

# CSCI 5408

## DATA MANAGEMENT AND WAREHOUSING

### 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](https://git.cs.dal.ca/prashanthv/csci5408_tiny_db.git)

## Table of Contents

Background Research.....	3
Architecture Diagram.....	4
Pseudocode.....	5
Testcases.....	12
Register & Authentication: .....	12
Queries:.....	14
Meeting records.....	20

## Background Research

### **Data Structures used:**

- 1) Arrays are chosen for their simplicity and efficiency in handling linear data. Arrays provide fast access to elements using indices, which is beneficial for quick lookups and modifications. They are also memory-efficient when dealing with a fixed size of elements.
- 2) HashMap is used to provide efficient storage and retrieval of data through key-value pairs. They offer constant time complexity for insertions, deletions, and lookups, making them highly efficient for managing data where quick access and updates are required. This is particularly useful for operations like database transactions, where data needs to be accessed and modified frequently. Hash maps also handle dynamic resizing more efficiently compared to arrays, allowing the data structure to grow as needed without a significant performance impact.

### **Custom File Format:**

Here, we use a simple plain text file to store the data and metadata for this TinyDB project. Different delimiters are used to differentiate the data from each other. For example, "@" is used to denote the table name, and "?" is used to indicate each column and its properties. Additionally, "#" is used to separate multiple data items in one line. For instance, to store a user ID and its password, we store them as userID#password in one line.

```
1      @sample_table
2      ?id#INT#0#false#false#false
3      ?name#VARCHAR#0#false#false#false
```

*Figure 1: Delimiter use*

## Architecture Diagram

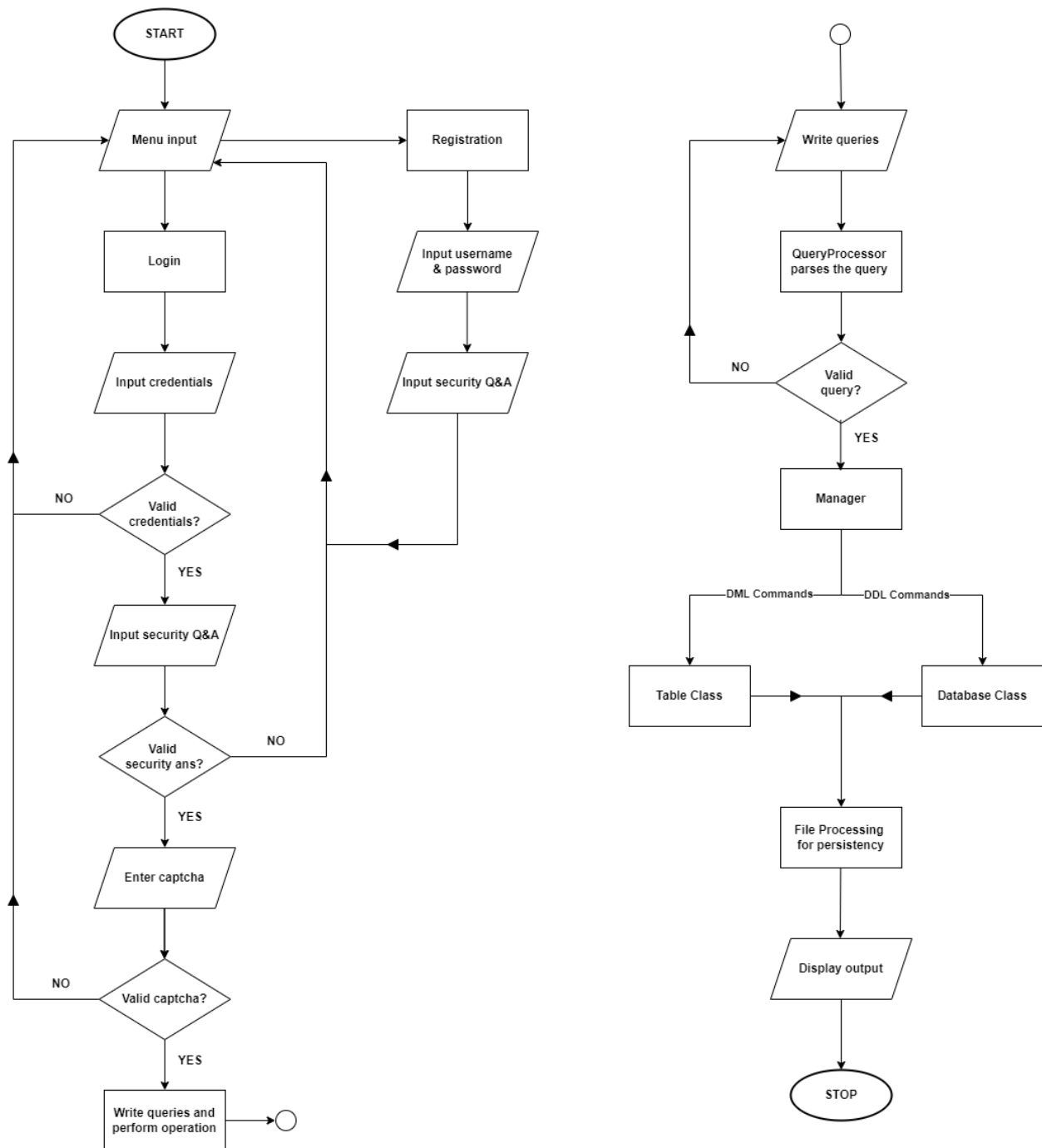


Figure 2: Application workflow

## Pseudocode

### **1) Create Database:**

```
function createDatabase(dbName):  
    if databaseExists(dbName):  
        return False  
    createDirectory(dbName)  
    return True
```

```
function databaseExists(dbName):  
    return directoryExists(dbName)
```

```
function createDirectory(dbName):  
    mkdir(dbName)
```

### **2) Create Table:**

```
function createTable(dbName, tableName, attributes):  
    if databaseNotSelected(dbName):  
        return False  
    if tableExists(dbName, tableName):  
        return False  
    writeTableSchema(dbName, tableName, attributes)  
    return True
```

### **3) Use Database:**

```
function processUseQuery(parts):  
    if length(parts) < 2 or upper(parts[1]) != "DATABASE":  
        print("Invalid USE DATABASE query format.")  
    if dbName not in databases:  
        return False  
    content = read("DataBase_FILE_PATH/" + dbName + "/" + dbName + ".txt")  
    for each currLine in content:  
        if startsWith(currLine, "@"):  
            add(tables, trim(substring(currLine, 1)))  
    return True
```

### **4) Select Query:**

```
function processSelectQuery(query) {  
    parts = split(query, "\\s+FROM\\s+", 2)  
    if length(parts) != 2  
        print("Invalid SELECT query format.")  
    selectPart = trim(substring(parts[0], length("SELECT"))).trim()  
    if selectPart == ""
```

```

        columns = null
    else
        columns = split(selectPart, ",")
    fromPart = trim(parts[1])
    if contains(upper(fromPart), "WHERE") {
        tableAndConditionParts = split(fromPart, "\\s+WHERE\\s+", 2)
        tableName = trim(tableAndConditionParts[0])
        conditionParts = split(tableAndConditionParts[1], "=", 2)
        if length(conditionParts) != 2
            print("Invalid WHERE clause format.")
        conditionColumn = trim(conditionParts[0])
        conditionValue = trim(replaceAll(conditionParts[1], "^'(.*)'$", "$1"))
    else
        tableName = trim(fromPart)
        conditionColumn = null
        conditionValue = null
    print("Table Name: " + tableName)
    print("Columns: " + columns)
    print("Condition Column: " + conditionColumn)
    print("Condition Value: " + conditionValue)

```

### 5) Insert Query:

```

function processInsertQuery(query):
    parts = split(query.trim(), " ")
    if length(parts) < 5 or upper(parts[0]) != "INSERT" or upper(parts[1]) != "INTO":
        print("Invalid INSERT query format.")
    columnsPart = substring(query, indexOf(query, "(") + 1, indexOf(query, ")")).trim()
    valuesPart = substring(query, lastIndexOf(query, "(") + 1, lastIndexOf(query, ")")).trim()
    columns = split(columnsPart, ",")
    values = split(valuesPart, ",")

    // Clean values
    for i = 0 to length(values) - 1:
        values[i] = trim(replaceAll(values[i], "^'(.*)'$", "$1"))

    // Validate database and table existence
    if currentDatabase == null or isBlank(currentDatabase)
        return False
    if not contains(tables, tableName)
        return False

    // Validate and convert values
    for i = 0 to length(values) - 1:
        value = values[i]

```

```

attribute = attributes[i]
if not validateAndConvert(value, attribute):
    return False

// Check unique constraints
existingRecords = readRecords(tableName)
for record in existingRecords:
    for j = 0 to length(positions) - 1:
        if record[positions[j]] == uniqueValues[j]:
            print "Unique constraint violated!"
            return False

// Write to datastore
content = join(values, "#")
return write("DataStore/" + currentDatabase + "/" + tableName + ".txt", [content], False)

```

## 6) Update Query:

```

function processUpdateQuery(query):
    setIndex = indexOf(upper(query), "SET")
    whereIndex = indexOf(upper(query), "WHERE")
    if setIndex == -1 or whereIndex == -1 or setIndex >= whereIndex:
        print("Invalid UPDATE query format.")

    setPart = trim(substring(query, setIndex + 3, whereIndex))
    setParts = split(setPart, "=", 2)

    if length(setParts) != 2:
        print("Invalid SET clause format.")

    column = trim(setParts[0])
    value = trim(replaceAll(setParts[1], "^'(.*)'$", "$1"))

    wherePart = trim(substring(query, whereIndex + 5))
    whereParts = split(wherePart, "=", 2)

    if length(whereParts) != 2:
        print("Invalid WHERE clause format.")

    conditionColumn = trim(whereParts[0])
    conditionValue = trim(replaceAll(whereParts[1], "^'(.*)'$", "$1"))

// Validate database and table existence
if currentDatabase == null or isBlank(currentDatabase):
    return False

```

```

if not contains(tables, tableName):
    return False

// Update operation
loadTableData(tableName, currentDatabase)
for each row in rows:
    if conditionColumn is null or row[findColumnIndex(conditionColumn)] == conditionValue:
        convertValue(attributes[columnIndex], value)

        if attributes[columnIndex].isUnique():
            for each otherRow in rows:
                if otherRow != row and otherRow[columnIndex] == value:
                    return False
            row[columnIndex] = value
            updated = True
saveTableData(tableName, currentDatabase)

```

### 7) Delete Query:

```

function processDeleteQuery(query):
    parts = split(trim(query), "\\s+")

    if length(parts) < 4 or upper(parts[0]) != "DELETE" or upper(parts[1]) != "FROM":
        print("Invalid DELETE query format.")

    tableName = parts[2]

    conditionColumn = null
    conditionValue = null

    if contains(query, "WHERE"):
        wherePart = trim(substring(query, indexOf(query, "WHERE") + 5))
        whereParts = split(wherePart, "=")

        if length(whereParts) != 2:
            throw IllegalArgumentException("Invalid WHERE clause format.")

        conditionColumn = trim(whereParts[0])
        conditionValue = trim(whereParts[1])

// Validate database and table existence
if currentDatabase == null or isBlank(currentDatabase):
    return False
if not contains(tables, tableName):
    return False

```



```

// Delete operation
loadTableData(tableName, currentDatabase)
conditionIndex = findColumnIndex(conditionColumn)
if conditionIndex == -1:
    return False
for iterator = iterator(rows):
    row = next(iterator)
    if row[conditionIndex] == conditionValue:
        remove(iterator)
        deleted = True
if deleted:
    saveTableData(tableName, currentDatabase)
    return True
else:
    return False

```

### **8) Drop Query:**

```

function processDropQuery(query):
    parts = split(trim(query), "\\s+")

    if length(parts) < 3 or upper(parts[0]) != "DROP" or upper(parts[1]) != "TABLE":
        return False

    tableName = parts[2]

    // Validate database and table existence
    if currentDatabase == null or isBlank(currentDatabase):
        return False
    if not contains(tables, tableName):
        return False

    // Check if table file exists
    tableFilePath = "DataBase_FILE_PATH/" + currentDatabase + "/" + tableName + ".txt"
    if not isDirectoryFileExist(tableFilePath):
        return False

    // Read database file content
    content = read("DataBase_FILE_PATH/" + currentDatabase + "/" + currentDatabase + ".txt")

    // Identify and mark lines to delete
    deleteLineNumbers = []
    for i = 0 to length(content) - 1:

```

```

if startsWith(content[i], "@" + tableName):
    add(deleteLineNumbers, i)
    while i < length(content) and startsWith(content[i], "?"):
        add(deleteLineNumbers, i)
        i = i + 1
    break

// Update database file and delete table file
if write("DataBase_FILE_PATH/" + currentDatabase + "/" + currentDatabase + ".txt", content,
True):
    deleteDirectoryOrFile(tableFilePath)
    remove(tables, tableName)
    return True
else:
    return False

```

## 9) Authentication

- **function registerUser(user)**  
 if user.getUserID() exists in registeredUsers  
     return false  
 encode user credentials and security questions into content list  
 write content to USER\_PROFILE.txt file  
 add user to registeredUsers  
 return true if write operation successful, else false
- **function loginUser(userID, password)**  
 if userID does not exist in registeredUsers  
     return false  
 if hashed password for userID matches stored hashed password  
     if verifySecurityQuestion(user)  
         return true  
     else  
         return false  
 else  
     return false
- **function verifySecurityQuestion(user)**  
 randomly select any one of the security question from user's list  
 if user-provided answer matches stored answer for selected question  
     return true  
 else  
     return false

- **function hashPassword(password)**  
generate MD5 hash of password  
encode hash using Base64  
return encoded hash
- **function isUserExist(userID)**  
if registeredUsers is empty  
    loadUsers()  
iterate through registeredUsers  
    if user.getUserID() equals userID  
        return true  
return false
- **function loadUsers()**  
if isUserProfileFileLoaded is true  
    return registeredUsers  
clear registeredUsers  
read content from USER\_PROFILE.txt file  
decode content into users list  
set isUserProfileFileLoaded to true  
return registeredUsers

## Testcases

### Register & Authentication:

#### 1) Login with a user that does not exist:

```
Menu:
1. Login
2. Register
3. Exit
1
Enter username: testuser
Enter password: testpass
User Doesn't Exist!
```

Figure 3: Login with non-existent user details

- User does not exist in user's file

1	prashanth#wtoRtzsX1+TjENio10SknA==
2	@Cat?#orange
3	

Figure 4: User\_Profile.txt

#### 2) Register using a username that already exists:

```
Enter new username: prashanth
Enter password: daw
Enter security question: dwadaw
Enter answer: dwa
Do you want to add another security question? (yes/no):
no
UserID already exist!
```

Figure 5: User registration with existing user values

- Users file stays the same.

1	prashanth#wtoRtzsX1+TjENio10SknA==
2	@Cat?#orange
3	

Figure 6: User\_Profile.txt

3) Register using a new username:

```
2
Enter new username: testuser
Enter password: testpass
Enter security question: testquestion
Enter answer: yes
Do you want to add another security question? (yes/no):
no
```

Figure 7: User registration with new user values

- User\_Profile.txt got updated as shown below

```
1 prashanth#wtoRtzsX1+TjENio10SknA==
2 @Cat?#orange
3 testuser#F5rUXGziy5fPECniEgRugQ==
4 @testquestion#yes
5 |
```

Figure 8: User\_Profile.txt (updated)

4) Login using an existing username:

```
1
Enter username: testuser
Enter password: testpass
testquestion?
yes
Captcha: #o82q
Please enter above captcha: #o82q
Login successful!
```

Figure 9: Login with existent user details

5) Login using an existing username but with wrong password:

```
1
Enter username: testuser
Enter password: wrongpass
Incorrect Credentials!
```

Figure 10: Invalid credentials prompt in Login

6) Login with wrong security answer:

```
1
Enter username: testuser
Enter password: testpass
testquestion?
no
Wrong Security Answer!
```

Figure 11: Invalid security answer prompt in Login

7) Login using the wrong captcha combination:

```
1
Enter username: testuser
Enter password: testpass
testquestion?
yes
Captcha: ojszs
Please enter above captcha: opgm
Invalid captcha
```

Figure 12: Invalid captcha prompt in Login

Queries:

1) Creating and using a database:

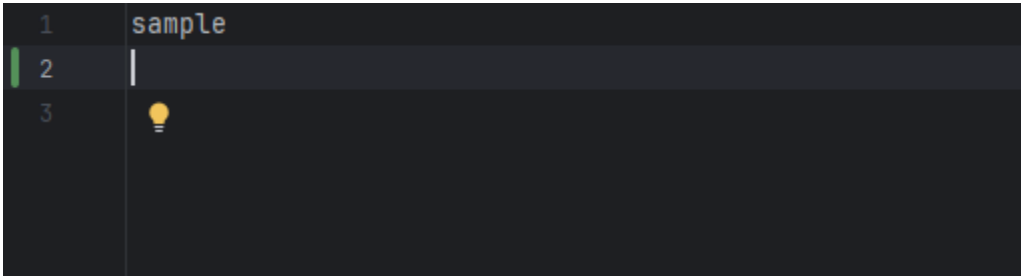


Figure 13: Metadata.txt before creating a new DB

```
Enter your SQL queries (type 'EXIT' to end):
SQL> CREATE DATABASE test;
SQL> USE DATABASE test;
```

Figure 14: Creating and using a DB

1	sample
2	test
3	

Figure 15: Metadata.txt after creating the test database

```

v DataStore
  > db
  > sample
  metadata.txt
  User_Profile.txt

```

Figure 16: DataStore directory before creating the test database

```

v DataStore
  > db
  > sample
  v test
    test.txt
  metadata.txt
  User_Profile.txt

```

Figure 17: DataStore directory after creating the test database

```

> static members of Main
v manager = {Manager@1106}
  > databases = {ArrayList@1292}
  > currentDatabase = "test"
  > database = {Database@1294}
  > DataBase_FILE_PATH = "DataStore"

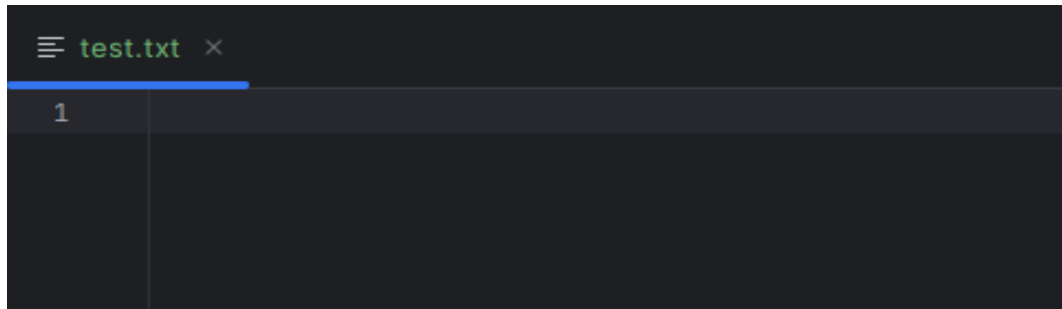
```

Figure 18: The test database is chosen after the "USE" command

## 2) Creating a table:

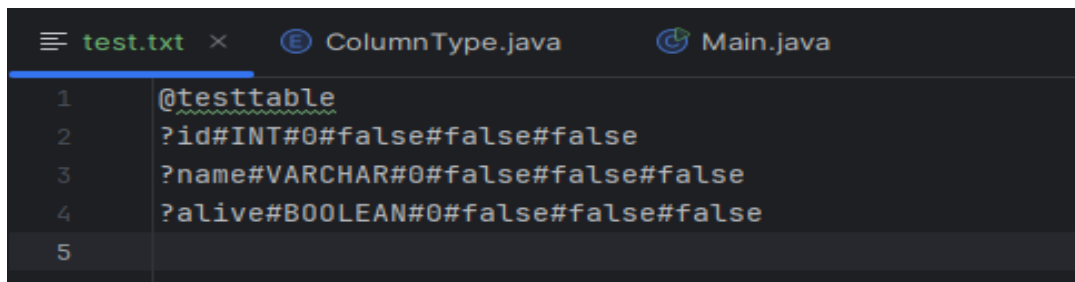
```
Enter your SQL queries (type 'EXIT' to end):
SQL> USE DATABASE test;
SQL> CREATE TABLE testtable (id INT,name VARCHAR, alive BOOLEAN);
SQL>
```

Figure 19: Creating a table



The screenshot shows a text editor window titled 'test.txt'. The file is empty, with only a line number '1' visible in the left margin.

Figure 20: test.txt database file is empty before creating a table



The screenshot shows a text editor window with three tabs: 'test.txt', 'ColumnType.java', and 'Main.java'. The 'test.txt' tab is active and contains the following text:

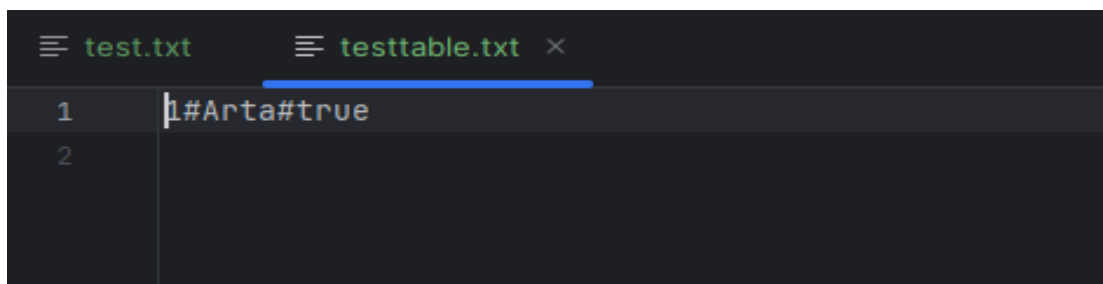
	@testtable
1	
2	?id#INT#0#false#false#false
3	?name#VARCHAR#0#false#false#false
4	?alive#BOOLEAN#0#false#false#false
5	

Figure 21: test.txt file is updated after the table creation

## 3) Insert into the table:

```
SQL> INSERT INTO testtable VALUES (1,Arta,true);
SQL>
```

Figure 22: Inserting values into the table



The screenshot shows a text editor window with two tabs: 'test.txt' and 'testtable.txt'. The 'testtable.txt' tab is active and contains the following text:

	1#Arta#true
1	
2	

Figure 23: New record is added to the table txt file

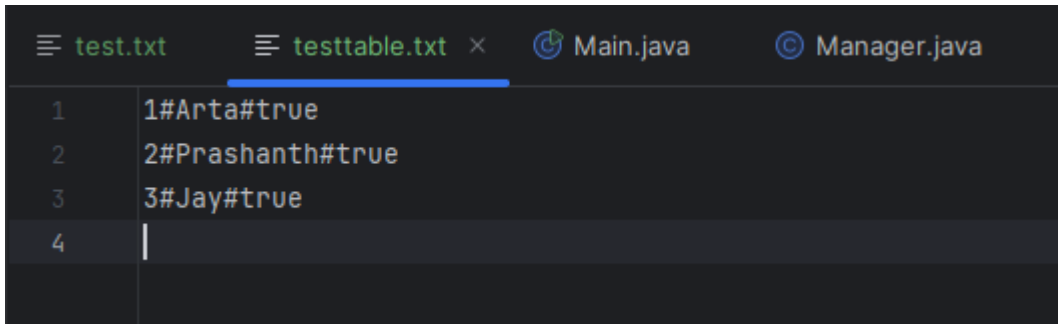


4) Add a table with a name that already exists in the database:

```
SQL> CREATE TABLE testtable (id INT,name VARCHAR, alive BOOLEAN);
Table 'testtable' already exists in database 'test'.
```

Figure 24: Table already exists error

5) Select from a table without a WHERE clause:



1	1#Arta#true
2	2#Prashanth#true
3	3#Jay#true
4	

Figure 25: Records in table txt file

```
SQL> SELECT * FROM testtable;
Query result:
id: 1   name: Arta   alive: true
id: 2   name: Prashanth alive: true
id: 3   name: Jay    alive: true
```

Figure 26: Displaying output from SELECT query

6) Select from a table with a WHERE clause:

```
SQL> SELECT name FROM testtable WHERE id = 1;
Query result:
name: Arta
```

Figure 27: Displaying output from SELECT query (with WHERE clause)

7) Update a table with a WHERE clause:

```
SQL> UPDATE testtable SET alive = false WHERE name = Arta;
Table 'testtable' updated successfully.
SQL>
```

Figure 28: Updating a table

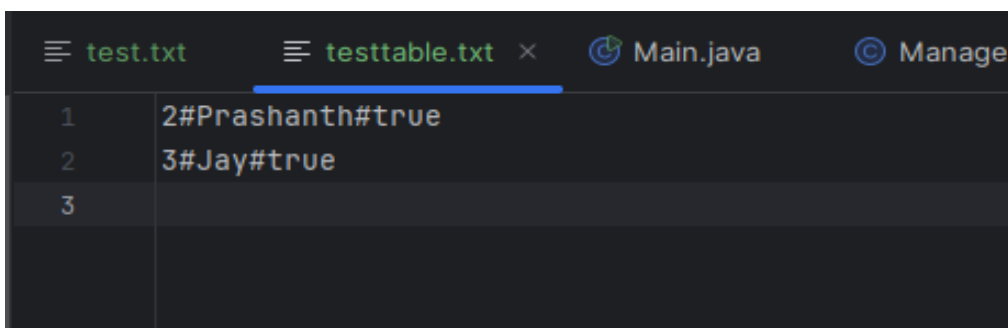
```
SQL> SELECT * FROM testtable;
Query result:
id: 1   name: Arta   alive: false
id: 2   name: Prashanth alive: true
id: 3   name: Jay    alive: true
```

Figure 29: Displaying output after updation

#### 8) Deleting a record from the table:

```
SQL> DELETE FROM testtable WHERE name = Arta;
testtable
name
Arta
Row(s) deleted from table 'testtable'.
SQL> SELECT * FROM testtable;
Query result:
id: 2   name: Prashanth alive: true
id: 3   name: Jay    alive: true
```

Figure 30: Displaying output after deletion



1	2#Prashanth#true
2	3#Jay#true
3	

Figure 31: The record is deleted from the table.txt file after the operation

#### 9) Dropping a table:

```
SQL> DROP TABLE testtable;
Directory deleted successfully
Table 'testtable' dropped successfully.
```

Figure 32: Success prompt after dropping a table

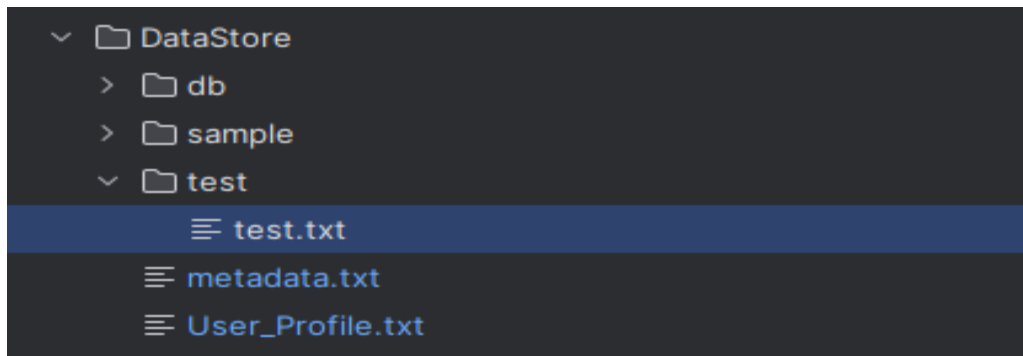


Figure 33: Table text file is removed from the Datastore directory

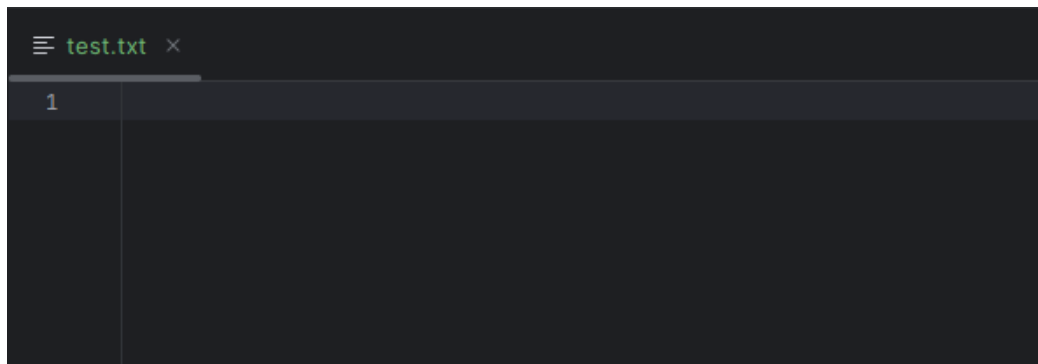


Figure 34: Table information is removed from the database .txt file

#### 10) SELECT, UPDATE & DELETE from a table that does not exist in the database:

```

Table 'testtable' doesn't exist.
SQL> SELECT * FROM testtable;
Table 'testtable' doesn't exist in database 'test'.

SQL> UPDATE testtable SET name = Arta WHERE id = 2;
Table 'testtable' doesn't exist in database 'test'.

SQL> DELETE FROM testtable WHERE id = 2;
Table 'testtable' doesn't exist in database 'test'.

```

Figure 35: Doesn't exist in the database prompt for CRUD operation

## Meeting records

<b>Date</b>	<b>Time</b>	<b>Attendees</b>	<b>Agenda</b>	<b>Meeting type</b>	<b>Meeting link</b>
09/06/2024	7:00 - 7:30 PM	Arta, Jay, Prashanth	Thorough analysis of project requirement	Online	<a href="https://bit.ly/3zm7VIE">https://bit.ly/3zm7VIE</a>
14/06/2024	7:30 – 8:00 PM	Arta, Jay, Prashanth	Finalizing project structure	Online	<a href="https://bit.ly/3XFhiHf">https://bit.ly/3XFhiHf</a>
21/06/2024	5:00 – 5:25 PM	Arta, Jay, Prashanth	Analyzing code and merging branches	Online	<a href="https://bit.ly/3XOa6sj">https://bit.ly/3XOa6sj</a>
26/06/2024	11:00 – 11:20 PM	Arta, Jay, Prashanth	Discussion on progress so far	Online	<a href="https://bit.ly/4eIGhzu">https://bit.ly/4eIGhzu</a>

# 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](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

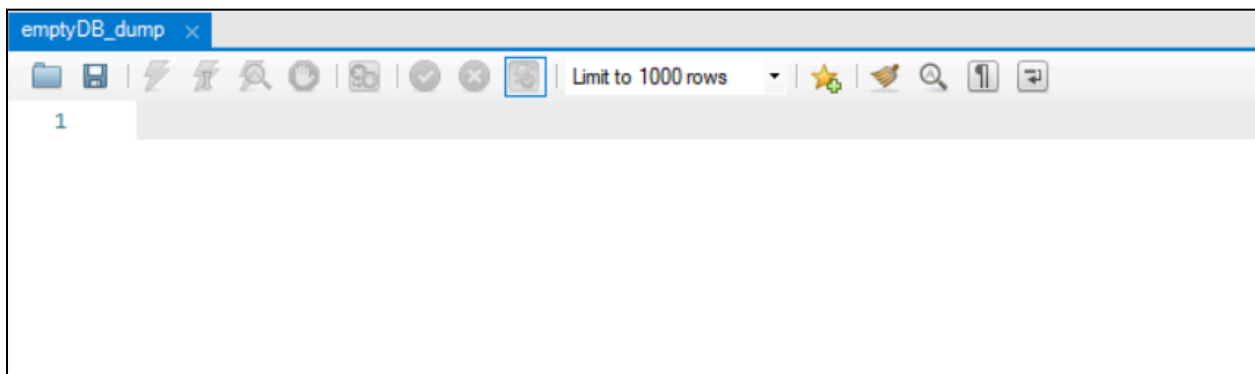


Fig 2: Dump file for empty DB

#### 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

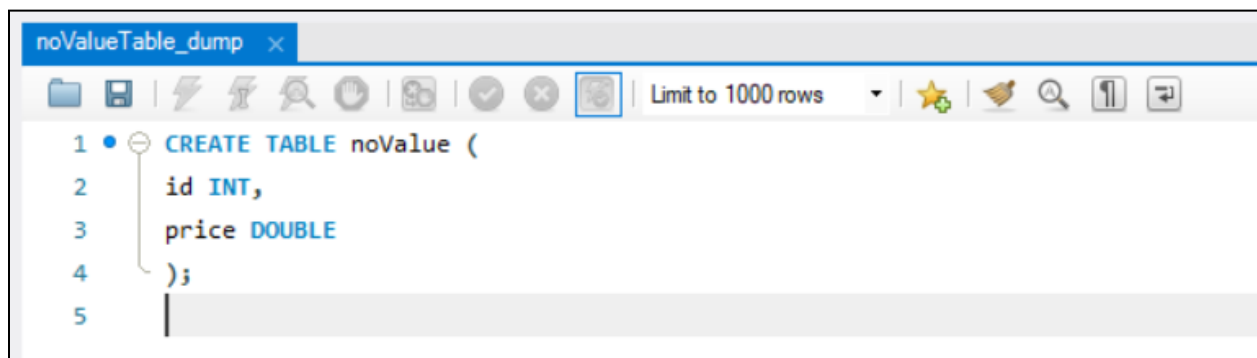


Fig 5: Dump file with single table (w/o values)

### 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

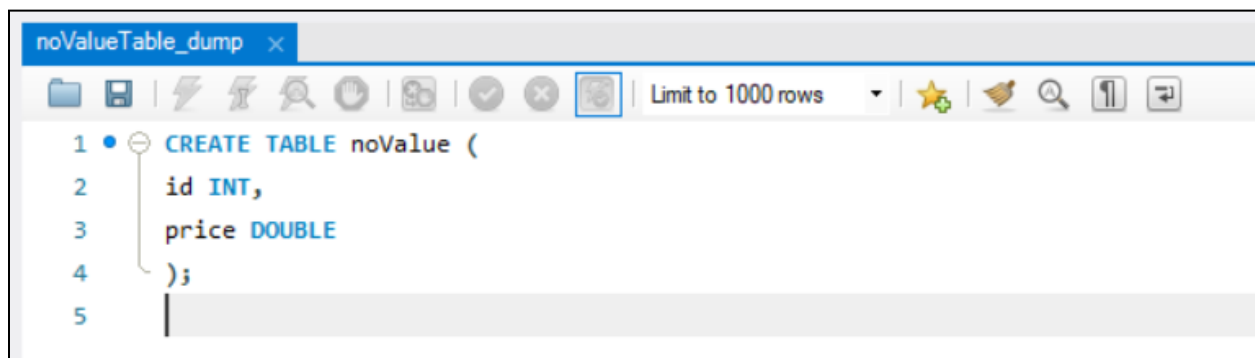


Fig 11: Dump file for table w/o foreign key

##### 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

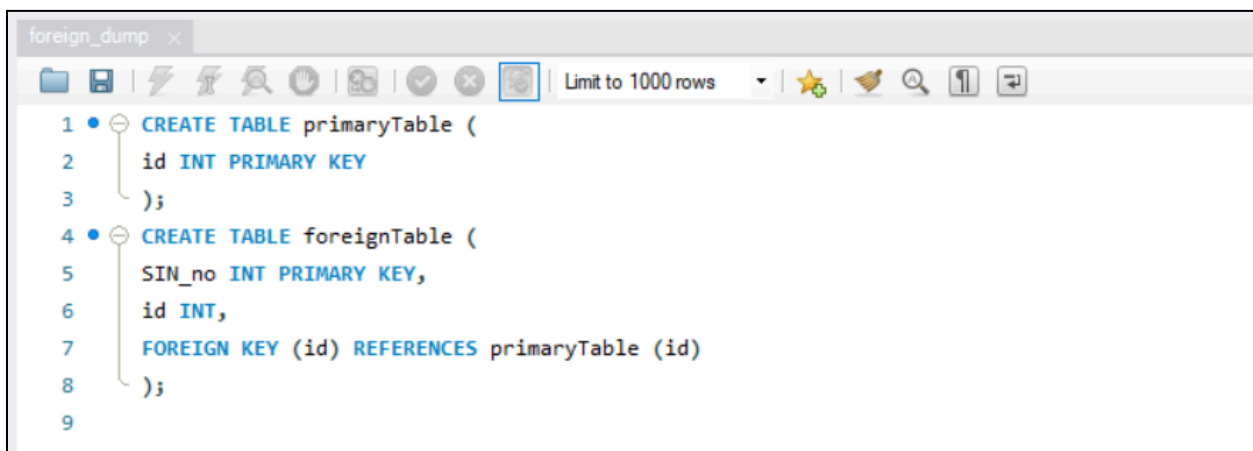


```

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

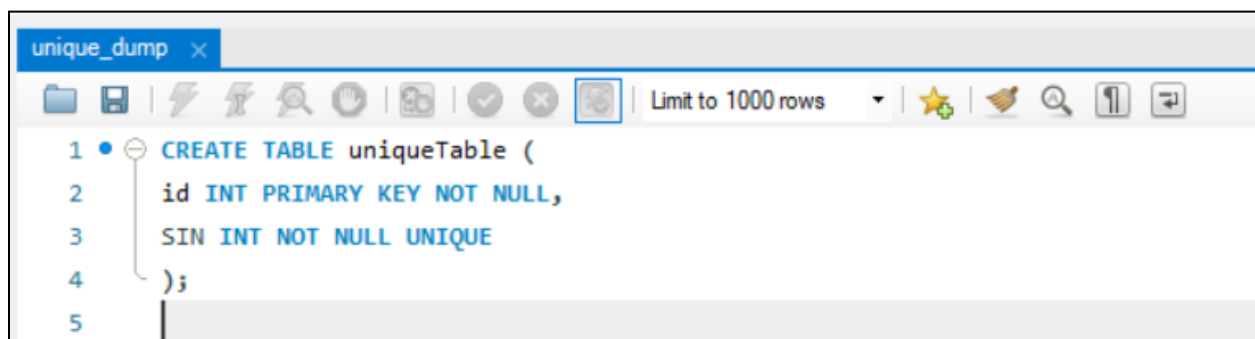


Fig 18: Dump file for a table w/ unique and not null constraints

## 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:

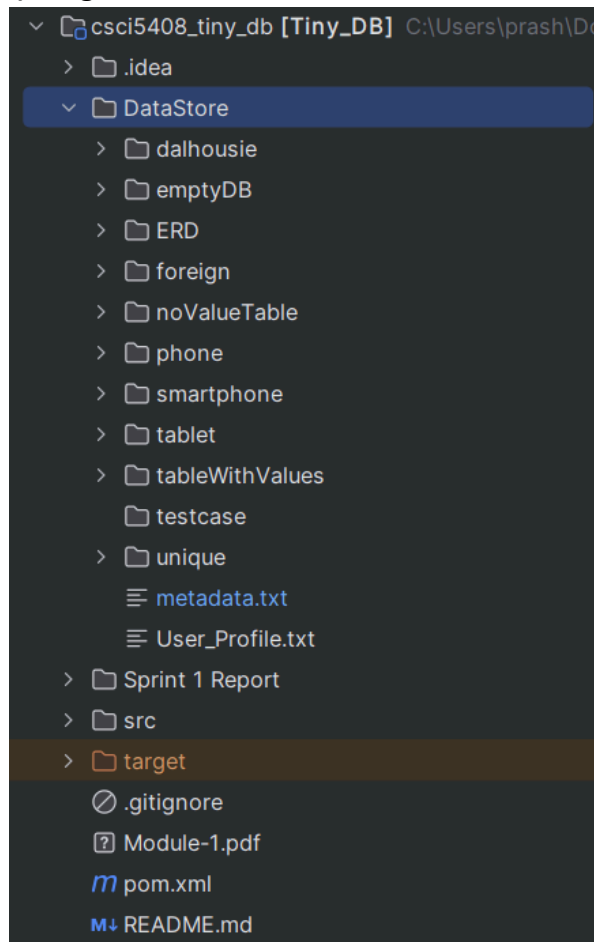


Fig 20: No log found in directory before operation

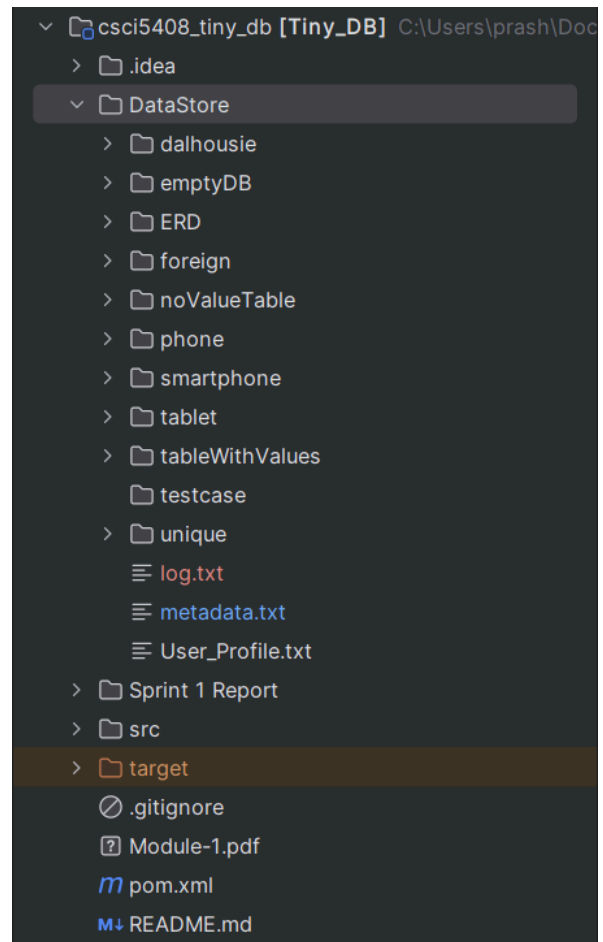


Fig 21: Log found in directory after operation

**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

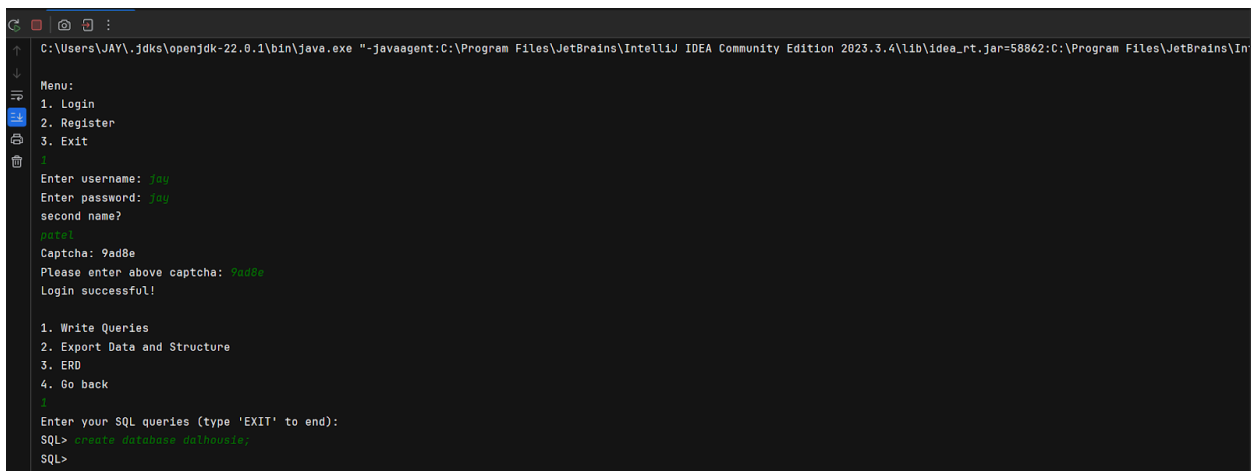
## 7) User-related operations:

[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: joy
Enter password: joy
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	<a href="https://bit.ly/3Y35kY6">https://bit.ly/3Y35kY6</a>
10/07/2024	11:30 – 11:45 AM	Arta, Jay, Prashanth	Weekly standup meeting to measure progress	Online	<a href="https://bit.ly/3WkvXGx">https://bit.ly/3WkvXGx</a>

# **CSCI 5408**

## **DATA MANAGEMENT AND WAREHOUSING**

### **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](https://git.cs.dal.ca/prashanthv/csci5408_tiny_db.git)

## Table of Contents

Pseudocode.....	3
Test cases.....	5

## **Pseudocode:**

### **TransactionManager**

#### **Initialization:**

```
function TransactionManager():  
    set inTransaction to false
```

Set Current Database Name:

```
function setCurrentDBName(currentDBName):  
    set this.currentDBName to currentDBName  
    set originalDatabasePath to "DataStore/" + currentDBName
```

#### **Get Instance:**

```
function getInstance():  
    if instance is null:  
        lock TransactionManager class:  
            if instance is null:  
                set instance to new TransactionManager()  
    return instance
```

#### **Start Transaction:**

```
function startTransaction():  
    if inTransaction:  
        print "Transaction already in progress"  
        throw CustomException("Transaction already in progress")  
  
    if currentDBName is null:  
        print "No database set currently"  
        throw CustomException("No database set currently")  
  
    set tempDatabasePath to "DataStore/" + currentDBName + "_temp"  
    call copyDirectory with originalDatabasePath and tempDatabasePath  
  
    set inTransaction to true
```

#### **Commit Transaction:**

```
function commitTransaction():  
    if not inTransaction:  
        throw CustomException("No transaction in progress")
```

```
    call deleteDirectory with originalDatabasePath
    move tempDatabasePath to originalDatabasePath with
StandardCopyOption.ATOMIC_MOVE
    set inTransaction to false
```

### **Rollback Transaction:**

```
function rollbackTransaction():
    if not inTransaction:
        throw CustomException("No transaction in progress")

    call deleteDirectory with tempDatabasePath
    set inTransaction to false
```

### **Copy Directory:**

```
function copyDirectory(source, target):
    for each src in walk source:
        set dest to target.resolve(source.relativeize(src))
        try:
            copy src to dest with StandardCopyOption.REPLACE_EXISTING
        catch IOException as e:
            print stack trace of e
```

### **Delete Directory:**

```
function deleteDirectory(path):
    for each file in walk path in reverse order:
        delete file
```

### **Get Database Path:**

```
function getDatabasePath():
    return inTransaction ? tempDatabasePath : originalDatabasePath
```

## Test Cases:

### 1. Start a transaction

```
SQL> use test;  
SQL> start transaction;  
SQL>
```

Figure 1: start a transaction in the console

```
[Successful] [2024-07-27 17:15:13.342] User 'testuser' logged in successfully.  
[Successful] [2024-07-27 17:15:20.362] Using database 'test'.  
[Successful] [2024-07-27 17:15:25.979] start transaction; executed in 10 ms. State= Database: test| Tables:
```

Figure 2: transaction successfully started

### 2. Start a transaction before using a database

```
SQL> start transaction;  
No database set currently  
SQL>
```

Figure 3: Starting a transaction without using a database in the console

```
[Successful] [2024-07-27 17:09:51.855] User 'testuser' logged in successfully.  
[Failed] [2024-07-27 17:11:59.172] Error processing query 'start transaction; ms: No database set currently
```

Figure 4: Logs show that the transaction cannot start because no database is selected

### 2. Start a transaction while another transaction is active

```
SQL> use test;  
SQL> start transaction;  
SQL> start transaction;  
Transaction already in progress
```

Figure 5: start a transaction in the console while another transaction is running

```
[Successful] [2024-07-27 17:15:20.362] Using database 'test'.  
[Successful] [2024-07-27 17:15:25.979] start transaction; executed in 10 ms. State= Database: test| Tables:  
[Failed] [2024-07-27 17:17:14.493] Error processing query 'start transaction; ms: Transaction already in progress
```

Figure 6: Logs show that the new transaction cannot start because another transaction is in progress



### 3. Commit without starting a transaction

```
Enter your SQL queries (type 'EXIT' to end):
SQL> use test;
SQL> commit;
SQL>
```

Figure 7: using commit without starting a transaction in the console

```
[Successful] [2024-07-27 17:09:51.855] User 'testuser' logged in successfully.
[Successful] [2024-07-27 17:19:50.198] Using database 'test'.
[Failed] [2024-07-27 17:19:52.951] Error processing query 'commit; ms: No transaction in progress
```

Figure 8: Logs show that commit cannot be done because no transaction is in progress

### 4. Rollback without starting a transaction

```
Enter your SQL queries (type 'EXIT' to end):
SQL> use test;
SQL> rollback;
SQL>
```

Figure 9: using rollback without starting a transaction in the console

```
[Successful] [2024-07-27 17:09:51.855] User 'testuser' logged in successfully.
[Successful] [2024-07-27 17:21:11.102] Using database 'test'.
[Failed] [2024-07-27 17:21:14.163] Error processing query 'rollback; ms: No transaction in progress
```

Figure 10: Logs show that rollback cannot be done because no transaction is in progress

### 5. Full transaction process with commit

```
Enter your SQL queries (type 'EXIT' to end):
SQL> use test;
SQL> start transaction;
SQL> create table test-table (id INT,name varchar);
Enter Field Name: exit
SQL> insert into test-table values (1,Arta);
SQL> update test-table set name = Prashanth where id = 1;
SQL> create table test-table2 (id INT);
Enter Field Name: exit
SQL> drop table test-table2;
Deleted successfully
SQL> insert into test-table values (2,jay);
SQL> delete from test-table where id = 2;
SQL>
```

Figure 11: running a transaction in the console with Create, Insert, update, delete, and drop commands

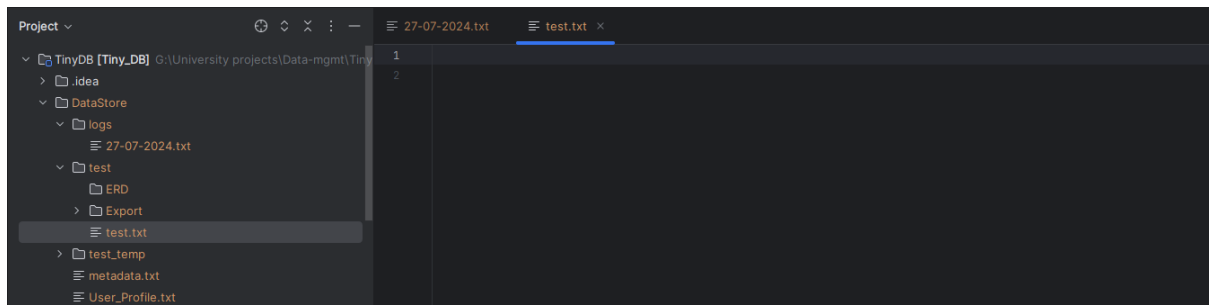


Figure 12: files are not changed before committing the transaction

```

Enter your SQL queries (type 'EXIT' to end):
SQL> use test;
SQL> start transaction;
SQL> create table test-table (id INT,name varchar);
Enter Field Name: exit
SQL> insert into test-table values (1,Arta);
SQL> update test-table set name = Prashanth where id = 1;
SQL> create table test-table2 (id INT);
Enter Field Name: exit
SQL> drop table test-table2;
Deleted successfully
SQL> insert into test-table values (2,jay);
SQL> delete from test-table where id = 2;
SQL> commit;

```

Figure 13: the transaction is committed

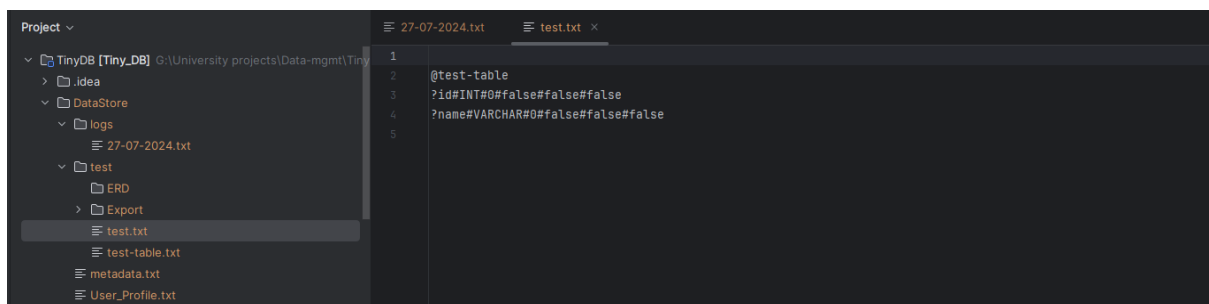


Figure 14: changes appeared on the database file

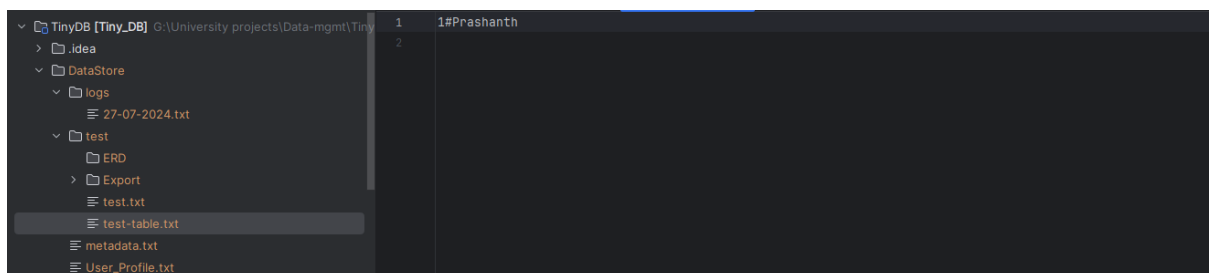


Figure 15: A new table is created with the latest changes

## 6. Full transaction process with rollback

```
Enter your SQL queries (type 'EXIT' to end):
SQL> use test;
SQL> start transaction;
SQL> create table test-table2 (id int);
Enter Field Name: exit
SQL> drop table test-table;
Deleted successfully
SQL> insert into test-table2 values (1);
SQL> update test-table2 set id = 2 where id = 1;
SQL>
```

Figure 16: running a transaction in the console with Create, Insert, update, delete, and drop commands

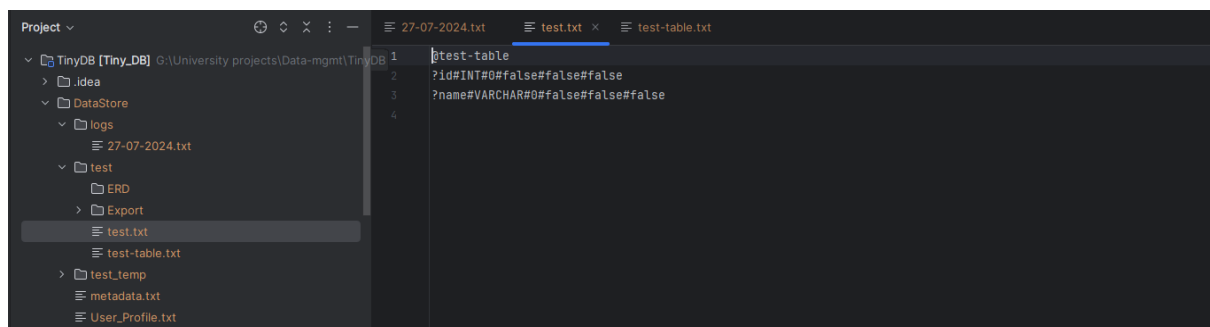


Figure 17: files are not changed before rolling back the transaction

```
Enter your SQL queries (type 'EXIT' to end):
SQL> use test;
SQL> start transaction;
SQL> create table test-table2 (id int);
Enter Field Name: exit
SQL> drop table test-table;
Deleted successfully
SQL> insert into test-table2 values (1);
SQL> update test-table2 set id = 2 where id = 1;
SQL> rollback;
```

Figure 18: the transaction is rolled back

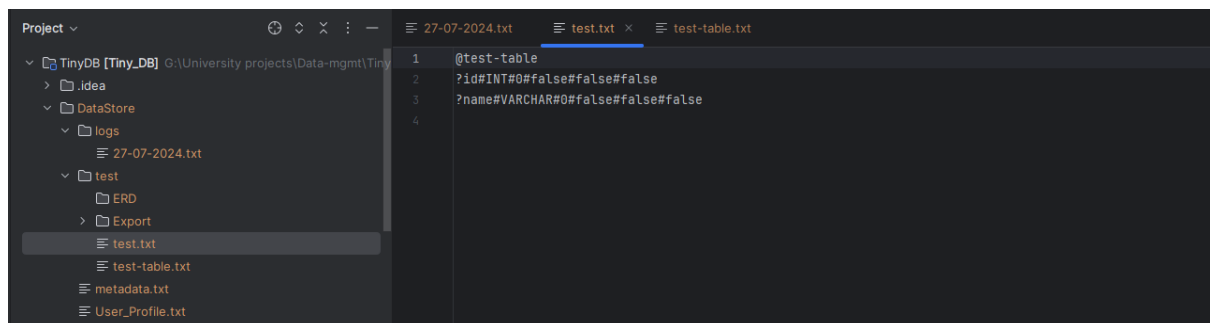


Figure 19: no changes are made in database files and no new tables are created

## 7. Stop the program while a transaction is running

```
Enter your SQL queries (type 'EXIT' to end):
SQL> use test;
SQL> start transaction;
SQL> create table test-table2 (name varchar);
Enter Field Name: exit
SQL>
```

Figure 20: started a transaction and created a table without committing or rolling back

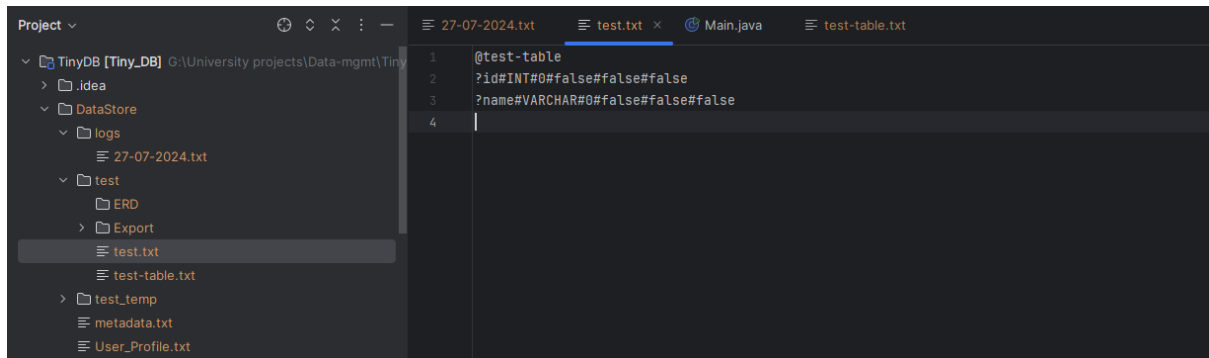


Figure 21: state of the database files before stopping the program

After stopping and running the program again:

```
Enter your SQL queries (type 'EXIT' to end):
SQL> use test;
SQL> commit;
SQL> rollback;
SQL> start transaction;
SQL>
```

Figure 22: trying to commit, rollback and start a new transaction in the console

```
[Successful] [2024-07-27 17:46:50.815] User 'testuser' logged in successfully.
[Successful] [2024-07-27 17:47:01.792] Using database 'test'.
[Failed] [2024-07-27 17:47:05.078] Error processing query 'commit; ms: No transaction in progress
[Failed] [2024-07-27 17:47:08.469] Error processing query 'rollback; ms: No transaction in progress
[Successful] [2024-07-27 17:47:15.24] start transaction; executed in 4 ms. State= Database: test| Tables: (test-table, 1 records),
```

Figure 23: logs show that there is no transaction in progress and the new transaction has successfully started

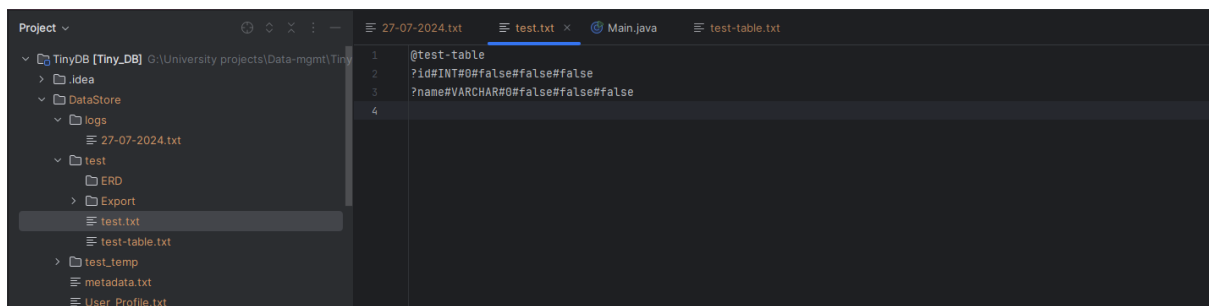


Figure 23: state of the database files after stopping and running the program again