CSCI 5408

DATA MANAGEMENT AND WAREHOUSING

SPRINT -1 DOCUMENT

Group-3
Jay Alpeshkumar Patel (B00969013)
Manish Shankar Jadhav (B00969328)

GitLab Link: https://git.cs.dal.ca/patel38/csci_5408_s24_3/-/tree/main?ref_type=heads

Table of Contents

Background Research

A database management system (DBMS) is a software system that creates and manages databases [1]. In this project, we are creating a DBMS using Java with the help of data structure concepts.

For Sprint1, we have implemented Module1, Module2 and Module3. One of the main objectives of DBMS is storing data, so we have analyzed this and decided to go with a custom file format for storing information in Txt files.

We did our research and came to this conclusion. Every database created will be inside the Tiny DB directory and tables in those databases will be inside those directories. Each table will have two files, one is a schema file, and the other is a data file. The schema file will store the metadata information and the data file will store the data related to the table.

Then we did research about the data structure we will need for storing data. In our code, we use a combination of a LinkedHashMap and a List for data storage. The reason for using LinkedHashMap is to ensure that the data is stored in a linear order and maintains the appending order when retrieving the data.

Our main data structure is Map<String, LinkedHashMap<String, List<String>>>, where:

- The first key represents the table name.
- The LinkedHashMap stores the columns, with the inner key representing the column name and the associated List<String> holding the column values.

For the table schema, we use List<Map<String, String>>, where:

- The key represents the column name.
- The value represents the schema of the column.

This structure is intended for temporary storage.

We treat each command as a transaction. After each command we perform changes to our data structure and if it is committed or when the program ends. This design has really helped us solve Module 2 and Module 3 together since it cleared up the thought process of storing the data part.

The background research really played a huge role in setting up a base which made the implementation parts much more easier.

Architecture Diagram

1) Overall Architecture Diagram

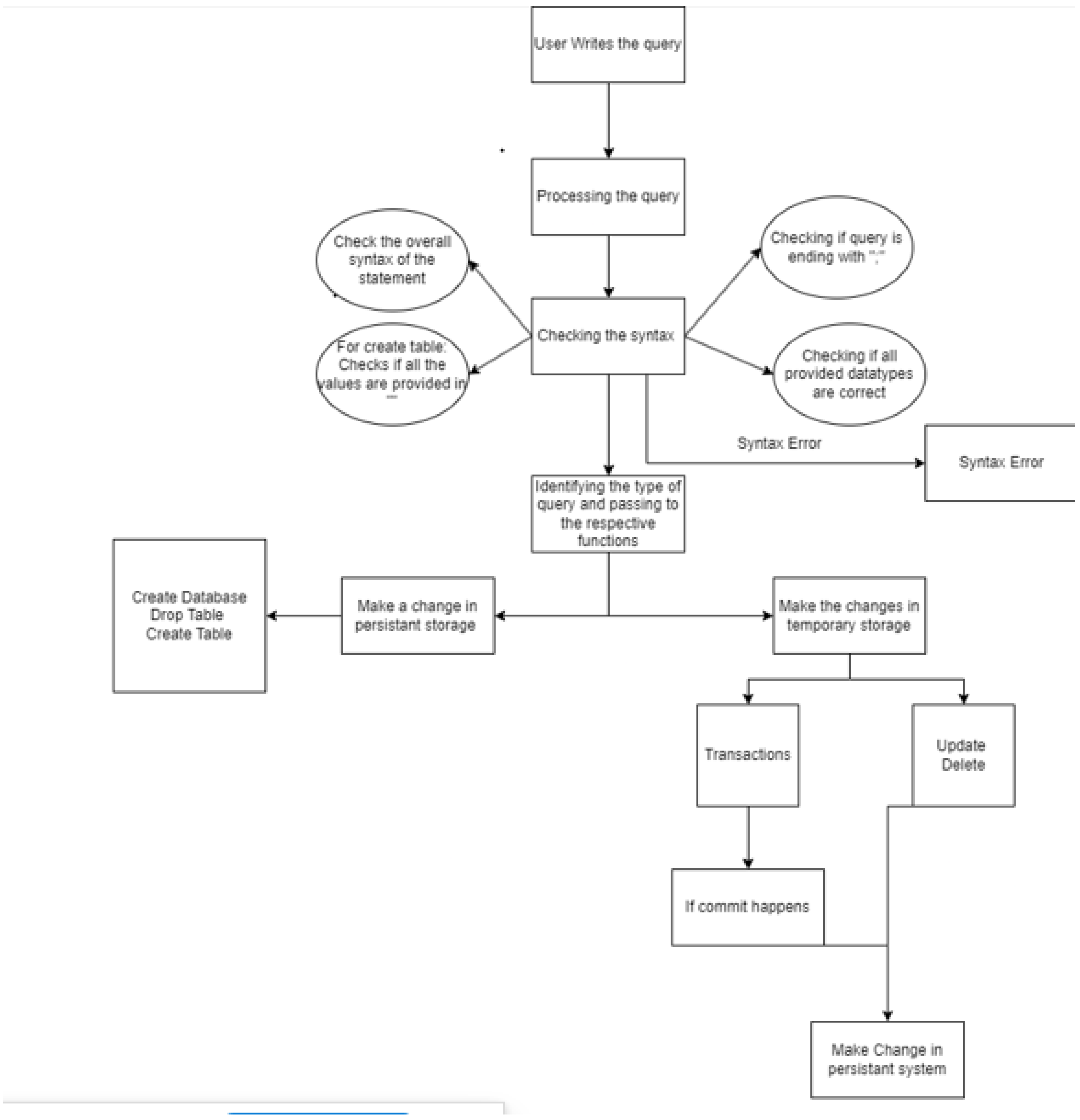

User Writes the query
Processing the query
Check the overall syntax of the statement
For create table: Checks if all the values are provided in
Checking the syntax
Checking if query is ending with ";"
Checking if all provided datatypes are correct
Syntax Error
Syntax Error
Identifying the type of query and passing to the respective functions
Create Database Drop Table Create Table
Make a change in persistant storage
Make the changes in temporary storage
Transactions
Update Delete
If commit happens
Make Change in persistant system

Figure 1 Overall architecture

2) Create Database

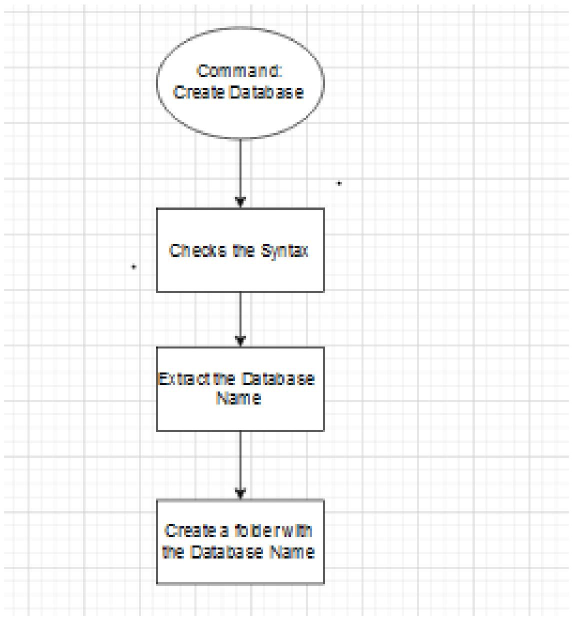

Command: Create Database
Checks the Syntax
Extract the Database Name
Create a folder with the Database Name

Figure 2 : Create Database Architecture.

3) Use Database

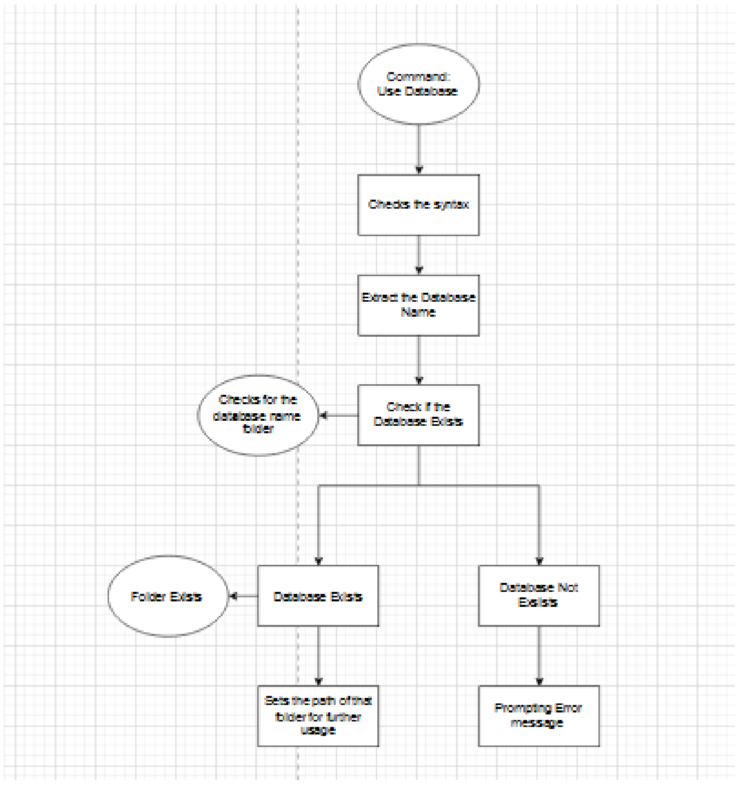
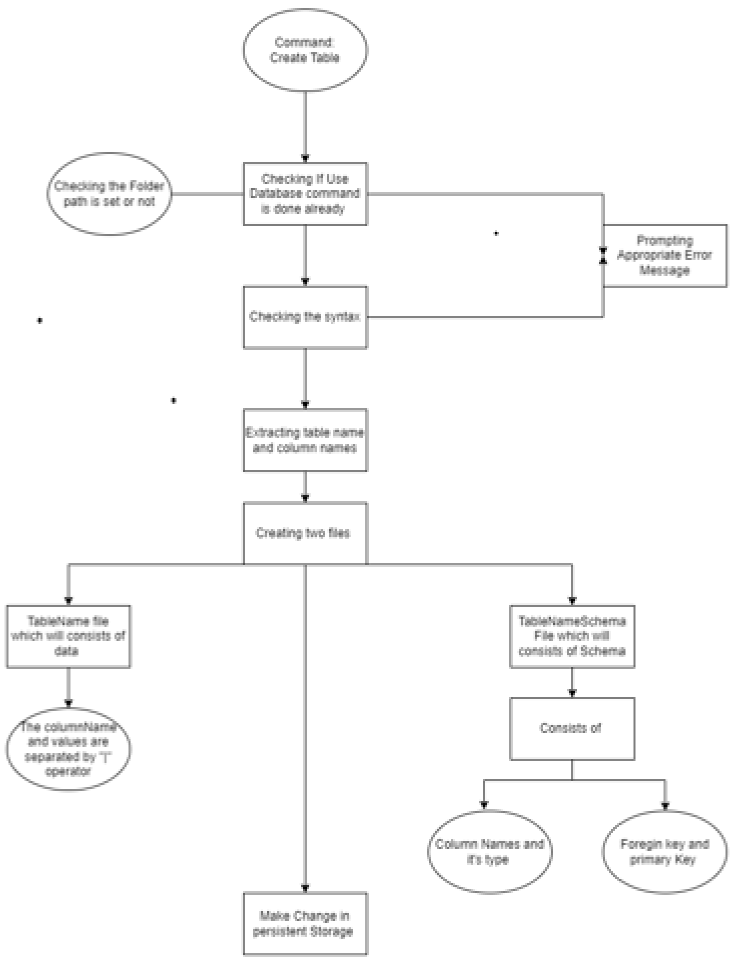

Command: Use Database
Checks the syntax
Extract the Database Name
Check if the Database Exists
Checks for the database name folder
Database Exists
Database Not Exists
Folder Exists
Sets the path of that folder for further usage
Prompting Error message

Figure 3 : Use Database Architecture.

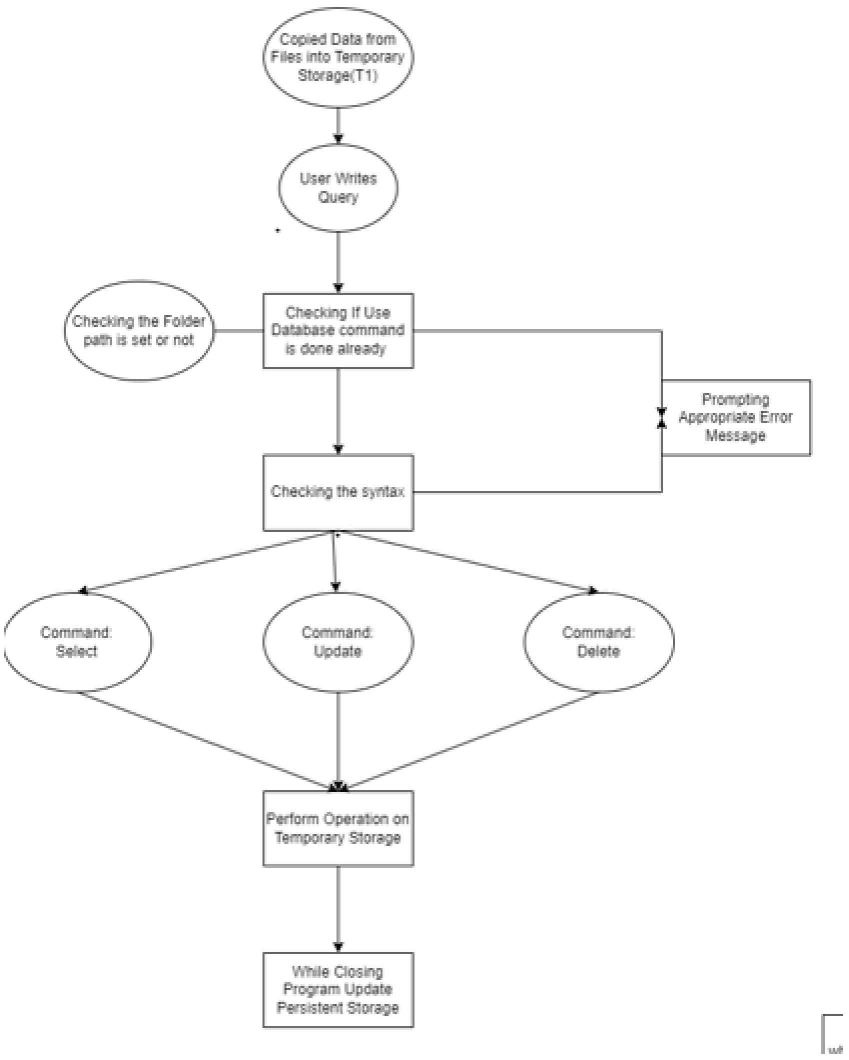
4) Create Table



Command: Create Table
Checking the Folder path is set or not
Checking If Use Database command is done already
Prompting Appropriate Error Message
Checking the syntax
Extracting table name and column names
Creating two files
TableName file which will consists of data
The columnName and values are separated by "|" operator
TableNameSchema File which will consists of Schema
Consists of
Column Names and it's type
Foregin key and primary Key
Make Change in persistent Storage

Figure 4 : Create Table Architecture.

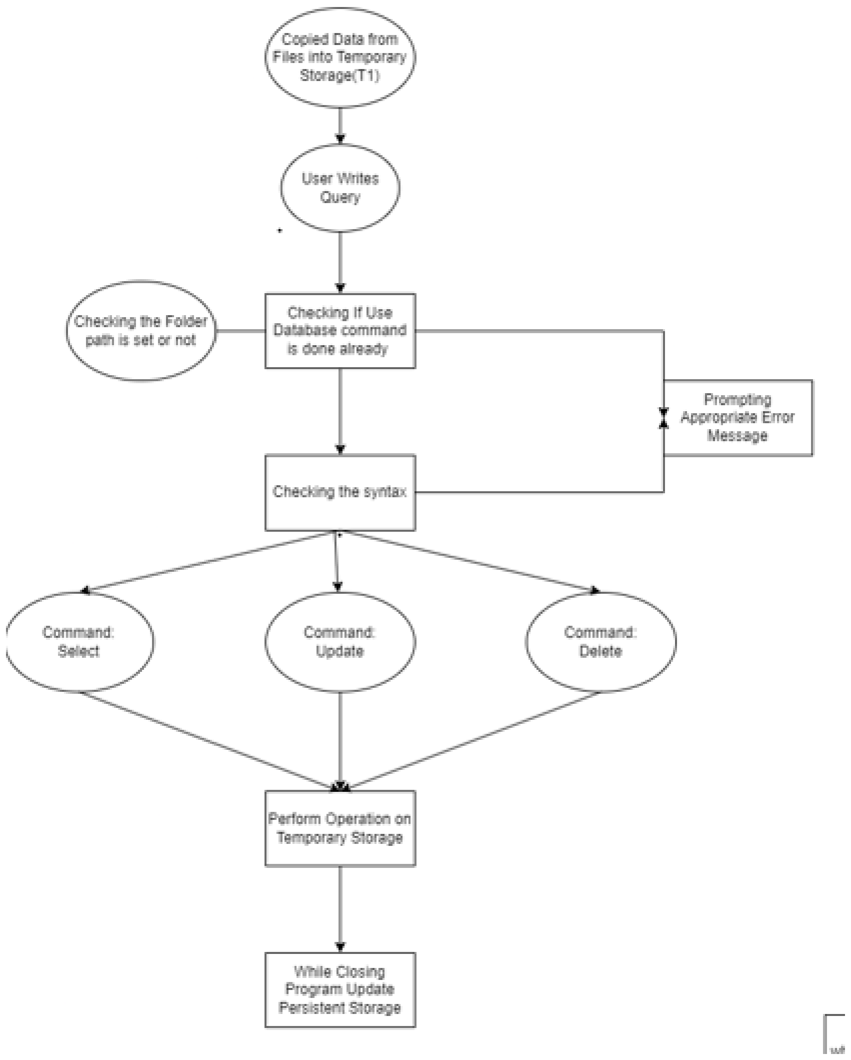
5) Update, Delete, and Select Commands



Copied Data from Files into Temporary Storage(T1)
User Writes Query
Checking the Folder path is set or not
Checking If Use Database command is done already
Prompting Appropriate Error Message
Checking the syntax
Command: Select
Command: Update
Command: Delete
Perform Operation on Temporary Storage
While Closing Program Update Persistent Storage

Figure 5: Update, Delete, and Select commands Architecture.

6) Drop Database and Drop Table Commands



Copied Data from Files into Temporary Storage(T1)
User Writes Query
Checking the Folder path is set or not
Checking If Use Database command is done already
Prompting Appropriate Error Message
Checking the syntax
Command: Select
Command: Update
Command: Delete
Perform Operation on Temporary Storage
While Closing Program Update Persistent Storage

Figure 6: Drop Database and Drop table commands Architecture.

Pseudocode

Create Database:

```
FUNCTION createDatabase(query)
   CALL queryProcessor(query)

FUNCTION queryProcessor(query)
  //Checking The Syntax of the Query
   IF checkSyntax(query) IS FALSE
      RETURN "Syntax Error"
   // Return The database name from the given query
   databaseName = extractDatabaseName(query)
   // CHECK IF folder WITH name databaseName EXISTS IN "TinyDB" folder
   IF databaseExists(databaseName)
      RETURN "Database already created"
   createDatabaseFolder(databaseName)
   RETURN "Database created successfully"

FUNCTION createDatabaseFolder(databaseName)
   CREATE folder WITH name databaseName IN "TinyDB" folder
```

Use Database:

```
FUNCTION useDatabase(query)
   CALL queryProcessor(query)

FUNCTION queryProcessor(query)
  //Checking The Syntax of the Query
   IF checkSyntax(query) IS FALSE
      RETURN "Syntax Error"
    // Return The database name from the given query
   databaseName = extractDatabaseName(query)
   IF NOT databaseExists(databaseName)
      RETURN "Database not present"
   SET currentDatabasePath TO databaseName
   RETURN "Database set to " + databaseName

FUNCTION databaseExists(databaseName)
   // CHECK IF folder WITH name databaseName EXISTS IN "TinyDB" folder
```

Create Table:

```
FUNCTION createTable(query)
  CALL queryProcessor(query)

FUNCTION queryProcessor(query)
   // Logic to check SQL syntax for data types
  // Return TRUE if SQL syntax is correct, otherwise FALSE
  IF checkSyntax(query) IS FALSE
    RETURN "Syntax Error"
  // Pseudo code logic to extract table name from query
  tableName = extractTableName(query)
  // Logic to extract columns and data types from query
  // Return a list of column names and data types
  columnsAndDataTypes = extractColumnsAndDataTypes(query)
  CALL createTableFile(tableName, columnsAndDataTypes)
  CALL createTableSchemaFile(tableName, columnsAndDataTypes)
  RETURN "Table created successfully"


FUNCTION createTableFile(tableName, columnsAndDataTypes)
  CREATE file WITH name tableName IN currentDatabasePath
  WRITE column names SEPARATED BY "|" TO file

FUNCTION createTableSchemaFile(tableName, columnsAndDataTypes)
  CREATE file WITH name tableName_schema IN currentDatabasePath
  WRITE primary and foreign keys TO file
```

Insert Data:

```
FUNCTION insertData(query)
 // Pseudo code logic to copy database to a temporary structure
  // Return the temporary structure
  temporaryDatabase = copyDatabaseToTemporaryStructure(currentDatabasePath)
  CALL queryProcessor(query, temporaryDatabase)

FUNCTION queryProcessor(query, temporaryDatabase)
  // Checking the syntax of query
  IF checkSyntax(query) IS FALSE
    RETURN "Syntax Error"
  //Extracting Table Name from the query
  tableName = extractTableName(query)
  // Logic to extract values from the query
  // Return the extracted values
  values = extractValues(query)
  // Checks file exists in current database Path
  IF NOT tableExists(tableName, currentDatabasePath)
    RETURN "Table not present"
   // Getting the schema from tableName_Schema file
  databaseSchemaPath = getDatabaseSchemaPath(tableName, currentDatabasePath)
   // Logic to match data types with schema
  // Return TRUE if types match, otherwise FALSE
  IF NOT matchDataTypes(values, databaseSchemaPath)
    RETURN "Data type mismatch"
  // Logic to insert values into temporary structure
  CALL insertIntoTemporaryStructure(temporaryDatabase, tableName, values)
  RETURN "Data inserted into temporary structure"
```

```
FUNCTION persistDataOnClose(temporaryDatabase, path)
  // Logic to write data from temporary structure back to persistent storage (Into Files)
```

Select Query:

```
FUNCTION selectData(query)

  // Logic to copy database to a temporary structure
  // Return the temporary structure
  temporaryDatabase = copyDatabaseToTemporaryStructure(currentDatabasePath)
  CALL queryProcessor(query, temporaryDatabase)

FUNCTION queryProcessor(query, temporaryDatabase)

  //Checking the Syntax error in the Query
  IF checkSyntax(query) IS FALSE
    RETURN "Syntax Error"

  // Logic to extract table name from the query
  tableName = extractTableName(query)

  // Logic to extract column names from the query
  // Return the extracted column names
  columnNames = extractColumnNames(query)

  //Checks If the table Name exists in the temporary Storage
  IF NOT tableExists(tableName, temporaryDatabase)
    RETURN "Table not present"

  // Checks if the column name is present in the temporary database for that query
  IF NOT columnsExist(columnNames, tableName, temporaryDatabase)
    RETURN "Column not present"
  result = extractData(temporaryDatabase, tableName, columnNames)
  RETURN result
```

Update Command:

FUNCTION updateData(query)

```
  temporaryDatabase = copyDatabaseToTemporaryStructure(currentDatabasePath)
  CALL queryProcessor(query, temporaryDatabase)
```

FUNCTION queryProcessor(query, temporaryDatabase)

```
  // Checking the syntax for the query
  IF checkSyntax(query) IS FALSE
    RETURN "Syntax Error"

  // Logic to extract table name from the query
  tableName = extractTableName(query)

  // Logic to extract column name, new value, and condition from the query
  // RETURN extracted details (columnName, newValue, condition)
  columnName, newValue, condition = extractUpdateDetails(query)

  // Checks If the table exists in temporary Storage
  IF NOT tableExists(tableName, temporaryDatabase)
    RETURN "Table not present"

  // Checks if the ColumnName exists in the temporary storage
  IF NOT columnExists(columnName, tableName, temporaryDatabase)
    RETURN "Column not present"

  // Logic to update the required information in the temporary storage
  CALL updateTemporaryStorage(temporaryDatabase, tableName, columnName,
newValue, condition)
  RETURN "Data updated in temporary storage"

  // logic to write data from temporary structure back to persistent storage
  // Called when program closes

  CALL UpdatePersistentStorage(temporaryDatabase)
```

Delete Command:

```
FUNCTION deleteData(query)

  temporaryDatabase = copyDatabaseToTemporaryStructure(currentDatabasePath)
  CALL queryProcessor(query, temporaryDatabase)

FUNCTION queryProcessor(query, temporaryDatabase)

  IF checkSyntax(query) IS FALSE
    RETURN "Syntax Error"

  tableName = extractTableName(query)
  condition = extractCondition(query)


  IF NOT tableExists(tableName, temporaryDatabase)
    RETURN "Table not present"

  CALL deleteFromTemporaryStorage(temporaryDatabase, tableName, condition)
  RETURN "Data deleted in temporary storage"
   // logic to write data from temporary structure back to persistent storage
  // Called when program closes
  CALL UpdatePersistentStorage(temporaryDatabase)
```

Start Transaction:

```
FUNCTION startTransaction()

  T1 = copyDatabaseToTemporaryStructure(currentDatabasePath)
  C1 = copyTemporaryStructure(T1)
  RETURN "Transaction started"

FUNCTION processQuery(query)

  // Modify C1 storage based on the query
  // If Commit Found, perform operations accordingly

FUNCTION commitTransaction()

  T1 = copyTemporaryStructure(C1)
  copyTemporaryStructureToPersistentStorage(T1)

  END transaction
  RETURN "Transaction committed"
```

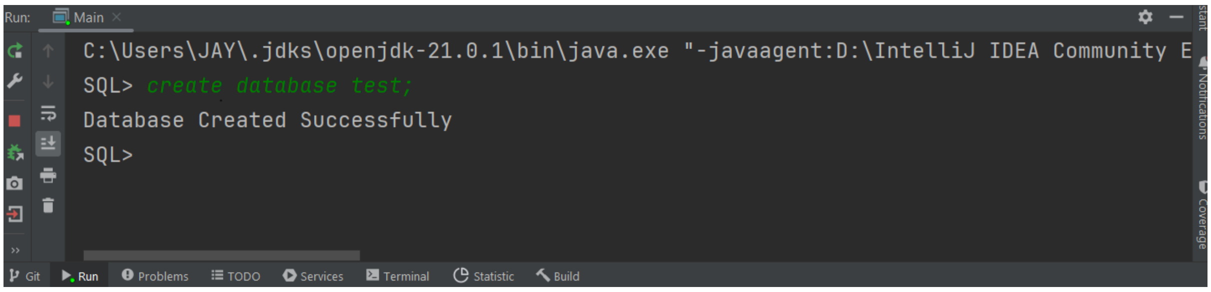
```
FUNCTION rollbackTransaction()

   // No action needed for rollback in this design

FUNCTION copyTemporaryStructureToPersistentStorage(temporaryStorage)

   // Write data from temporary storage T1 back to persistent storage
```

Test Cases and Evidence of Testing

CREATE DATABASE COMMAND

1) Create Database (Valid)

   Command: create database test;



```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> create database test;
Database Created Successfully
SQL>
```

Figure 7: Executing command create database test.

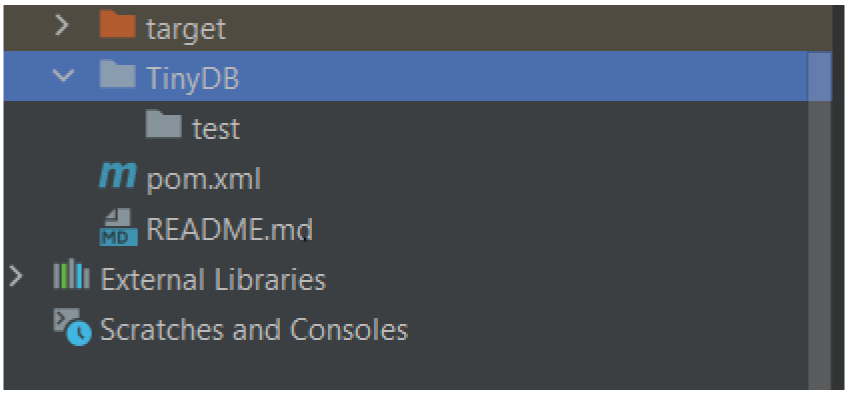
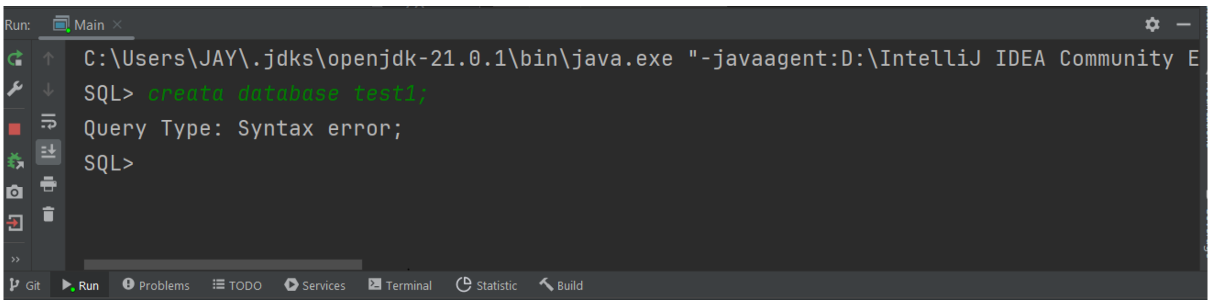


Figure 8: Folder gets added inside TinyDB with the name of the database.

2) Create Database (Spelling Mistake)
   Command: creata database test1;

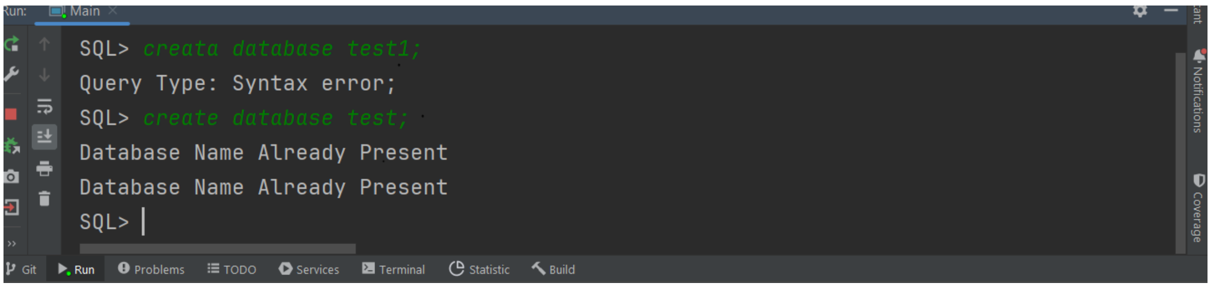


```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> creata database test1;
Query Type: Syntax error;
SQL>
```

Figure 9: When given the wrong command spelling, it prints a Syntax error and no folder gets created.

3) Database name already exists

Command: create database test;



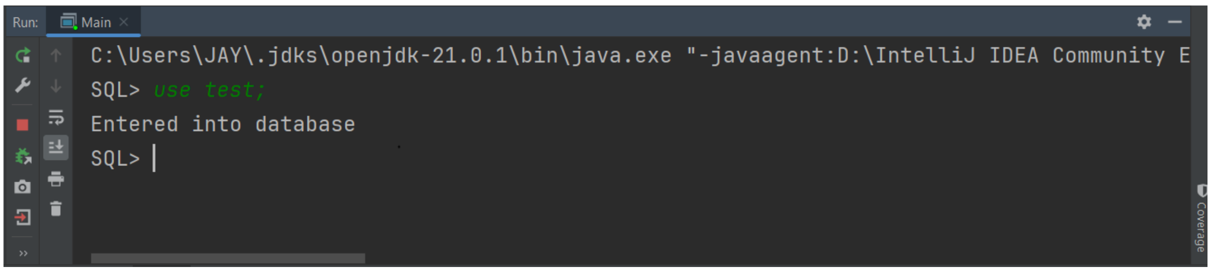
```
SQL> creata database test1;
Query Type: Syntax error;
SQL> create database test;
Database Name Already Present
Database Name Already Present
SQL> |
```

Figure 10: Executing create database command with a database name that already exists.

USE DATABASE COMMAND

1) Use Database (Valid)

Command: use database test;

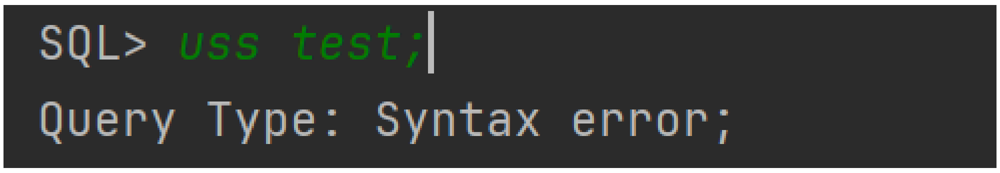


```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> |
```

Figure 11: Executing use database command for the database that exists.

2) Use Database (Spelling Mistake)

Command: uss database test;

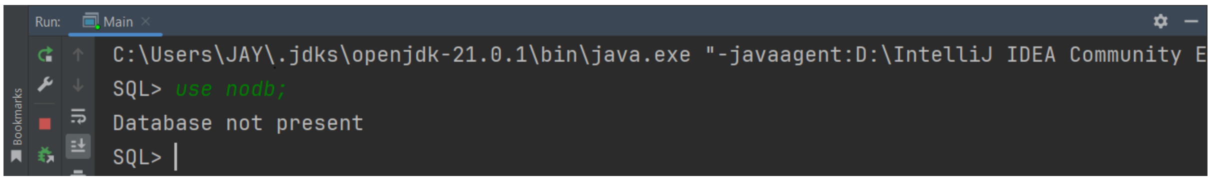


```
SQL> uss test;|
Query Type: Syntax error;
```

Figure 12: When given the wrong command spelling, it prints a Syntax error and no folder gets created.

3) Use Database (Invalid)

Command: use database test23;



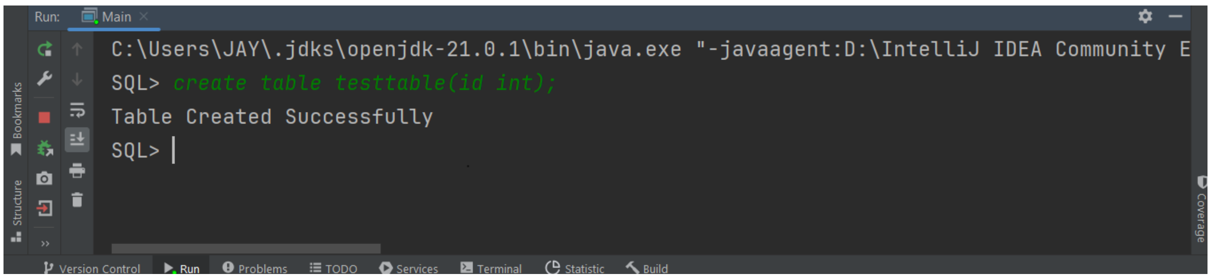
```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use nodb;
Database not present
SQL> |
```

Figure 13: Executing use database command for the database which does not exist.

CREATE TABLE COMMAND

1) Create Table (Valid)

Command: create table testtable(id int);



```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> create table testtable(id int);
Table Created Successfully
SQL> |
```

Figure 14: Executing the create table command.

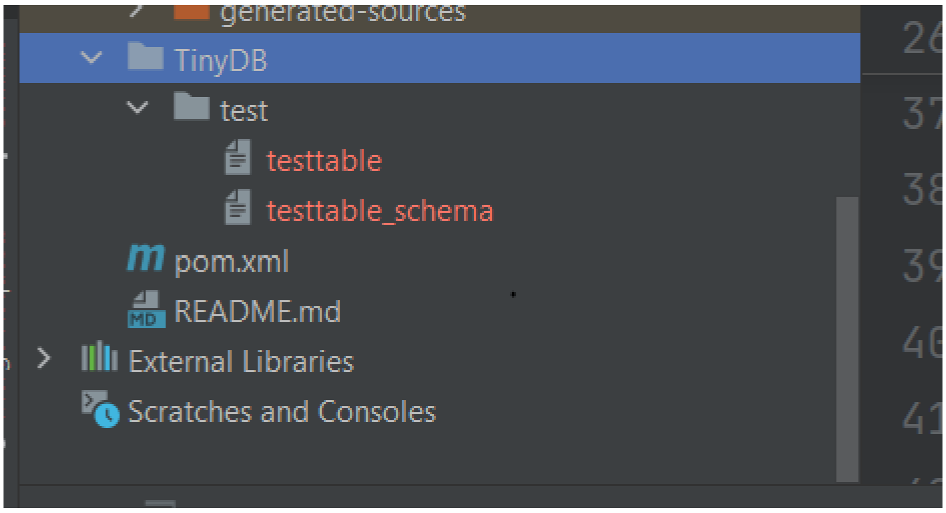
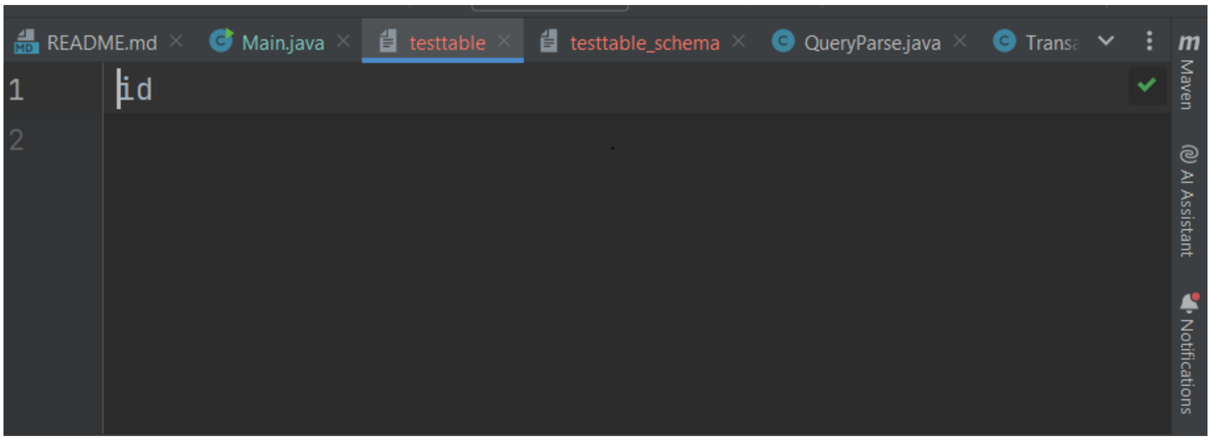
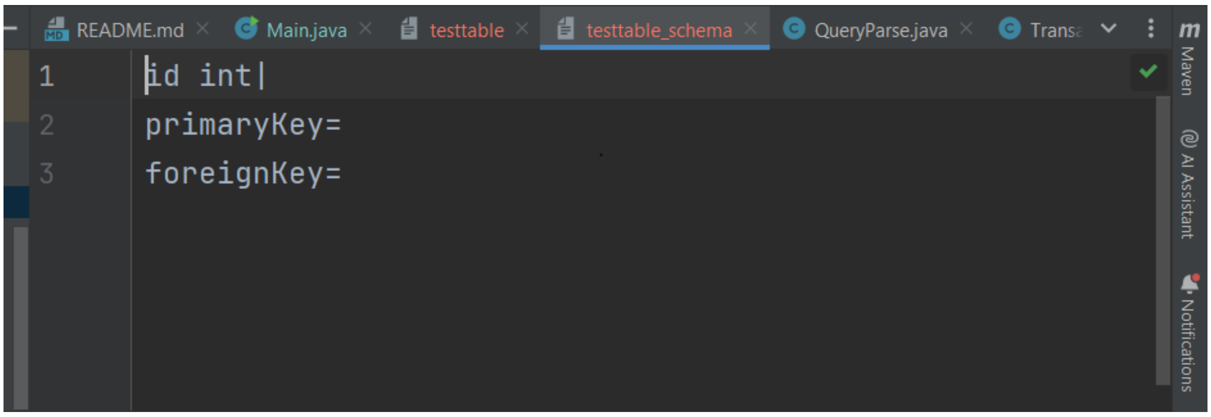


Figure 15: Table data file and schema file get created.



```
id
```

Figure 16: Table data file content.

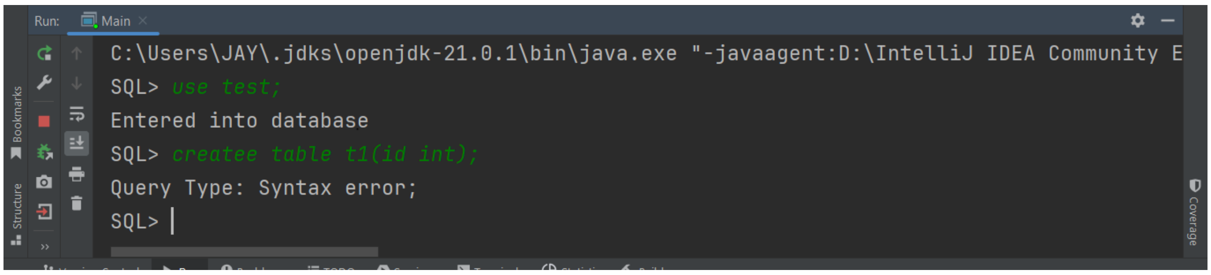


```
id int
primaryKey=
foreignKey=
```

Figure 17: Schame file content.

2) Create Table (Spelling Mistake)

Command: createe table t1(id int);

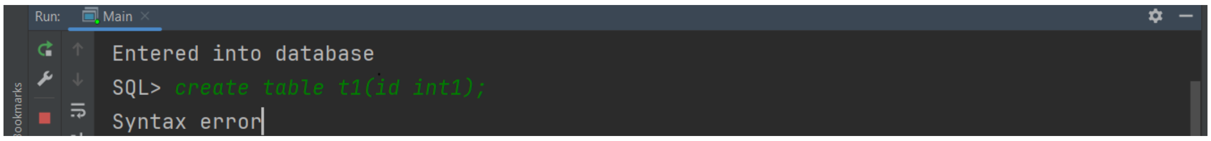


```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> createe table t1(id int);
Query Type: Syntax error;
SQL>
```

Figure 18: When given the wrong command spelling, it prints a Syntax error and table does not get created.

3) Create a Table (Spelling Mistake in variable type)

Command: create table t1(id int1);



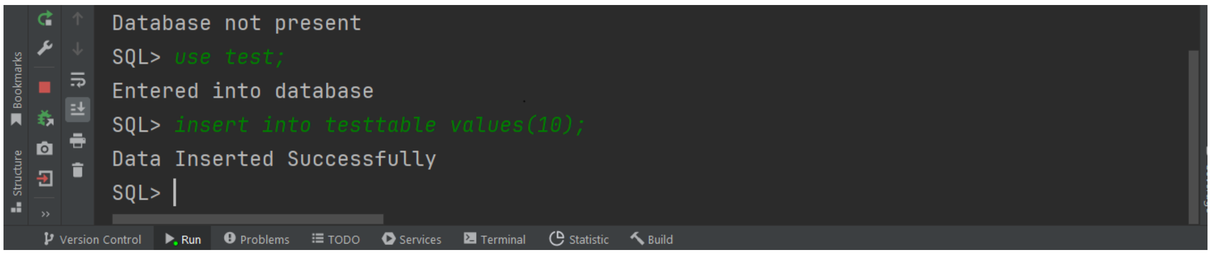
```
Entered into database
SQL> create table t1(id int1);
Syntax error
```

Figure 19: When given the wrong command spelling(variable type), it prints a Syntax error and the table does not get created.

INSERT INTO TABLE COMMAND

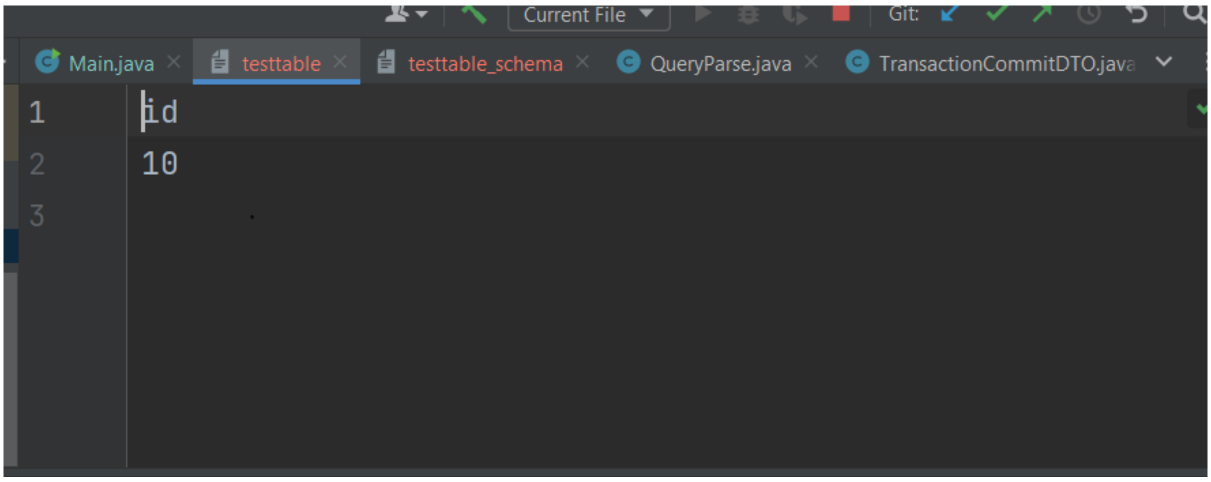
1) Insert Into Table (Valid)

Commands: use database test;
insert into testtable values(10);



```
Database not present
SQL> use test;
Entered into database
SQL> insert into testtable values(10);
Data Inserted Successfully
SQL>
```

Figure 20: Executing use database command for the database that exists.

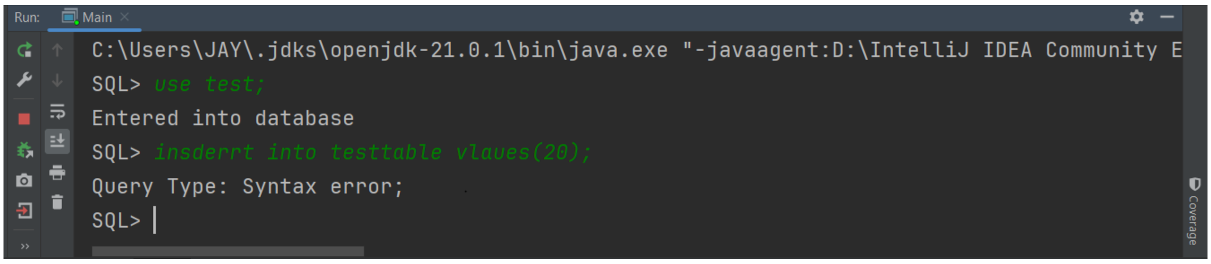


```
id
10
```

Figure 21: Data added to the data file.

2) Insert Into Table (Spelling Mistake)

Command: insderrt into testtable vlaues(20);

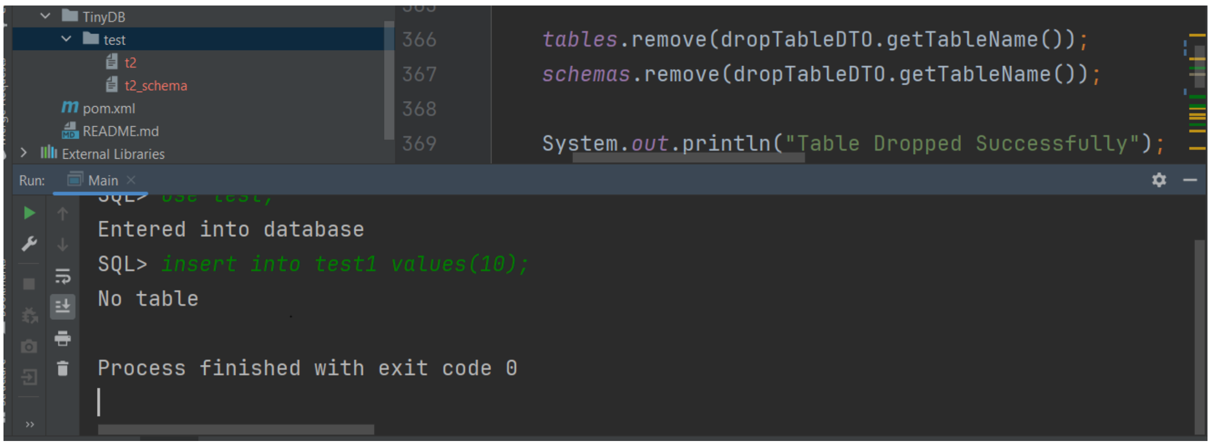


```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> insderrt into testtable vlaues(20);
Query Type: Syntax error;
SQL> |
```

Figure 22: When given the wrong command spelling, it prints a Syntax error and no data gets added.

3) Insert into table (Table does not exist)

Command: insert into table test1 values(10);



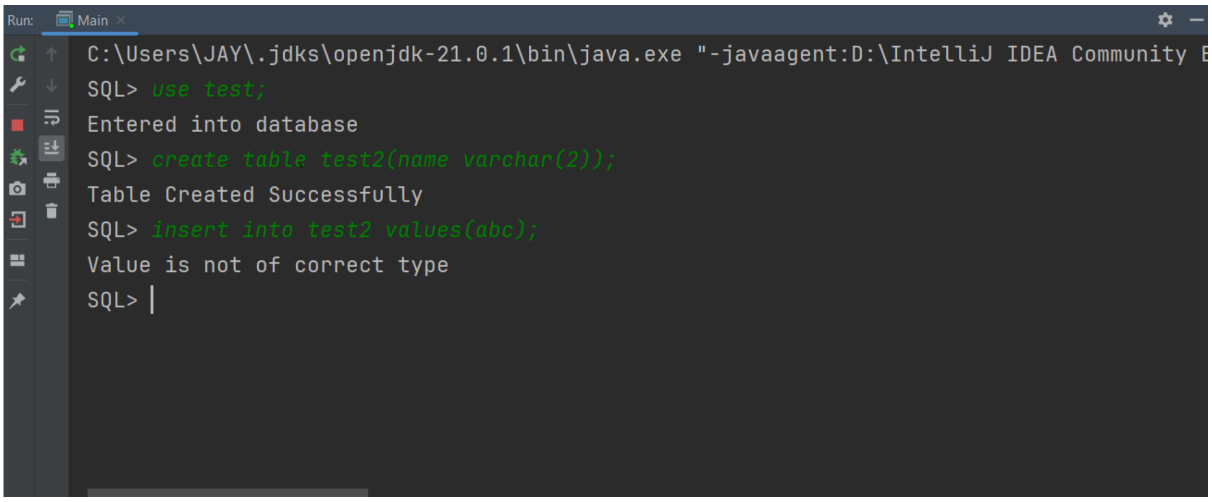
```
        tables.remove(dropTableDTO.getTableName());
        schemas.remove(dropTableDTO.getTableName());

        System.out.println("Table Dropped Successfully");
```

```
Entered into database
SQL> insert into test1 values(10);
No table

Process finished with exit code 0
```

Figure 23: Executing insert table command for which table does not exist.

4) Insert into table (Inserting invalid length values)

Command: use test;
create table test2(name varchar(2));
insert into table test2 values(abc);

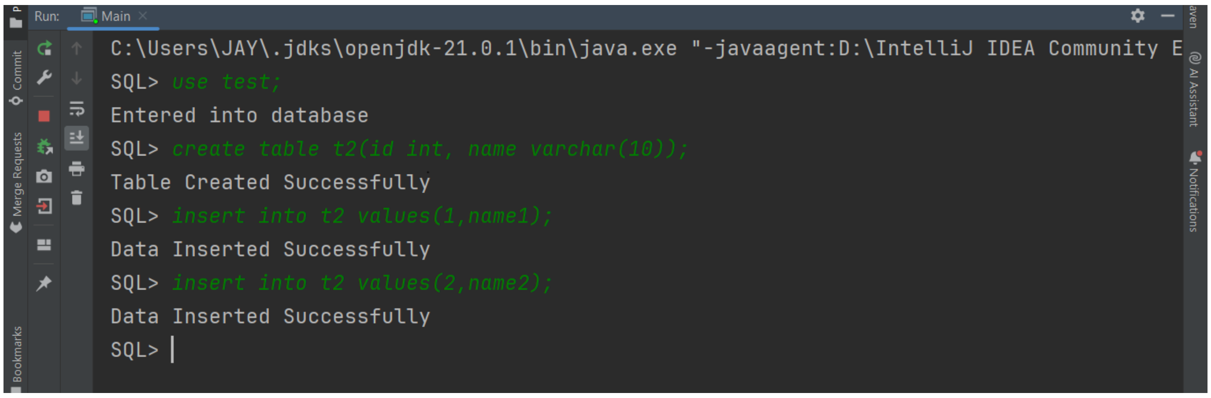


```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> create table test2(name varchar(2));
Table Created Successfully
SQL> insert into test2 values(abc);
Value is not of correct type
SQL> |
```

Figure 24 : Inserting values of invalid type.

SETUP FOR THE NEXT FEW COMMANDS

Commands:
create table t2(id int, name varchar(10));
insert into t2 values(1,name1);
insert into t2 values(2,name2);



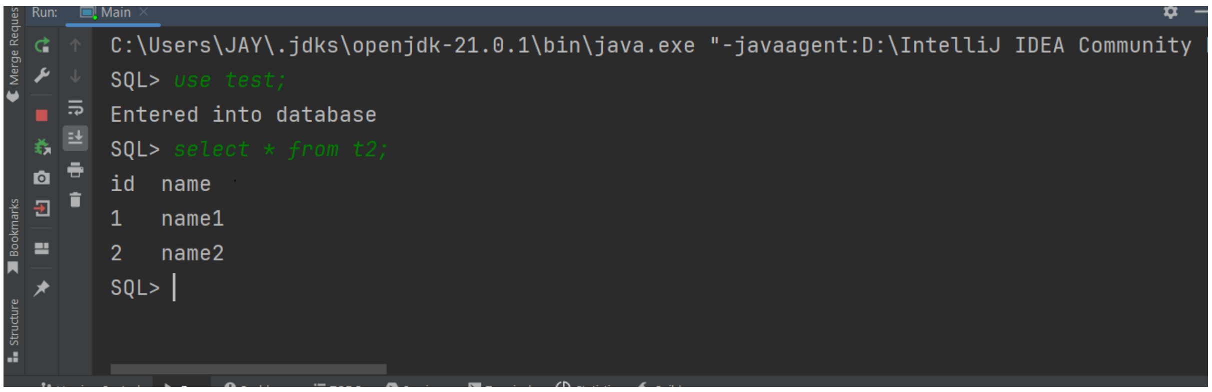
```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> create table t2(id int, name varchar(10));
Table Created Successfully
SQL> insert into t2 values(1,name1);
Data Inserted Successfully
SQL> insert into t2 values(2,name2);
Data Inserted Successfully
SQL> |
```

Figure 25:Executing set up commands.

SELECT FROM TABLE COMMAND

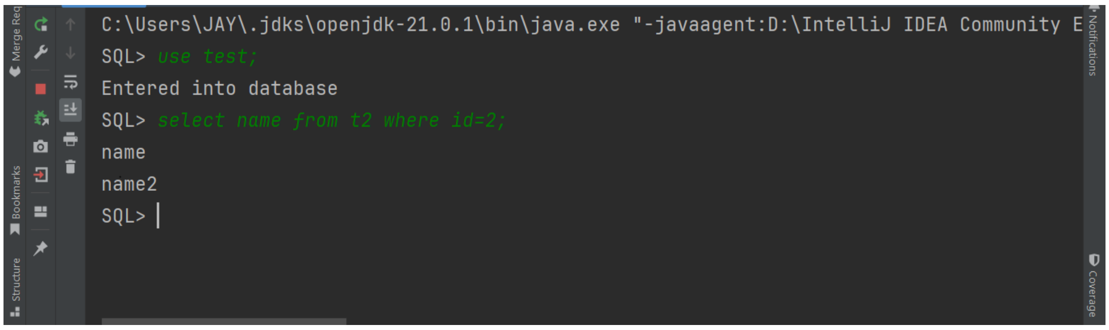
1) Select from table (Valid)

   Commands: use database test;
   select * from t2;



```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community
SQL> use test;
Entered into database
SQL> select * from t2;
id  name
1   name1
2   name2
SQL>
```

*Figure 26: Executing select * command.*

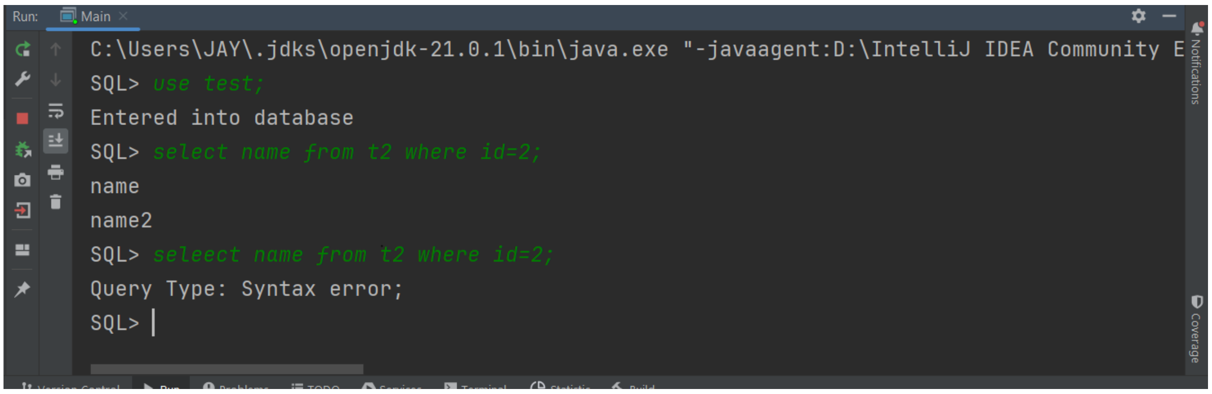


```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> select name from t2 where id=2;
name
name2
SQL>
```

Figure 27: Executing select name command from table t2.

2) Select from table (Spelling Mistake)

Command: seleect name from t2 where id=2;

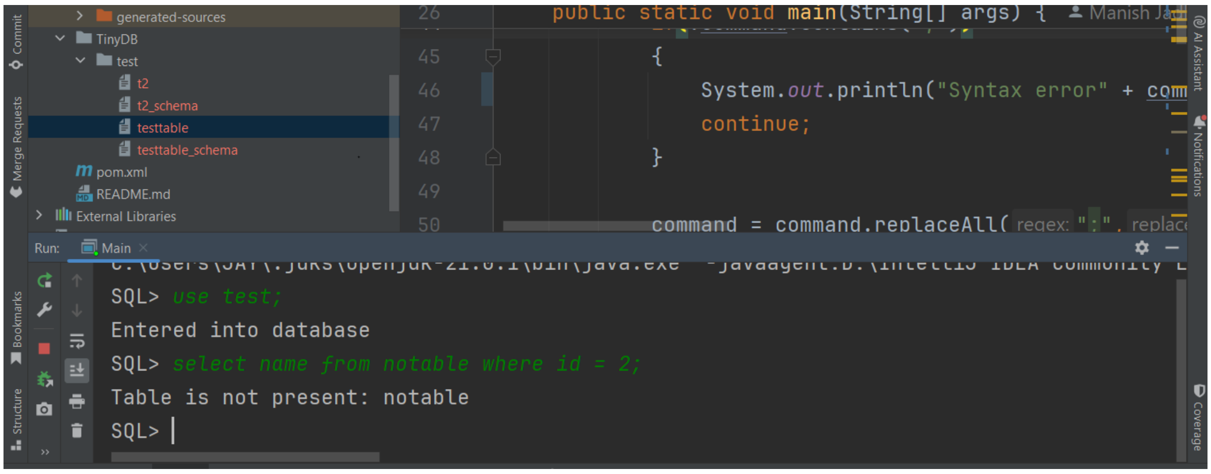


```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> select name from t2 where id=2;
name
name2
SQL> seleect name from t2 where id=2;
Query Type: Syntax error;
SQL>
```

Figure 28: When given the wrong command spelling, it prints a Syntax error and no data gets fetched.

3) Insert into table (Table does not exist)

Command: use test;
select name from notable where id = 2;



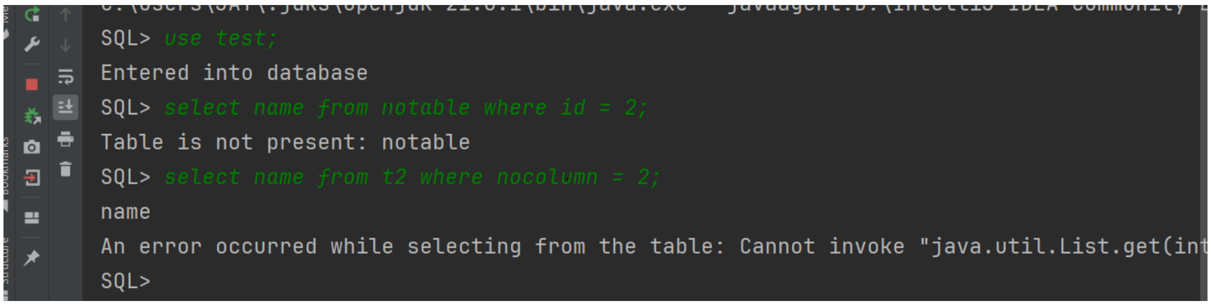
```
public static void main(String[] args) {
    {
        System.out.println("Syntax error" + com
        continue;
    }

    command = command.replaceAll( regex: ";", replac
```

```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe -javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> select name from notable where id = 2;
Table is not present: notable
SQL> |
```

Figure 29: Executing select table command for which table does not exist.

4) Insert into table (Column does not exist)

Command: use test;
select name from t2 where nocolumn = 2;



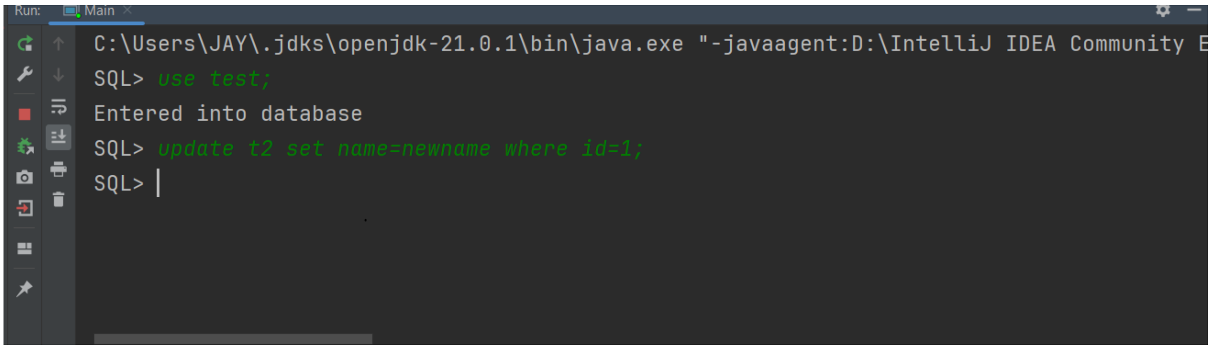
```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe -javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> select name from notable where id = 2;
Table is not present: notable
SQL> select name from t2 where nocolumn = 2;
name
An error occurred while selecting from the table: Cannot invoke "java.util.List.get(int
SQL>
```

Figure 30: Executing select table command for which column does not exist. It catches the exception and returns a message with the error that occurred and the exception message.

UPDATE FROM TABLE COMMAND

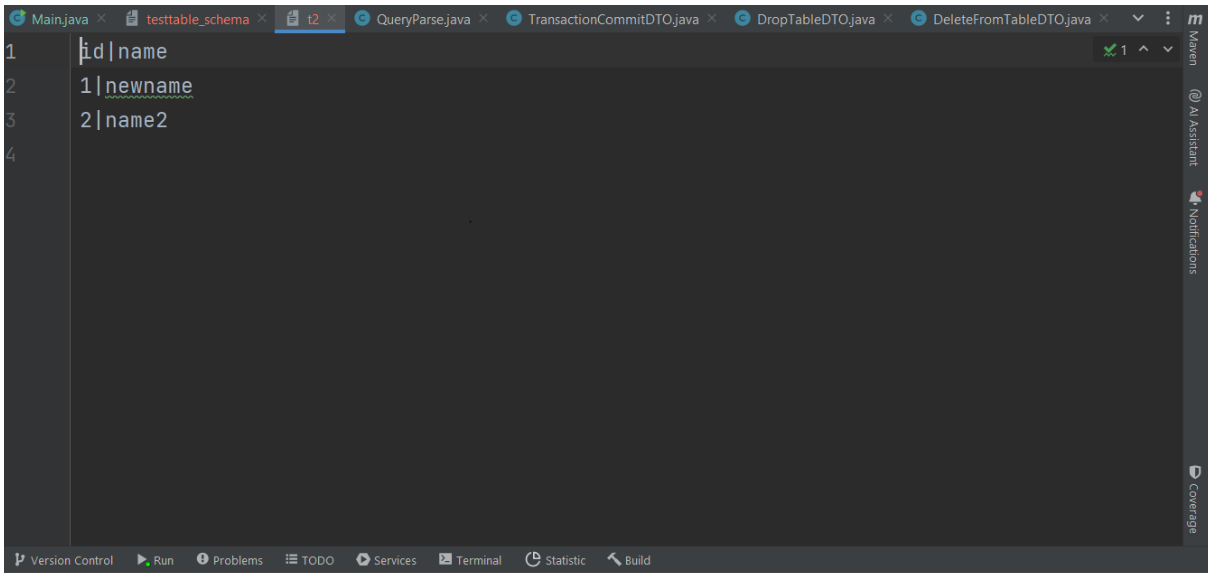
1) Update table (Valid)

Commands: use database test;
update t2 set name=newname where id=1;



```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> update t2 set name=newname where id=1;
SQL>
```

Figure 31: Executing update command.

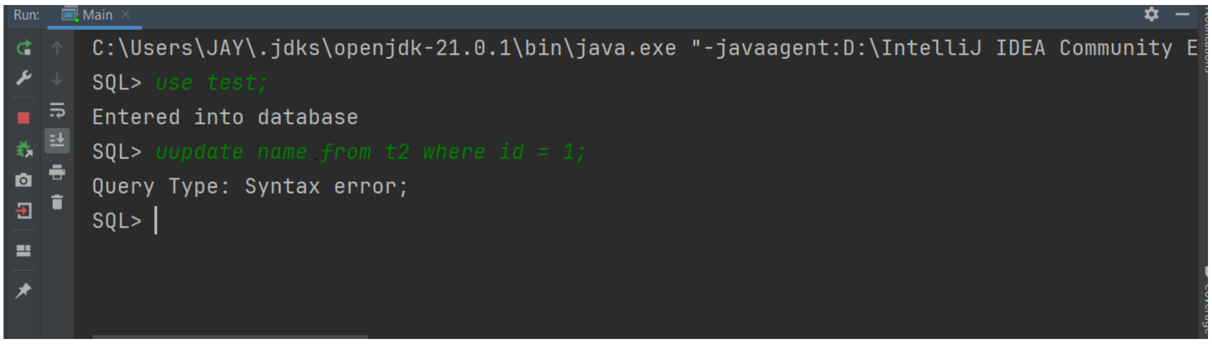


```
id|name
1|newname
2|name2
```

Figure 32: Updated values.

2) Update table (Spelling Mistake)

Command: uupdate name from t2 where id=2;

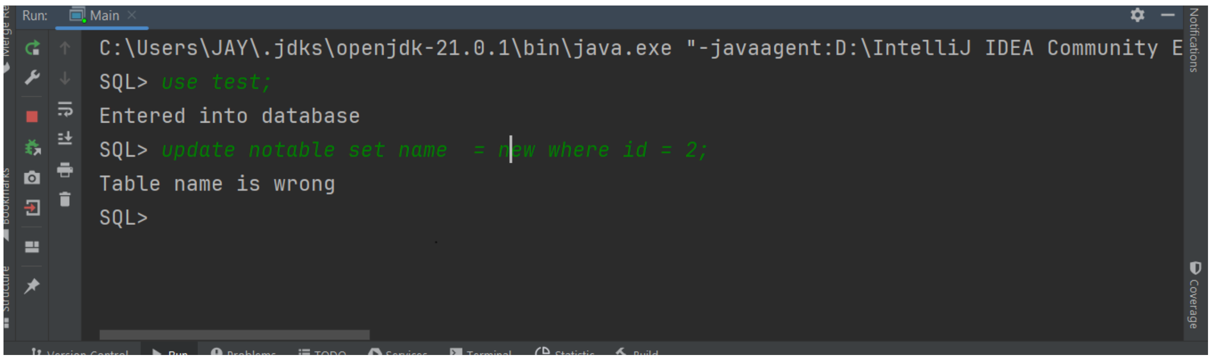


```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> uupdate name from t2 where id = 1;
Query Type: Syntax error;
SQL> 
```

Figure 33: When given the wrong command spelling, it prints a Syntax error and no data gets fetched.

3) Update table (Table does not exist)

Command: use test;
update notable set name = new where id = 2;

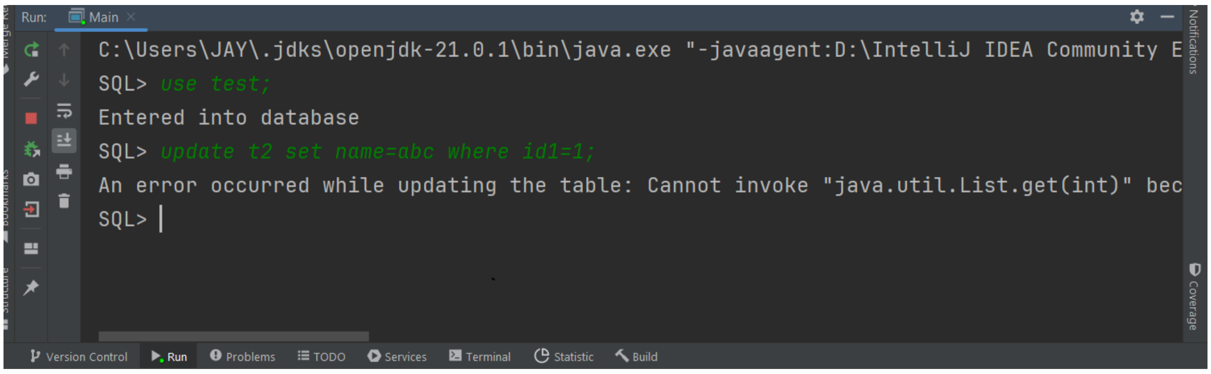


```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> update notable set name  = new where id = 2;
Table name is wrong
SQL>
```

Figure 34: Executing update table command for which table does not exist.

4) Update table (Column does not exist)

Command: use test;
update t2 set name=abc where id1=1;



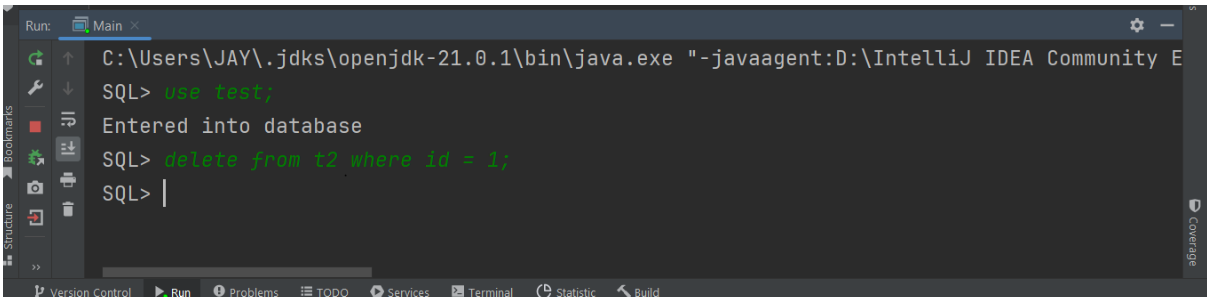
```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> update t2 set name=abc where id1=1;
An error occurred while updating the table: Cannot invoke "java.util.List.get(int)" bec
SQL> |
```

Figure 35: Executing update table command for which column does not exist. It catches the exception and returns a message with the error that occurred and the exception message.

DELETE COMMAND

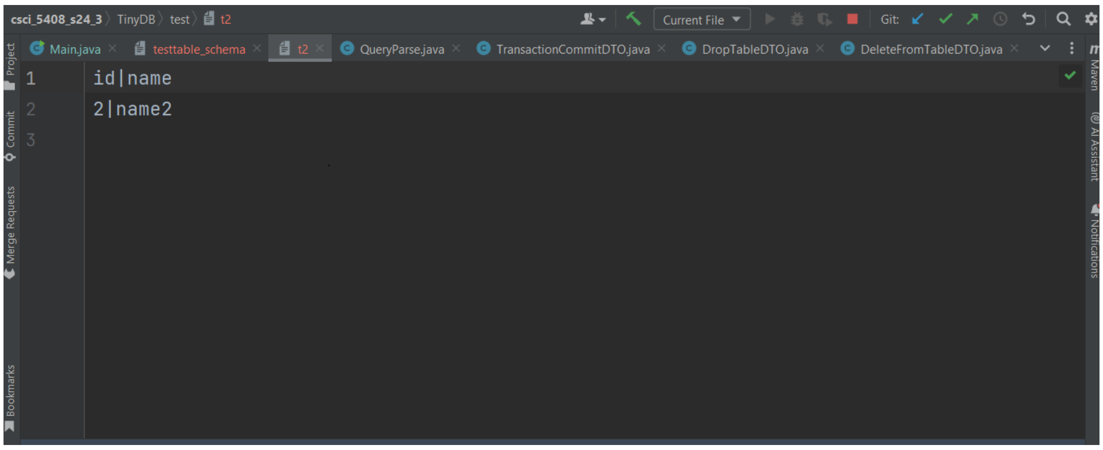
1) Delete from table (Valid)

Commands: use database test;
delete from t2 where id = 2;



```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> delete from t2 where id = 1;
SQL> |
```

Figure 36: Executing delete command.

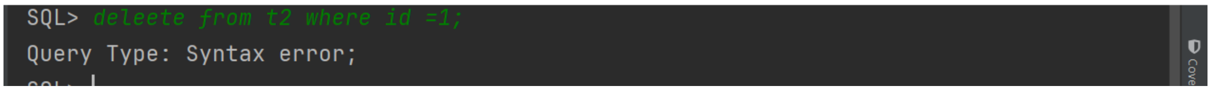


```
id|name
2|name2
```

Figure 37: After executing the delete command 1st row gets deleted

2) Delete from table (Spelling Mistake)

Command: deleete from t2 where id =1;

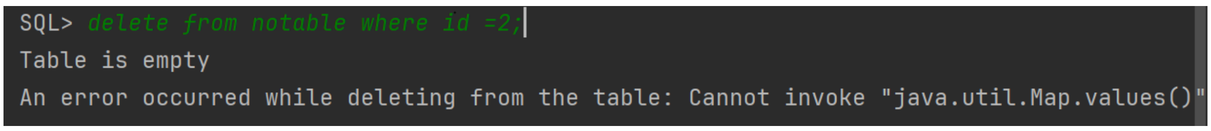


```
SQL> deleete from t2 where id =1;
Query Type: Syntax error;
```

Figure 38: When given the wrong command spelling, it prints a Syntax error and no data gets deleted.

3) Delete from table (Table does not exist)

Command: use test;
delete from notable where id =2;

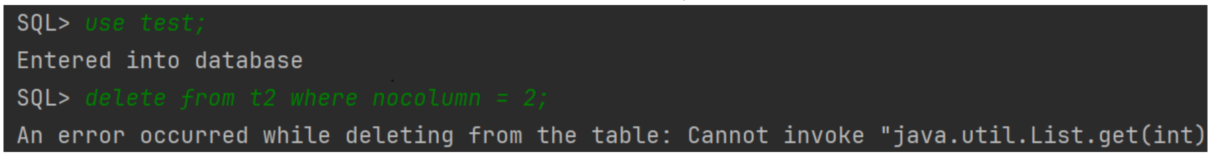


```
SQL> delete from notable where id =2;
Table is empty
An error occurred while deleting from the table: Cannot invoke "java.util.Map.values()"
```

Figure 39: Executing delete from table command for which table does not exist.

4) Delete from table (Column does not exist)

Command: use test;
delete from t2 where nocolumn = 2;



```
SQL> use test;
Entered into database
SQL> delete from t2 where nocolumn = 2;
An error occurred while deleting from the table: Cannot invoke "java.util.List.get(int)
```

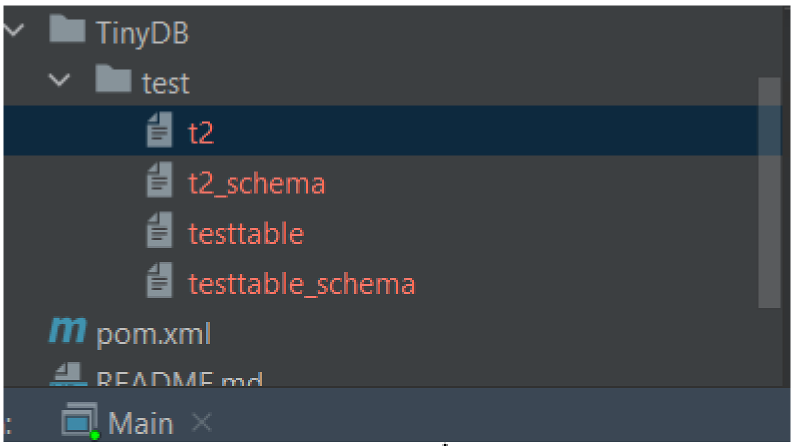
Figure 40: Executing delete from table command for which column does not exist. It catches the exception and returns a message with the error that occurred and the exception message.

DROP TABLE COMMAND

Commands:
create table t2(id int, name varchar(10));
insert into t2 values(1,name1);
insert into t2 values(2,name2);

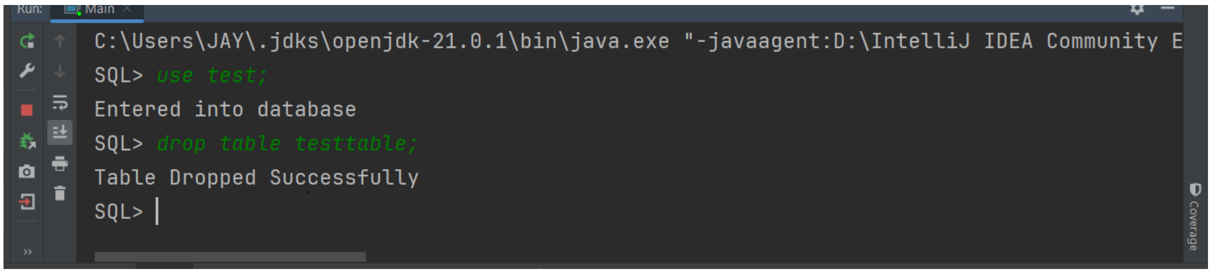
1) Drop table (Valid)

Command: use test;
drop table t2;



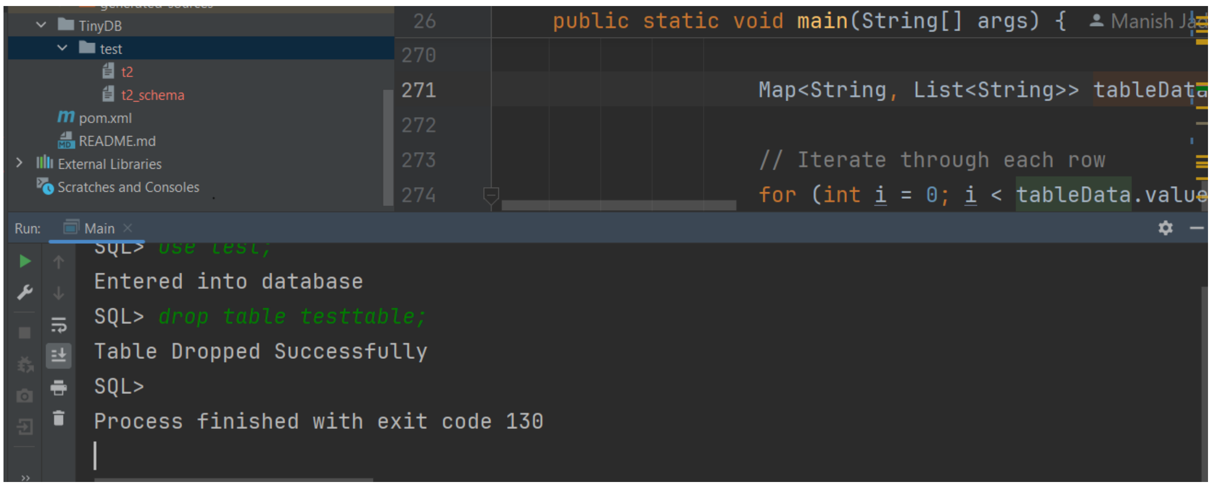
```
TinyDB
  test
    t2
    t2_schema
    testtable
    testtable_schema
  pom.xml
  README.md
Main
```

Figure 41: Before dropping table testtable.



```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> drop table testtable;
Table Dropped Successfully
SQL> |
```

Figure 42: Executing command drop table testtable.



```
TinyDB
  test
    t2
    t2_schema
  pom.xml
  README.md
External Libraries
Scratches and Consoles
```

```
public static void main(String[] args) {
    Map<String, List<String>> tableData

    // Iterate through each row
    for (int i = 0; i < tableData.value
```

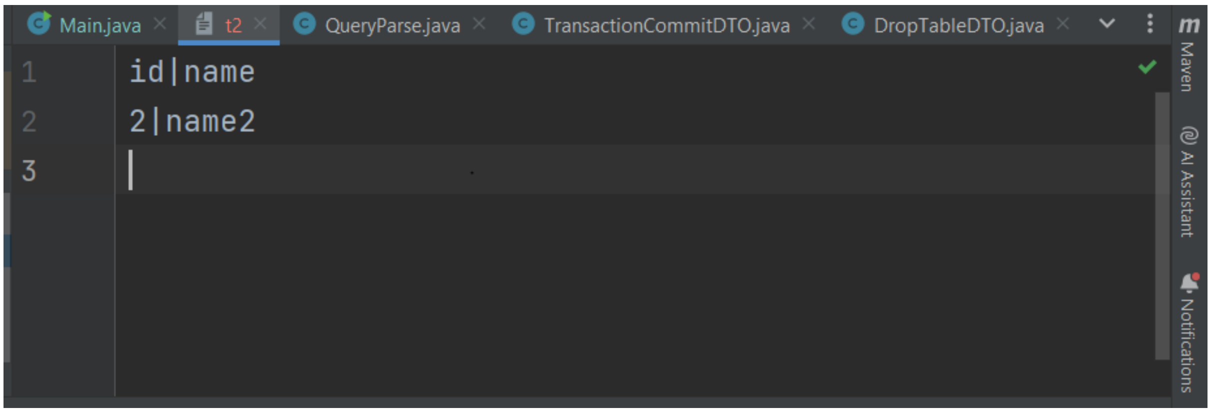
```
SQL> use test;
Entered into database
SQL> drop table testtable;
Table Dropped Successfully
SQL>
Process finished with exit code 130
```

Figure 43: After dropping table testtable.

TRANSACTION

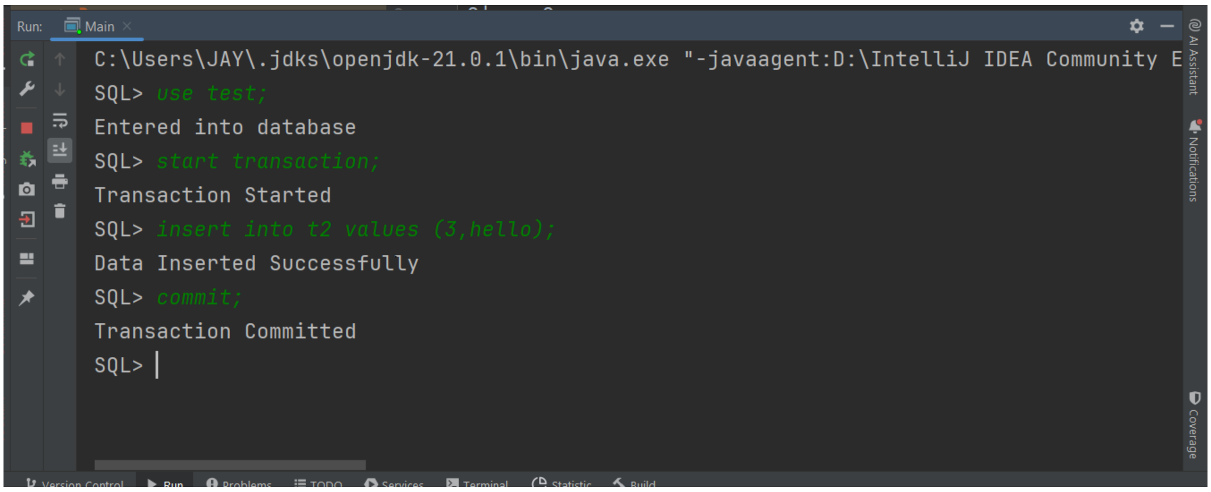
2) Transaction (Valid)

Command: use test;
start transaction;
insert into t2 values (3,hello);
commit;



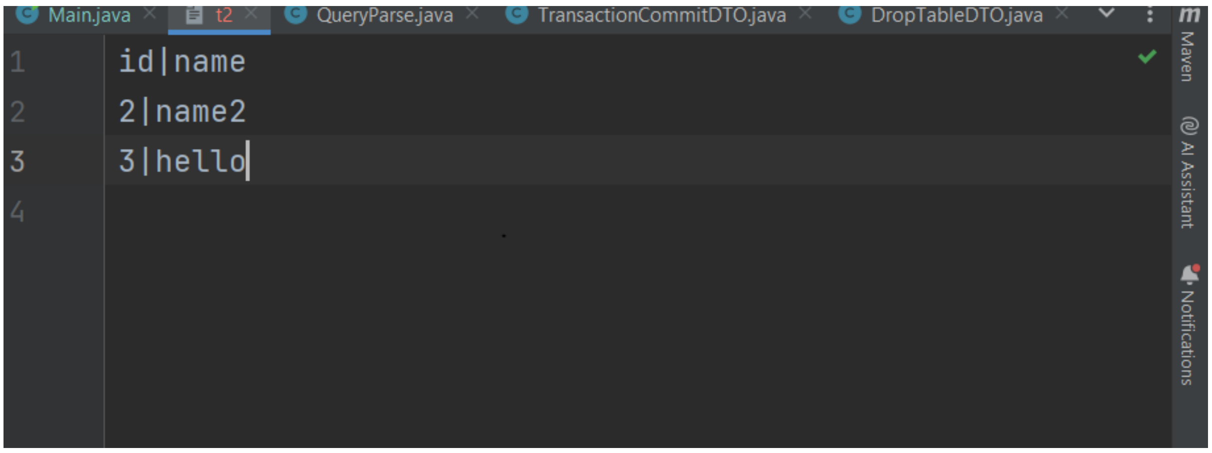
```
id|name
2|name2
```

Figure 44: Before transaction started.



```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> start transaction;
Transaction Started
SQL> insert into t2 values (3,hello);
Data Inserted Successfully
SQL> commit;
Transaction Committed
SQL>
```

Figure 45: Executing transaction and committing.

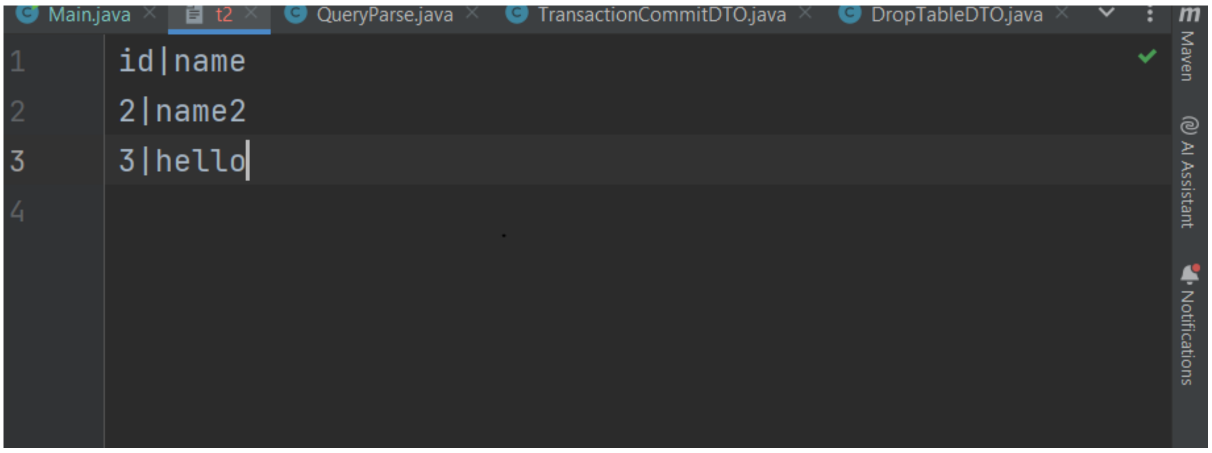


```
id|name
2|name2
3|hello
```

Figure 46: After the transaction committed

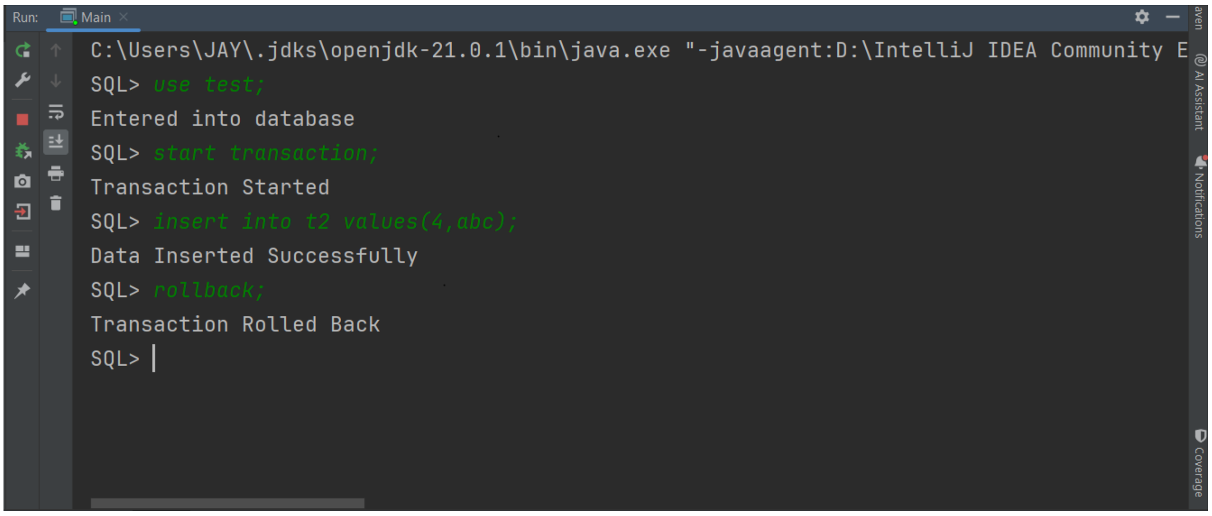
3) Transaction (Valid)

Command: use test;
start transaction;
insert into t2 values (4,abc);
rollback;



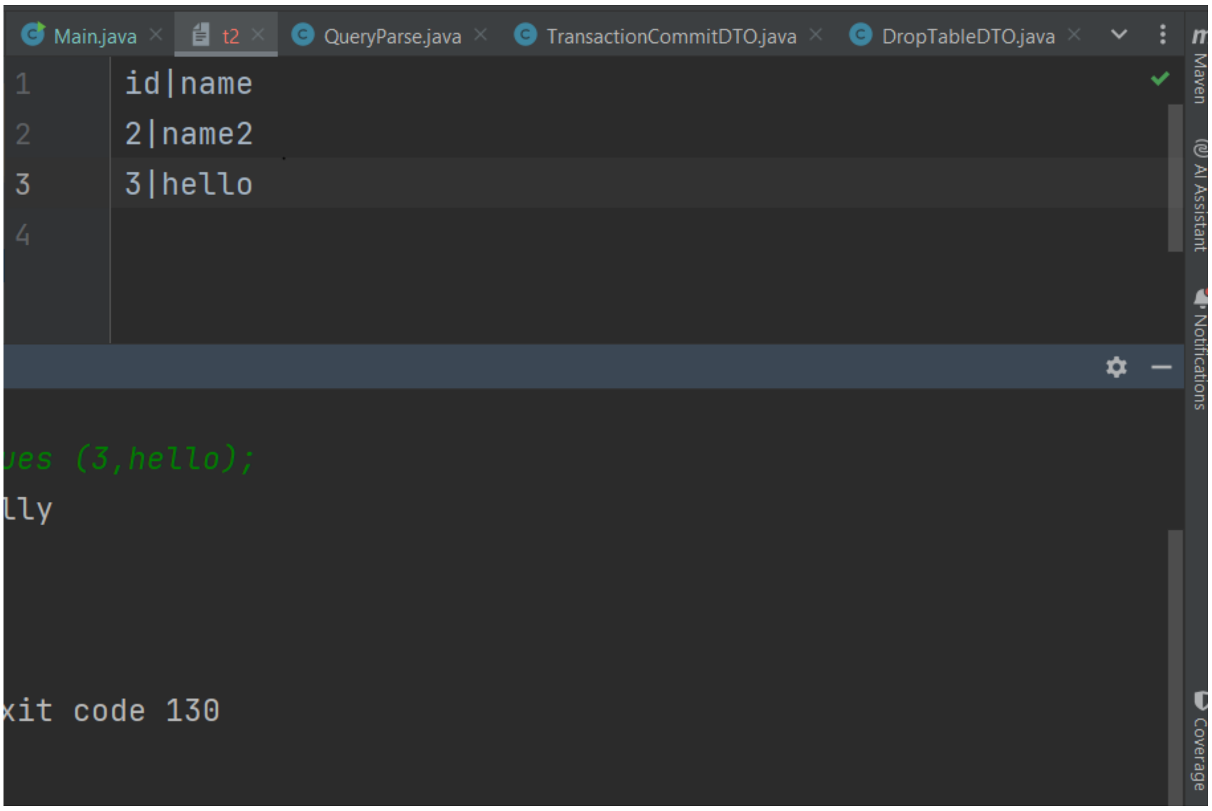
```
id|name
2|name2
3|hello
```

Figure 47: Before the transaction started.



```
C:\Users\JAY\.jdks\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA Community E
SQL> use test;
Entered into database
SQL> start transaction;
Transaction Started
SQL> insert into t2 values(4,abc);
Data Inserted Successfully
SQL> rollback;
Transaction Rolled Back
SQL> |
```

Figure 48: Executing transaction and committing.



```
id|name
2|name2
3|hello
```

```
ues (3,hello);
lly

xit code 130
```

Figure 49: After rollback data.

Meeting Logs

Date	Time	Attendees	Agenda	Meeting Type	Meeting Recording Link
29-05-2024	5:00-5:30	Manish, Jay	Analysis of Project requirement	Online	https://bit.ly/3zsaoLy
05-06-2024	5:00-6:00	Manish, Jay	Finalizing the Temporary and Persistent Storage	Online	https://bit.ly/3xDU0H4
12-06-2024	5:00-5:45	Manish, Jay	Division Of second Module (Queries)	Online	https://bit.ly/3xCj8y0
19-06-2024	5:00-5:40	Manish, Jay	Discussion of Transaction Implementation	Online	https://bit.ly/4cnjCar
24-06-2024	5:00-6:30	Manish, jay	Testing the code and review the code	Online	https://bit.ly/3W532Gk

References

[1] Kinza Yasar, "Database Management System (DBMS)," [Online]. Available: https://www.techtarget.com/searchdatamanagement/definition/database-management-system. [Accessed: June 29, 2024].