CSCI 5408
Data Management, Warehousing Analytics

Project – Sprint 01 – Report

Submitted to

Dr. Saurabh Dey

Department of Computer Science

Dalhousie University.

Submitted by Group 5

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

Contents

Overview 3
Background Research 3
Architecture Diagram: 4
Pseudo code: 5
Test Cases and evidence of testing: 9
Module 1 & 2: 15
Module 7: 27
Gitlab Repository Link: 30
Sprint 1 Meeting Logs: 30
References: 31

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 LinkedList 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:

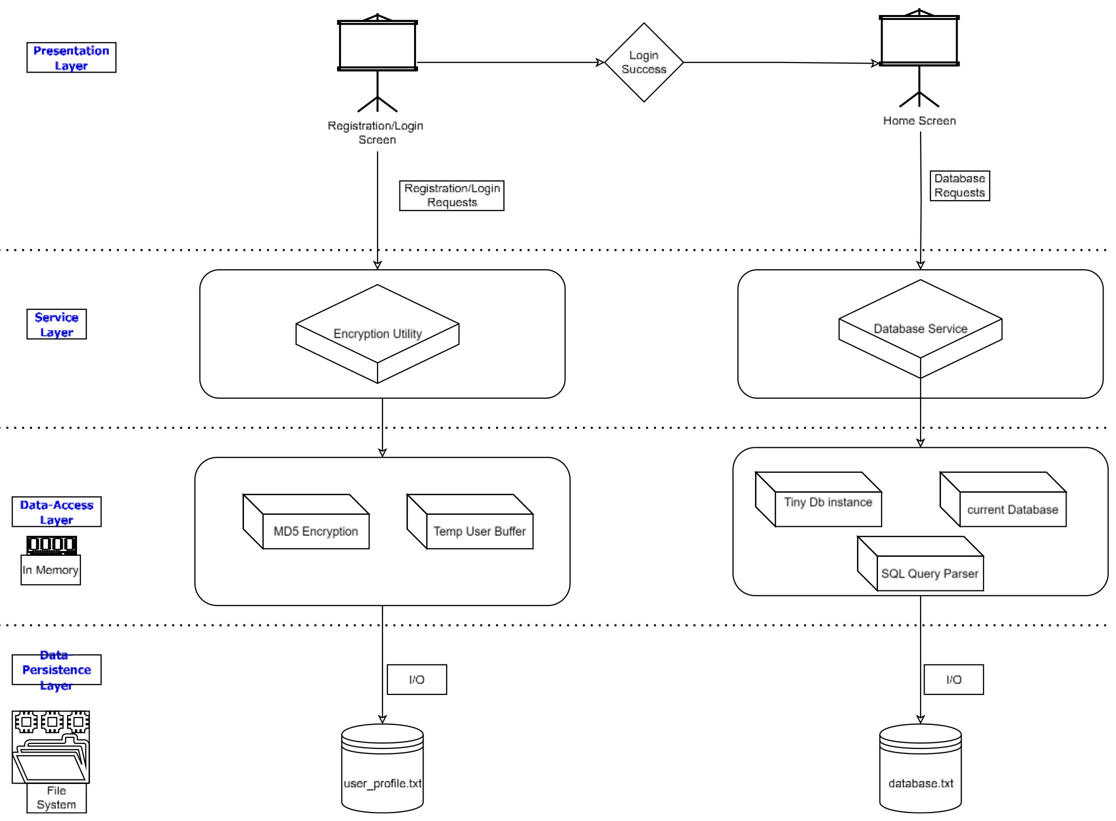


Figure1: architectural diagram of the application.

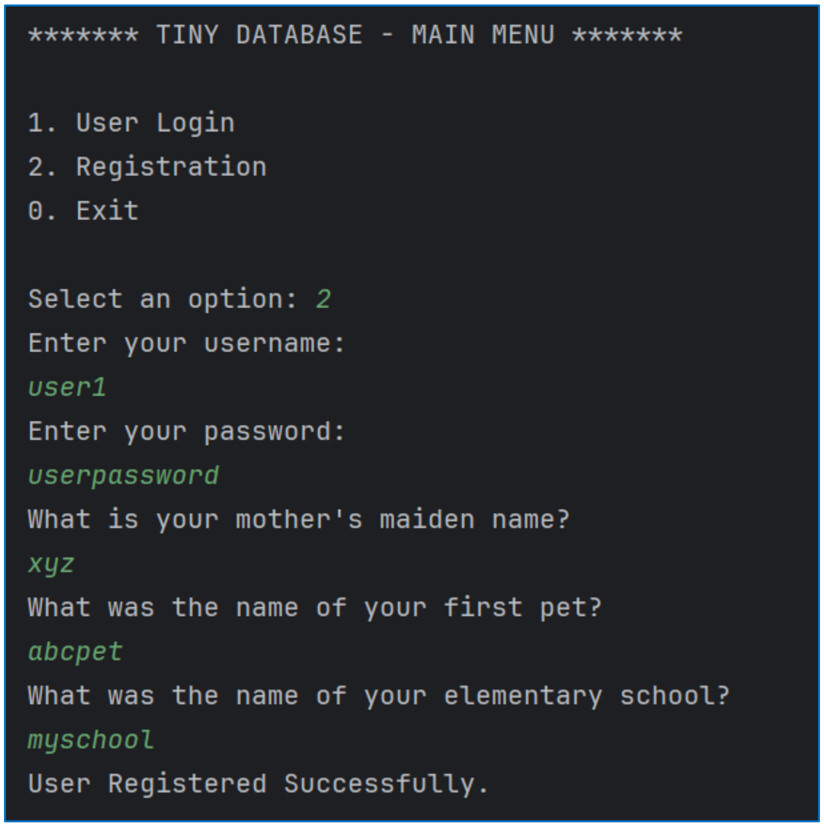
Pseudo code:

- DatabaseService:-

```
// Function to initialize DatabaseService
function initializeDatabaseService():
    tinyDb = new TinyDb()
    loadDatabaseFromFile()
    sqlQueryParser = new SQLQueryParser()
    currentDatabase = null

// Function to load database from file
function loadDatabaseFromFile():
    try:
        tinyDb.loadFromFile("database.txt")
    catch IOException:
        print "The database configurations don't exist."

// Function to save database to file
function saveToFile():
    try:
        tinyDb.saveToFile("database.txt")
    catch IOException:
        throw new RuntimeException(e)

// Function to process queries
function processQueries(query):
    object = sqlQueryParser.parse(query)
    if object is instance of UseDatabaseQuery:
        databaseName = object.getDatabaseName()
        currentDatabase = tinyDb.getDatabaseByName(databaseName)
    else if object is instance of CreateDatabaseQuery:
        databaseToCreate = object.getDatabaseName()
        tinyDb.addNewDatabase(databaseToCreate)
    else if object is instance of CreateTableQuery:
        if currentDatabase is null:
            print "No Database is Selected"
        else:
            currentDatabase.addTable(
                object.getTableName(),
                object.getColumnNames(),
                object.getColumnDataTypes(),
                object.getColumnSizes(),
                object.getPrimaryKeyColumn()
            )
    else if object is instance of InsertQuery:
        if currentDatabase is null:
            print "No Database is Selected"
        else:
            result = currentDatabase.insertData(
                object.getTableName(),
                object.getColumnNames(),
                object.getValues()
            )
            if result == "-1":
                print "Invalid entry in the insert query."
            else:
                print "Successfully added entry with Id:" + result
    else if object is instance of SelectQuery:
        if currentDatabase is null:
            print "No Database is Selected"
        else:
            result = currentDatabase.selectData(
                object.getTableName(),
                object.getColumns(),
                object.getConditionColumn(),
                object.getConditionValue()
            )
            for row in result:
                print row.getDataValue()
    else if object is instance of UpdateQuery:
        if currentDatabase is null:
            print "No Database is Selected"
        else:
            result = currentDatabase.updateData(
                object.getTableName(),
                object.getSetColumn(),
                object.getSetValue(),
                object.getConditionColumn(),
                object.getConditionValue()
            )
            print "Successfully updated the entry:" + result
    else if object is instance of DeleteQuery:
        if currentDatabase is null:
            print "No Database is Selected"
        else:
            result = currentDatabase.deleteData(
                object.getTableName(),
                object.getConditionColumn(),
                object.getConditionValue()
            )
    else if object is instance of DropTableQuery:
        if currentDatabase is null:
            print "No Database is Selected"
        else:
            result = currentDatabase.dropTable(object.getTableName())
            if result == "1":
                print "Successfully dropped the table with name:" +
object.getTableName()
            else if result == "-1":
                print "Table does not exist"
            else:
                print "Something went wrong!"
    else:
        print "Invalid Query"
        tinyDb.saveToFile("database.txt")
```

Test Cases and evidence of testing:

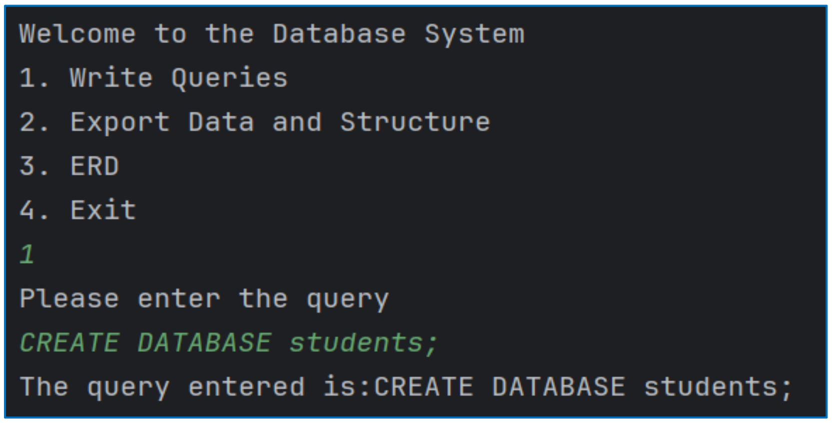
- User Registration: -



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

1. User Login
2. Registration
0. Exit

Select an option: 2
Enter your username:
user1
Enter your password:
userpassword
What is your mother's maiden name?
xyz
What was the name of your first pet?
abcpet
What was the name of your elementary school?
myschool
User Registered Successfully.
```

Figure 2: user registration screen.

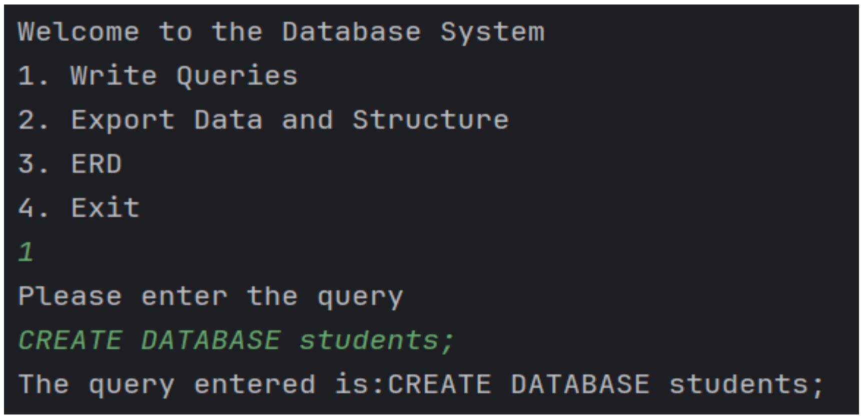
- User Login: -



```
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 3: user login screen.

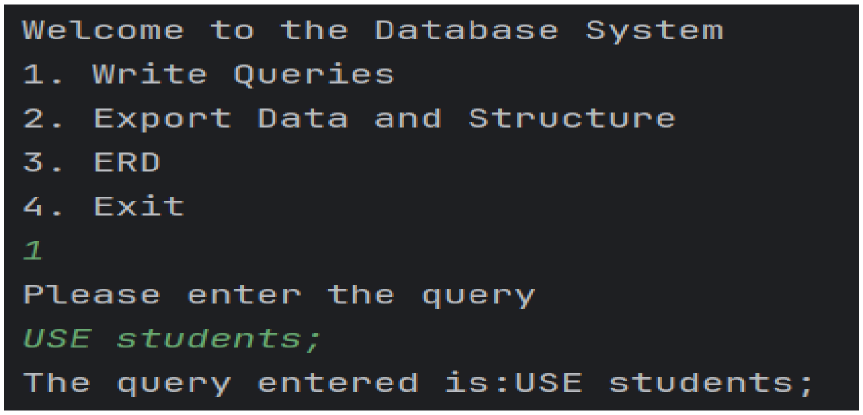
- Creating the Database: -



```
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 4: database creation.

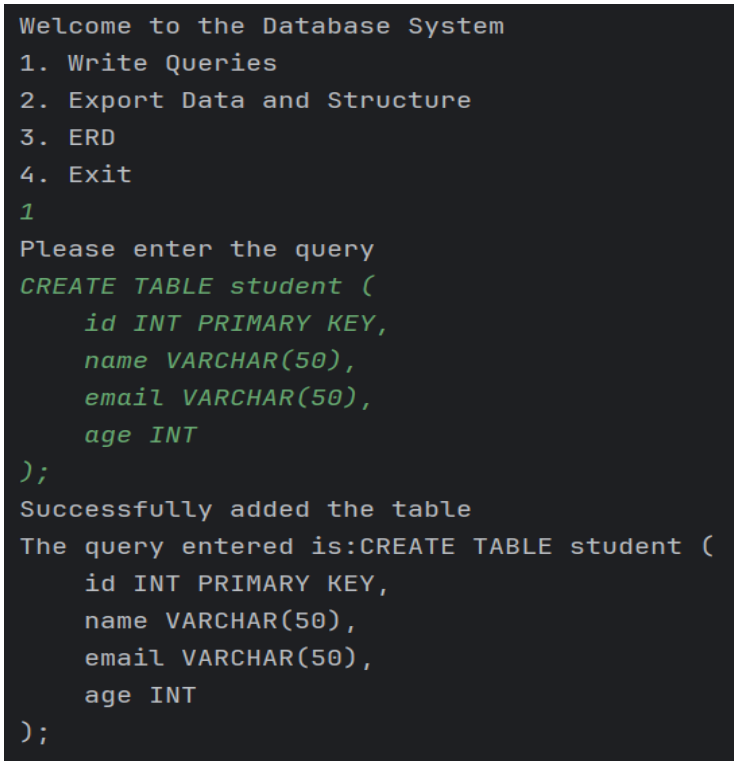
- Using the created Database: -



```
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 5: use database query execution.

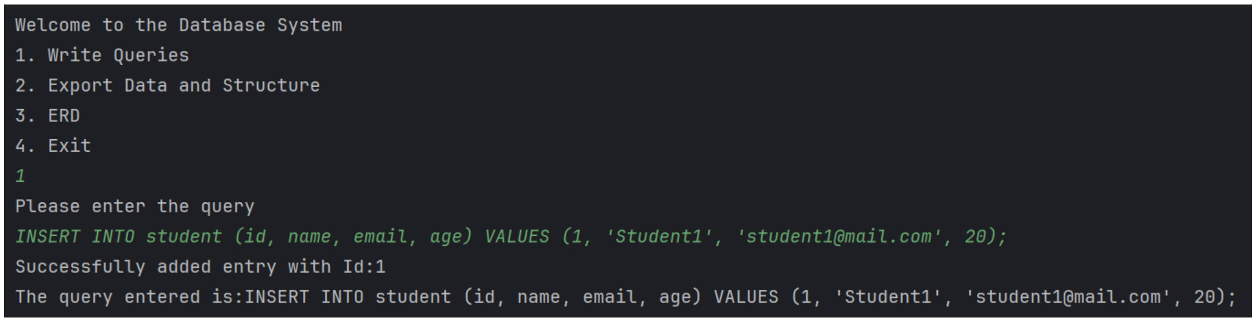
- Creating the table in the selected database: -



```
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 6: create table query execution.

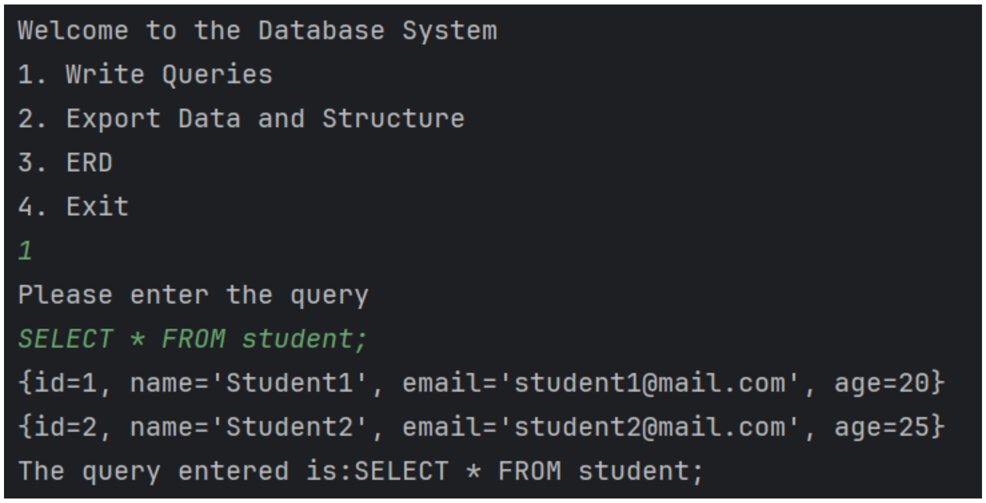
- Inserting the data in the Table: -



```
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 7: insert data into the table query execution.

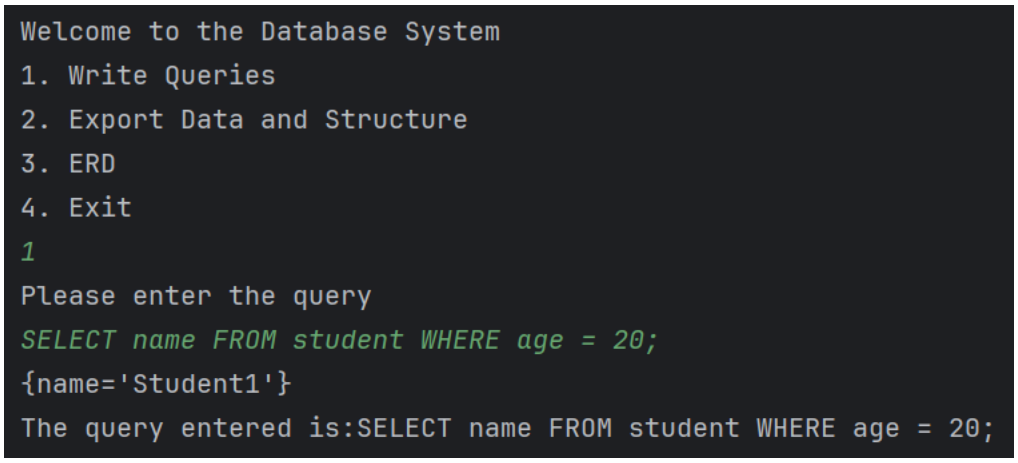
- Selecting everything from the table: -



```
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 8: selecting all the data from the table query execution

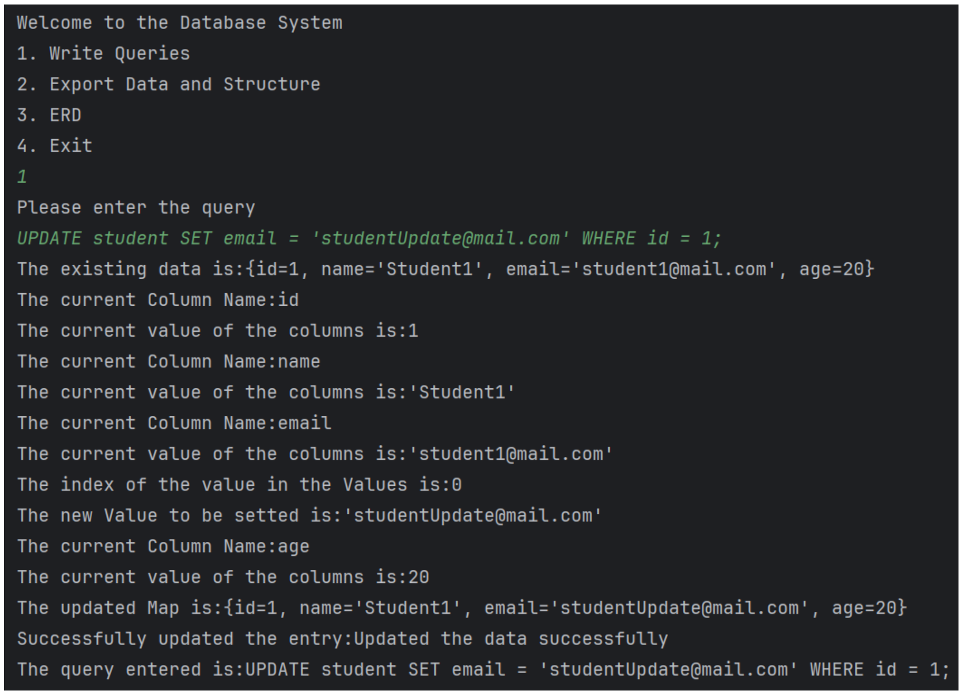
- Selecting specific data from the table: -



```
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 9: selecting the specific data from the table query execution

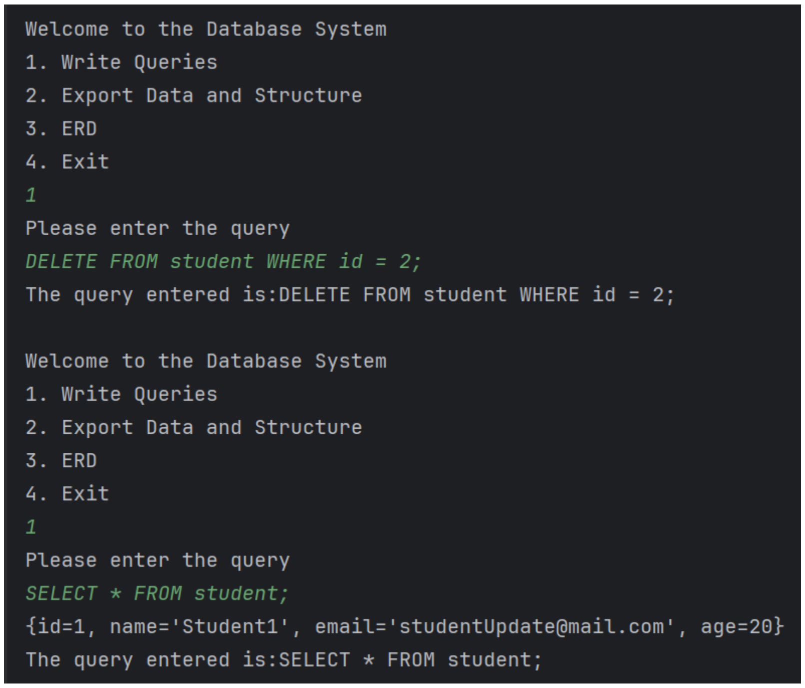
- Updating the data in the table: -



```
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 10: updating the data in the table query execution

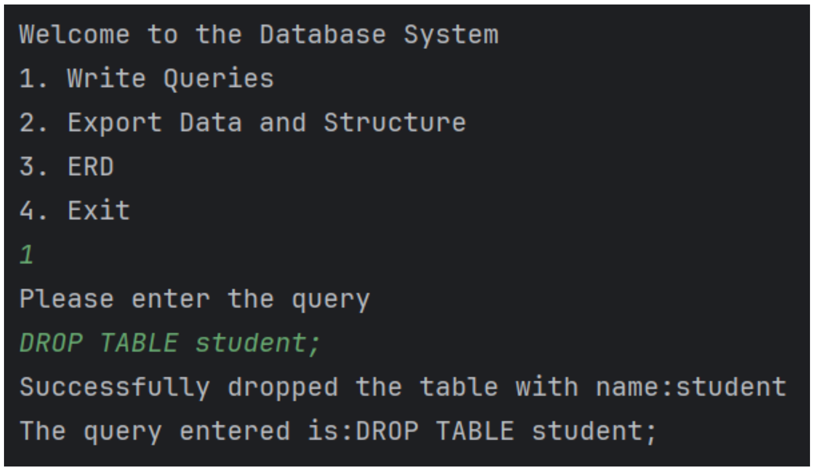
- Deleting the data from the table: -



```
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 11: 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 12: dropping the table query execution

Module 1 & 2:

This module has two main components as below: -

1. SQLQueryParser
2. TinyDb database instance

Explanation of Methods in SQLQueryParser

parseCreateDatabaseQuery

- This method parses a SQL CREATE DATABASE query and extracts the database name.
- It uses a regular expression to match the CREATE DATABASE statement followed by the database name.
- If a match is found, it extracts the database name and returns a CreateDatabaseQuery object.
- If no match is found, it returns null.

parseUseDatabaseQuery

- This method parses a SQL USE database query and extracts the database name.
- It uses a regular expression to match the USE statement followed by the database name.
- If a match is found, it extracts the database name and returns a UseDatabaseQuery object.
- If no match is found, it returns null.

parseCreateTableQuery

- This method parses a SQL CREATE TABLE query and extracts the table name and column definitions.
- It uses a regular expression to match the CREATE TABLE statement, the table name and the column definitions.
- If a match is found, it processes the column definitions to extract column names, data types, sizes, and the primary key column.
- It returns a CreateTableQuery object with the extracted information.
- If no match is found, it returns null.

parseInsertQuery

- This method parses a SQL INSERT INTO table query and extracts the table name, column names, and values.
- It uses a regular expression to match the INSERT INTO statement, the table name, column names, and values.
- If a match is found, it splits the column names and values into lists.
- It returns an InsertQuery object with the extracted information.
- If no match is found, it returns null.

parseSelectQuery

- This method parses a SQL SELECT from table query and extracts the table name, columns, and an optional WHERE condition.
- It uses a regular expression to match the SELECT statement, column names, table name, and an optional WHERE condition.
- If a match is found, it processes the columns and the WHERE condition if present.
- It returns a SelectQuery object with the extracted information.
- If no match is found, it returns null.

parseUpdateQuery

- This method parses a SQL UPDATE query and extracts the table name, SET clauses and WHERE condition.
- It uses a regular expression to match the UPDATE statement, the table name, SET clauses and WHERE condition.
- If a match is found, it processes the SET clauses and the WHERE condition.
- It returns an UpdateQuery object with the extracted information.
- If no match is found, it returns null.

parseDeleteQuery

- This method parses a SQL DELETE FROM query and extracts the table name and an optional WHERE condition.
- It uses a regular expression to match the DELETE FROM statement, the table name, and an optional WHERE condition.
- If a match is found, it processes the WHERE condition if present.
- It returns a DeleteQuery object with the extracted information.
- If no match is found, it returns null.

parseDropTableQuery

- This method parses a SQL DROP TABLE query and extracts the table name.
- It uses a regular expression to match the DROP TABLE statement followed by the table name.
- If a match is found, it extracts the table name and returns a DropTableQuery object.
- If no match is found, it returns null.

Explanation of Query Classes:-

CreateDatabaseQuery

Fields: databaseName stores the name of the database to be created.

Constructor: Initializes the databaseName field.

Getter: getDatabaseName() returns the database name.

UseDatabaseQuery

Fields: databaseName stores the name of the database to be used.

Constructor: Initializes the databaseName field.

Getter: getDatabaseName() returns the database name.

CreateTableQuery

Fields:

- tableName: Name of the table.
- columnNames: List of column names.
- columnDataTypes: List of data types for each column.
- columnSizes: List of sizes for each column.
- primaryKeyColumn: The primary key column.

Constructor: Initializes all fields.

Getter: Provide access to all fields.

InsertQuery

Fields:

- tableName: Name of the table.
- columnNames: List of column names.
- values: List of values to insert.

Constructor: Initializes all fields.

Getter: Provide access to all fields.

SelectQuery

Fields:

- tableName: Name of the table.
- columns: List of columns to select (null for all columns).
- conditionColumn: Column used in the WHERE clause.
- conditionOperator: Operator used in the WHERE clause.
- conditionValue: Value used in the WHERE clause.

Constructor: Initializes all fields.

Getter: Provide access to all fields.

UpdateQuery

Fields:

- tableName: Name of the table.
- setColumn: List of columns to update.
- setValue: List of values to set.
- conditionColumn: Column used in the WHERE clause.
- conditionOperator: Operator used in the WHERE clause.
- conditionValue: Value used in the WHERE clause.

Constructor: Initializes all fields.

Getter: Provide access to all fields.

DeleteQuery

Fields:

- tableName: Name of the table.
- conditionColumn: Column used in the WHERE clause.
- conditionValue: Value used in the WHERE clause.
- conditionOperator: Operator used in the WHERE clause.

Constructor: Initializes all fields.

Getter: Provide access to all fields.

DropTableQuery

Fields: tableName stores the name of the table to be dropped.

Constructor: Initializes the tableName field.

Getter: getTableName() returns the table name.

Explanation of main Database Entities:-

Class: TinyDb

Class Overview:

The TinyDb class represents a lightweight, in-memory database management system. It provides functionalities to create, retrieve, and delete databases, as well as to serialize and deserialize the entire database system to and from a file.

Attributes:

- List<Database> databases: A list of Database objects managed by this TinyDb instance.

Constructors:

- TinyDb(): Initializes a new TinyDb instance with an empty list of databases.

Methods:

Database Management:

- Database getDatabaseByName(String databaseName): Retrieves a database by its name. Returns the Database object if found, otherwise returns null.
- String addNewDatabase(String databaseName): Adds a new database with the specified name. If the database already exists, it does nothing. Returns a success message.
- void dropDatabase(String databaseName): Removes a database with the specified name from the list of databases. Prints a success message upon successful removal.

File Operations:

- void saveToFile(String filename) throws IOException: Saves the serialized form of the TinyDb instance to a file with the specified filename.
- void loadFromFile(String filename) throws IOException: Loads the serialized form of the TinyDb instance from a file with the specified filename.

Serialization and Deserialization:

- String serialize(): Serializes the TinyDb instance into a string format. The serialized string includes all databases, tables, columns, and rows.
- void deserialize(String str): Deserializes the provided string to reconstruct the TinyDb instance, including its databases, tables, columns, and rows.

Usage:

- The TinyDb class is used to manage multiple databases in memory, providing functionalities to add, retrieve, and delete databases. It also supports saving the entire state to a file and loading it back from a file, making it suitable for lightweight persistence and transfer of database states.

Class: Database

Class Overview:

The Database class is designed to represent a database, holding multiple tables and providing various methods to manage and manipulate these tables and their data. It allows the creation, deletion, and manipulation of tables within the database, as well as the insertion, selection, updating, and deletion of data from these tables.

Attributes:

- String databaseName: Stores the name of the database.
- List<Table> tables: Holds a list of Table objects representing the tables within the database.

Constructors:

- Database (String databaseName): Initializes a new Database instance with the specified databaseName and an empty list of tables.

Methods:

Database Management:

- String getDatabaseName(): Returns the name of the database.
- void addTable(String tableName, List<String> columnNames, List<String> columnDataTypes, List<Integer> columnSizes, String primaryKeyColumnName): Adds a new table to the database with the specified columns and primary key. Does nothing if the table already exists.
- String dropTable(String tableName): Removes the table with the specified name from the database. Returns "1" if successful, "-1" if the table does not exist.

Data Manipulation:

- String insertData(String tableName, List<String> columnName, List<String> values): Inserts data into the specified table. Returns the result of the operation.
- List<Row> selectData(String tableName, List<String> columnNames, String columnName, String value): Retrieves data from the specified table based on a condition. Returns the selected rows or null if the table does not exist.
- String updateData(String tableName, List<String> columnNames, List<String> values, String whereColumn, String whereValue): Updates data in the specified table based on a condition. Returns the result of the operation.
- String deleteData(String tableName, String columnName, String value): Deletes data from the specified table based on a condition. Returns the result of the operation.

Serialization and Deserialization:

- String serialize (String indent): Serializes the database and its tables into a string format with the specified indentation.
- void deserialize (String str): Deserializes the provided string to reconstruct the database and its tables.

Usage:

- The Database class is a central component for managing tables and their data within a database. It allows for easy addition, removal, and modification of tables and their rows, and it supports serialization for persistent storage or transfer of the database state.

Class: Table

Class Overview:

The Table class is designed to represent a table within a database, including its columns and data rows. It provides methods for managing table structure and data, including adding columns, inserting, selecting, updating, and deleting data, as well as serialization and deserialization operations.

Attributes:

- String tableName: Stores the name of the table.
- List<Column> columns: Holds a list of Column objects representing the columns of the table.
- List<Row> data: Holds a list of Row objects representing the data rows in the table.
- String primaryKeyColumnName: Stores the name of the primary key column.
- static int id: A static field to generate unique IDs for rows.

Constructors:

- Table (String tableName): Initializes a new Table instance with the specified tableName, empty lists for columns and data, and resets the primary key column name and row ID.

Methods:

Table Management:

- String getTableName(): Returns the name of the table.
- void setTableName(String tableName): Sets the name of the table.
- List<Column> getColumns(): Returns the list of columns.
- void setColumns(List<Column> columns): Sets the list of columns.
- List<Row> getData(): Returns the list of data rows.
- void setData(List<Row> data): Sets the list of data rows.
- static int getId(): Returns the current row ID.
- static void setId(int id): Sets the row ID.

Column Management:

- void addColumn(String columnName, String columnDataType, int dataTypeSize, String primaryKeyColumnName): Adds a new column to the table with the specified name, data type, and size. Sets the primary key column name.

Data Manipulation:

- String insertData(List<String> columnNames, List<String> values): Inserts data into the table. Validates data types before insertion. Returns the result of the operation.
- List<Row> getDataFromTable(List<String> columnNames, String columnName, String value): Retrieves data from the table based on a condition. Returns the selected rows.
- String updateData(List<String> columnNames, List<String> values, String whereColumn, String whereValue): Updates data in the table based on a condition. Returns the result of the operation.
- String deleteData(String whereColumn, String whereValue): Deletes data from the table based on a condition. Returns the result of the operation.

Serialization and Deserialization:

- String serialize (String indent): Serializes the table, its columns, and its data into a string format with the specified indentation.
- void deserialize (String str): Deserializes the provided string to reconstruct the table, its columns, and its data.

Private Methods:

- boolean validateDataTypes(List<Column> columnsInDatabase, List<String> columnNames, List<String> values): Validates the data types of the values to be inserted. Returns true if all values are valid, false otherwise.
- List<Row> filterData(List<String> columnNames, List<Row> rows): Filters the data rows to include only the specified columns.

Usage:

The Table class is a central component for managing the structure and data of a table within a database. It allows for the dynamic addition of columns, validation, and manipulation of data rows, and supports serialization for persistent storage or transfer of the table state.

Class: Column

Class Overview:

The Column class represents a column within a database table. It includes the column name, data type, and size. The class provides methods for serializing and deserializing column data, allowing it to be easily stored and reconstructed.

Attributes:

- String columnName: Stores the name of the column.
- String columnDataType: Stores the data type of the column (e.g., INT, VARCHAR).
- int dataTypeSize: Stores the size of the data type.

Constructors:

- Column (String columnName, String columnDataType, int dataTypeSize): Initializes a new Column instance with the specified columnName, columnDataType, and dataTypeSize.

Methods:

Getters and Setters:

- String getColumnName(): Returns the name of the column.
- void setColumnName(String columnName): Sets the name of the column.
- String getColumnDataType(): Returns the data type of the column.

Serialization and Deserialization:

- String serialize (String indent): Serializes the column into a string format with the specified indentation. The serialized string includes the column name, data type, and size.
- void deserialize (String str): Deserializes the provided string to reconstruct the column's attributes. The string should follow the format used by the serialize method.

Usage:

The Column class is used to define the structure of a column within a database table. It supports basic operations such as setting and getting column attributes, as well as serialization for persistent storage or transfer of the column state.

Class: Row

<u>Class Overview:</u>

The Row class represents a single row within a database table. It includes methods for adding, updating, and comparing row data, as well as serializing and deserializing row data for persistent storage or transfer.

<u>Attributes:</u>

Map<String, String> dataValue: Stores the data of the row, where the key is the column name, and the value is the data for that column.

<u>Constructors:</u>

- Row (): Initializes a new Row instance with an empty dataValue map.

<u>Methods:</u>

Getters and Setters:

- void setDataValue(Map<String, String> dataValue): Sets the data of the row.
- Map<String, String> getDataValue(): Returns the data of the row.

Data Manipulation:

- String addDataRow(List<String> columnNames, List<String> values, String primaryKey, int currentIdCounter): Adds data to the row. It takes a list of column names, a list of values, a primary key, and the current ID counter. If the primary key value is not provided, it assigns a new ID from the current ID counter. Returns the primary key data.
- String updateDataRow(List<String> columnNames, List<String> values): Updates the data of the row based on the provided column names and values. Returns a success message upon successful update.
- boolean equals(Row rowToCompare): Compares the current row with another row. Returns true if the rows are equal, otherwise false.

Serialization and Deserialization:

- String serialize (String indent): Serializes the row into a string format with the specified indentation. The serialized string includes all column names and their respective values.
- void deserialize (String str): Deserializes the provided string to reconstruct the row's data. The string should follow the format used by the serialize method.

Usage:

- The Row class is used to manage and manipulate the data of a single row in a database table. It supports adding, updating, and comparing row data, as well as serialization for persistent storage or transfer.

Module 7:

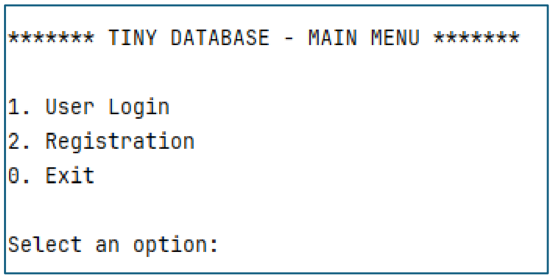
This module focuses on the user authentication part and to complete its implementation this module was divided into following tasks:

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.

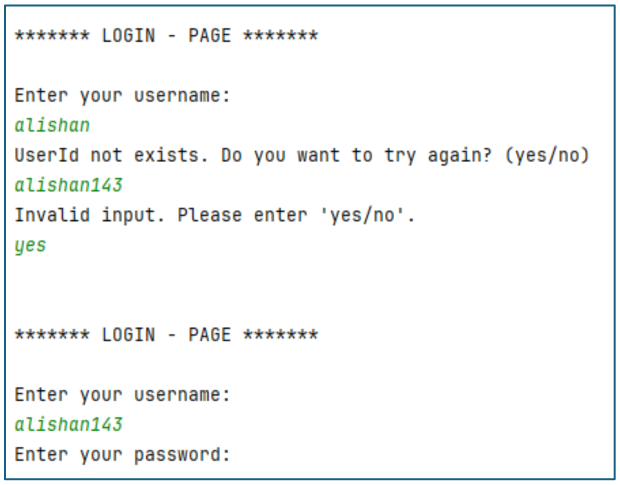
The other approach would be to use List of User type objects but that would compare all objects until a match was found. This could be easy in terms of implementation but could have increased the space complexity.

Then we implemented the presentation layer for the user for registration and login. Which could be referred in the figure 14 and figure 15.



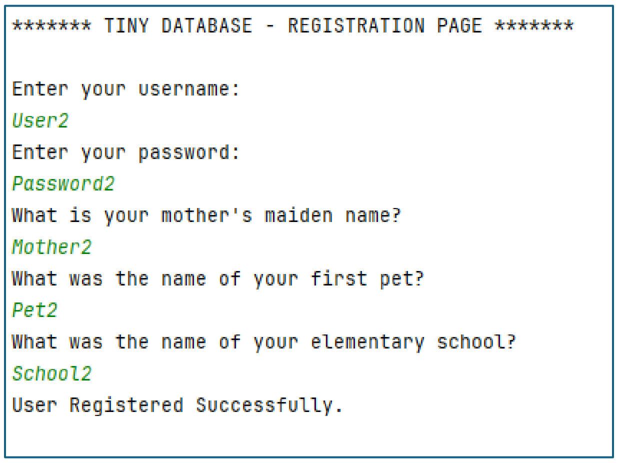
```
******* TINY DATABASE - MAIN MENU *******

1. User Login
2. Registration
0. Exit

Select an option:
```

Figure 13: 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:
```

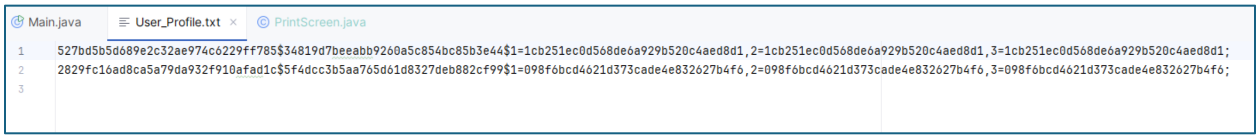
Figure 14: Illustrating the login page screen.



```
******* 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 15: Illustrating the registration page.

Whenever the user gets registered a record has been entered into the User Profile which can be referred to figure 16.

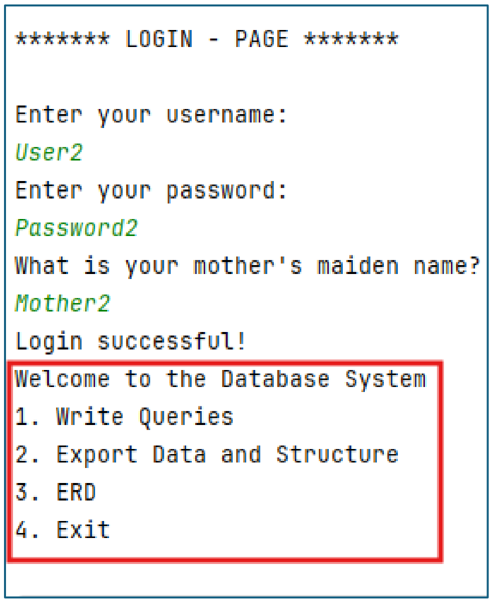


```
527bd5b5d689e2c32ae974c6229ff785$34819d7beeabb9260a5c854bc85b3e44$1=1cb251ec0d568de6a929b520c4aed8d1,2=1cb251ec0d568de6a929b520c4aed8d1,3=1cb251ec0d568de6a929b520c4aed8d1;
2829fc16ad8ca5a79da932f910afad1c$5f4dcc3b5aa765d61d8327deb882cf99$1=098f6bcd4621d373cade4e832627b4f6,2=098f6bcd4621d373cade4e832627b4f6,3=098f6bcd4621d373cade4e832627b4f6;
```

Figure 16: User details stored in the User Profile after hashing.

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 17.



```
******* 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 17: 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.

Gitlab Repository Link:

https://git.cs.dal.ca/alishan/csci_5408_s24_group5

Sprint 1 Meeting Logs:

Table 1: Shows sprint1 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

References:

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