CSCI 5408
Data Management, Warehousing Analytics

Project – Sprint 02 – Report

Submitted to

Dr. Saurabh Dey

Department of Computer Science

Dalhousie University.

Submitted by Group 5

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

Table of Contents

Gitlab Repository Link: 3

Pseudo Code: 3

Module 3: Transaction 3

Module 4: Log Management 4

Module 5: Data Modelling – Reverse Engineering 6

Evidence of testing: 11

Module 3: Transaction 11

Module 4: Log Management 29

Module 5: Data Modelling – Reverse Engineering 31

Gitlab Repository Link:

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

Pseudo Code:

Module 3: Transaction

DatabaseService:

method processQueries():

identify the type of the query

fetch the transactional command given by the user

return transactional command

condition1: if the command is ***start transaction***

set the flag for the transaction

create a buffer for temporary database state(**databaseCopy**)

continue taking the next input from the user

execute the following queries on the buffer(**databaseCopy**)

condition2: if the command is ***commit***

reset the flag for the transaction to false

copy the buffer state to the actual database state

persist the state to the file

continue taking the next input from the user in non-transaction mode

condition3: if the command is ***rollback***

reset the flag for the transaction to false

clear the buffer created

continue taking the next input from the user in non-transaction mode

Module 4: Log Management

Query Logger:

```
class QueryLogger:
    constant DATE_FORMATTER = current datetime.

    method getCurrentTimestamp():
        initialize variable with format "yyyy-MM-dd HH:mm:ss"
        return initialized variable

    method writeLog(filePath, logEntry):
        try:
            use file at filePath in append mode
            write logs in new lines
        catch IOException:
            print stack trace of exception

    method logGeneral(generalType, details):
        create log as a log string in JSON format with:
           set current timestamp
            set logType
            set details
        call writeLog with GENERALLOGFILE and log

    method logEvent(eventType, description):
            create log as a log string in JSON format with:
            set current timestamp
```

```
        set eventType
        set description
    call writeLog with EVENTLOGFILE and log

method logQuery(userId, queryType, query):
    create log as a log string in JSON format with:
        set UserId
        set current timestamp
        set queryType
    call writeLog with QUERYLOGFILE and log
```

Module 5: Data Modelling – Reverse Engineering

method generateERD(foreignKeyFile, databaseFile):

```
try:
    databaseInfo = parseDatabaseFile(databaseFile)
    try:
        reader = open(foreignKeyFile)
        sb = new StringBuilder()
        while line = reader.readLine():
            sb.append(line + "\n")
        entries = sb.toString().split("\n\n")

        for entry in entries:
            lines = entry.split("\n")

            databaseName = null
            tableName = null
            foreignKey = null
            referenceTable = null
            referenceTableColumn = null

            for currentLine in lines:
                if currentLine.trim().startsWith("Database: "):
                    databaseName = currentLine.trim().substring(10)
                else if currentLine.trim().startsWith("Table: "):
                    tableName = currentLine.trim().substring(7)
                else if currentLine.trim().startsWith("ForeignKeys: "):
                    foreignKey = currentLine.trim().substring(13)
                else if currentLine.trim().startsWith("ReferenceTables: "):
                    referenceTable = currentLine.trim().substring(17)
```

```
            else if currentLine.trim().startsWith("ReferenceTableColumns: "):
                referenceTableColumn = currentLine.trim().substring(22)

        referenceTableCardinality = if databaseInfo.isPrimaryKey(referenceTable,
referenceTableColumn) then "1" else "N"
        foreignKeyCardinality = if databaseInfo.isPrimaryKey(tableName, foreignKey) then
"1" else "N"
        cardinality = referenceTableCardinality + " to " + foreignKeyCardinality

        erdFileName = databaseName + "_erd.txt"
        try:
            writer = open(erdFileName, "true")
            writer.println(referenceTable + " (" + referenceTableColumn + ") is related to " +
tableName + " (" + foreignKey + ") [" + cardinality + "]")
            print "Successfully generated ERD in file: " + erdFileName
        catch IOException:
            print "Failed to write ERD file: " + e.getMessage()
    catch IOException:
        print "Error reading from file: " + e.getMessage()
```

method parseDatabaseFile(databaseFile):

```
  databaseInfo = new DatabaseInfo()
  try:
    reader = open(databaseFile)
    currentTable = null
    while line = reader.readLine():
      line = line.trim()
      if line.startsWith("- Table: "):
        currentTable = line.substring(9)
      else if line.startsWith("PrimaryKey: "):
```

```
            primaryKey = line.substring(12).trim()
            if primaryKey != "null" and currentTable != null:
                databaseInfo.addPrimaryKey(currentTable, primaryKey)
    catch IOException:
        print "Error reading database file: " + e.getMessage()
    return databaseInfo
```

class DatabaseInfo:

```
    // Constructor to initialize DatabaseInfo
    function DatabaseInfo():
        tablePrimaryKeys = new HashMap()
```

method addPrimaryKey(table, primaryKey):

```
      if not tablePrimaryKeys.containsKey(table):
          tablePrimaryKeys.put(table, new HashSet())
      tablePrimaryKeys.get(table).add(primaryKey)
```

method isPrimaryKey(table, column):

```
        return tablePrimaryKeys.containsKey(table) and
tablePrimaryKeys.get(table).contains(column)
```

class ForeignKey:

method serialize(databaseName, tableName):

```
        sb = new StringBuilder()
        sb.append("Database: ").append(databaseName).append("\n")
        sb.append("Table: ").append(tableName).append("\n")
        sb.append("ForeignKeys: ").append(foreignKeys.join(",")).append("\n")
        sb.append("ReferenceTables: ").append(referenceTables.join(",")).append("\n")
```

```
    sb.append("ReferenceTableColumns: ").append(referenceTableColumns.join(",")).append("\n")
    return sb.toString()
```

method saveToFile(filename, databaseName, tableName):

```
    try:
        writer = open(filename, "a") // Append mode
        writer.write(serialize(databaseName, tableName))
    catch IOException as e:
        print "Error saving to file: " + e.getMessage()
```

method loadFromFile(filename):

```
    try:
        reader = open(filename)
        sb = new StringBuilder()
        while line = reader.readLine():
            sb.append(line).append("\n")
        deserialize(sb.toString())
    catch IOException as e:
        print "Error loading from file: " + e.getMessage()
```

method deserialize(str):

```
    lines = str.split("\n")
    foreach line in lines:
        if line.trim().startsWith("ForeignKeys: "):
            foreignKeys = line.trim().substring(13).split(",")
        else if line.trim().startsWith("ReferenceTables: "):
```

```
            referenceTables = line.trim().substring(17).split(",")
        else if line.trim().startsWith("ReferenceTableColumns: "):
            referenceTableColumns = line.trim().substring(22).split(",")
```

method generateForeignKeyMetadata(filename, databaseName, tableName):

```
    if not foreignKeys.isEmpty():
        saveToFile(filename, databaseName, tableName)
```

Evidence of testing:

Module 3: Transaction

Insert Query: -

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

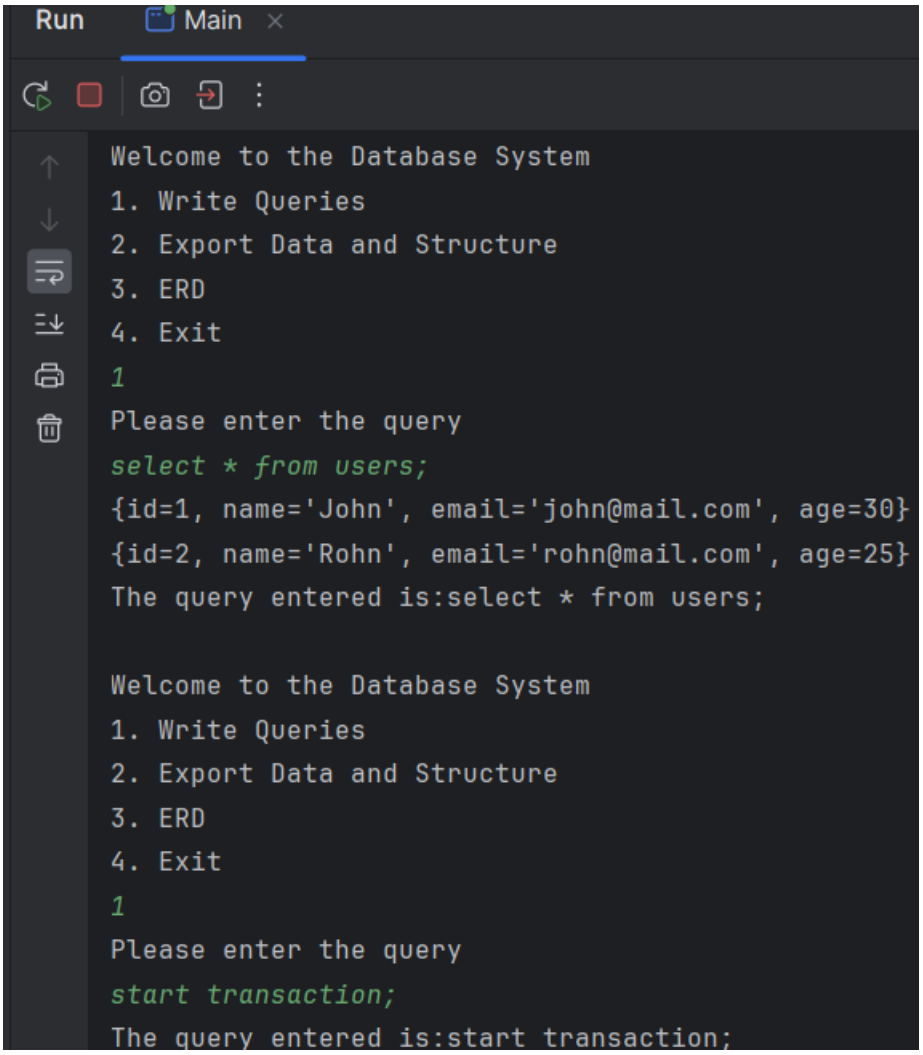

Figure 1:initial state before insert transaction operation

```
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 2: 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 3: 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 4: 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 5: The insert query while in the transaction state followed by 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
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 6: 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.

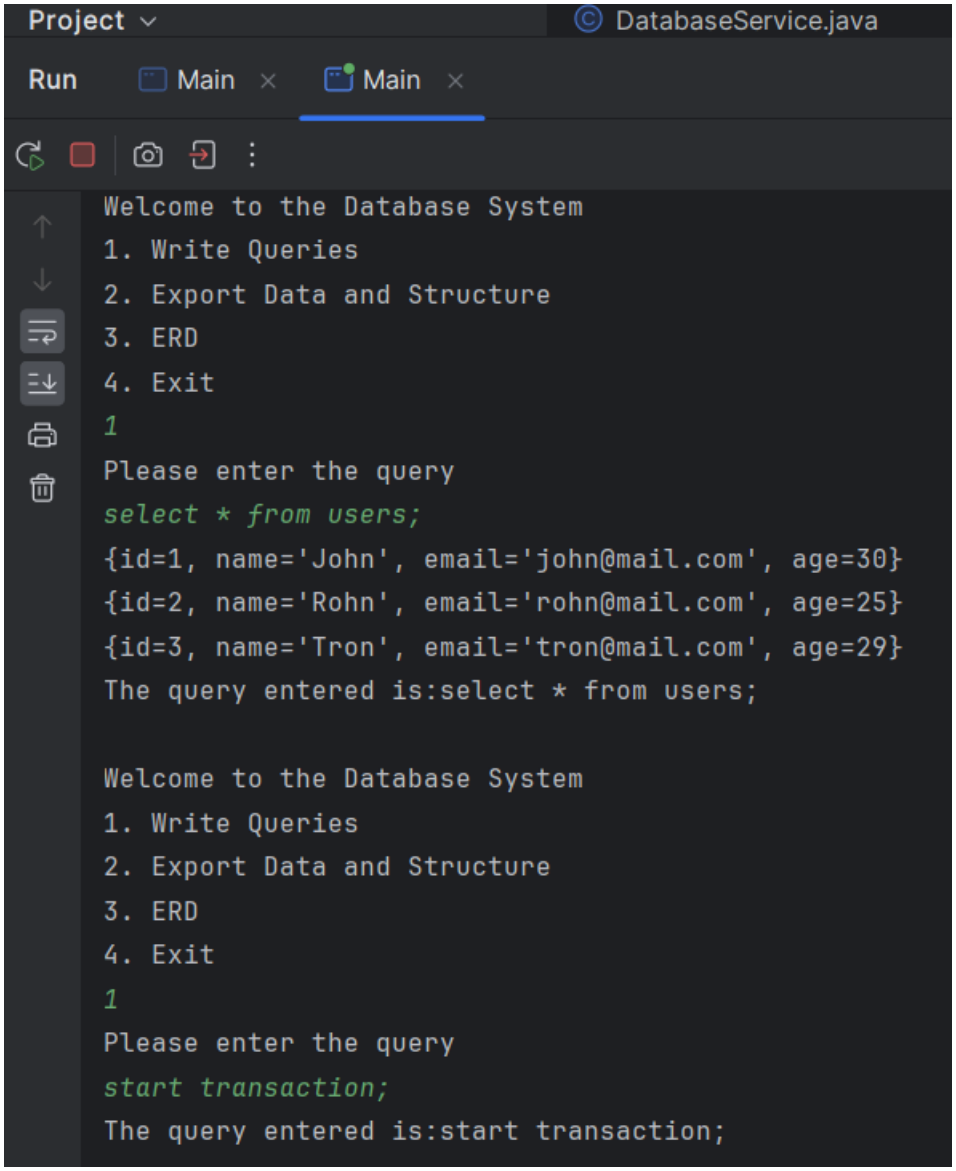

Figure 7: initial state of the db before transaction with update query

```
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 8: 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 9: 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.

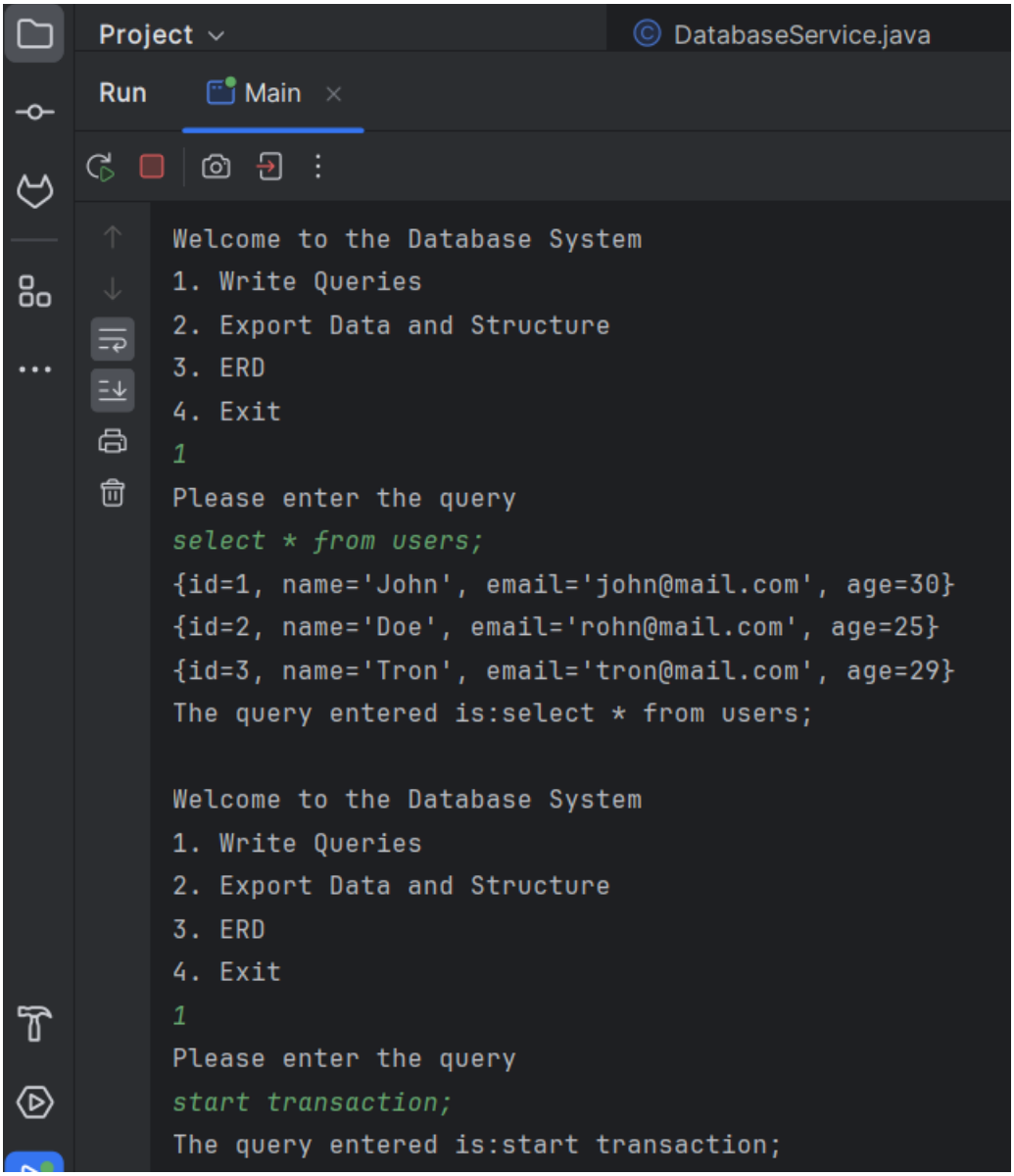

Figure 10: the db records before the transaction update query followed by rollback

```
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 11: 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 12: 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.

```
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=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 13: 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 14: 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 15: 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 16: 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 17: 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 18: 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

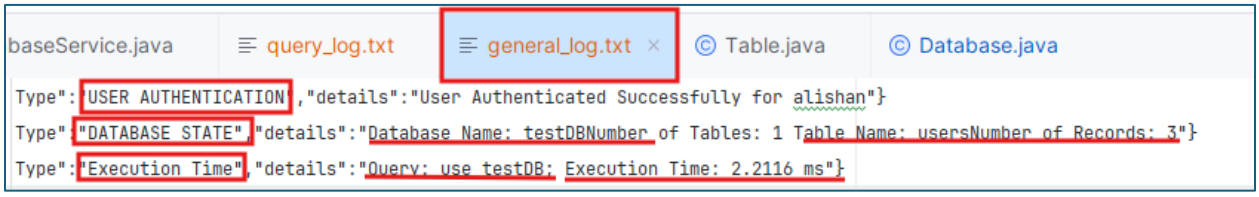

Figure 19: Shows typical general log example.

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.

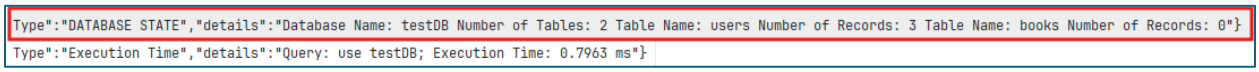

Figure 20: Shows newly added books table on the logs with 0 records.

Figure 21: 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 22: 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.


```
{"timestamp":"2024-07-13 15:31:06","logType":"EVENT","eventType":"test","description":"users is locked to test because of transaction."}
{"timestamp":"2024-07-13 15:31:11","logType":"EVENT","eventType":"test","description":"Releasing all locks from the tables."}
{"timestamp":"2024-07-13 15:46:21","logType":"EVENT","eventType":"test","description":"users is locked to test because of transaction."}
```


Figure 23: Shows the transaction lock applied and released on tables.


```
{"timestamp":"2024-07-13 18:58:12","logType":"EVENT","eventType":"users","description":"TRANSACTION DETECTED"}
{"timestamp":"2024-07-13 18:58:18","logType":"EVENT","eventType":"users","description":"TRANSACTION COMMIT"}
```


Figure 24: 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.

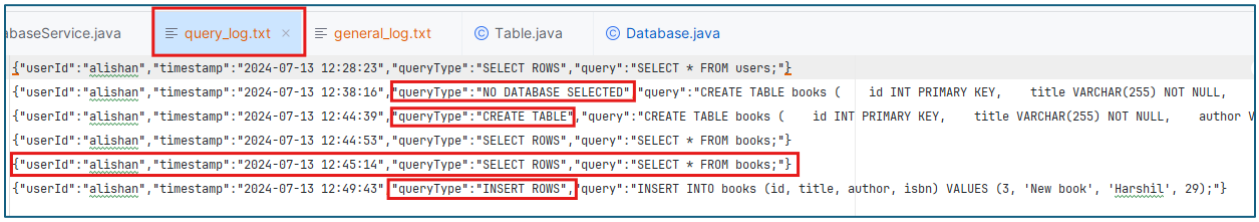

Figure 25: Shows the output of query logs.

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 19: 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 20: 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 21: 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 22: 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 23: 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 24 :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 25: 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 26: Referencing attribute of table which does not exist

Sprint 2 Meeting Log:

Table 1: Sprint 2 meeting log for group 5

Date	Time	Attendees	Agenda	Meeting Type	Meeting Recording Link
3 July, 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