PROJECT REPORT

Title:

Distributed Database Management System with File System Storage

Date: **20th April, 2021**

Group 22 Members: Tasneem Yusuf Porbanderwala (B00873256)

Tejaswi Chaudhary (B00858613)

Venkata Kanakayya Prashant Vadlamani (B00883901)

Problem Statement:

Create a distributed database management system which supports the CREATE, SELECT, INSERT, UPDATE, DELETE and DROP queries along with WHERE clause support. Transactions are supported. User sign-on is implemented and passwords are stored in an encrypted format. The system has a unique file system management concept for persistent storage and handles the loading and storage of data as well as metadata. Appropriate logs are generated when queries are run and generation of textual Entity Relationship Diagram and SQL Dump file is also supported. Further, the two locations for the database are considered as two separate folders existing on the same machine.

Note: The distributed concept has been modified after the QnA session into containing two separate instances of the project being run and communication between them is implemented using SSH. This has been documented in the Project Extra submission.

Coordination and Contribution:

Team Coordination:

As a team, we have achieved to implement a distributed management system with its file system storage and communication between two assumed sites, treated as different folders. The queries are parsed and executed, along with transactions. ERD, SQL dumps and logs are generated. The system supports the sign-on of a single user at the local site.

Group Contribution:

The group has coordinated via Microsoft Teams throughout the project duration. Meetings were held as necessary to discuss individual progress and establish a plan of action for the next milestone to be achieved. Doubts and bugs were solved in meetings as well. All meetings were attended by every group member.

Individual Contribution:

The modules implemented by each group member is highlighted under their named sub-section:

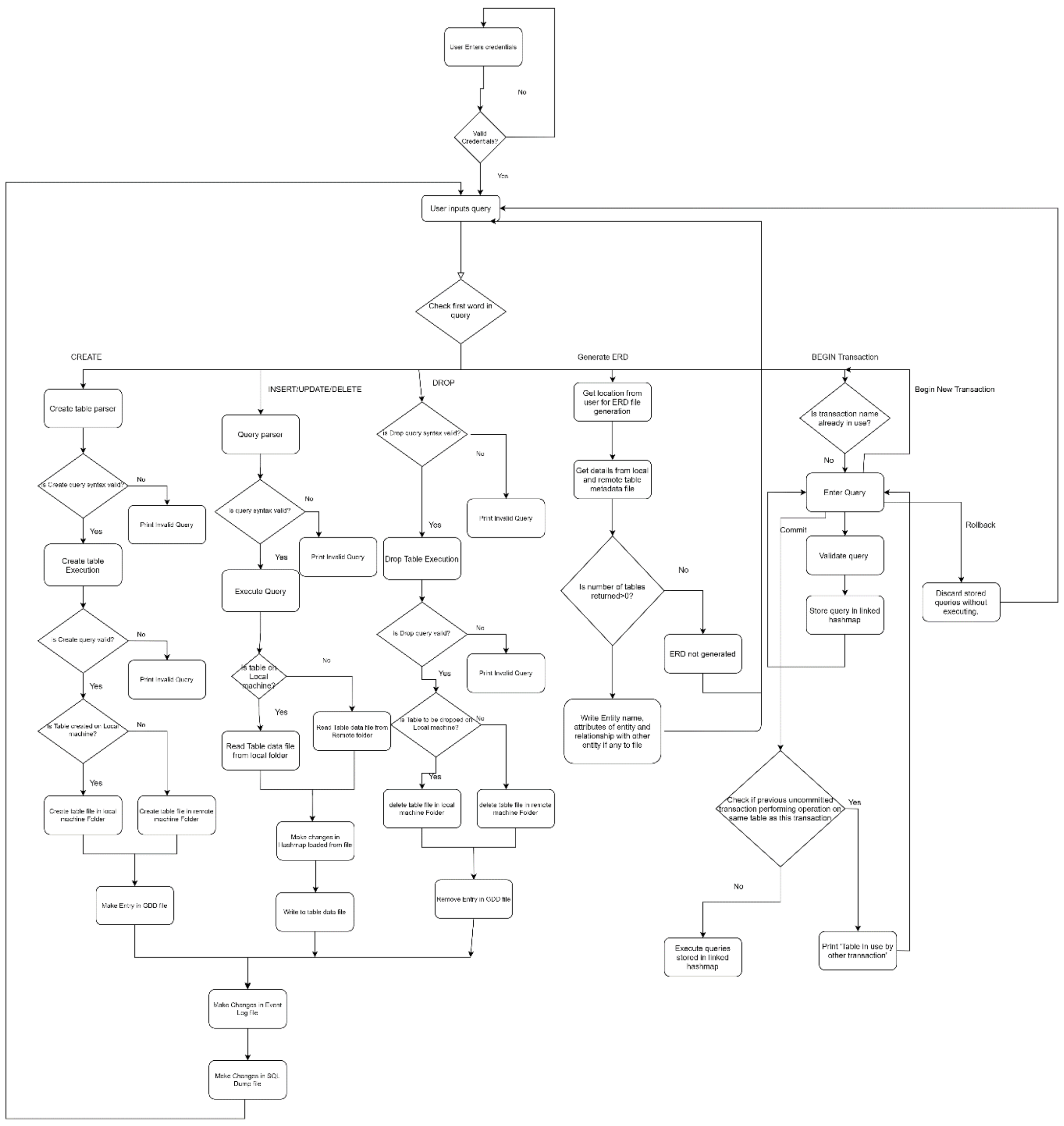
- Tasneem Yusuf Porbanderwala:
    - Create query parsing and execution
    - Select query parsing and execution
    - File persistent storage (Reading and writing table data, metadata and Global Data Dictionary)
    - Distribution of database via folders.

  Note: Post QnA session, the distribution of database is created using SSH connection between remote and local sites. This has been documented in the Project Extra submission.
- Venkata Kanakayya Prashant Vadlamani:
    - Main driver class
    - Event logs
    - SQL dump
    - Credential validation
    - Drop table query parser and execution
    - Delete table metadata
- Tejaswi Rameshkumar Chaudhary:
    - Insert query parser and execution.
    - Update query parser and execution.
    - Delete query parser and execution.
    - Generate ERD of entities stored in a database.

- Multiple transaction with Table level locking and single transaction.

Implementation Plan:

The overview of implementation of the project is depicted in the flowchart below:


User Enters credentials
No
Valid Credentials?
Yes
User inputs query
Check first word in query
CREATE
INSERT/UPDATE/DELETE
DROP
Generate ERD
BEGIN Transaction
Begin New Transaction
Create table parser
Is Create query syntax valid?
No
Print Invalid Query
Yes
Create table Execution
is Create query valid?
No
Print Invalid Query
Yes
Is Table created on Local machine?
No
Yes
Create table file in local machine Folder
Create table file in remote machine Folder
Make Entry in GDD file
Query parser
is query syntax valid?
No
Print Invalid Query
Yes
Execute Query
Is table on Local machine?
No
Yes
Read Table data file from local folder
Read Table data file from Remote folder
Make changes in Hashmap loaded from file
Write to table data file
is Drop query syntax valid?
No
Print Invalid Query
Yes
Drop Table Execution
is Drop query valid?
No
Print Invalid Query
Yes
Is Table to be dropped on Local machine?
No
Yes
delete table file in local machine Folder
delete table file in remote machine Folder
Remove Entry in GDD file
Make Changes in Event Log file
Make Changes in SQL Dump file
Get location from user for ERD file generation
Get details from local and remote table metadata file
Is number of tables returned>0?
No
ERD not generated
Write Entity name, attributes of entity and relationship with other entity if any to file
Is transaction name already in use?
No
Enter Query
Commit
Rollback
Validate query
Store query in linked hashmap
Discard stored queries without executing.
Check if previous uncommitted transaction performing operation on same table as this transaction
Yes
No
Execute queries stored in linked hashmap
Print 'Table in use by other transaction'

Implementation Details:

The implementation details for all modules are described in the following sections.

Configuration Setup:

- When the application is run, it checks if there exists a folder named 'data' present in its current working directory.
- If not present, the folders 'data' and 'dataRemote' are created and the following files are created
  - gdd.txt – only in 'data' folder
  - tableMetaData.txt – in both folders

   - userMetaData.txt – only in ‘data’ folder

  The user is asked for the root user password and this password is stored in the userMetaData.txt file in an encrypted format.
- The getDataPathLocal() and getDataPathRemote() return the fully qualified paths of the local ‘data’ folder and remote ‘dataRemote’ folder respectively. These functions are used by other modules to obtain values for the dataPath variable to decide where to read or write the files from.

Main Driver class:

- The main driver class is used to run the application.
- The first step of the main driver class is to ask the user about the credentials.
- Once the credentials are validated the system asks the user to enter the queries.
- When the user enters the queries, these queries are parsed by the parsing class and then the query is executed.
- The system asks the user to enter the queries till the user enters exit to stop the application.

Login Validation:

- Credential validation is used to validate the password of the root user.
- The credentials are stored in userMetaData.txt file in which the password is encrypted. The system asks the user to enter the password for root user.
- After the user enters the password, the password is encrypted using BCryptPassword encoder provided by Spring Security [1].
- The encrypted password and username are stored in usermetadata.txt file
- When the user logins again the system reads the input with the user and compares with the credentials that are stored.
- If the credentials are valid then the system asks the user to enter the query and if the credentials are invalid then the system asks the user to enter the credentials till the user enters valid credentials.

Create Query:

- The application supports Create query parsing and execution with the following syntax:

  <location> CREATE TABLE <tablename> (<column name> <datatype> PRIMARY KEY, .. , <column name> <datatype> FOREIGN KEY REFERENCES <referenced_table_name> <referenced column name>);

  Where <location> keyword can be either ‘local’ or ‘remote’ to denote where the user wishes the table to be created. Primary key is mandatory for the table, whereas foreign key is optional.

- The function isValid(query) validates the syntax of the create table query by performing string split and comparison operations.
- The datatypes of each column name are checked to be valid and supported by the application.
- The GDD is checked to make sure a table with the same table name does not already exist either in local or remote folder.
- If the table location is local, the dataPath variable is loaded with ‘data’ folder path or it gets loaded with ‘dataRemote’ folder path.
- The foreign key referenced table is checked for its existence as well as the data type of the foreign key column and the column it references is checked to be equal.
- If all checks are passed, a file gets created with the same name as table name
- The appropriate entries are made in event log file and sql dump file of the relevant location, specified by the dataPath variable.

- The GDD gets updated with the new table name created
- An entry for the newly created table gets added in table metadata file with its metadata details.

Select Query:

- The application supports Select query parsing and execution with the following possible syntax:
    - *SELECT * FROM <table name>;*
    - *SELECT <column1 name>, <column2 name>… FROM <table name>;*
    - *SELECT * FROM <table name> WHERE <column name>=<value>;*

  The where operation can be equal to, less than, greater than, less than or equal to and greater than or equal to.
- The function isValid(query) in the SelectQueryParser class validates the syntax of the SELECT query and loads the where clause details in appropriate variables of its parser object.
- The executeSelect() function is passed this parser object and a boolean "isLocal" variable which denoted if the table exists in local folder or remote folder.
- The GDD is checked to understand where the table exists. If table is not found in either location, a 'Table does not exist' error is thrown.
- According to the table location, the table data is read using the readTable class into a Linked Hashmap, where keys are column names and values are column data.
- Based on the following scenarios, the appropriate function is called to display the data:
    - All columns to be displayed
    - Columns to be displayed specified
        - The column names specified in query are checked against the table hashmap keys read to verify they are correct
    - All columns to be displayed with WHERE clause specified
    - Columns to be displayed specified with WHERE clause
- For the queries with WHERE clause, the following additional steps are followed:
    - The meta data of the table is checked to verify existence of column name specified in WHERE clause
    - Based on datatype of WHERE clause column, the operation is either performed on string values or integer/float values.
    - If the value specified is of incorrect datatype, appropriate error is thrown
    - For text datatype, only the '='operation is valid. If any other operation is specified, error is thrown.

Update Query:

- The system asks the user to write an update query. The system accepts query that has multiple as well as single where conditions with multiple operations such as <,>,<=,>= and =. It also updates multiple cells of the table in single query execution.
- Syntax of the query:
    - *UPDATE <tableName> SET <column name>=<value> where <column name>=<value>;*
    - *UPDATE <tableName> SET <column name>=<value>,<column name>=<value> where <column name>=<value> AND <column name><=<value>;*
- Update parser will parse the update query. If a query is written correctly, the query will be passed for execution.
- It will store column name whose values are to be updated and the new value to the hashmap, and it will also store columns, operations, and its value specified in the where clause.
- The UpdateQueryParser class looks for the table exists or not by checking in the local folder and if the table is not available in the local folder, then the class looks for the table in a remote folder.
- If the table is found in one of the local or remote folders, then all the data of the table will be loaded in the LinkedHashMap. After that, it will validate column name whose values are to be

updated is already exists in the table or not. If the user has provided the wrong column name that does not exist in the table then the program will show an error message to the user, and if the column name exists in the system then the program will check new values data type that matches with tables column datatype. If the data type does not match then the program will throw an error. The same process mention above will be performed on columns or columns specified in the where clause.

- Before executing the query and updating the value of the table, it will check the following condition, and if the conditions satisfy then It will update the values of those columns in the table according to the conditions specified in the where clause.
  - ➢ While updating the value of the primary key column, or not. If the new value already exists in the primary key column, it will show an error message to the user else it will execute the query successfully.
  - ➢ While updating the foreign key column value, it will check whether referenced table's primary key column contains that new value. If referenced table primary key column contains a new value then it will update the foreign key column with a new value.
  - ➢ While updating the primary key column value, it will also check whether any other table is currently referencing it or not. If another table is referencing the current table, it will check another table's foreign key column contains an old value, or not. If another table contains the current primary key column value, the program will show an error and will not perform an update operation on a table. If another table does not contain the current table's primary key column value, the program will update the new value to the table.
- After executing query successfully, program will write back the data of table with new changes stored in the LinkedHashMap to the file.

Delete Query:

- The system asks the user to write a delete query. The system accepts query that has single where conditions with multiple operations such as <,>,<=,>= and =. It also deletes all the data from the table.
- Syntax for the query:
  DELETE <tableName> WHERE <column name>=<value>;
- The delete parser will parse the delete query. If a query is written correctly, the query will be passed for execution.
- It will store column, operation, and its value specified in the where clause.
- The deleteParser class looks for the table exists or not by checking in the local folder and if the table is not available in the local folder, then the class looks for the table in a remote folder.
- If the table is found in one of the local or remote folders then all the data of the table will be loaded in the LinkedHashMap. After that, it will check whether the column name specified in the where clause exists in the table. If the column name exists then the program will delete data of rows that satisfy the where condition, else program will throw an error.
- Before deleting data from the table, it will check the following conditions, and if the conditions satisfy then It will delete data from the table successfully.
  - ➢ Before deleting the primary key column value, it will check whether any other table is referencing it or not. If another table is referencing the current table primary key column, then it will check whether the current primary key column value exists in another table foreign key column or not. If another table contains the current table primary key column value, then the program will show an error message to the user and will not perform deletion on the table.
- After executing query successfully, program will write back the data of table with new changes stored in the LinkedHashMap to the file.

Insert Query:

- The system asks the user to write an insert query.

- The insert parser will parse the insert query. If a query is written correctly, the query will be passed for execution.
- Syntax for the query:

  INSERT INTO <tableName> VALUES<val1,val2…>;
- It will store values in the linkedhashmap that will be inserted into the table.
- The insert helper class looks for the table exists or not by checking in the local folder and if the table is not available in the local folder, then the class looks for the table in a remote folder.
- If the table is found in one of the local or remote folders, then all the data of the table will be loaded in the LinkedHashMap. After that, it will check datatype of a column values that are specified in insert query. If data type of current values matches with data type of table columns, program will insert data into the table.
- Before inserting new data in the table, it will check the following conditions, and if the conditions satisfy then It will insert data in the table successfully.
  - ➢ While inserting value, it will check whether new value of primary key column exists in table or not. If already exists then program will show an error to user and doesn't perform insertion on the table.
  - ➢ While inserting value to the foreign key column, it will check whether referenced table's primary key column contains that new value. If referenced table primary key column doesn't contain a new value then program will not allow to insert that data in the table and givers an error to the user.
- After executing query successfully, program will write back the data of table with new changes stored in the LinkedHashMap to the file.

Drop Query:

- Drop table parser is used to parse the drop query that is received as an input.
- Once the query is parsed then the drop table execution is run to remove the table from the database.
- When the user enters drop query, the query is parsed by drop query parser
- If the query is valid then the query is executed by droptable class.
- The dropTable class looks for the table in the local folder and if the table is not available in local folder, then the class looks for the table in remote folder.
- If the table is found in one of the local or remote folder, then the table is dropped and the table is also deleted from gdd.
- If the table is not found, then it throws a message to the user saying that the table is not found.
- The table cannot be deleted if the table is referenced as a foreign key by another table. To delete this table all the tables that are referring to this table should be deleted.
- After the table is dropped this event is stored in the eventlog file and all the queries related to the table being dropped are deleted from the sqldump file.

Backend Store and Load:

The backend persistent file storage is implemented with the concept of using '^'symbol as a separator for values, and are stored with the .txt extension.

Each table has a dedicated text file, with the table name as the file name. A folder named 'data' is created by the Configuration Setup module and all the files related to local database, including metadata files, are stored in this folder. It acts as a database folder. A folder named 'dataRemote' is created by the Configuration Setup module and all the files related to remote database are stored here.

The Global Data Dictionary file is kept in 'data' folder and is shared by both the folders. Each folder has its own tableMetaData.txt, sqldump.txt and eventlog.txt files.

The following are the subsections within backend storage and loading of the various file types:

- **Read Table Data:**
  The data is read from persistent storage and loaded into a Linked Hashmap with keys as column names of the table being read and values as a list of strings of the data in that column. The appropriate order of columns is maintained by the linked hashmap, which matches the order in which they are specified in the table metadata.
  It is also checked whether a file with the same name as table exists in the folder. A dataPath variable is passed to the module functions to determine if the local or remote folder needs to be accessed to read the data.
  The steps followed are as below:
  1. Read metadata of table into Metadata object via readmetaData function.
  2. Initialise LinkedHashMap <String,ArrayList<String>> table
  3. Read table data file which exists on dataPath location.
  4. Set keys for hashmap table based on columns from metadata object
  5. Read the file and use '^' for splitting each line. Load the values into each column key
  6. If data is not present, return hashmap table with only keys.
  7. Return table

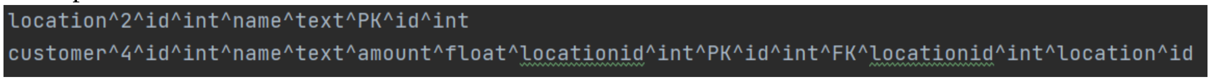
- **Write Table Data:**
  Data is written from the Linked Hashmap, which is passed to the function, into a text file, by using the '^' symbol as a separator for the column values. While writing, the rows are transposed into columns. Column names are not stored as this information is present in table metadata file. A dataPath variable is passed to the module functions to determine if the local or remote folder needs to be accessed to write the data.
  The steps followed are as follows:
  1. Prepare to write to table file which exists at dataPath location
  2. Iterate through the key values in the table hashmap
  3. Build a string of the values for each column with '^' as a separator between them
  4. Store this into a file on newline
  5. Repeat steps 4 and 5 for each key in hashmap
  6. Return

- **MetaData:**
  A MetaData class object is used to hold the following details about each table whose metadata is read from or written into the table metadata text file:
  - tableName
  - number Of Columns
  - LinkedHashMap<String, String> columns: Key is column name, value is datatype
  - Primary Key Column name
  - Primary Key Data Type
  - Foreign key Column
  - Foreign key Data Type
  - Referenced Column Name
  - Referenced Table Name

- **Read metadata:**
  The table metadata is stored in the following format in a text file named 'tableMetaData.txt':

  <tablename>^<number of columns>^<column1 name>^<column1 datatype>^...^PK^<primary key column name>^<primary key datatype> ^FK^<Foreign key column name>^<Foreign key datatype> ^<referenced table>^<referenced column name>

Example:



```
location^2^id^int^name^text^PK^id^int
customer^4^id^int^name^text^amount^float^locationid^int^PK^id^int^FK^locationid^int^location^id
```

This text file is created by Configuration setup when the application is run for the first time on a site. The data is loaded into the Metadata object for each table listed. A hashmap of all table names as keys and their corresponding Metadata object as values is returned by the readmetaData() function. A dataPath variable is passed to the module functions to determine if the local or remote folder needs to be accessed to read the table metadata. Steps followed are outlined below:

1. Initialise HashMap<String, MetaData> tables variable
2. Prepare to read from tableMetaData.txt file which exists at dataPath location.
3. Iterate through the table details listed in file by each line
4. Read the file and use '^' for splitting each line. Load key of map tables as table name and load the respective values into all parameters of its MetaData object, which is the corresponding value.
5. Add this entry into tables map
6. Repeat steps 5 and 6 for all table entries
7. Return tables

- **Write Metadata:**
  Data from the Metadata object is written to the tableMetaData file at the time of table creation, by using the '^' symbol as a separator. The format explained in previous section is used to write the values. According to the location of the table, the appropriate tableMetaData.txt file of local or remote folder is updated. A dataPath variable is passed to the module functions to determine if the local or remote folder needs to be accessed to write the table metadata. Steps followed are:
  1. Prepare to write to tableMetaData.txt file which exists at dataPath location.
  2. Build string 'value' of values in format specified by using the values specified in MetaData object
  3. Append string line to local tableMetaData.txt file

- **Delete Metadata:**
  This is used to delete the table metadata from tablemetadata.txt file.
  When the table is dropped the metadata of the table is deleted from tablemetadatafile.

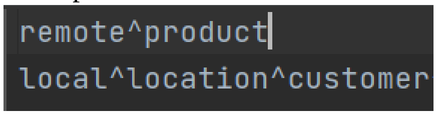
- **GDD:**
  The GDD object holds an HashMap with keys as 'local' and 'remote' and their respective values being a list of table names situated on the local and remote sites respectively.
  The other modules use the getInstance() method from this class to get this hashmap.

  The Global Data Dictionary text file is called gdd.txt and the same file is accessed for queries on tables which exist in local data folder or the remote site data folder.

  Note: Post QnA session changes include the copy of the file present on remote as well as local site. Each time the gdd.txt file is changed, for example, when a table is created or dropped, the updated gdd.txt file is sent to the other site.

- **<u>Read GDD:</u>**
  The gdd.txt file is read and its details are loaded in the HashMap of GDD. The keys are ‘local’ and ‘remote’ whereas values are a list of tables on local and remote site.
  The text file contains the following format:

  local^<table1>^<table2>..
  remote^<table1>^<table2>..

  Example:
  

```
remote^product
local^location^customer
```

  At the time of initial setup, this gdd.txt file is created with only the keywords of ‘local’ and ‘remote’ present in it. The steps followed are:
  1. Setup variables to read gdd.txt file
  2. Iterate through the lines present in file
  3. Load remote keyword in hashmap key followed by values of table names as list of strings
  4. Load local keyword in hashmap key followed by values of table names as list of strings.

- **<u>Write GDD:</u>**
  The GDD is updated when a table is either created at local/remote folder or if a table is dropped from either folder
  The updated GDD hashmap is written to the file gdd.txt.

Event Logs:

- Event logs are used to store the events that are triggered by the user, the events that are stored consists of the type of query, table on which the query is being executed, updated data for update and delete query, date and time of the event.
- These events are stored in eventlog.txt file and the local and remote locations have their own eventlog file. These fields are separated with an “^”.
- The system stores the events or the queries that are successfully executed.
- These events are stored in eventlog.txt file.
- There are two different eventlog.txt files, one is on the local folder and the other is on the remote folder.
- These eventlog files consists of the fields query type, table name, updated data for update and delete query, date and time of the event.
- These fields are separated with a “^”.
- Select queries are not stored in the eventlog file.

SQL Dump:

- Sqldump is used to store the successfully executed queries to get the instance of the current database at any particular time.
- These queries are stored in sqldump.txt file and the local and remote locations have their own sqldump file.
- If a table is dropped in any location, then the queries related to that table is all removed from sqldump file. Select query is not stored in sqldump.
- The system stores the queries that are successfully executed in a sqldump.txt file.
- This dump is used to create an instance of the current database at any particular time.
- There are two sqldump files, one for the local folder and the other for the remote folder.
- Select query is not stored in sqldump file.

- If a table is dropped from the database, then the queries related to that particular table is deleted from the sqldump file.

ERD Generation:

- It will fetch all the tables from the local folder and remote folder by reading the metadata file, and store columns and data of the table in LinkedHashMap.
- It will write data stored in the LinkedHashMap to the file. If a table contains a foreign key then it will fetch its referenced table, referenced table primary key column name, and its datatype by reading metadata file, and it will write all the fetched data in the erd text file. It will also store the cardinality of the table. If the table has a foreign key column then it will write many to one cardinality of a table with its referenced table and in the reference table, it will write one to many with a table that is referencing to it.

Transactions:

- Our program supports single as well as multiple serializable transactions.
- The syntax for a transaction:
- Start with: begin<transactionName>
    - Perform queries.
- End with: commit<transactionName> or rollback<transactionName>
- In a single transaction, transaction all the queries fired by a user will be stored in the LinkedHashMap. When the user commits or rollbacks the transaction, queries stored in the LinkedHashMap will be executed. When the user commits the current transaction, all the queries will be executed and the stored result of executed queries will be re-write to the table file. If the user rollbacks the transaction, the program will simply discard all the stored queries of the transaction in the LinkedHashMap and does not execute those queries.
- In multiple transactions, each transaction stores queries fired by a user in the LinkedHashMap. When the user commits or rollbacks the transaction, it will check whether the current table is locked by the previous transaction, or not. If a previous transaction is performing any write operation on that column and yet not committed then the current transaction will not be allowed to execute query until the previous transaction is committed. When a transaction is committed by the user, all the queries stored in the LinkedHashMap will be executed and the stored result of executed queries will be re-write to the table file. This way we have achieved table-level locking to maintain the ACID property. If the user rollbacks the transaction, the program will simply discard all the stored queries of the transaction in the LinkedHashMap and does not execute those queries.

<u>Use Cases:</u>

Login Validation:

- User cannot enter queries until valid credentials are provided.
- System asks the user to enter the credentials till valid credentials are provided.

Main driver class:

- The system asks the user to enter the credentials
- The system asks the user to enter the queries only if the valid credentials are entered.
- If the user enters invalid credentials, the system needs to ask the credentials till the user enter the valid credentials.
- The application stops if the user enters exit.

Create Query:

- Works for creating tables in local as well as remote database folder from local site, containing primary key mandatorily.
- Works for query containing one foreign key specified.
- Works for query without foreign key specified.
- Works for any number of columns specified.

Select Query:

- Query works for displaying all columns using ‘*’
- Query works for displaying specified columns in query.
- Query works with where clause specified. The where clause operations supported are =, <=, >=, <, >.
- Tables existing on local as well as remote database folder can be queried.

Update Query:

- Query works for updating single row values or multiple rows values.
- If table specified in the query exists, and conditions specified in the where clause matches with table data then query will be executed and table data will be updated.

Delete Query:

- Query works for deleting single data, multiple data or all the data from the table.
- If table with given name in the query exists in the system, it will delete data from the table as mentioned in the query.

Insert Query:

- Query works for inserting data into the table. If primary and foreign key exists in the table, and written query does not violate primary and foreign key constraint then it will insert data into the table successfully.
- For null column value, user needs to specify null value while writing insert query, if he does not want to gives any value to specific column.

Drop query:

- The syntax of the query should be drop table tablename;
- The table gets dropped if the table is found and the table is not being referred by another table.
- The table can be either in local or remote folder.
- There will be a unique tablename for the tables in local and remote folder.

Event log:

- Only successful events are stored in the eventlog file.
- Select query is not stored in eventlog file.

SQL Dump:

- The queries that are successfully executed are stored in sqldump file.
- The queries that are related to the table that is being dropped should be deleted upon successful deletion of the table.
- Select queries are not stored in sqldump file.

ERD:

- When user wants to know relationships between entities stored in the database. It will generate ERD for user, which stored name of the entities, attributes of the entity and relationship of that entity with other entity if exists.

Transactions:

- Multiple Transaction implemented to achieve ACID property.
- User needs to specify for which transaction they are running query.

Test Cases:

Login Validation:

Test	Preferred Result
User enter invalid credentials	System throws an error saying invalid credentials and asks the user to enter credentials again
User enters valid credentials	System asks the user to enter the query

Main Driver class:

Test	Preferred Result
When user enters valid credentials	System asks the user to enter the query
User enters invalid credentials	System throws an error saying invalid credentials and asks the user to enter credentials again
When user enters exit as a query	The application stops running
When user enters query other than exit	The system asks the user to enter the query

Create Query:

Test Case	Preferred Result
Syntax specified is correct.	Invalid query.
The table name provided is unique.	Table already exists
Datatype specified for each column is correct and supported by the application.	Invalid datatype.
Primary key is specified.	Invalid query
Foreign key referenced table exists.	Foreign key not valid.
The datatype of foreign key column matches the data type of referenced column.	Foreign key not valid.

Select Query:

Test Case	Preferred Result
Syntax specified is correct.	Invalid Syntax
The table name provided does not exist.	Table does not exist
Column name specified exists	Check Column Name
Column specified in where clause exists	Where column doesn’t exist
Where clause operation is supported by application	Incorrect operation
Value specified for where clause column comparison is of correct datatype	Where value datatype is wrong
Text column support only = operator	Incorrect operation

Update Query:

Test	Preferred Result
Update query with incorrect syntax (eg. Query without semicolon, missing one of the keywords, etc).	Invalid syntax.
Update table data with either invalid column name, invalid column value datatype, or column name in where clause in update query.	Gives an error. Specified column does not exist in the table.
Update table data with valid column name, column value datatype in update query with valid column name in where clause.	Update table data successfully.
Update primary key column value with new value that creates duplication in the table.	Gives an error (Primary key column already contain value).
Update primary key column value with new value that doesn’t exist in the table.	Update table data successfully.
Update table data with update query that parsed successfully but no row has match with where condition.	Query run successfully but 0 rows will be affected in the table.
Update table data with update query that parsed successfully, and single row has match with where condition.	Query run successfully, and only 1 row will be affected in the table.
Update table data that parsed successfully, and multiples row have match with where condition.	Update table data that parsed successfully, and multiples row have match with where condition.
Update foreign key column value that violates primary and foreign key constraints.	Gives an error. It violates primary and foreign key constraints.
Update table data with update query that parsed successfully but one of the where clause conditions does not satisfy by any rows.	Query run successfully but 0 rows will be affected in the table.

Delete Query:

Test	Preferred Result
Delete query with incorrect syntax (eg. Query without semicolon, missing one of the keywords, etc).	Invalid syntax.
Delete table data with invalid where clause column name that doesn’t exist in the table.	Gives an error. Specified column does not exist in the table.
Delete table data with valid column name, column value datatype in delete query with valid column name in where clause.	Delete data from table successfully.
Delete primary key column value which is used in another table’s foreign key column.	Gives an error. As a parent table can’t delete if child table using that value.
Delete foreign key column value.	Delete value from the table successfully.
Delete all the data from the table.	Delete data from the table successfully.
Delete data from the table with delete query that parsed successfully, and single row has match with where condition.	Query execute successfully, and 1 row is deleted.
Delete data from the table with delete query that parsed successfully, and multiple rows have match with where condition.	Query execute successfully, and multiple rows are deleted.
Delete data from the table with delete query that parsed successfully but one of the where clause conditions does not satisfy by any rows.	Query execute successfully but 0 rows deleted.

Insert query:

Test	Preferred Result
Insert query with incorrect syntax (eg. Query without semicolon, missing one of the keywords, etc).	Invalid syntax.
Insert data with insert query that has valid column value with valid data type.	Insert data in the table successfully.
Insert duplicate primary key column value.	Gives an error as primary key contains value already.
Insert primary key column value that doesn't exist in the table.	Insert new data in the table successfully.
Insert foreign key column value that doesn't exist in the referenced table primary key column table.	Gives an error as primary key column of referenced table does not contain value.
Insert foreign key column value that exists in the referenced table primary key column table.	Insert new data in the table successfully.
Insert data with insert query that has one of the column value's datatype invalid.	Gives an error. Invalid datatype.
Insert data with insert query that has one or more than one column values as null.	Insert new data in the table successfully.
Insert data with insert query that has one of the column value missing.	Gives an error.

Drop Table:

Test	Preferred Result
When user enters invalid syntax	The system shows a message saying incorrect query
When user enters valid query and table is found	Table is deleted
When user enters valid query and table is not available	The system shows a message saying table does not exists
When user tries to delete a table, which is referred by another table	The system shows a message saying table cannot be dropped

Generate ERD:

Test	Preferred Result
Try to generate ERD when no database exists.	Gives an error.
Try to generate ERD when database has single table.	Successfully creates ERD.
Try to generate ERD when multiple tables exist and has relationship between each other.	Successfully creates ERD.
Try to generate ERD when table has foreign key column.	Successfully creates ERD.
Try to generate ERD when multiple tables exist and has no relationship between each other.	Successfully creates ERD.

Transaction:

Test	Preferred Result
Begin single transaction with valid single query and its syntax.	Run query successfully and change table according performed query when transaction committed, and discard query when transaction is rollback.

Begin single transaction with valid multiple queries and its syntax.	Run query successfully and change table according performed query when transaction committed, and discard query when transaction is rollback.
Begin multiple transaction with valid multiple queries in which first transaction if performing only read on the table, and second transaction is performing either run read or write on the same table.	Both transactions run successfully only if transaction 1 is committed and after that transaction 2 executes.
Begin multiple transaction with valid multiple queries in which first transaction if performing either read or write on the table, and second transaction is performing only read on the same table.	Both transactions run successfully only if transaction 1 is committed and after that transaction 2 executes.
Begin multiple transaction with valid multiple queries in which first transaction if performing either read or write on the table, and second transaction is performing either run read or write on the same table.	Both transactions run successfully only if transaction 1 is committed and after that transaction 2 executes.

Event Log:

Test	Preferred Result
When the query is successfully executed	The event is stored in the eventlog file except for select query
When the query is not executed	The event is not stored in eventlog file

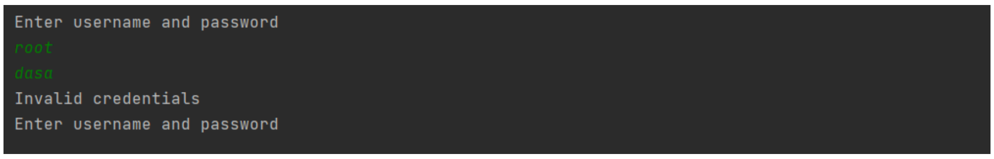
SQL Dump:

Test	Result
When the query is successfully executed	The query is stored in the sqldump file except for select query
When the query is not executed	The event is not stored in sqldump file

Evidence of Testing:

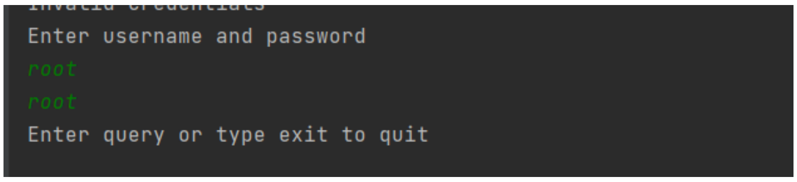
Login Validation:

- Invalid credentials



```
Enter username and password
root
dasa
Invalid credentials
Enter username and password
```

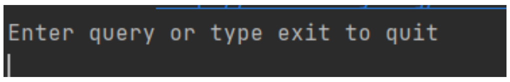
- Valid credentials



```
Enter username and password
root
root
Enter query or type exit to quit
```

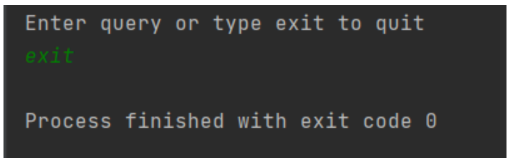
Main driver class:

- After user enters valid credentials then the system asks the user to enter queries



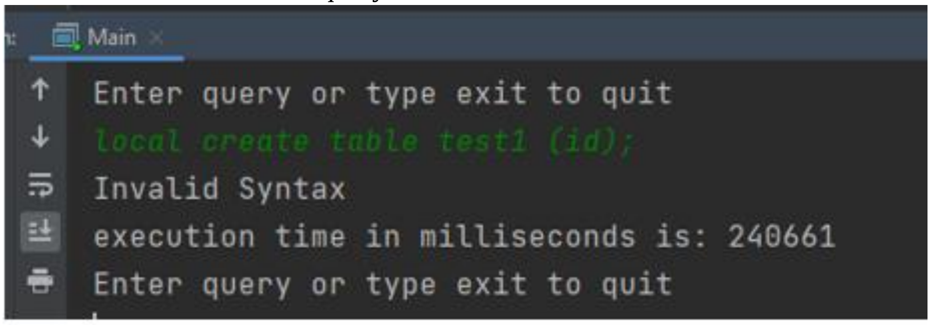
```
Enter query or type exit to quit
```

- The application ends when the user enters exit

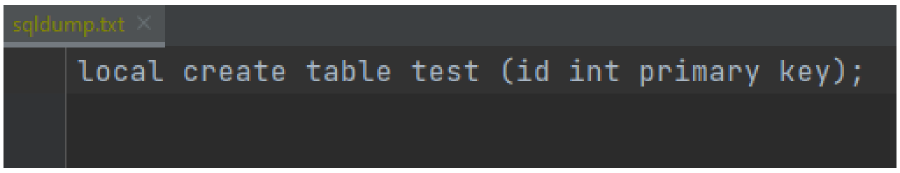

```
Enter query or type exit to quit
exit

Process finished with exit code 0
```

Sqldump:

- Unsuccessful execution of query

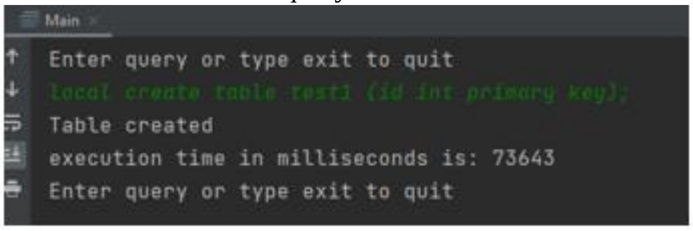


```
Enter query or type exit to quit
local create table test1 (id);
Invalid Syntax
execution time in milliseconds is: 240661
Enter query or type exit to quit
```

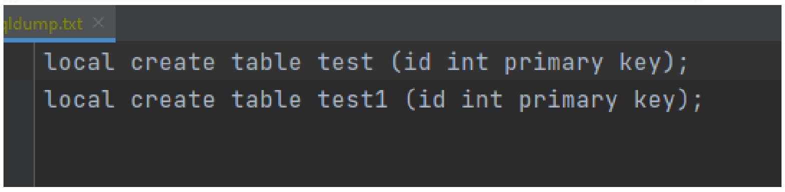


```
local create table test (id int primary key);
```

- Successful execution of query



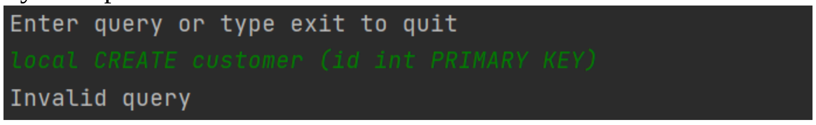
```
Enter query or type exit to quit
local create table test1 (id int primary key);
Table created
execution time in milliseconds is: 73643
Enter query or type exit to quit
```



```
local create table test (id int primary key);
local create table test1 (id int primary key);
```

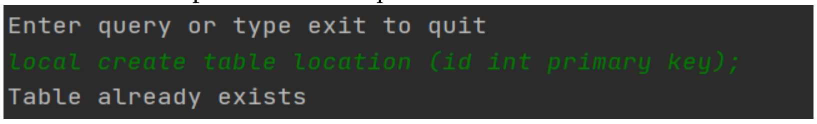
Create Query:

- Syntax specified is correct.



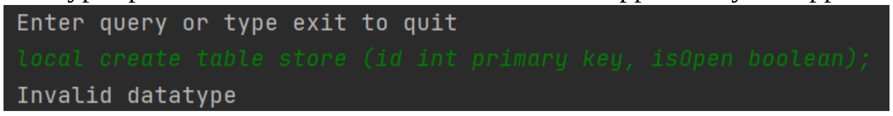
```
Enter query or type exit to quit
local CREATE customer (id int PRIMARY KEY)
Invalid query
```

- The table name provided is unique.



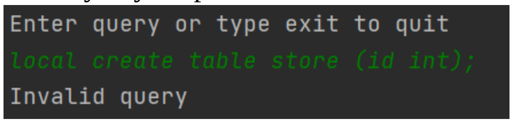
```
Enter query or type exit to quit
local create table location (id int primary key);
Table already exists
```

- Datatype specified for each column is correct and supported by the application.



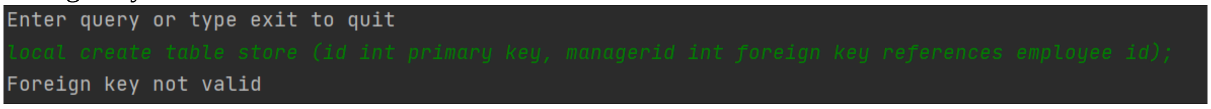
```
Enter query or type exit to quit
local create table store (id int primary key, isOpen boolean);
Invalid datatype
```

- Primary key is specified.



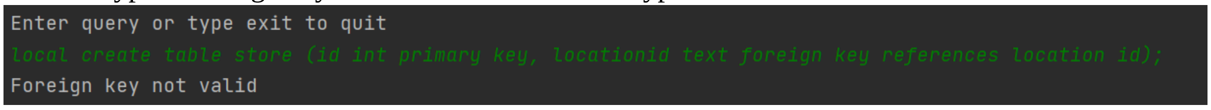
```
Enter query or type exit to quit
local create table store (id int);
Invalid query
```

- Foreign key referenced table exists.



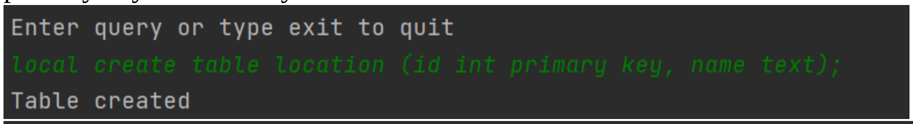
```
Enter query or type exit to quit
local create table store (id int primary key, managerid int foreign key references employee id);
Foreign key not valid
```

- The datatype of foreign key column matches the data type of referenced column.

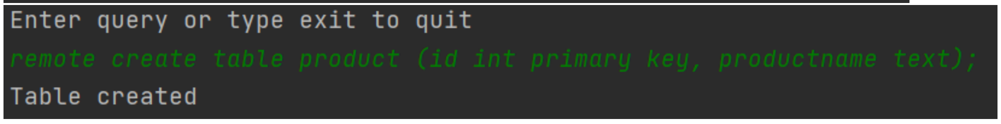


```
Enter query or type exit to quit
local create table store (id int primary key, locationid text foreign key references location id);
Foreign key not valid
```

- Works for creating tables in local as well as remote database folder from local site, containing primary key mandatorily.

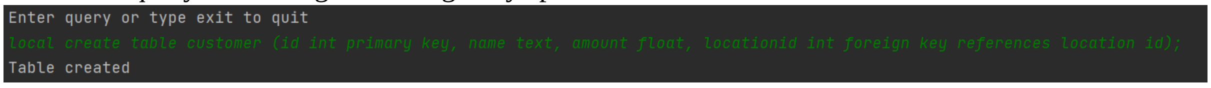


```
Enter query or type exit to quit
local create table location (id int primary key, name text);
Table created
```



```
Enter query or type exit to quit
remote create table product (id int primary key, productname text);
Table created
```

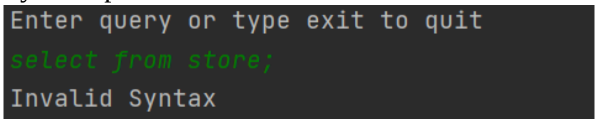
- Works for query containing one foreign key specified.



```
Enter query or type exit to quit
local create table customer (id int primary key, name text, amount float, locationid int foreign key references location id);
Table created
```

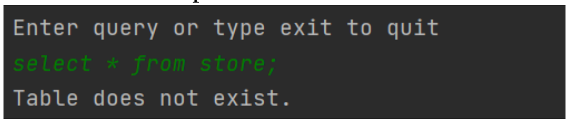
Select Query:

- Syntax specified is correct.



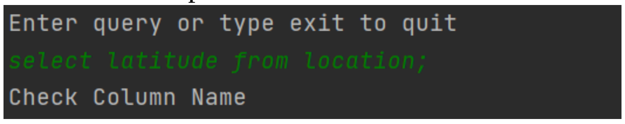
```
Enter query or type exit to quit
select from store;
Invalid Syntax
```

- The table name provided exists.



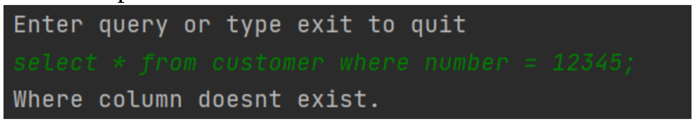
```
Enter query or type exit to quit
select * from store;
Table does not exist.
```

- Column name specified exists



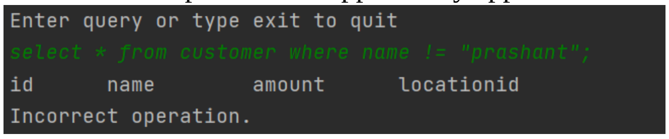
```
Enter query or type exit to quit
select latitude from location;
Check Column Name
```

- Column specified in where clause exists



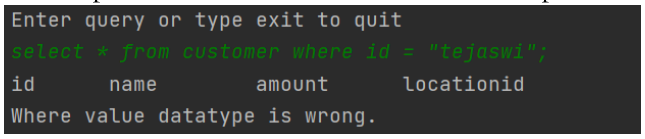
```
Enter query or type exit to quit
select * from customer where number = 12345;
Where column doesnt exist.
```

- Where clause operation is supported by application



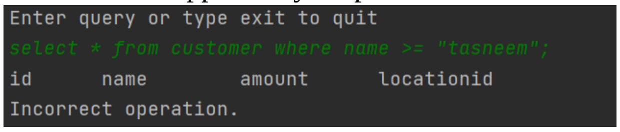
```
Enter query or type exit to quit
select * from customer where name != "prashant";
id      name        amount      locationid
Incorrect operation.
```

- Value specified for where clause column comparison is of correct datatype



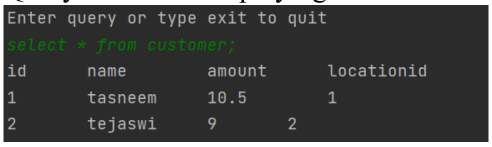
```
Enter query or type exit to quit
select * from customer where id = "tejaswi";
id      name         amount      locationid
Where value datatype is wrong.
```

- Text column support only = operator



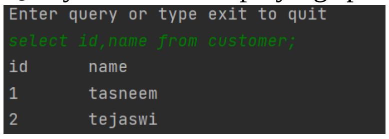
```
Enter query or type exit to quit
select * from customer where name >= "tasneem";
id      name         amount      locationid
Incorrect operation.
```

- Query works for displaying all columns using '*'



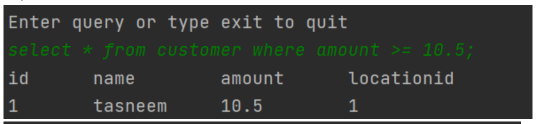
```
Enter query or type exit to quit
select * from customer;
id      name         amount      locationid
1       tasneem      10.5        1
2       tejaswi      9       2
```

- Query works for displaying specified columns in query.

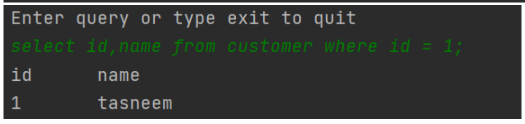


```
Enter query or type exit to quit
select id,name from customer;
id      name
1       tasneem
2       tejaswi
```

- Query works with where clause specified. The where clause operations supported are =, <=, >=, <, >.

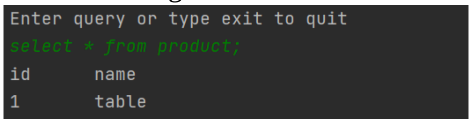


```
Enter query or type exit to quit
select * from customer where amount >= 10.5;
id      name         amount      locationid
1       tasneem      10.5        1
```



```
Enter query or type exit to quit
select id,name from customer where id = 1;
id      name
1       tasneem
```

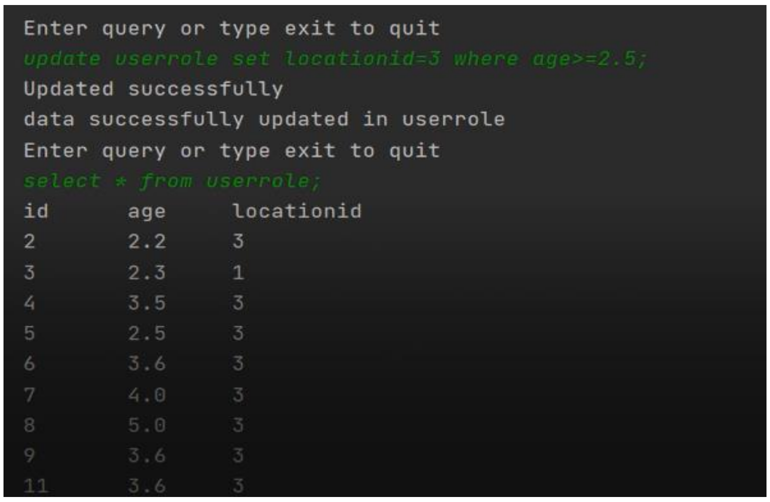
- Tables existing on local as well as remote database folder can be queried.



```
Enter query or type exit to quit
select * from product;
id      name
1       table
```

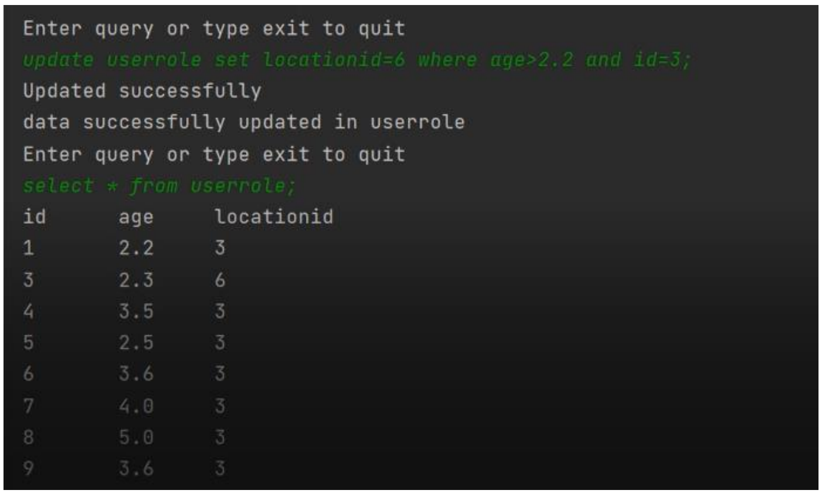
Update Query:

- Update single column value with single where condition:



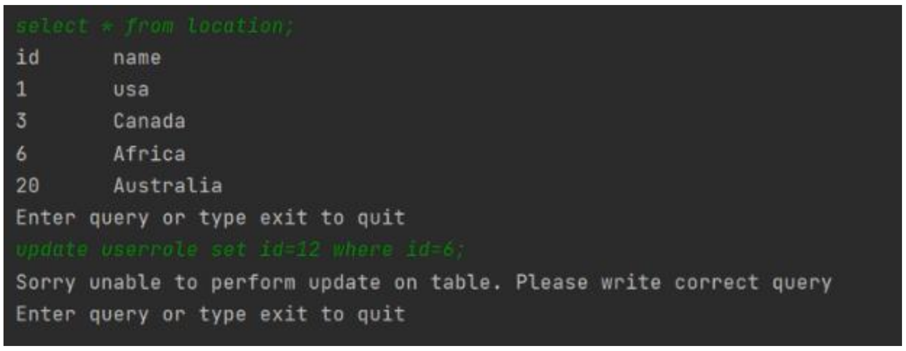
```
Enter query or type exit to quit
update userrole set locationid=3 where age>=2.5;
Updated successfully
data successfully updated in userrole
Enter query or type exit to quit
select * from userrole;
id       age      locationid
2        2.2      3
3        2.3      1
4        3.5      3
5        2.5      3
6        3.6      3
7        4.0      3
8        5.0      3
9        3.6      3
11       3.6      3
```

- Update single column with multiple where condition as well as it s updating foreign key column value:



```
Enter query or type exit to quit
update userrole set locationid=6 where age>2.2 and id=3;
Updated successfully
data successfully updated in userrole
Enter query or type exit to quit
select * from userrole;
id      age     locationid
1       2.2     3
3       2.3     6
4       3.5     3
5       2.5     3
6       3.6     3
7       4.0     3
8       5.0     3
9       3.6     3
```

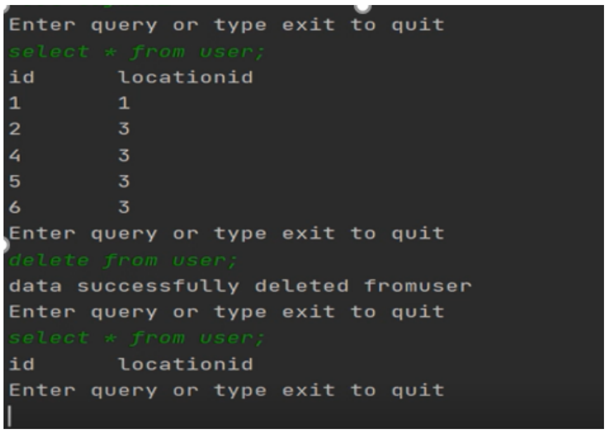
- Update primary column value of table which is referenced by another table and value of primary key column exists in it’s child table.



```
select * from location;
id      name
1       usa
3       Canada
6       Africa
20      Australia
Enter query or type exit to quit
update userrole set id=12 where id=6;
Sorry unable to perform update on table. Please write correct query
Enter query or type exit to quit
```

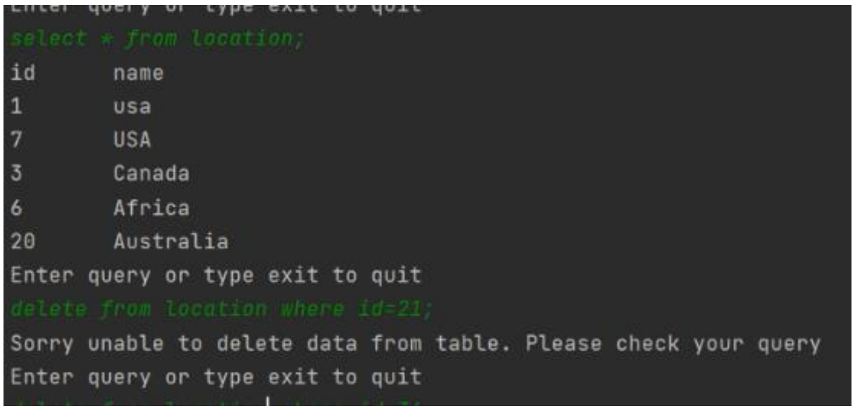
Delete Query:

- Delete all the data from the table:



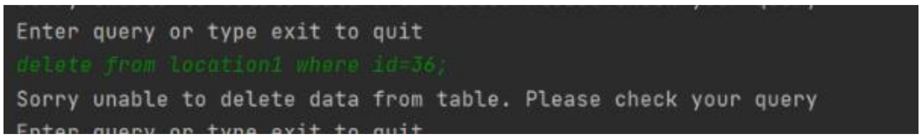
```
Enter query or type exit to quit
select * from user;
id       locationid
1        1
2        3
4        3
5        3
6        3
Enter query or type exit to quit
delete from user;
data successfully deleted fromuser
Enter query or type exit to quit
select * from user;
id       locationid
Enter query or type exit to quit
```

- Delete data from table with primary key which does not exist in table:



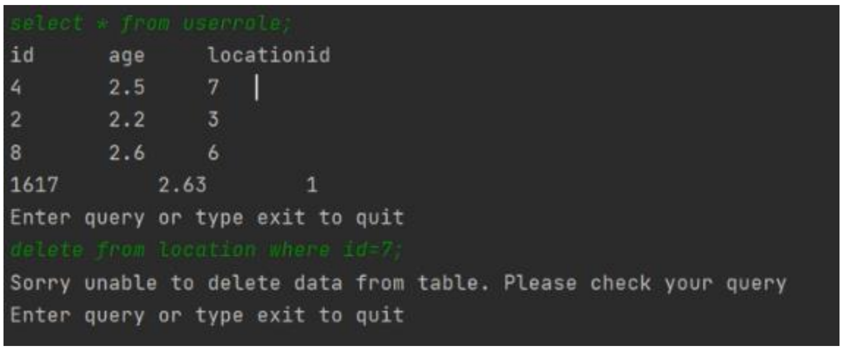
```
Enter query or type exit to quit
select * from location;
id      name
1       usa
7       USA
3       Canada
6       Africa
20      Australia
Enter query or type exit to quit
delete from location where id=21;
Sorry unable to delete data from table. Please check your query
Enter query or type exit to quit
```

- Delete data from table which does not exist.



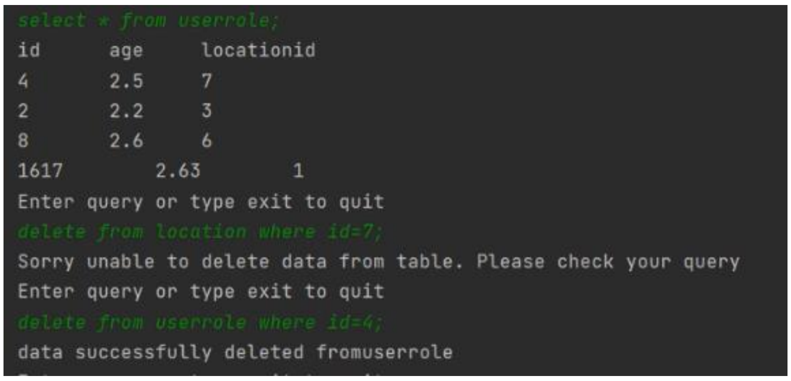
```
Enter query or type exit to quit
delete from location1 where id=36;
Sorry unable to delete data from table. Please check your query
Enter query or type exit to quit
```

- Delete value of column which exists in another table foreign key column that is referencing current table.



```
select * from userrole;
id      age     locationid
4       2.5     7
2       2.2     3
8       2.6     6
1617        2.63        1
Enter query or type exit to quit
delete from location where id=7;
Sorry unable to delete data from table. Please check your query
Enter query or type exit to quit
```

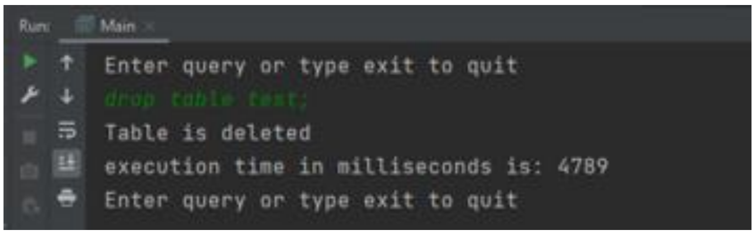
- Delete value of column which does not exist in another table foreign key column that is referencing current table.



```
select * from userrole;
id      age     locationid
4       2.5     7
2       2.2     3
8       2.6     6
1617        2.63        1
Enter query or type exit to quit
delete from location where id=7;
Sorry unable to delete data from table. Please check your query
Enter query or type exit to quit
delete from userrole where id=4;
data successfully deleted fromuserrole
```

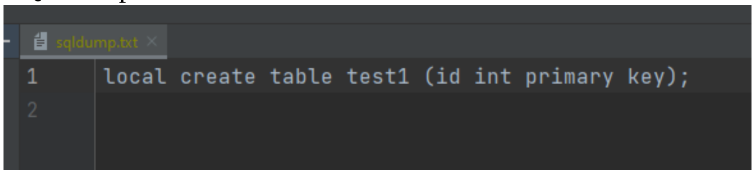
Drop table:

- Successful deletion of table



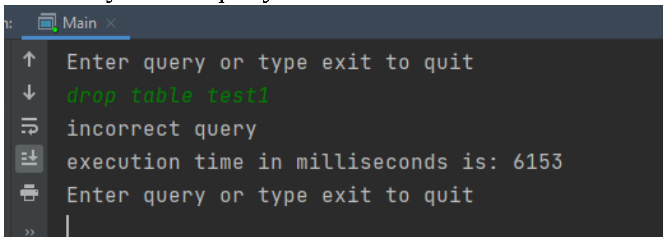
```
Enter query or type exit to quit
drop table test;
Table is deleted
execution time in milliseconds is: 4789
Enter query or type exit to quit
```

SQL dump on successful delete:



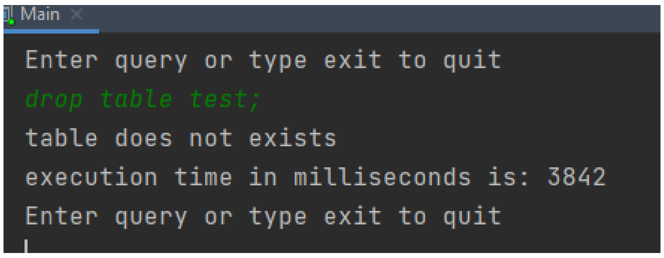
```
local create table test1 (id int primary key);
```

- Invalid syntax of query



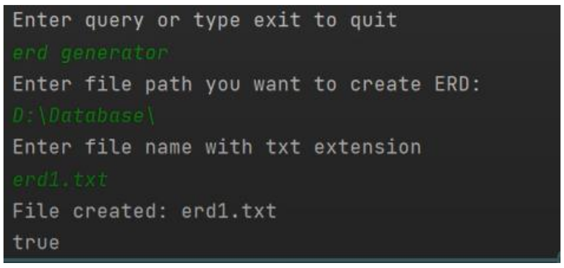
```
Enter query or type exit to quit
drop table test1
incorrect query
execution time in milliseconds is: 6153
Enter query or type exit to quit
```

- Table is not available



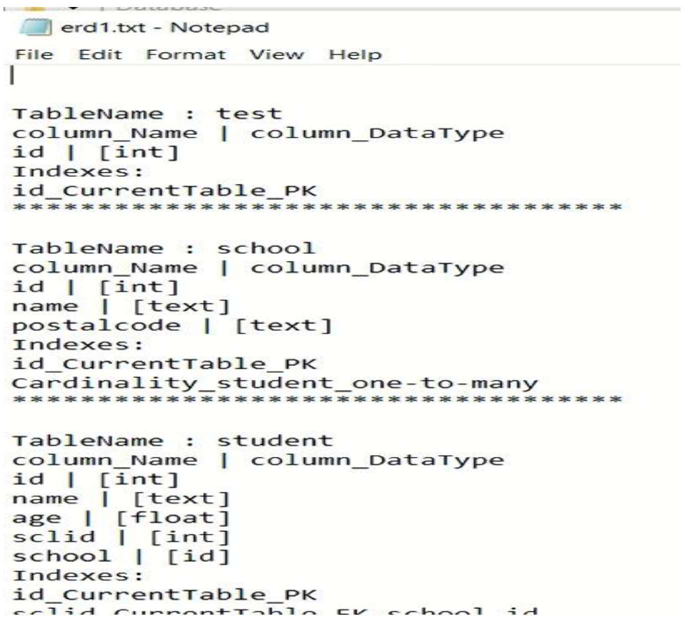
```
Enter query or type exit to quit
drop table test;
table does not exists
execution time in milliseconds is: 3842
Enter query or type exit to quit
```

Generate ERD:



```
Enter query or type exit to quit
erd generator
Enter file path you want to create ERD:
D:\Database\
Enter file name with txt extension
erd1.txt
File created: erd1.txt
true
```

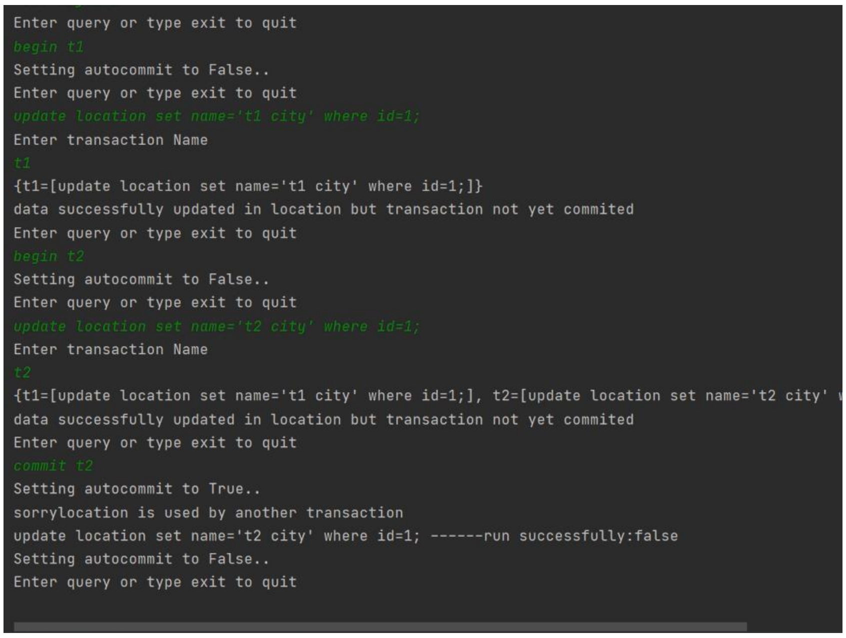
Generated ERD file:



```
TableName : test
column_Name | column_DataType
id | [int]
Indexes:
id_CurrentTable_PK
*************************************

TableName : school
column_Name | column_DataType
id | [int]
name | [text]
postalcode | [text]
Indexes:
id_CurrentTable_PK
Cardinality_student_one-to-many
*************************************

TableName : student
column_Name | column_DataType
id | [int]
name | [text]
age | [float]
sclid | [int]
school | [id]
Indexes:
id_CurrentTable_PK
sclid_CurrentTable_FK_school_id
```

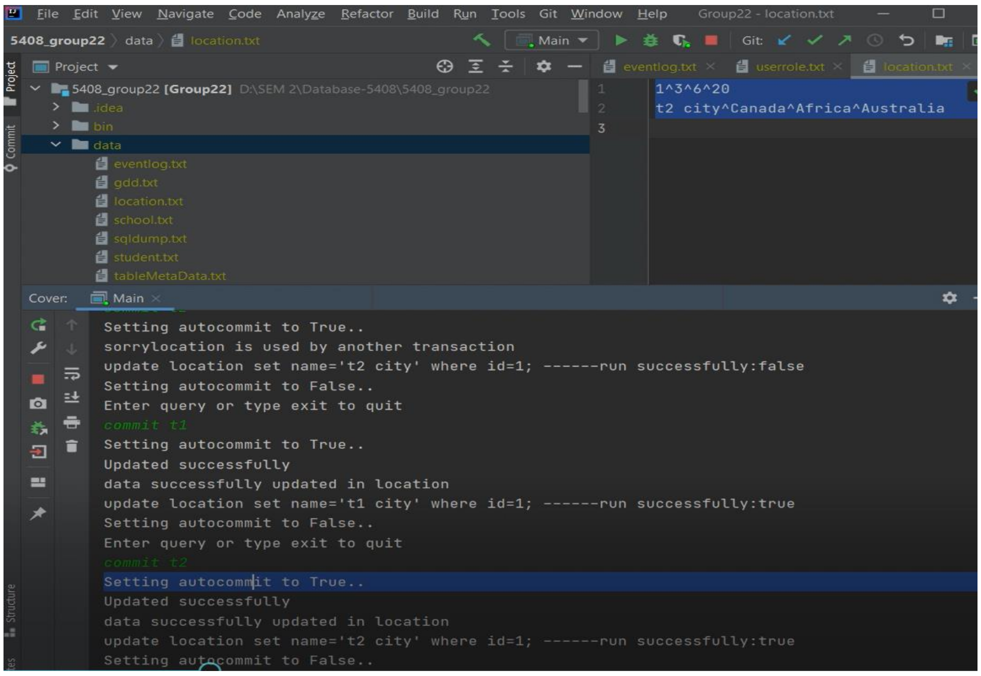
Transactions:

- Multiple transaction where both transactions try to write data of same table, and transaction 2 is not allowed to commit before transaction 1 because table is locked by transaction 1.



```
Enter query or type exit to quit
begin t1
Setting autocommit to False..
Enter query or type exit to quit
update location set name='t1 city' where id=1;
Enter transaction Name
t1
{t1=[update location set name='t1 city' where id=1;]}
data successfully updated in location but transaction not yet commited
Enter query or type exit to quit
begin t2
Setting autocommit to False..
Enter query or type exit to quit
update location set name='t2 city' where id=1;
Enter transaction Name
t2
{t1=[update location set name='t1 city' where id=1;], t2=[update location set name='t2 city' 
data successfully updated in location but transaction not yet commited
Enter query or type exit to quit
commit t2
Setting autocommit to True..
sorrylocation is used by another transaction
update location set name='t2 city' where id=1; ------run successfully:false
Setting autocommit to False..
Enter query or type exit to quit
```

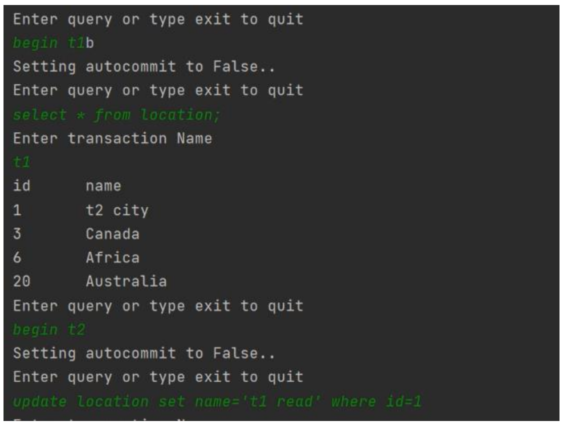
T2 is allowed to execute query after T1 is committed.



```
1^3^6^20
t2 city^Canada^Africa^Australia
```

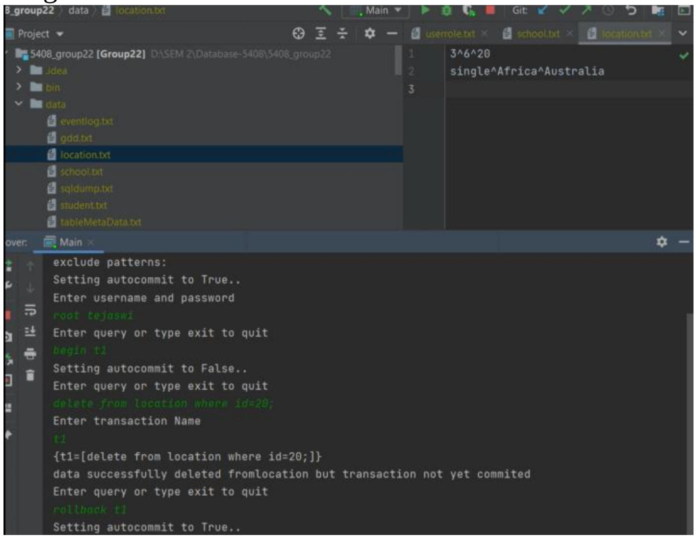
```
Setting autocommit to True..
sorrylocation is used by another transaction
update location set name='t2 city' where id=1; ------run successfully:false
Setting autocommit to False..
Enter query or type exit to quit
commit t1
Setting autocommit to True..
Updated successfully
data successfully updated in location
update location set name='t1 city' where id=1; ------run successfully:true
Setting autocommit to False..
Enter query or type exit to quit
commit t2
Setting autocommit to True..
Updated successfully
data successfully updated in location
update location set name='t2 city' where id=1; ------run successfully:true
Setting autocommit to False..
```

- Multiple transaction where T1 reads before T2 writes data on the same table, and T2 tires to commit before T1 commits.



```
Enter query or type exit to quit
begin t1b
Setting autocommit to False..
Enter query or type exit to quit
select * from location;
Enter transaction Name
t1
id      name
1       t2 city
3       Canada
6       Africa
20      Australia
Enter query or type exit to quit
begin t2
Setting autocommit to False..
Enter query or type exit to quit
update location set name='t1 read' where id=1
```

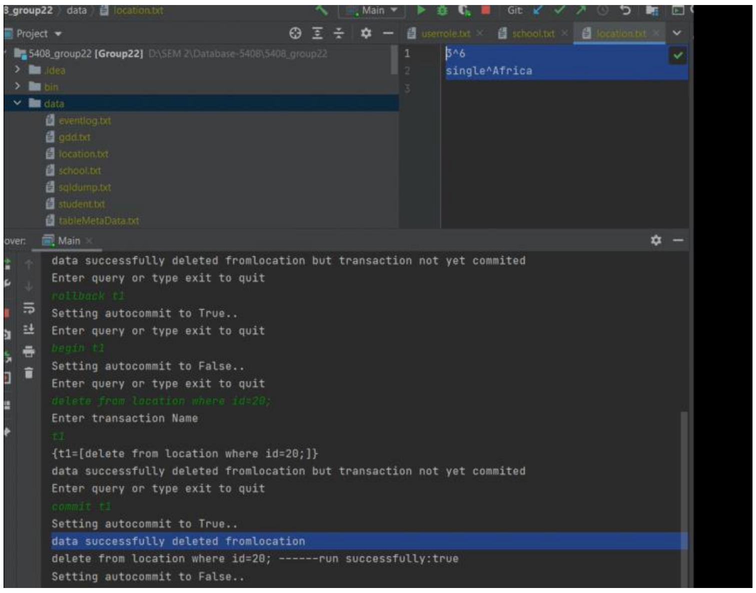
- Single transaction with rollback.



```
3^6^20
single^Africa^Australia
```

```
exclude patterns:
Setting autocommit to True..
Enter username and password
root tejaswi
Enter query or type exit to quit
begin t1
Setting autocommit to False..
Enter query or type exit to quit
delete from location where id=20;
Enter transaction Name
t1
{t1=[delete from location where id=20;]}
data successfully deleted fromlocation but transaction not yet commited
Enter query or type exit to quit
rollback t1
Setting autocommit to True..
```

- Single transaction with commit:

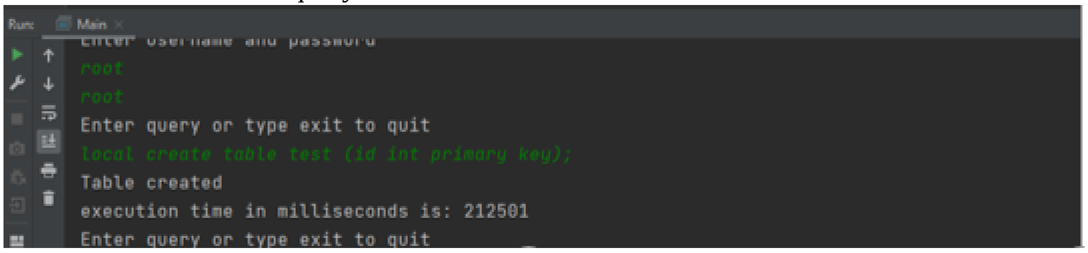


```
5^6
single^Africa
```

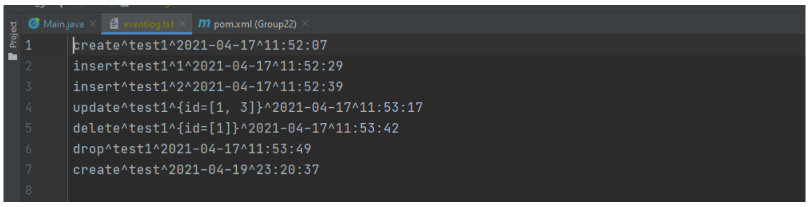
```
data successfully deleted fromlocation but transaction not yet commited
Enter query or type exit to quit
rollback t1
Setting autocommit to True..
Enter query or type exit to quit
begin t1
Setting autocommit to False..
Enter query or type exit to quit
delete from location where id=20;
Enter transaction Name
t1
{t1=[delete from location where id=20;]}
data successfully deleted fromlocation but transaction not yet commited
Enter query or type exit to quit
commit t1
Setting autocommit to True..
data successfully deleted fromlocation
delete from location where id=20; ------run successfully:true
Setting autocommit to False..
```

Event Log:

- Successful execution of query

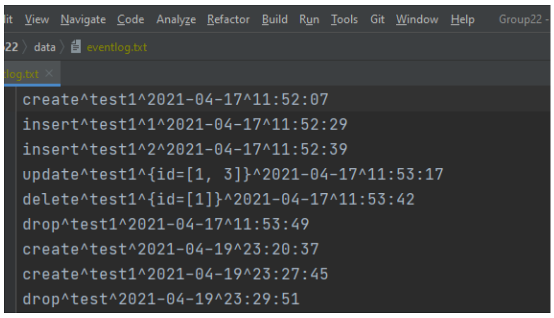


```
Enter username and password
root
root
Enter query or type exit to quit
local create table test (id int primary key);
Table created
execution time in milliseconds is: 212501
Enter query or type exit to quit
```

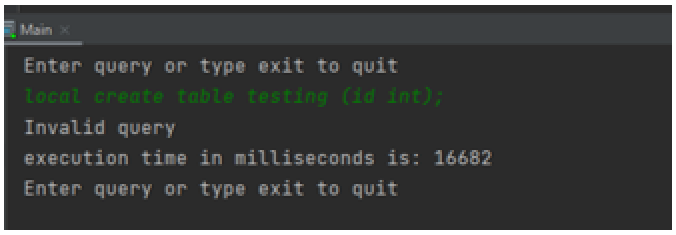


```
create^test1^2021-04-17^11:52:07
insert^test1^1^2021-04-17^11:52:29
insert^test1^2^2021-04-17^11:52:39
update^test1^{id=[1, 3]}^2021-04-17^11:53:17
delete^test1^{id=[1]}^2021-04-17^11:53:42
drop^test1^2021-04-17^11:53:49
create^test^2021-04-19^23:20:37
```

- Unsuccessful execution of query



```
create^test1^2021-04-17^11:52:07
insert^test1^1^2021-04-17^11:52:29
insert^test1^2^2021-04-17^11:52:39
update^test1^{id=[1, 3]}^2021-04-17^11:53:17
delete^test1^{id=[1]}^2021-04-17^11:53:42
drop^test1^2021-04-17^11:53:49
create^test^2021-04-19^23:20:37
create^test1^2021-04-19^23:27:45
drop^test^2021-04-19^23:29:51
```



```
Enter query or type exit to quit
local create table testing (id int);
Invalid query
execution time in milliseconds is: 16682
Enter query or type exit to quit
```

Limitation and Future Work:

Limitations

The limitations present in the database management system are:

1. At one local machine, only one database is present.
2. At a time, a single user can use the database from the local machine.
3. The datatypes supported are text, real numbers and integers only.
4. The ORDER BY and GROUP BY clauses are not supported by the database management system
5. No aggregate functions are provided by the DBMS
6. Table cannot be altered after creation.
7. A single transaction can work on the database at a particular time
8. Certain syntax specific whitespaces need to be provided, for example, a whitespace after table name in CREATE query.
9. Only one foreign key column can be present in a table

Future Work:

The distributed database management system can be extended further in the following ways:

1. Support for multiple foreign keys
2. Support ORDER BY and GROUP BY clauses
3. Support aggregate functions
4. Support for multiple distributed transactions
5. Storing old value in the logs as well when an update query is executed
6. Support for multiple user logins could be implemented.

References:

[1] "BCryptPasswordEncoder (spring-security-docs-manual 5.4.6 API)", Docs.spring.io. [Online]. Available: https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.html. [Accessed: 20- Apr- 2021]