

CSCI 5408

DATA MANAGEMENT AND WAREHOUSING

Sprint 2 Report Group 1 - TinyDB

Group Members:

- 1) Arta Baghdadi (B00981005)
- 2) Jay Sanjaybhai Patel (B00982253)
- 3) Prashanth Venkatesan (B00980291)

GitLab Project Link: https://git.cs.dal.ca/prashanthv/csci5408_tiny_db.git

Table of Contents

| | |
|----------------------|----|
| Pseudocode..... | 3 |
| Testcases..... | 9 |
| Meeting records..... | 23 |

Pseudocode

A) MODULE: Log Management

- Logger Constructor:

```
function Logger():  
  set logFilePath to "<LogFilePath>/log.txt"  
  create directory if not exists  
  clear log file
```

- Get Instance:

```
function getInstance():  
  if logger instance is null:  
    create new logger instance  
  return logger instance
```

- Set File Path:

```
function setFilePath(logFilePath):  
  set logFilePath  
  create directory if not exists  
  clear log file
```

- Create Directory If Not Exists:

```
function createDirectoryIfNotExists():  
  if directory does not exist:  
    create directory
```

- Clear Log File:

```
function clearLogFile():  
  try:  
    clear content of log file  
  catch error:  
    print error message
```

- Log:

```
function log(status, logMessage):  
  try:  
    append logMessage with timestamp and status to log file
```

```
    return true
catch error:
    print error message
    return false
```

B) MODULE: Data Modelling – Reverse Engineering

- Load MetaData:

```
function loadMetaData():
    clear list of databases
    read content from metadata file

    if content is empty:
        print "Do not have any database!"
        return false

    print "List of Databases:"
    for each line in content:
        add line to list of databases
        print line

    return true
```

- Draw ERD:

```
function drawERD():
    if not loadMetaData():
        return

    prompt user to enter database name
    if database name is not in list of databases:
        print "Database Not Found!"
        return

    set dbName to user input
    if not load database file:
        return

    create ERD text file
```

- **Create Txt File:**

```
function createTxtFile():  
    initialize content as empty list  
  
    for each table in list of table names:  
        add table name to content  
        get attributes of table  
  
        for each attribute in attributes:  
            check if attribute is foreign key  
            if attribute is primary key:  
                if attribute is also foreign key:  
                    add attribute info with (PK) (FK) to content  
                else:  
                    add attribute info with (PK) to content  
            else if attribute is foreign key:  
                add attribute info with (FK) to content  
  
        add newline to content  
  
    if ERD directory does not exist:  
        create ERD directory  
  
    write content to ERD file
```

- **Check Is Foreign Key:**

```
function checkIsForeignKey(attribute, tableName):  
    for each cardinality in list of cardinalities:  
        if cardinality matches attribute and table name:  
            return cardinality info  
  
    return null
```

- **Load Database File:**

```
function loadDatabaseFile():  
    if dbName is empty:  
        print "No Database Selected!"  
        return false  
  
    clear existing table names, attributes, and cardinalities  
    read content from database schema file
```

```

if content is empty:
    return true

for each line in content:
    if line starts with "@":
        add table name to list of table names
        initialize attributes as empty list

        while next line starts with "?" or "&":
            if line starts with "&":
                parse and add cardinality
                continue

            parse attribute from line
            add attribute to attributes

        add attributes to list of table attributes

return true

```

C) MODULE: Export Structures and Values

- Export Database:

```

function exportDatabase(dbName):
    set dbFolder to "<DatabaseDirectory>/" + dbName
    set schemaFile to dbFolder + "/" + dbName + ".schema"
    read schemaLines from schemaFile

    if schemaLines is empty:
        print "Failed to read schema file."
        return

    initialize sqlDump as empty string
    initialize currentTable as null
    initialize currentTableSchema as empty string

    for each line in schemaLines:
        if line starts with "@":
            if currentTable is not null:
                add create table statement to sqlDump
                add insert statements to sqlDump
            set currentTable to substring of line after "@"

```

clear currentTableSchema

append line to currentTableSchema

if currentTable is not null:

add create table statement to sqlDump

add insert statements to sqlDump

write sqlDump to file

- **Generate Create Table Statement:**

function generateCreateTableStatement(tableName, tableSchema):

initialize createTable as "CREATE TABLE " + tableName + " (\n"

split tableSchema into schemaLines by newline

initialize foreignKeys as empty list

for each line in schemaLines:

if line starts with "?":

split line into parts by "#"

set columnName to parts[0]

set dataType to parts[1]

if dataType is VARCHAR without size, set to "VARCHAR(255)"

set isUnique to parts[3] is "true"

set isPrimaryKey to parts[4] is "true"

add columnName and dataType to createTable

if isPrimaryKey, add "PRIMARY KEY" to createTable

else if isUnique, add "UNIQUE" to createTable

add ",\n" to createTable

else if line starts with "&":

split line into parts by "#"

add foreign key constraint to foreignKeys

for each fk in foreignKeys:

add fk to createTable

remove last comma and newline from createTable

add ");\n" to createTable

return createTable

- **Generate Insert Statements:**

```
function generateInsertStatements(dbFolder, tableName):  
  initialize insertStatements as empty string  
  set tableFile to dbFolder + "/" + tableName + ".data"  
  read dataLines from tableFile  
  
  if dataLines is not empty:  
    for each row in dataLines:  
      split row into values by "#"  
      add "INSERT INTO " + tableName + " VALUES (" to insertStatements  
      for each value in values:  
        add "" + value + ", " to insertStatements  
      remove last comma and space from insertStatements  
      add ");\n" to insertStatements  
  return insertStatements
```

- **Write SQL Dump to File:**

```
function writeSqlDumpToFile(dbName, sqlDump):  
  set dumpFile to "<DatabaseDirectory>" + dbName + "/" + dbName + "_dump.sql"  
  try:  
    write sqlDump to dumpFile  
  catch error:  
    print "Failed to write SQL dump: " + error.message
```


Testcases

A) MODULE: Export Structures and Values

Note: The dump files are made with the extension .sql. So the files are opened in SQL workbench for viewing.

1) Export empty database

SQL dump file wouldn't have anything as no tables are defined. Database creation commands wouldn't be displayed as they are not shown in the original SQL dump files as well.

```
1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
2
Enter database name:
emptyDB
```

Fig 1: Exporting dump for empty database

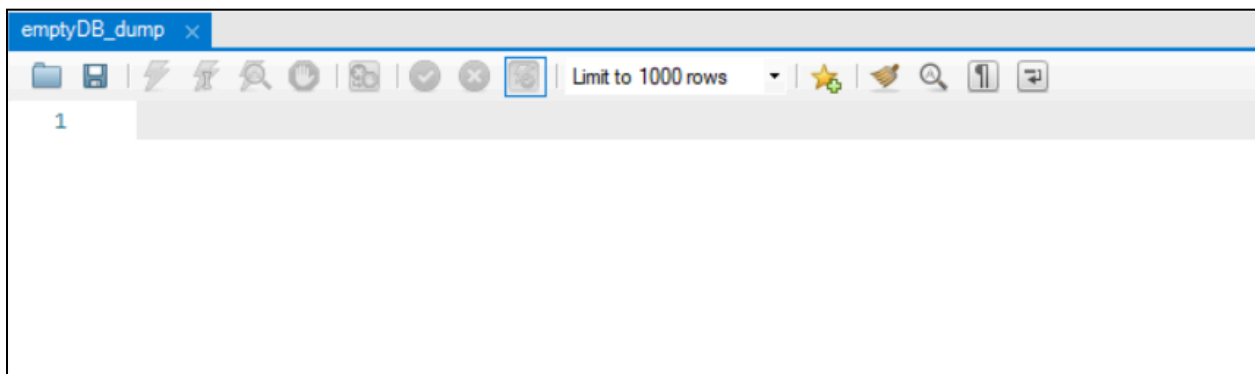


Fig 2: Dump file for empty DB

2) Export a single table without any values

```
Enter your SQL queries (type 'EXIT' to end):
SQL> create database noValueTable;
SQL> use database noValueTable;
SQL> create table noValue (id int, price double);
```

Fig 3: Creating single table without values

```

1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
2
Enter database name:
noValueTable

```

Fig 4: Exporting dump for single table w/o values

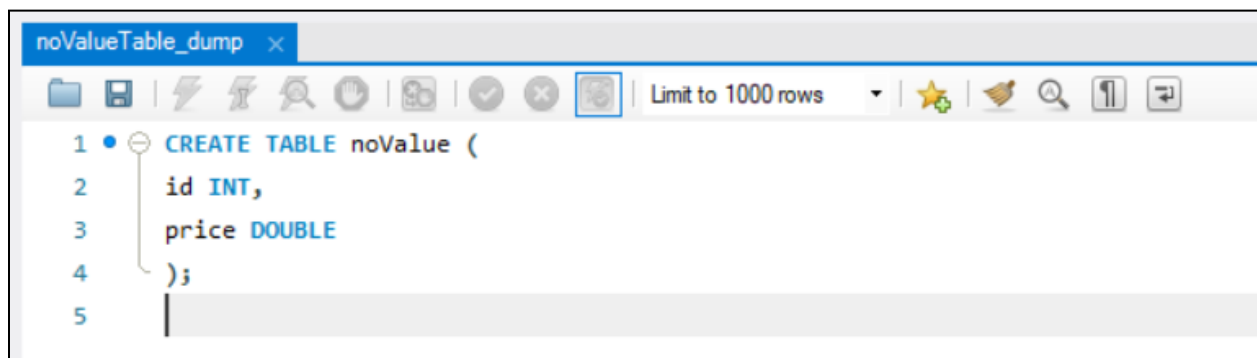


Fig 5: Dump file with single table (w/o values)

3) Export a single table with values

```

1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
1
Enter your SQL queries (type 'EXIT' to end):
SQL> create database tableWithValues;
SQL> use database tableWithValues;
SQL> create table tableValues (id int, price double);
Enter Field Name: exit
SQL> insert into tableValues values (1, 2.3);
SQL>

```

Fig 6: Creating single table with values

```

1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
2
Enter database name:
tableWithValues

```

Fig 7: Exporting dump for single table w/ values

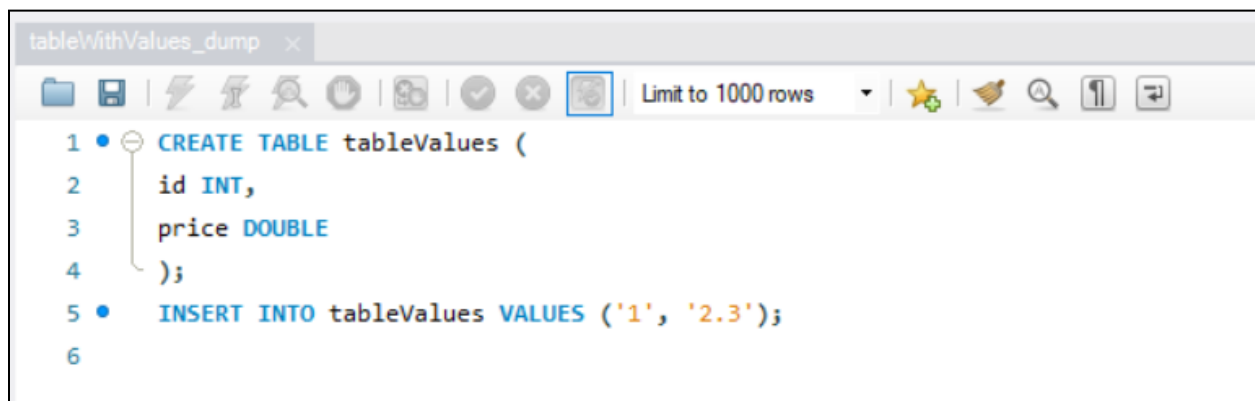


Fig 8: Dump file for single table (w/ values)

4) Export table without foreign keys

```

Enter your SQL queries (type 'EXIT' to end):
SQL> create database noValueTable;
SQL> use database noValueTable;
SQL> create table noValue (id int, price double);

```

Fig 9: Creating table w/o foreign key

```

1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
2
Enter database name:
noValueTable

```

Fig 10: Exporting dump for table w/o foreign key

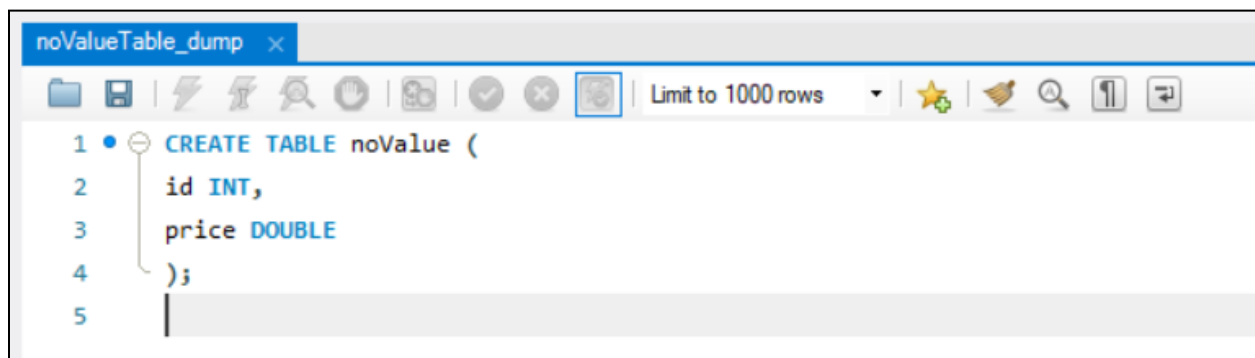


Fig 11: Dump file for table w/o foreign key

5) Export table with foreign key relation

```

Enter your SQL queries (type 'EXIT' to end):
SQL> create database foreign;
SQL> use database foreign;
SQL> create table primaryTable (id int PRIMARY KEY);

```

Fig 12: Creating a table which another table will reference

```

SQL> create table foreignTable (SIN_no int PRIMARY KEY, id int );
Enter Field Name: id
primaryTable
Enter Table Name: primaryTable
id
Enter Field Name: id
Foreign Key Added Successfully!

```

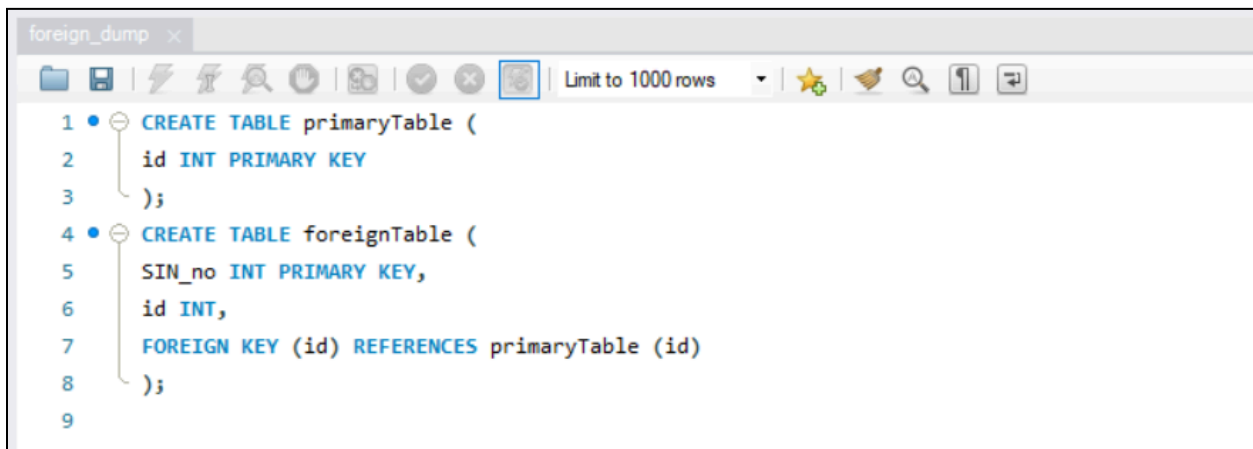
Fig 13: Creating table w/ foreign key relation

```

1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
2
Enter database name:
foreign

```

Fig 14: Exporting dump for table w/ foreign key



```

foreign_dump x
Limit to 1000 rows
1 CREATE TABLE primaryTable (
2   id INT PRIMARY KEY
3 );
4 CREATE TABLE foreignTable (
5   SIN_no INT PRIMARY KEY,
6   id INT,
7   FOREIGN KEY (id) REFERENCES primaryTable (id)
8 );
9

```

Fig 15: Dump file for table w/ foreign key

6) Export table with UNIQUE and NOT NULL constraints

```

1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
1
Enter your SQL queries (type 'EXIT' to end):
SQL> create database unique;
SQL> use database unique;
SQL> create table uniqueTable (id int PRIMARY KEY, SIN int NOT NULL UNIQUE KEY );

```

Fig 16: Creating table w/ unique and not null constraints

```
1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
2
Enter database name:
unique
```

Fig 17: Exporting dump for table w/ constraints

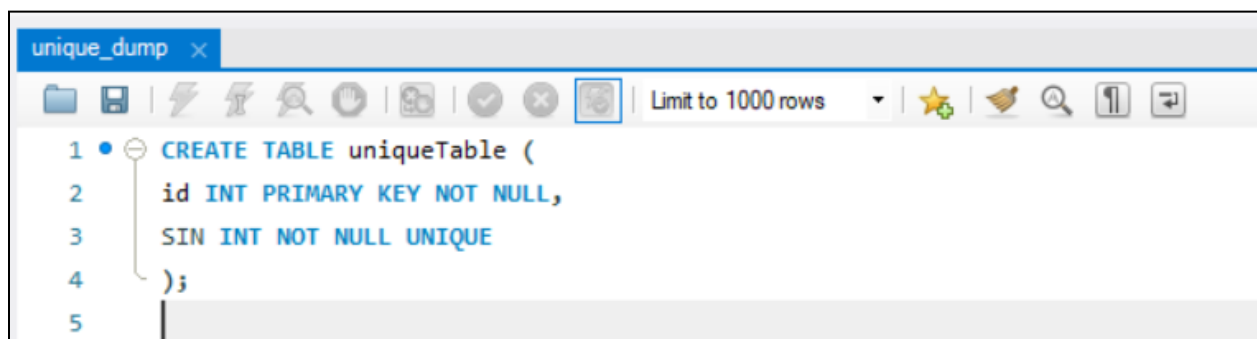


Fig 18: Dump file for a table w/ unique and not null constraints

7) Export non-existing database folder

It will display an error when trying to export a non-existent database.

```
1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
2
Enter database name:
dbDoesntExist
Database not found!
```

Fig 19: Error prompt for no db found

B) MODULE: Log Management

1) Log creation:

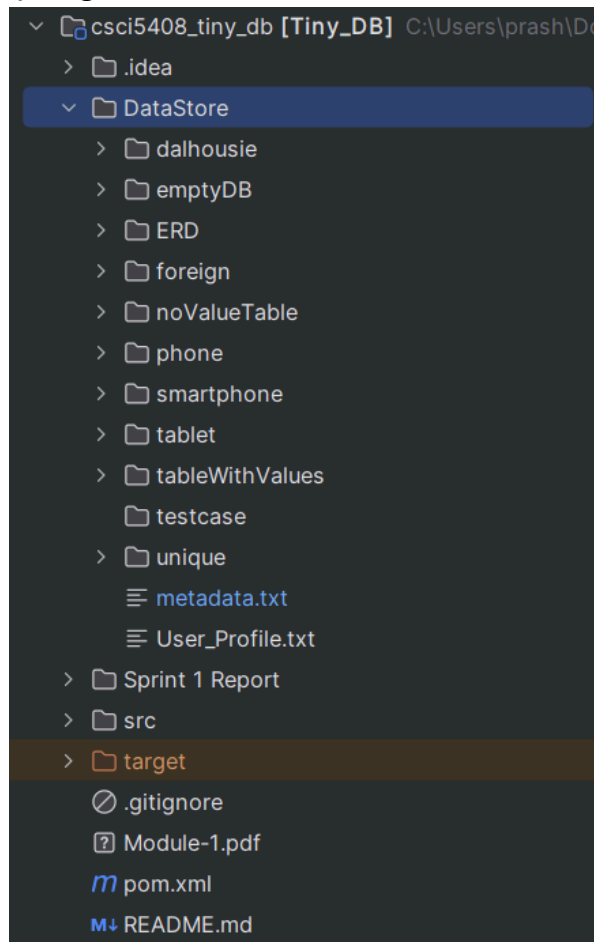


Fig 20: No log found in directory before operation

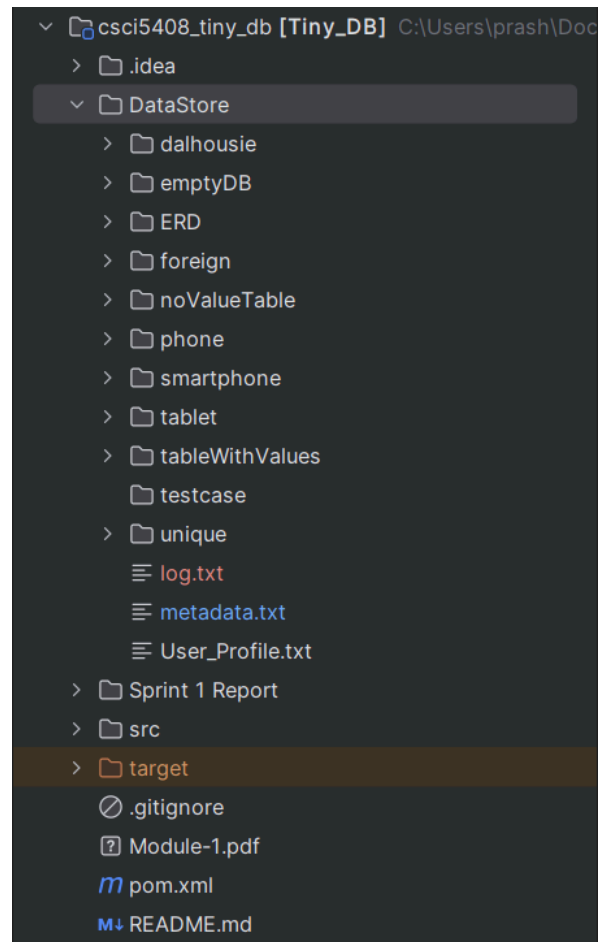


Fig 21: Log found in directory after operation

Note: All logs are implemented in one log file itself.

```
1 [Successful] [2024-07-13 22:16:58.318] User 'prashanth' logged in successfully.
2 [Successful] [2024-07-13 22:24:03.792] Database 'laptop' created successfully.
3 [Successful] [2024-07-13 22:24:08.736] Using database 'laptop'.
4 [Successful] [2024-07-13 22:24:30.656] create table models (id int, name varchar); executed in 2466 ms. State= Database: laptop| Tables: (models, 0 records),
5 [Successful] [2024-07-13 22:24:56.641] insert into models values (1, Macbook); executed in 0 ms. State= Database: laptop| Tables: (models, 1 records),
6 [Successful] [2024-07-13 22:25:08.726] insert into models values (2, Asus); executed in 3 ms. State= Database: laptop| Tables: (models, 2 records),
7 [Successful] [2024-07-13 22:25:29.618] select * FROM models; executed in 3 ms. State= Database: laptop| Tables: (models, 2 records),
8 [Failed] [2024-07-13 22:25:49.763] Table ' computer' doesn't exist in database 'laptop'.
9 [Successful] [2024-07-13 22:26:05.129] update models set id = 3 where id = 2; executed in 12 ms. State= Database: laptop| Tables: (models, 2 records),
10 [Successful] [2024-07-13 22:26:19.13] select * FROM models; executed in 0 ms. State= Database: laptop| Tables: (models, 2 records),
```

Fig 22: Log data

2) Log query execution time

Query executed: select * from smartphone;

Log:

[Successful] [2024-07-13 19:13:28.151] select * FROM smartphone; executed in 0 ms. State= Database: phone | Tables: (smartphone, 1 records)

3) Log state of the database

Query executed: select * from newPhones;

Log:

[Successful] [2024-07-13 19:19:48.126] select * FROM newPhones; executed in 2 ms. State= Database: phone | Tables: (newPhones, 2 records),

4) Log changes

- For Insert query:

[Successful] [2024-07-13 19:19:36.062] insert into newPhones values (2, S24, Samsung); executed in 4 ms. State= Database: phone | Tables: (newPhones, 2 records)

- For Update query:

[Successful] [2024-07-13 19:22:22.264] update newPhones set id = 3 where id = 2; executed in 10 ms. State= Database: phone | Tables: (newPhones, 2 records)

- For Delete query:

[Successful] [2024-07-13 19:24:24.747] DELETE FROM newPhones WHERE id = 3; executed in 0 ms. State= Database: phone | Tables: (newPhones, 1 records)

5) Failed queries examples:

- When the table doesn't exist:

[Failed] [2024-07-13 19:11:02.64] Table 'phone' doesn't exist in database 'phone'.

- When DB doesn't exist:

[Failed] [2024-07-13 19:28:15.175] Failed to use database 'invalidDB' as it does not exist.

- When a row doesn't exist:

[Failed] [2024-07-13 19:31:13.826] Error processing query 'delete FROM newPhones WHERE id = 5;' in 0 ms: No rows matched the condition.

6) Invalid queries:

Invalid queries being executed:

[Failed] [2024-07-13 19:27:14.166] Error processing query 'incorrect query;' in 0 ms: Unsupported SQL command: INCORRECT

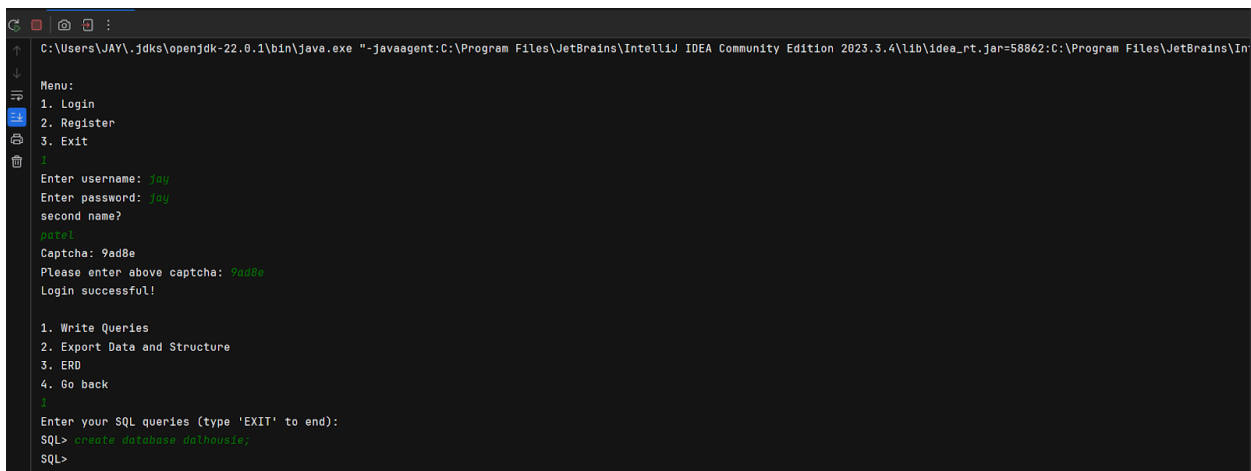
7) User-related operations:

[Successful] [2024-07-13 19:09:36.679] User 'prashanth' logged in successfully.

[Failed] [2024-07-13 19:34:02.353] User 'prashanth' failed captcha verification.

C) MODULE: Data Modelling – Reverse Engineering

- Here, we show how a user can create an ERD (Entity-Relationship Diagram) of any existing database and what that ERD looks like.
- We use a .txt file to illustrate the overall ERD.
- We also demonstrate how a user can add a foreign key to a table while creating the table.
- So, first, we log in using a registered user ID, then create a database, then make a table, and finally generate its ERD.



```
C:\Users\JAY\jdk\openjdk-22.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.4\lib\idea_rt.jar=58862:C:\Program Files\JetBrains\In

Menu:
1. Login
2. Register
3. Exit
4
Enter username: jay
Enter password: jay
second name?
patel
Captcha: 9ad8e
Please enter above captcha: 9ad8e
Login successful!

1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
5
Enter your SQL queries (type 'EXIT' to end):
SQL> create database dalhousie;
SQL>
```

Fig 23: Login and Create Database

- Here we create a database of the name “dalhousie”.
- Now we create three tables inside this database.



```
SQL> use database dalhousie;
SQL> create table student (id INT PRIMARY KEY, name VARCHAR 255, email VARCHAR 255, phone INT 10, address VARCHAR 200);
Enter Field Name: exit
SQL>
```

Fig 24: Create a table (Student) without any foreign key

- The first table is named “student”, with fields for student id, name, email, phone, and address.
- In this table, we set the student id as the primary key.

```
SQL> create table course (id INT PRIMARY KEY, name VARCHAR 255, description VARCHAR 255, instructor_name VARCHAR 200, level INT 5);
Enter Field Name: exit
```

Fig 25: Create a table (course) without any foreign key

- The second table is named “course”, with fields for course id, name, description, instructor_name, and level(undergrad or grad).
- In this table, we set the course id as the primary key.

```
SQL> create table student_course (id INT PARIMARY KEY, student_id INT, course_id INT);
Enter Field Name: name
Invalid Field Name!
```

Fig 26: Create a table (student_course)

- The third and last table is named “student_course”, with fields for id, student_id, and course_id.
- In this table, we set the id as the primary key, and now we set student_id and course_id as foreign keys.

```
SQL> create table student_course (id INT PARIMARY KEY, student_id INT, course_id INT);
Enter Field Name: name
Invalid Field Name!
Enter Field Name: student_id
student
course
Enter Table Name: mobile
Invalid Table Name!
Enter Field Name:
```

Fig 27: Enter the invalid table name

- Here, the user needs to enter the attribute name they want to set as a foreign key.
- In the screenshot above, we show what happens if they enter an attribute that doesn’t exist in the table they want to create.

```

SQL> create table student_course (id INT PRIMARY KEY, student_id INT, course_id INT);
Enter Field Name: name
Invalid Field Name!
Enter Field Name: student_id
student
course
Enter Table Name: mobile
Invalid Table Name!
Enter Field Name: student_id
student
course
Enter Table Name: student
id
Enter Field Name: student_mobile
Invalid Foreign Key Name!
Enter Field Name:

```

Fig 28: Enter invalid reference key name

- Now, after entering the attribute name from the current table, the user needs to enter the table name to which it needs to connect for the reference key.
- We also show the list of existing table names from the current database, so it will be easy for the user to see them and write the table name.
- In the screenshot above, we show what happens if the user enters a table name that isn't present in the given table names.

```

Enter Field Name: student_id
student
course
Enter Table Name: student
id
Enter Field Name: id
Foreign Key Added Successfully!
Enter Field Name: course_id
student
course
Enter Table Name: course
id
Enter Field Name: id
Foreign Key Added Successfully!
Enter Field Name: exit
SQL> |

```

Fig 29: Foreign keys (student_id and course_id) added successfully

- The previous screenshot shows the complete process from creating a table to adding a foreign key, including all the successful steps.

```

dalhousie.txt x
1  @student
2  ?id#INT#0#true#true#true
3  ?name#VARCHAR#255#false#false#false
4  ?email#VARCHAR#255#false#false#false
5  ?phone#INT#10#false#false#false
6  ?address#VARCHAR#200#false#false#false
7  @course
8  ?id#INT#0#true#true#true
9  ?name#VARCHAR#255#false#false#false
10 ?description#VARCHAR#255#false#false#false
11 ?instructor_name#VARCHAR#200#false#false#false
12 ?level#INT#5#false#false#false
13 @student_course
14 ?id#INT#0#false#false#false
15 ?student_id#INT#0#false#false#false
16 ?course_id#INT#0#false#false#false
17 &student_id#id#student
18 &course_id#id#course
19

```

Fig 30: Text file after successfully creating the table and adding a foreign key.

- We use the “&” keyword to separate the foreign key constraint from the normal attributes.
- You can see the last two lines in which we connect “student_id” from the “student_course” table to the “student” table, and “course_id” from “student_course” to the “course” table.
- Here, we store this information in three parts, and all parts are separated by the “#” keyword.
- The first part shows the foreign key, the second part shows the reference key and the last part shows the reference key table name.

```
1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
3

List of Databases:
phone
smartphone
dalhousie

Enter Database Name: |
```

Fig 31: Show a list of databases to create ERD

- Now, we see how a user can generate the ERD of the "dalhousie" database.
- When the user chooses option 3 to generate the ERD, we show a list of existing database names so that the user can see them and enter the database name for which they want to generate the ERD.

```
1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
3

List of Databases:
phone
smartphone
dalhousie

Enter Database Name: mobile
Database Not Found!

1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
```

Fig 32: Enter an invalid database name

- In the last screenshot, we show what happens if the user enters a database name that doesn't exist in the provided list.

```

1. Write Queries
2. Export Data and Structure
3. ERD
4. Go back
3

List of Databases:
phone
smartphone
dalhousie

Enter Database Name: dalhousie
ERD Printed Successfully!

```

Fig 33: ERD created successfully for database name "dalhousie"

- Now ERD is successfully generated for the provided database in this case "dalhousie".

```

1      student
2      id (PK)
3      name
4      email
5      phone
6      address
7
8
9      course
10     id (PK)
11     name
12     description
13     instructor_name
14     level
15
16
17     student_course
18     id
19     student_id(M:1) with student
20     course_id(M:1) with course
21
22

```

Fig 34: ERD(.txt file) for database name "dalhousie"

- Here, we can see the table names, their attributes, and cardinalities.
- It shows that one student enrolls in many courses, and one course enrolls many students.

- In this way, the user just needs to enter the database name, and its ERD is ready from which they can create a physical schema.

Meeting records

| Date | Time | Attendees | Agenda | Meeting Type | Meeting Link |
|------------|------------------|----------------------|--|--------------|---|
| 02/07/2024 | 6:10 - 6:20 PM | Arta, Jay, Prashanth | Discussing modules for the upcoming sprint | Online | https://bit.ly/3Y35kY6 |
| 10/07/2024 | 11:30 – 11:45 AM | Arta, Jay, Prashanth | Weekly standup meeting to measure progress | Online | https://bit.ly/3WkvXGx |