Data Management, Warehousing, and Analytics

Assignment 1

Submitted by –

Name - Alen Santosh John

Banner ID – B00930528

Email – al283652@dal.ca

Problem 1 -

Step 1: Visiting www.dal.ca and the subdomains.

Upon visiting the website and the subdomains. I was able to create multiple entities as listed in step 2.

Step 2: Entities are as follows -

The following entities were generated -

1. Students - {Name (First name, Middle name, Last Name), Student national id, Age (Derived attribute), Date of birth, Program, Student id, contact, faculty name, enrollment status }

   The "Students" entity encompasses a range of details about students, including their name, national ID number, age, date of birth, program, student ID, contact information, faculty name, and enrollment status. By utilizing the myDal platform, users can log in as a student and access this information. As a strong entity, the "Students" entity possesses a primary key that enables each row to be uniquely identified.

2. User - {Name, Email, User id}

   The “Users” entity stores the information of the user before logging in, the user can log in as faculty staff, or student and can access resources. This is a strong entity that does not depend on the other entities.

3. Program - {Faculty ID, email, program length, Department, campus, Degree type, Program Id}

   The “Program” entity stores the details of the various programs that the university provides. This entity does not depend on any other entities to exist and hence is a strong Entity

4. Course - {Timings, Course Name, Room No, Credits, Course Id, Building Name}

   The “Course” entity stores the various courses present in a program. Since the entity can have records without being dependent on other entities it is a strong entity.

5. News - {News id, Description, Date, Headline}

   The “News” entity consists of various features of news and headlines that www.dal.ca showcases.

6. Building - { Name, Location, Building Id, Type of building }

   The “Building” entity stores information about the various buildings in the university. These buildings have multiple rooms where classes for different courses can be conducted. The buildings may also be a library or administrative block Since the entity can exist without being dependent on other entities this is a strong entity

7. Faculty Staff - {Email, Name (First Name, Middle name, last name), Date of Birth, Faculty Id, qualifications, Years, Age(Derived attribute)}

   The entity “Faculty Staff”, contains the details of all the faculty belonging to the university.

8. Faculty - { Faculty name, Id }

   The entity “Faculty” stores information about the various faculties present in the university for example - The faculty of graduate studies, the Faculty of engineering etc.

9. Resources - {Phone, Name, Description, Email, Resource Id}

   The entity “Resources” stores the various resources the university provides.

The following weak entities were created -

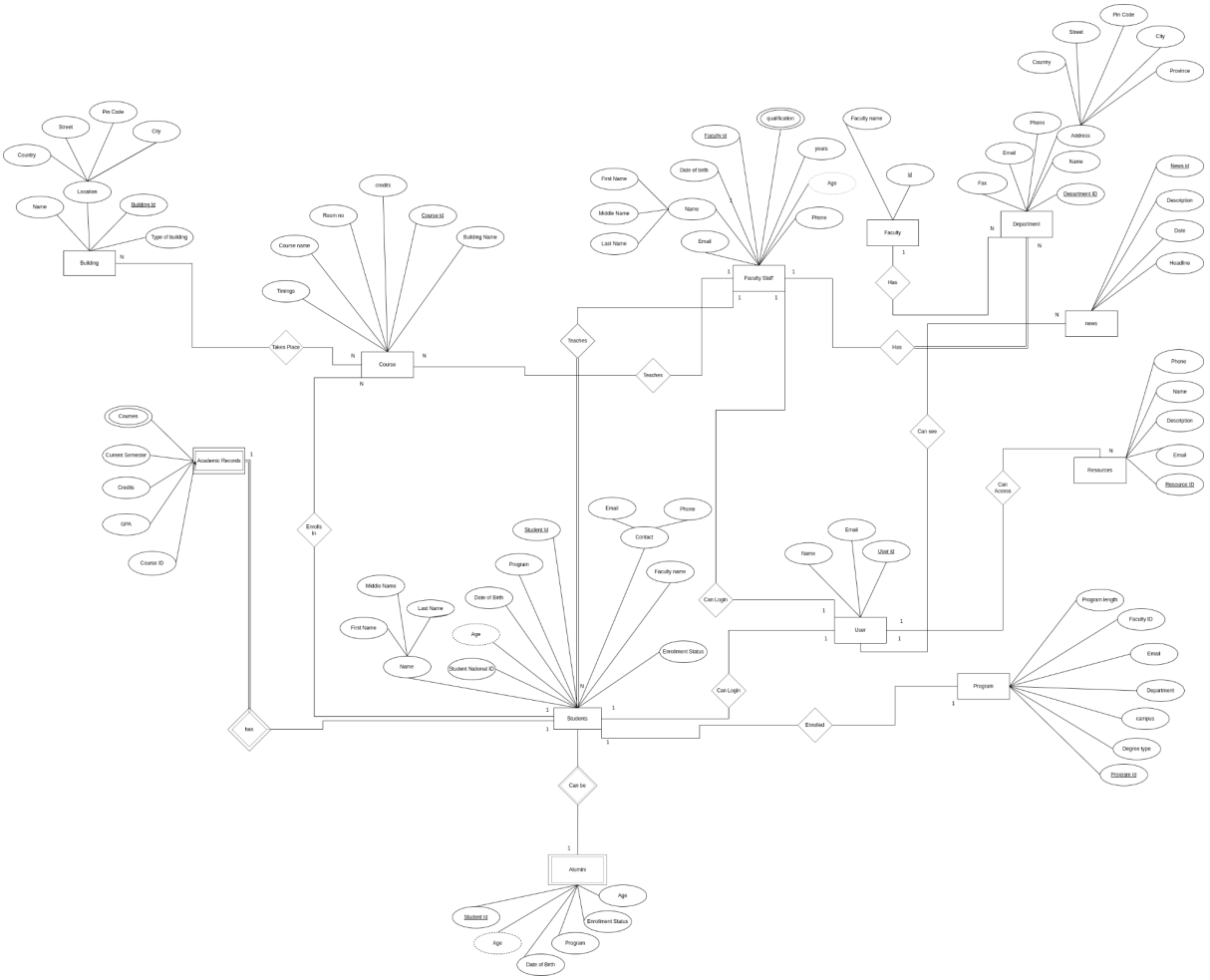
1. Academic records - {Courses, current semester, credits, Gpa}

   This entity contains the student's academic records. Since this entity cannot exist without the student entity this is a weak entity.

2. Alumni - {Student Id, Age, Date of birth, Program, Enrollment status, Age}

   This entity contains the details of the graduated students. Since this entity depends on the student's entity this is considered to be a weak entity.

Step 3:-



The Entity relation diagram was generated by referring www.dal.ca. The Entity relation diagram consists of nine strong entities and two weak entities.

The user entity is what and how the user actually visits the dal.ca website. The user can log in as a faculty or as a student. The user has **attributes** like the user's **email ID**, The **user ID**, through which we can understand if he is a faculty or a student and the **name** of the user. The userId is the primary key as it can access all the records uniquely. The user entity is connected by a **one-to-many** relation with the resource entity. This is because a single user can access multiple resources present through the website. The resource entity Contains the various resources that the university provides such as research Labs or career guidance services Hence the resource entity provides multiple resources wherein one user can access those multiple services.

The resource entity contains Attributes like a **phone** number which is the contact information of the resource the **name** of the resource the description of what the **resource** provides the contact **email** and the **resource ID** which is the primary key. The user can also view multiple news articles and headlines on the website. To enable this I have created the news entity Which consists of the **news ID** the **description** of

the news the **date** of public and the **headline**. A single user can view multiple news on the website. NewsID year is the primary key for the entity.

The user can also log in as a student so a particular user can only log in as a particular student hence the relation between the two entities is a one-to-one relationship. The student entity has various attributes like the **student's National id** card (eg - Passport), the **student's age,** which is a derived attribute from the **date of birth,** the student's current ongoing **program,** the student's previous programs, the **student ID,** the student's **contact** information, which is a **decomposable attribute** and has **email** and **phone** number as sub-attributes, the **Faculty name** and the **enrollment status.**

If the enrollment status of the student is false then the student may be alumni, this entity's existence solely depends on the student entity, Hence is termed as a weak entity the alumni entity has various attributes like the **student ID, age,** the date of birth, the program details and the enrollment status.

Each student entity may also have their own academic records. This entity also depends on the student entity to exist and hence is termed as a weak entity. The academic records entity has attributes like **courses, current semester, credits**, **GPA** and **course ID**. The course ID along with the current semester can use to identify unique records in the entity. this is the partial key of the entity. A single student can enroll in a single program hence the relation between the student entity and the program entity is one-to-one relation. The program entity has various attributes like the **program length,** The **Faculty ID,** the **email ID,** the **department,** the **campus,** the **degree type** and the **program ID.**

The users can also log in as faculty staff since a single user can only log in as a single faculty staff the relation is one-to-one between the two entities. The Faculty staff entity has attributes like **email ID, name,** which is a decomposable attribute, the date of birth, The Faculty ID, qualification which is a multivalued attribute, year of **experience, age** and **phone number**. The faculty entity has a one-to-many relation with the course entity, this is because a single professor can take various courses. The course entity has various attributes like timings, course timings, Course name, Room no, credits, course id and building name. The course entity has many to many relations with the building entity model. This is because many courses may be conducted inside a building and vice versa. The building entity has attributes like Name, Location, building ID, and type of building. The faculty entity stores the information about various faculties. These faculties have many departments. The relationship between the faculty and the department is one to many. The department has various attributes like fax, email, phone, Address, Name and Department ID.

The complete Er diagram is attached to the folder.

Step 4:-

Design traps occur when a relation between entities is wrongly interpreted. This may lead to poor database design and may also cause loss of data. My implementation of this database has no design traps.

Step 5: && Step 6:

Student Entity -

Name	Student National ID	Age	Date Of Birth	Program	Student ID	Contact	Faculty name	Enrollment Status

The functional dependencies are as follows -

1 . Student ID -> Program, Faculty name, Enrollment Status:
A student's identification number may be used to determine their program, the faculty or department associated with their program, and their enrollment status in that program.

2. Program -> Faculty name:
A program might be associated with a specific faculty or department within the university.

Normalization -

Breaking the table into 2 tables

Student Table -

Student ID	Student National ID	Name	Age	Date Of Birth	Program	Enrollment Status

Contact table -

Contact_id	Student_id	Email	Contact	Address

- Contact information is stored in a different table as it has a one-to-many relationship with student

Faculty table -

Faculty_id	Student_id	Faculty name

- Faculty details stored in a separate table as it has a one-to-many relationship with a student.

Program Table:

ProgramId	DegreeType	Campus	Department	Email	FacultyId	ProgramLength

1. Faculty -> Email: Each faculty has a unique email associated with it. This can be used to uniquely identify the row.
2. Program_id -> Degree type, Campus, department, program length
   The program_id uniquely identifies a program, and each program has a specific degree type, campus, department, and program length. Therefore, knowing the program_id determines these other attributes.

   I have normalized the table to remove the functional dependencies.

Normalization -

Program_table

Programid	Degree Type	Campus	Department	Program Length

Faculty Table

Faculty Id	Email

Resources Table:

Resource_id	Email	Description	Name	Phone

- No functional dependencies exists for this table.

News Table:

Headline	Date	description	NewsId

- No functional dependencies exists for this table.

Department

Fax	Email	Phone	Address	Name	DepartmentId

- No functional dependencies exists for this table.

Faculty Id

Email	Name	Date of birth	FacultyId	Qualification	Years	Age	Contact

- No functional dependencies exists for this table.

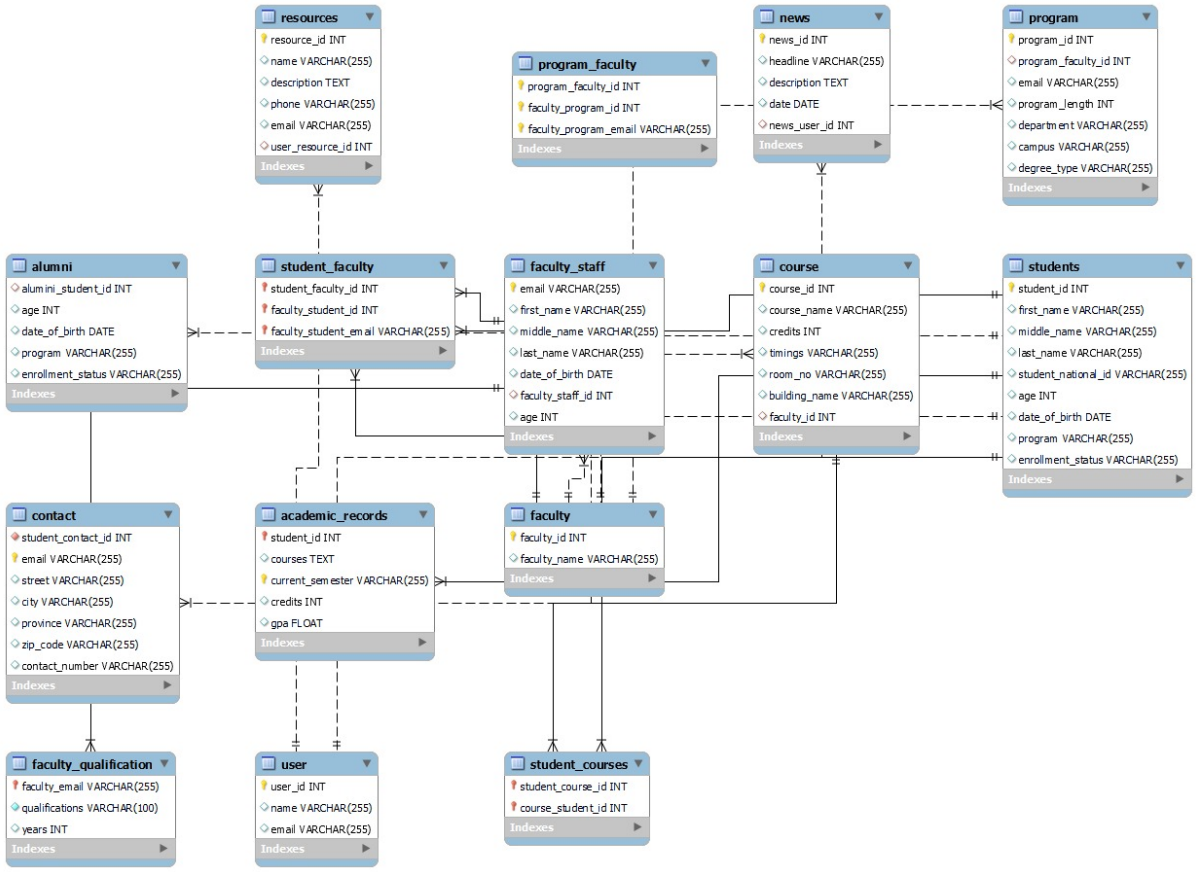
Faculty

Facultyid	Email	Name	Date of birth	Phone	Age

Faculty Qualification

Faculty ID	Qualification	Years

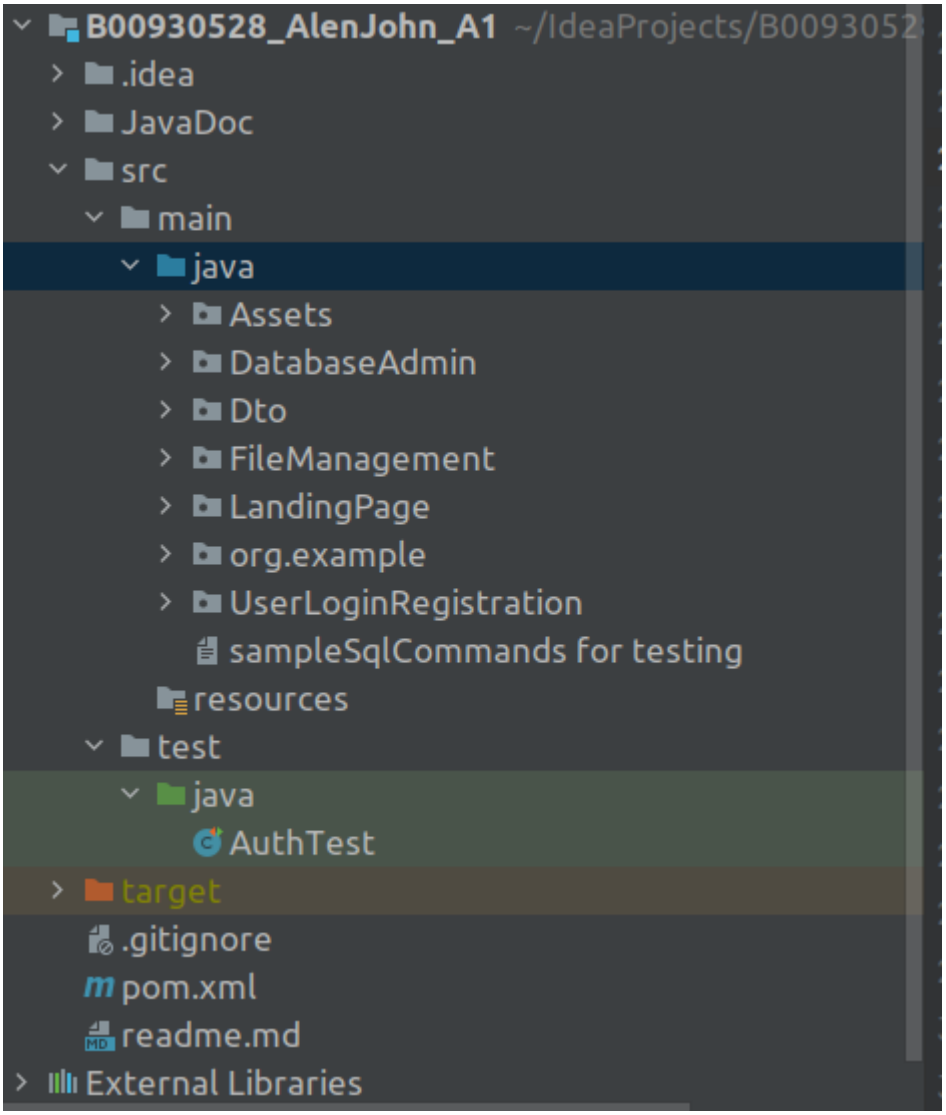
Step 7: The queries to generate the below physical model is attached a a .txt file to this project and the git lab repo.



Problem 2 -

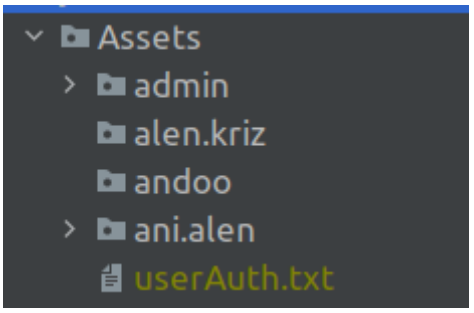
Lightweight DBMS -

Program Structure -



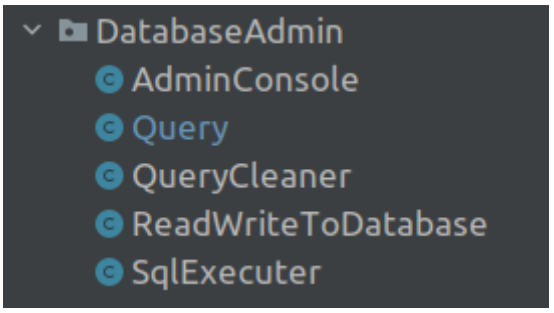
1. My implementation of the lightweight DBMS consists of 7 packages namely -
    a. Assets
    b. DatabaseAdmin
    c. Dto
    d. FileManagement
    e. LandingPage
    f. Org.example
    g. UserLoginRegistration

Assets -
The assets package stores all the generated files.



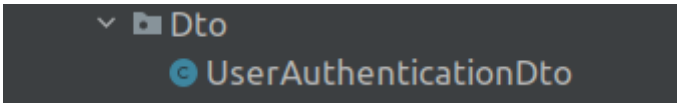
Here all the folders are databases that were created during the execution of the program (later explained)

DatabaseAdmin -
The Database admin package consists of the following -

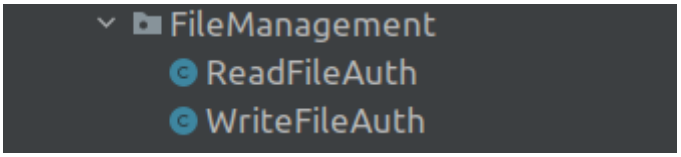


This package is accessed after the user successfully authenticates themself. This package is responsible to decompose the sql query extracting important information, and then processing the query to create/update/read/delete the data.

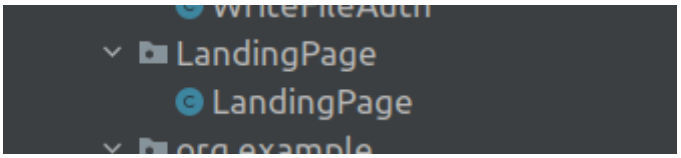
DTOs
This is the data transfer object class used for this implementation.



FileManagement
This package is responsible for the read and write operation of the files. (all data is stored with custom delimiters in .txt files)

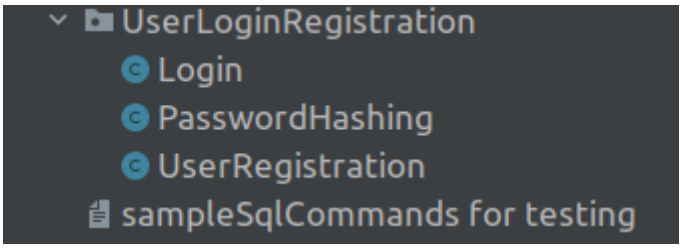


LandingPage
This package is used to interact with the user login registration module.



Org.Example
This renders the main.java

UserLoginRegistration
This package is responsible for login and user authentication.



Task 3 -

1. My implementation has followed the SOLID principles to ensure the best coding practices.

   Single responsible Principle - Each functionality in my implementation has followed this principle. My modules have a single responsibility and are loosely coupled with each other. For example - The user login registration only handles the strategies for the login and user authentication.

   Open/closed Principle - By using a loosely coupled and well-modular architecture. My implementation can easily be upgraded and modified by not modifying the existing classes and methods. This can be further improved by

   Liskov Substitution Principle: I have made use of this principle when decomposing the sql query. The query was decomposed and assigned to a method as a parameter. Depending on the keywords Select, update, Delete etc. the query was further processed. The query keyword was substituted to the params of the sub-type of the base class.

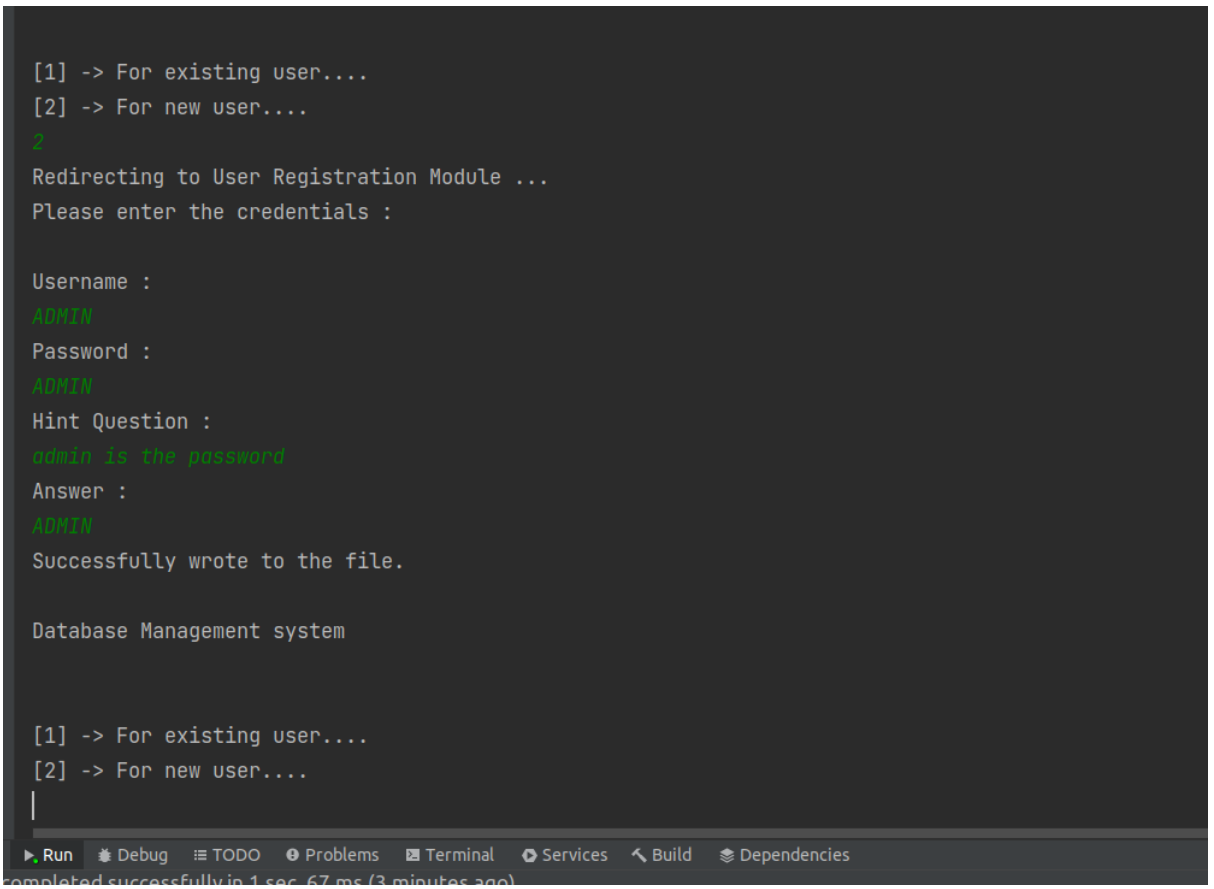
   Interface - Interface could be used to make the code even more robust, but it was out of the scope of this implementation.

Abstractions - My main classes in a package are not dependent on the small classes and their helper methods.
Task 4 -

My implementation is successfully allowing users to log in, register and create multiple databases by taking user input in sql format.

Detailed screenshots of the functioning are attached below

Task 5 -
Creating a new user -



New user has been created, now log in using that credentials -

Login Successful!

Task 6 -
Creating a database and then a table to store the user log in the .txt file

```
^^^^^^^^^^^^^^ CONSOLE LOGS ^^^^^^^^^^^^^^
.......... ALEN SQL Shell running ..........
this shell is whitespace sensitive ...
To create a new Database -> Eg : 'create database NAME_OF_DATABASE;'
To use a existing Database -> Eg : 'use database NAME_OF_DATABASE;'
To create a new table -> Eg : 'create table orders ( item_id int, orderDate varchar, nameOfOrder varchar, orderValue i
To read the table please use -> Eg : 'select * from orders;'
To insert into a table -> Eg : 'insert into orders values ( 1234, 23/oct/2023, Alen-order, $34000 );' NOTE - All the c
To update table -> Eg : 'update table orders ( 1234, 23/oct/2023, Alen-order, $34000 );'
To delete table -> Eg : 'delete table orders;'
.......... CHECK EXECUTION LOGS LOGGED ABOVE ..........
.......... Please execute query below ..........
Create database adminTestingReport;
Executing query ...
Database created!
^^^^^^^^^^^^^^ CONSOLE LOGS ^^^^^^^^^^^^^^
.......... ALEN SQL Shell running ..........
this shell is whitespace sensitive ...
To create a new Database -> Eg : 'create database NAME_OF_DATABASE;'
```

Now selecting that database,

```
To delete table -> Eg : 'delete table orders;'
.......... CHECK EXECUTION LOGS LOGGED ABOVE ..........
.......... Please execute query below ..........
use database adminTestingReport;
Executing query ...
Using Database
^^^^^^^^^^^^^^ CONSOLE LOGS ^^^^^^^^^^^^^^
.......... ALEN SQL Shell running ..........
this shell is whitespace sensitive ...
To create a new Database -> Eg : 'create database NAME_OF_DATABASE;'
To use a existing Database -> Eg : 'use database NAME_OF_DATABASE;'
To create a new table -> Eg : 'create table orders ( item_id int, orderDate varchar, nameOfOrder varchar, orderValue i
To read the table please use -> Eg : 'select * from orders;'
To insert into a table -> Eg : 'insert into orders values ( 1234, 23/oct/2023, Alen-order, $34000 );' NOTE - All the c
To update table -> Eg : 'update table orders ( 1234, 23/oct/2023, Alen-order, $34000 );'
To delete table -> Eg : 'delete table orders;'
.......... CHECK EXECUTION LOGS LOGGED ABOVE ..........
.......... Please execute query below ..........
```

Creating a table -

```
To delete table -> Eg : 'delete table orders;'
.......... CHECK EXECUTION LOGS LOGGED ABOVE ..........
.......... Please execute query below ..........
create table orders ( item_id int, orderDate varchar, nameOfOrder varchar, orderValue int );
Executing query ...
Table created successfully ...
^^^^^^^^^^^^^^ CONSOLE LOGS ^^^^^^^^^^^^^^
.......... ALEN SQL Shell running ..........
this shell is whitespace sensitive ...
To create a new Database -> Eg : 'create database NAME_OF_DATABASE;'
To use a existing Database -> Eg : 'use database NAME_OF_DATABASE;'
To create a new table -> Eg : 'create table orders ( item_id int, orderDate varchar, nameOfOrder varchar, orderValue i
To read the table please use -> Eg : 'select * from orders;'
To insert into a table -> Eg : 'insert into orders values ( 1234, 23/oct/2023, Alen-order, $34000 );' NOTE - All the c
To update table -> Eg : 'update table orders ( 1234, 23/oct/2023, Alen-order, $34000 );'
To delete table -> Eg : 'delete table orders;'
.......... CHECK EXECUTION LOGS LOGGED ABOVE ..........
.......... Please execute query below ..........
```

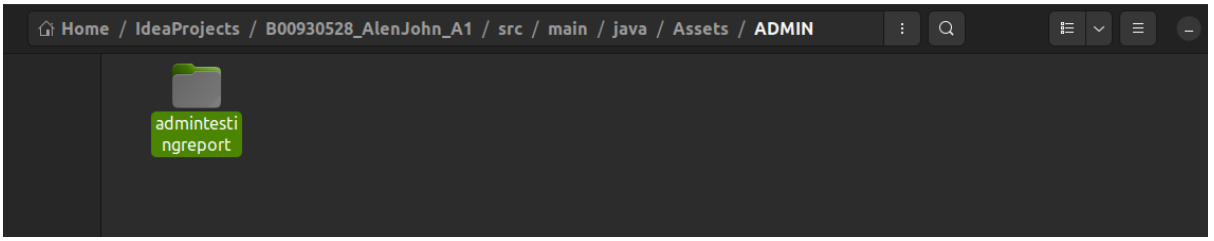
Checking out the user logs -

1. The moment the user creates a new account a new directory for the user is created

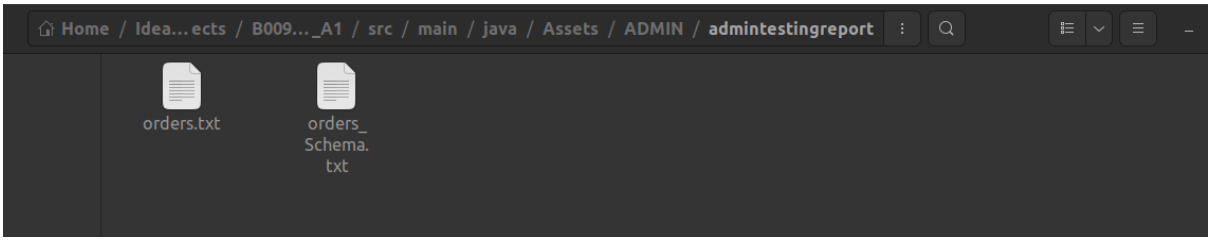
2. All the files and databases are stored in this directory.

3. The directory is under the assets folder in the root directory

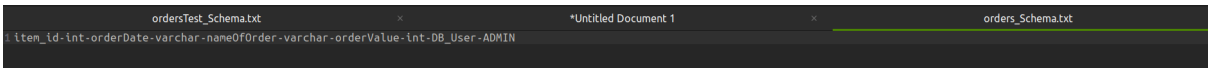
Inside the admin directory a new directory is created for the database -

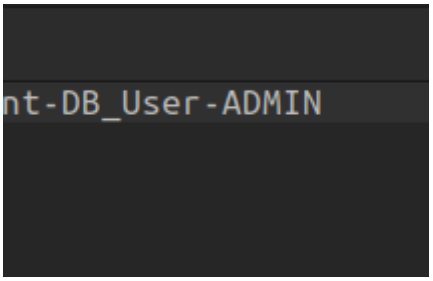


The Tables have been created -



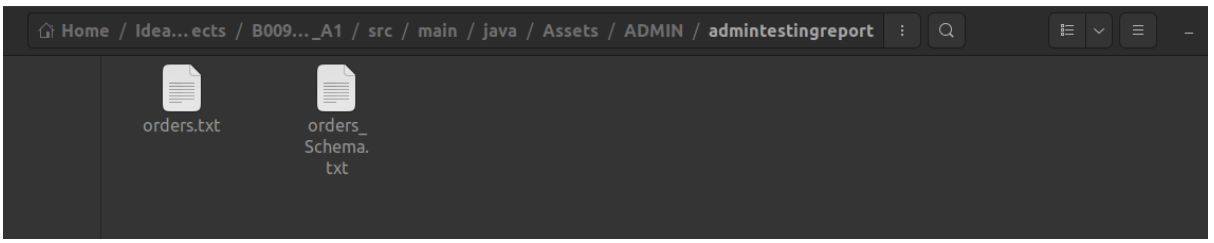
The user logs is stored in the schema -



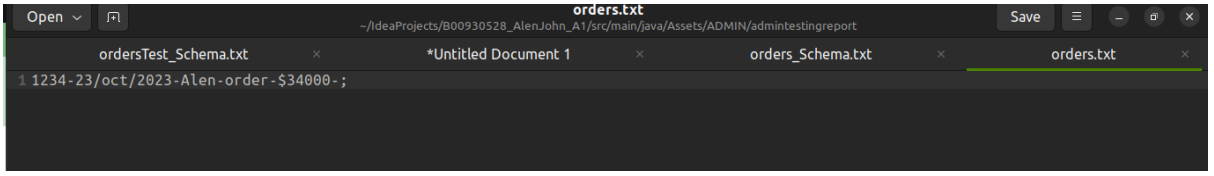


Task 7 (i) -
Creating -

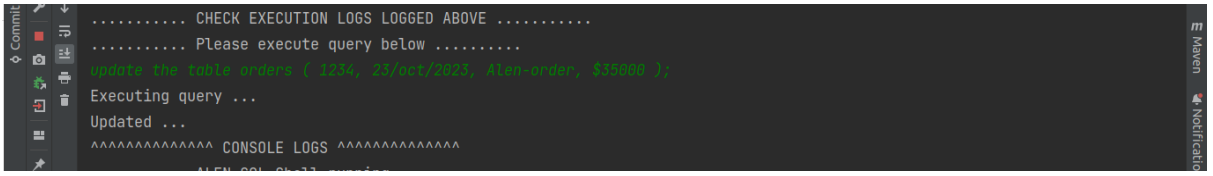
```
To delete table -> Eg : 'delete table orders;'
.......... CHECK EXECUTION LOGS LOGGED ABOVE ..........
.......... Please execute query below ..........
create table orders ( item_id int, orderDate varchar, nameOfOrder varchar, orderValue int );
Executing query ...
Table created successfully ...
^^^^^^^^^^^^^^ CONSOLE LOGS ^^^^^^^^^^^^^^
.......... ALEN SQL Shell running ..........
this shell is whitespace sensitive ...
To create a new Database -> Eg : 'create database NAME_OF_DATABASE;'
To use a existing Database -> Eg : 'use database NAME_OF_DATABASE;'
To create a new table -> Eg : 'create table orders ( item_id int, orderDate varchar, nameOfOrder varchar, orderValue i
To read the table please use -> Eg : 'select * from orders;'
To insert into a table -> Eg : 'insert into orders values ( 1234, 23/oct/2023, Alen-order, $34000 );' NOTE - All the c
To update table -> Eg : 'update table orders ( 1234, 23/oct/2023, Alen-order, $34000 );'
To delete table -> Eg : 'delete table orders;'
.......... CHECK EXECUTION LOGS LOGGED ABOVE ..........
.......... Please execute query below ..........
```

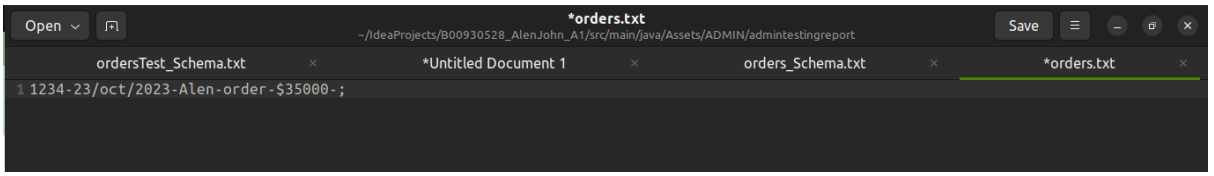


Inserting -

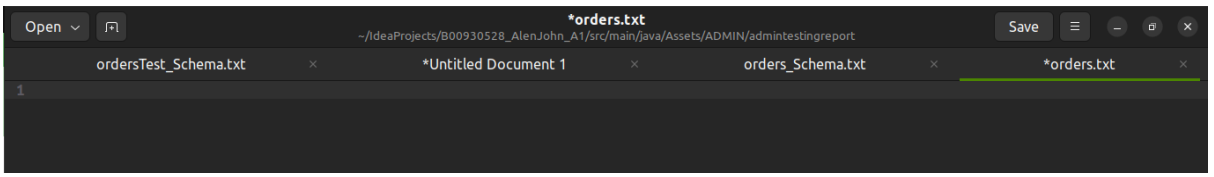


Updating -





Deleting -



References -

[1] J. Doe, "Course content on Brightspace LMS," Dalhousie University, Halifax, NS, Canada, 2023. Available: https://dal.brightspace.com/d2l/le/content/248902/viewContent/3512045/View. Accessed on: Mar. 3, 2023.

[2] E. Baeldung, "SOLID Principles," Baeldung, 2022. Available: https://www.baeldung.com/solid-principles. Accessed on: Mar. 3, 2023.

[3] E. Baeldung, "Java MD5," Baeldung, 2021. Available: https://www.baeldung.com/java-md5. Accessed on: Mar. 3, 2023.

[4] "Dalhousie University," Dalhousie University, Halifax, NS, Canada, 2023. Available: https://www.dal.ca/. Accessed on: Mar. 3, 2023.

[5]"MyDalHub," Dalhousie University, Halifax, NS, Canada, 2023. Available: https://dalu.sharepoint.com/sites/mydalhub. Accessed on: Mar. 3, 2023.

[6] A. John, "CSCI5408 Assignment 1," Dalhousie University, Halifax, NS, Canada, 2023. Available: https://git.cs.dal.ca/alen/csci5408_w23_b00930528_alen_john_assignment_1. Accessed on: Mar. 3, 2023.