%5F140625%2DMeeting%20Recording%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2E6b0c92a4%2Ddd10%2D46a3%2D9a7d%2D3dcdb9141675&ga=1
03/07/2024	2:45 – 3:30	Harshil, Kenil, Alishan	Module wise task allocation for sprint 2	Online	https://dalu-my.sharepoint.com/personal/al459703_dal_ca/_layouts/15/stream.aspx?id=%2Fpersonal%2Fal459703%5Fdal%5Fca%2FDocuments%2FRecordings%2FTeam%20Meet%20%2D%200%2D20240703%5F144434%2DMeeting%20Recording%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2E25c145f7%2Dd121%2D4ea2%2D9571%2D348702fde593

Note: We surely arranged more than 4 meetings throughout the project, but we have only recorded 4 sessions in the table. If required, we can show screenshots of the team call and meeting logs.

References:

[1] GeeksforGeeks, *"MD5 Hash in Java,"* GeeksforGeeks, 2019. [Online]. Available: https://www.geeksforgeeks.org/md5-hash-in-java/. [Accessed: 29-Jun-2024].