DATABASE MANAGEMENT SYSTEM AND CONNECTIVITY

Technical Report

Table of content

DBMS Section:

JDBC Section:

DBMS SECTION

1. Project Description:

A Computer Database is a structured collection of records or data that is stored in a computer system. On the other hand, a Database Management System (DBMS) is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database. DBMS are categorized according to their data structures or types. The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

2. Project Design:

Packaging Design

2.1.1. Interface package:

Contains both the Database Interface and the DBHandler Interface used with all saving and loading action.

2.1.2. View package:

Contains the SQL program view

2.1.3. Parse package:

Contains the validation class of any input query

2.1.4. ADTs package:

Contains the database data structure and the DBHandler implementation

2.1.5. Model package:

Contains the database interface.

Design Patterns

2.1.6. Composite design pattern:

For each database created or loaded a DatabaseManager object will be created that contains its relevant Tables –

contained in an array list of DatabaseTables- and these contains a List of Column objects.

Each column can be filled with a List of Elements that are created objects as well and user can set these objects at any desirable time. When anything is updated , an action occur to update them in the files of the table in the table.

3. Assumptions:

User Inputs one Query at a time (multiple sentence queries are not supported at the same time)

Any name could contain the following characters and only them (. - _ ‘) same as emails.

On Inserting/Updating a table with VARCHAR columns no single or double quotation is needed.

User can’t use the following statement unless all values written are one-to-one in correspondence with the given table.

“Insert into mytable values (name1, name2, name3)”

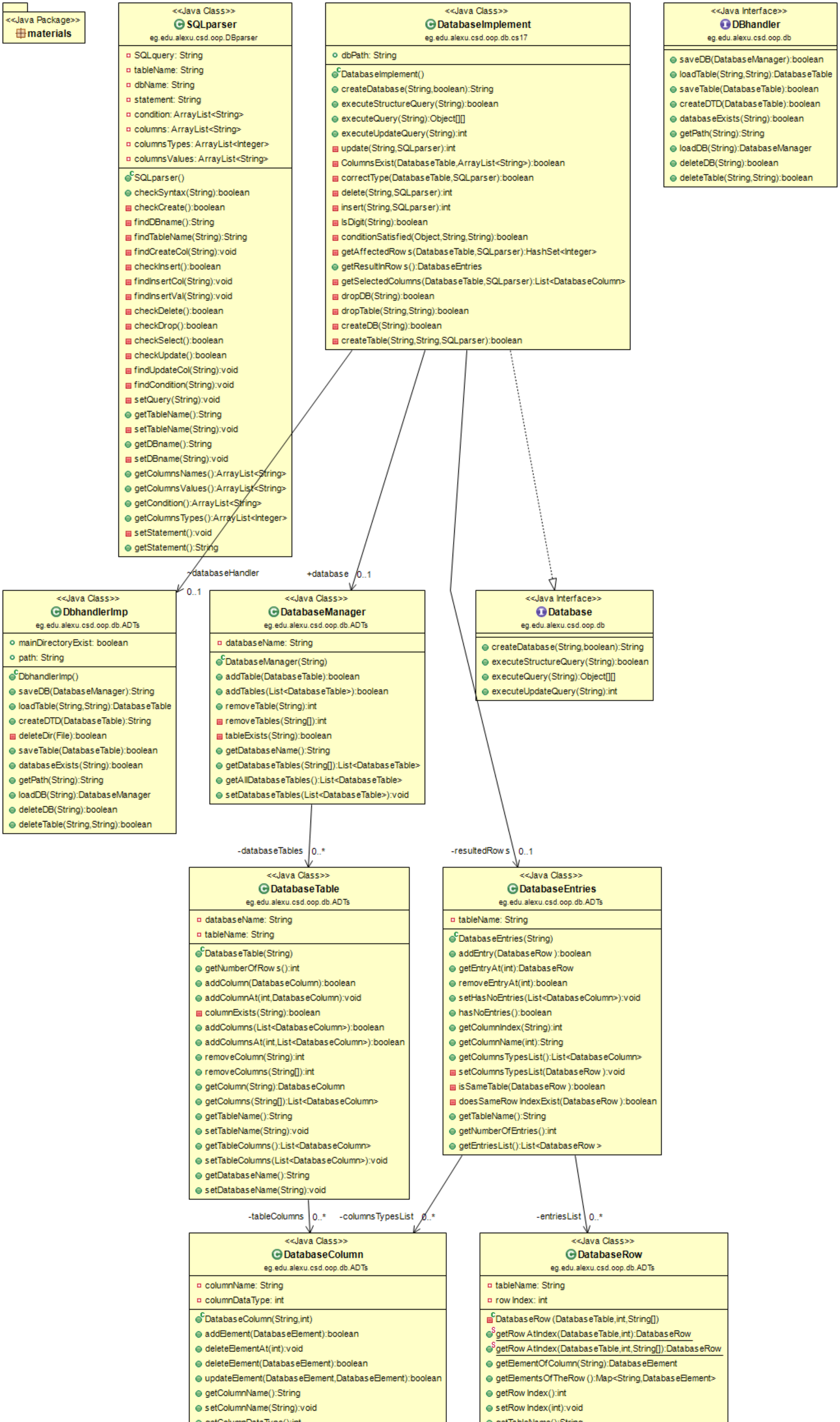
Due to the presence of 800+ words preserved for SQL, the user is given the right to name his table whatever they decide –ignoring the preserved words.

Both > , < are not supported for String values.

On creating any of the following, database, table or column while already a previous version exists an SQL error is shown.

4. UML Diagram:

- UML Class Diagram



<<Java Package>>
materials
<<Java Class>>
SQLparser
eg.edu.alexu.csd.oop.DBparser
SQLquery: String
tableName: String
dbName: String
statement: String
condition: ArrayList<String>
columns: ArrayList<String>
columnsTypes: ArrayList<Integer>
columnsValues: ArrayList<String>
SQLparser()
checkSyntax(String):boolean
checkCreate():boolean
findDBname():String
findTableName(String):String
findCreateCol(String):void
checkInsert():boolean
findInsertCol(String):void
findInsertVal(String):void
checkDelete():boolean
checkDrop():boolean
checkSelect():boolean
checkUpdate():boolean
findUpdateCol(String):void
findCondition(String):void
setQuery(String):void
getTableName():String
setTableName(String):void
getDBname():String
setDBname(String):void
getColumnsNames():ArrayList<String>
getColumnsValues():ArrayList<String>
getCondition():ArrayList<String>
getColumnsTypes():ArrayList<Integer>
setStatement():void
getStatement():String
<<Java Class>>
DatabaseImplement
eg.edu.alexu.csd.oop.db.cs17
dbPath: String
DatabaseImplement()
createDatabase(String,boolean):String
executeStructureQuery(String):boolean
executeQuery(String):Object[][]
executeUpdateQuery(String):int
update(String,SQLparser):int
ColumnsExist(DatabaseTable,ArrayList<String>):boolean
correctType(DatabaseTable,SQLparser):boolean
delete(String,SQLparser):int
insert(String,SQLparser):int
IsDigit(String):boolean
conditionSatisfied(Object,String,String):boolean
getAffectedRows(DatabaseTable,SQLparser):HashSet<Integer>
getResultInRows():DatabaseEntries
getSelectedColumns(DatabaseTable,SQLparser):List<DatabaseColumn>
dropDB(String):boolean
dropTable(String,String):boolean
createDB(String):boolean
createTable(String,String,SQLparser):boolean
<<Java Interface>>
DBhandler
eg.edu.alexu.csd.oop.db
saveDB(DatabaseManager):boolean
loadTable(String,String):DatabaseTable
saveTable(DatabaseTable):boolean
createDTD(DatabaseTable):boolean
databaseExists(String):boolean
getPath(String):String
loadDB(String):DatabaseManager
deleteDB(String):boolean
deleteTable(String,String):boolean
~databaseHandler
0..1
+database
0..1
<<Java Class>>
DbhandlerImp
eg.edu.alexu.csd.oop.db.ADTs
mainDirectoryExist: boolean
path: String
DbhandlerImp()
saveDB(DatabaseManager):String
loadTable(String,String):DatabaseTable
createDTD(DatabaseTable):String
deleteDir(File):boolean
saveTable(DatabaseTable):boolean
databaseExists(String):boolean
getPath(String):String
loadDB(String):DatabaseManager
deleteDB(String):boolean
deleteTable(String,String):boolean
<<Java Class>>
DatabaseManager
eg.edu.alexu.csd.oop.db.ADTs
databaseName: String
DatabaseManager(String)
addTable(DatabaseTable):boolean
addTables(List<DatabaseTable>):boolean
removeTable(String):int
removeTables(String[]):int
tableExists(String):boolean
getDatabaseName():String
getDatabaseTables(String[]):List<DatabaseTable>
getAllDatabaseTables():List<DatabaseTable>
setDatabaseTables(List<DatabaseTable>):void
<<Java Interface>>
Database
eg.edu.alexu.csd.oop.db
createDatabase(String,boolean):String
executeStructureQuery(String):boolean
executeQuery(String):Object[][]
executeUpdateQuery(String):int
-databaseTables
0..*
-resultedRows
0..1
<<Java Class>>
DatabaseTable
eg.edu.alexu.csd.oop.db.ADTs
databaseName: String
tableName: String
DatabaseTable(String)
getNumberOfRows():int
addColumn(DatabaseColumn):boolean
addColumnAt(int,DatabaseColumn):void
columnExists(String):boolean
addColumns(List<DatabaseColumn>):boolean
addColumnsAt(int,List<DatabaseColumn>):boolean
removeColumn(String):int
removeColumns(String[]):int
getColumn(String):DatabaseColumn
getColumns(String[]):List<DatabaseColumn>
getTableName():String
setTableName(String):void
getTableColumns():List<DatabaseColumn>
setTableColumns(List<DatabaseColumn>):void
getDatabaseName():String
setDatabaseName(String):void
<<Java Class>>
DatabaseEntries
eg.edu.alexu.csd.oop.db.ADTs
tableName: String
DatabaseEntries(String)
addEntry(DatabaseRow):boolean
getEntryAt(int):DatabaseRow
removeEntryAt(int):boolean
setHasNoEntries(List<DatabaseColumn>):void
hasNoEntries():boolean
getColumnIndex(String):int
getColumnName(int):String
getColumnsTypesList():List<DatabaseColumn>
setColumnsTypesList(DatabaseRow):void
isSameTable(DatabaseRow):boolean
doesSameRowIndexExist(DatabaseRow):boolean
getTableName():String
getNumberOfEntries():int
getEntriesList():List<DatabaseRow>
-tableColumns
0..*
-columnsTypesList
0..*
-entriesList
0..*
<<Java Class>>
DatabaseColumn
eg.edu.alexu.csd.oop.db.ADTs
columnName: String
columnDataType: int
DatabaseColumn(String,int)
addElement(DatabaseElement):boolean
deleteElementAt(int):void
deleteElement(DatabaseElement):boolean
updateElement(DatabaseElement,DatabaseElement):boolean
getColumnName():String
setColumnName(String):void
getColumnDataType():int
<<Java Class>>
DatabaseRow
eg.edu.alexu.csd.oop.db.ADTs
tableName: String
rowIndex: int
DatabaseRow(DatabaseTable,int,String[])
getRowAtIndex(DatabaseTable,int):DatabaseRow
getRowAtIndex(DatabaseTable,int,String[]):DatabaseRow
getElementOfColumn(String):DatabaseElement
getElementsOfTheRow():Map<String,DatabaseElement>
getRowIndex():int
setRowIndex(int):void
getTableName():String

JDBC SECTION

1. Project Description:

Java Database Connectivity (JDBC) provides Java developers with a standard API that is used to access databases, regardless of the driver and database product. JDBC presents a uniform interface to databases - change vendors and your applications only need to change their driver.

2. Project Design:

a. Packaging Design

i. **Abstract Classes package:**
Contains all the abstract class that implement the following interfaces : Connection, Statement, ResultSet and ResultSetMetaData.
These Classes only contains the unsupported methods

ii. **jdbc package**:
Contains the five main classes that extends the abstract classes method above.

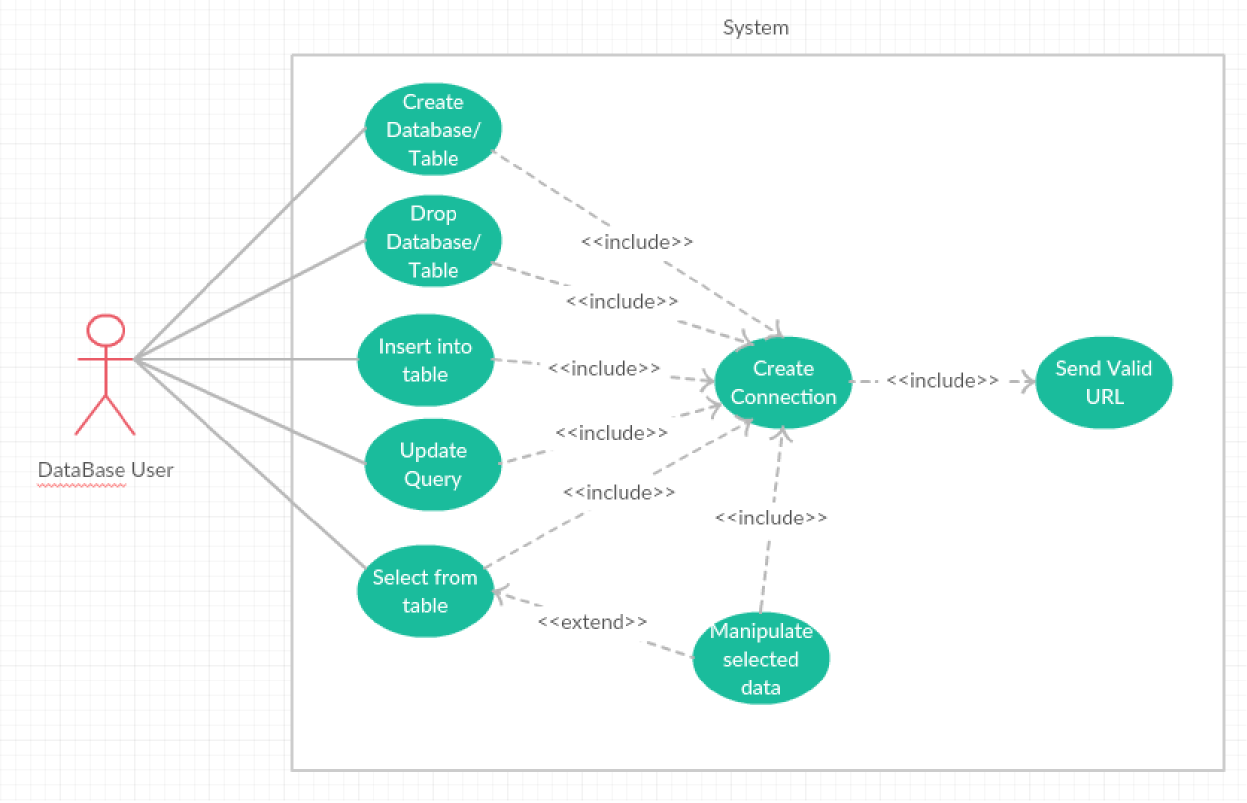
b. Design Patterns

i. **Adapter design pattern:**
Adapter design pattern was used for mainly two reason, one is its advantage of enabling the code to be flexible and reusable and the most important yet direct reason is organizing the code and not complicating the main classes with unnecessary methods that are unsupported.

ii. **MVC Design Pattern:**
Used to connect the Graphical user interface with the model, which the jdbc classes in our case.

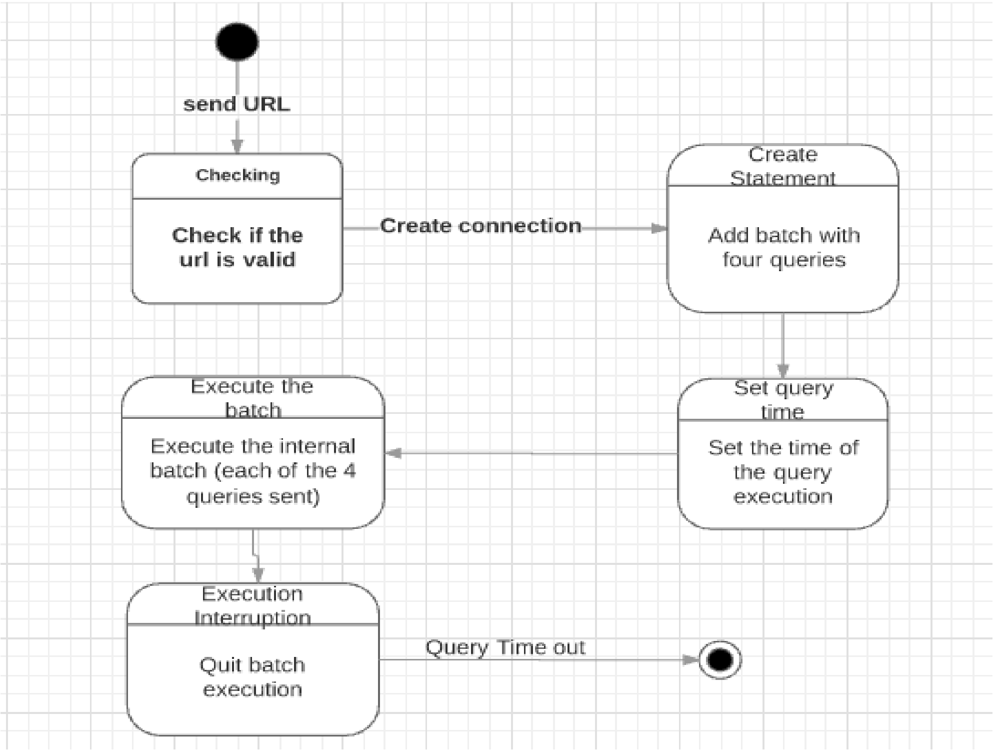
3. UML Diagrams:

- UML Use Case Diagram

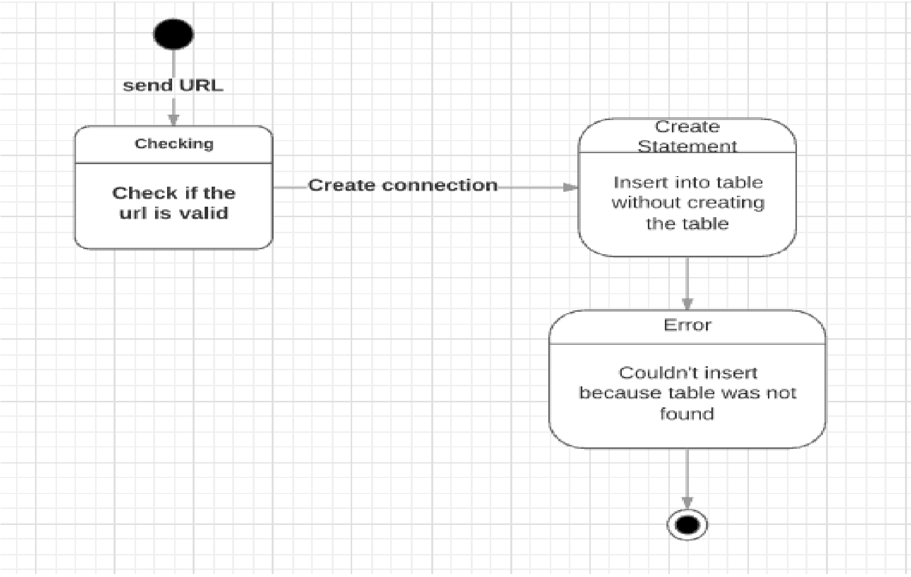


- Scenarios for State Diagrams

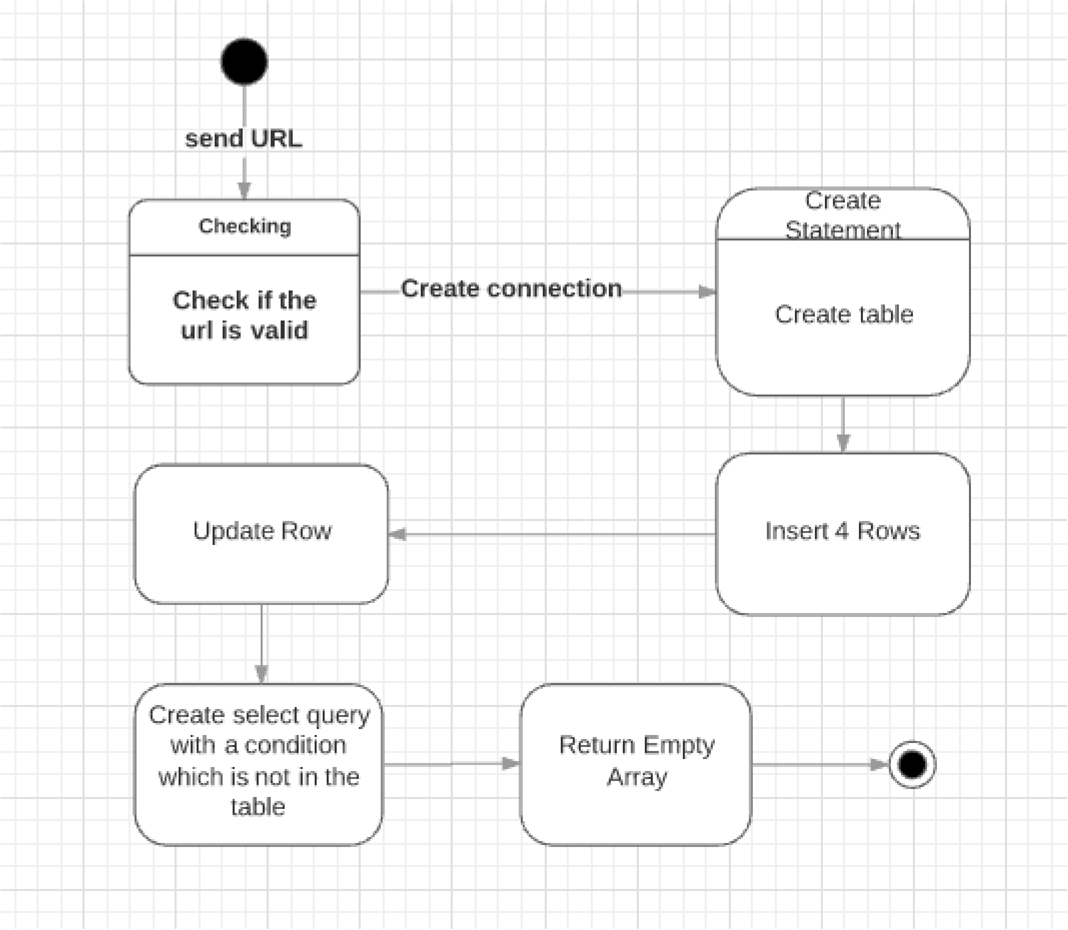
Scenario1:


send URL
Checking
Check if the url is valid
Create connection
Create Statement
Add batch with four queries
Set query time
Set the time of the query execution
Execute the batch
Execute the internal batch (each of the 4 queries sent)
Execution Interruption
Quit batch execution
Query Time out

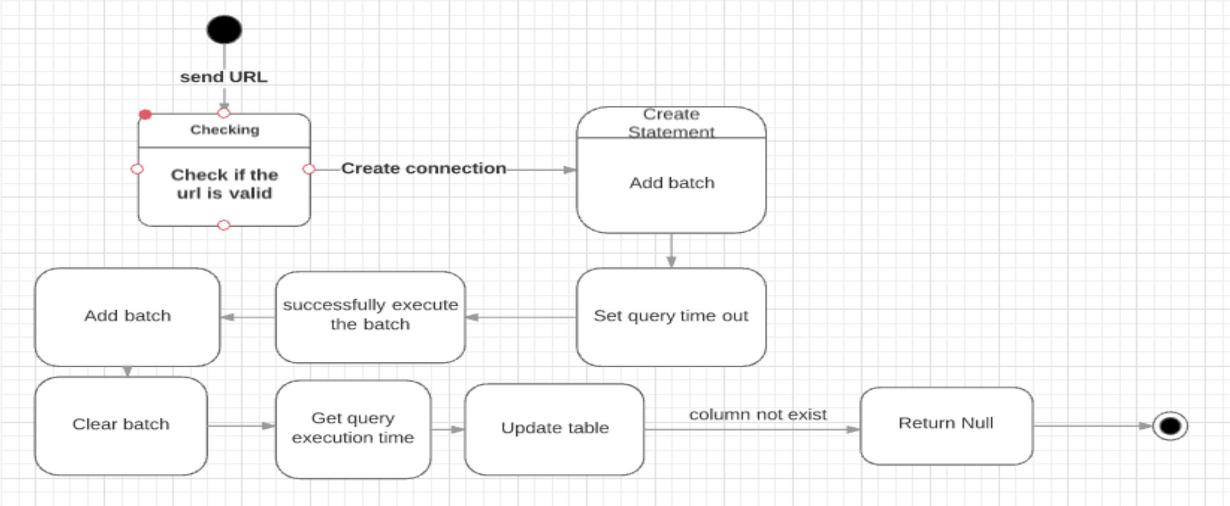
Scenario2:


send URL
Checking
Check if the url is valid
Create connection
Create Statement
Insert into table without creating the table
Error
Couldn't insert because table was not found

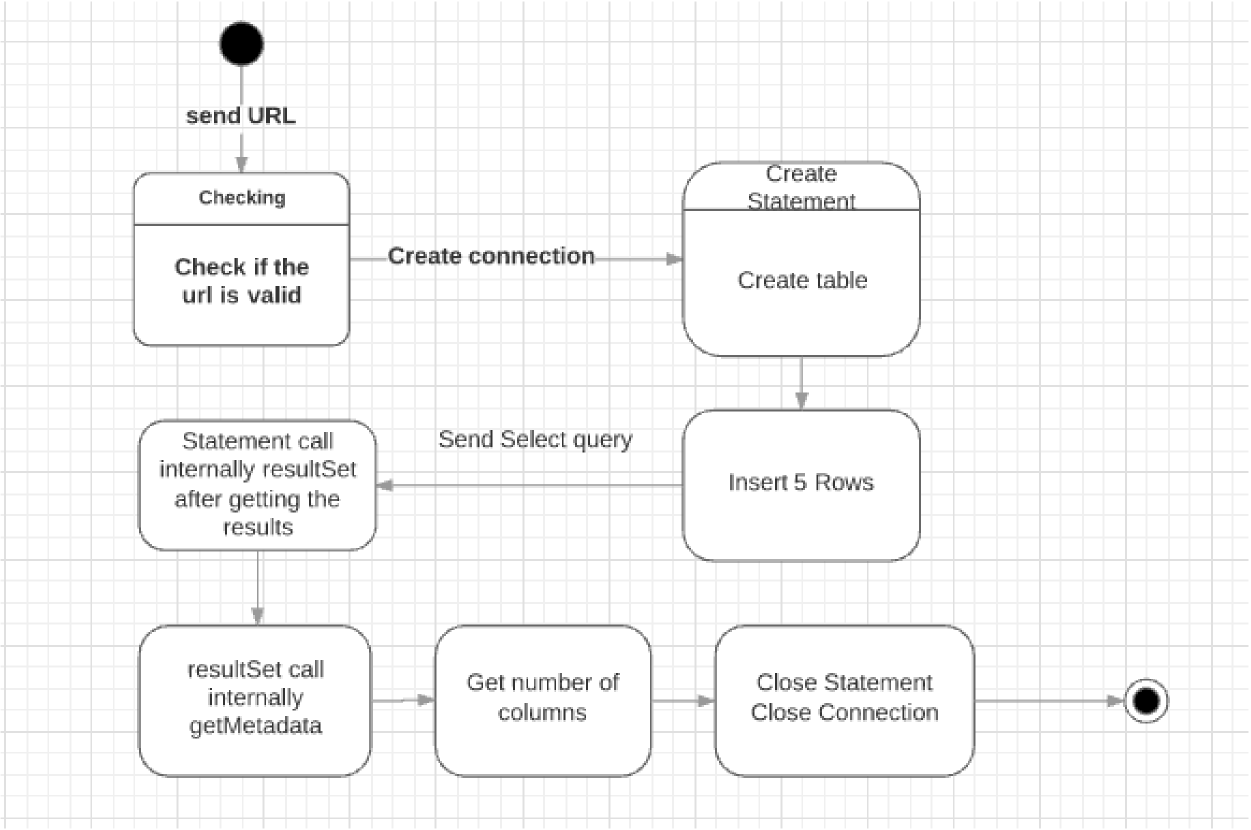
Scenario3:



Scenario4:

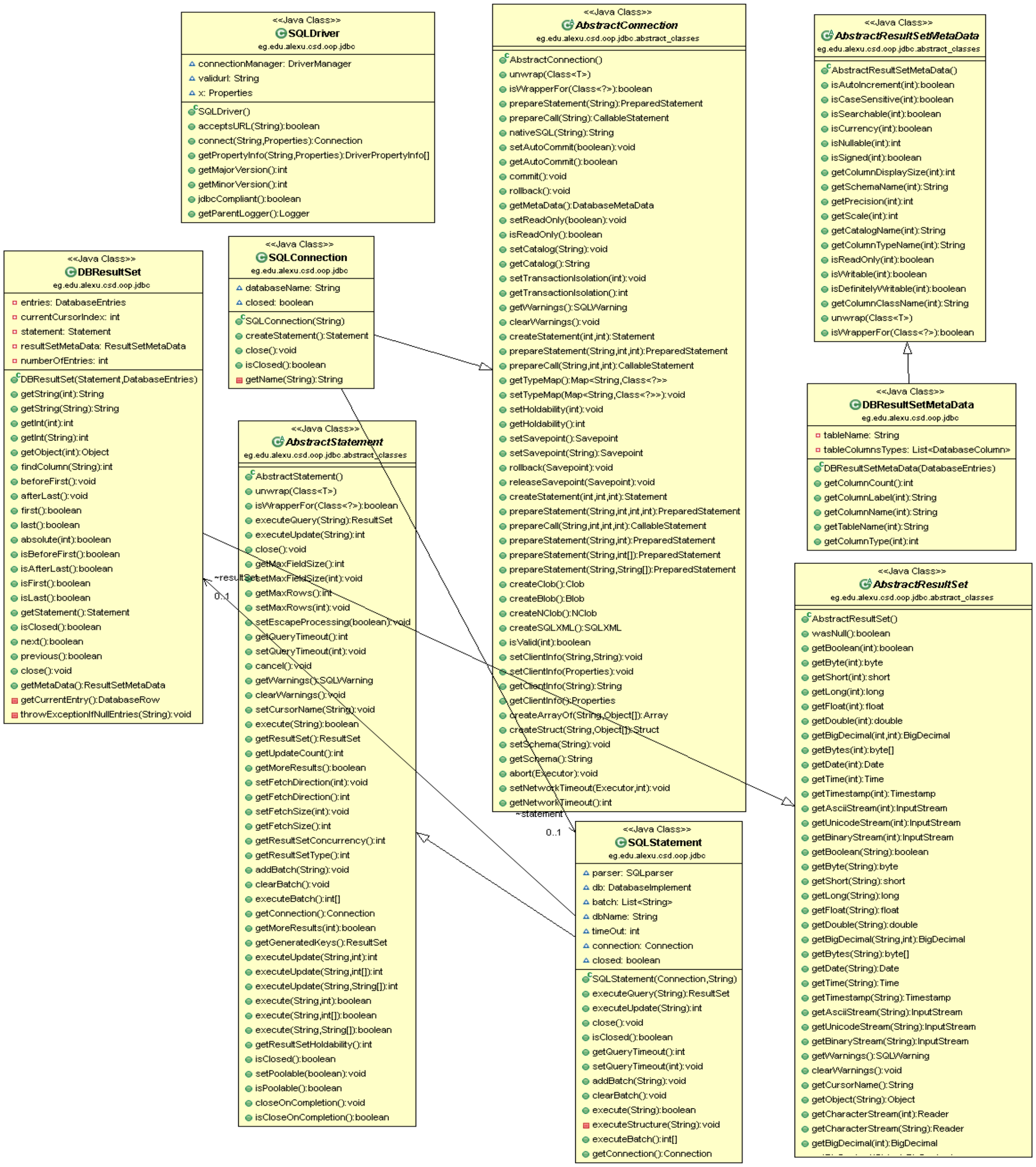


Scenario5:


send URL
Checking
Check if the url is valid
Create connection
Create Statement
Create table
Insert 5 Rows
Send Select query
Statement call internally resultSet after getting the results
resultSet call internally getMetadata
Get number of columns
Close Statement Close Connection

4. User

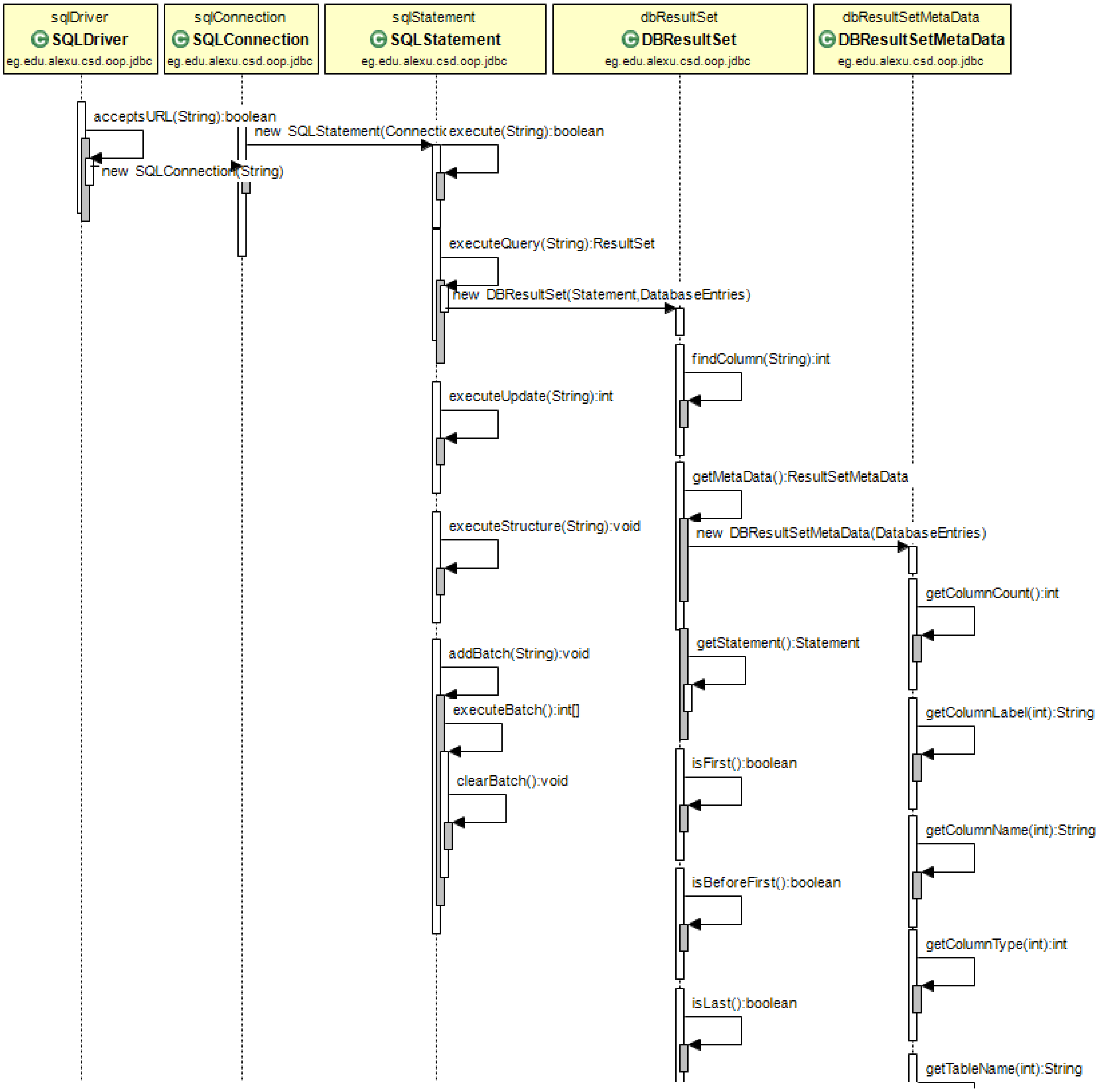
- UML Class Diagram
  Link : https://i.imgur.com/2oCpcWM.gif



<<Java Class>>
SQLDriver
eg.edu.alexu.csd.oop.jdbc
connectionManager: DriverManager
validurl: String
x: Properties
SQLDriver()
acceptsURL(String):boolean
connect(String,Properties):Connection
getPropertyInfo(String,Properties):DriverPropertyInfo[]
getMajorVersion():int
getMinorVersion():int
jdbcCompliant():boolean
getParentLogger():Logger
<<Java Class>>
AbstractConnection
eg.edu.alexu.csd.oop.jdbc.abstract_classes
AbstractConnection()
unwrap(Class<T>)
isWrapperFor(Class<?>):boolean
prepareStatement(String):PreparedStatement
prepareCall(String):CallableStatement
nativeSQL(String):String
setAutoCommit(boolean):void
getAutoCommit():boolean
commit():void
rollback():void
getMetaData():DatabaseMetaData
setReadOnly(boolean):void
isReadOnly():boolean
setCatalog(String):void
getCatalog():String
setTransactionIsolation(int):void
getTransactionIsolation():int
getWarnings():SQLWarning
clearWarnings():void
createStatement(int,int):Statement
prepareStatement(String,int,int):PreparedStatement
prepareCall(String,int,int):CallableStatement
getTypeMap():Map<String,Class<?>>
setTypeMap(Map<String,Class<?>>):void
setHoldability(int):void
getHoldability():int
setSavepoint():Savepoint
setSavepoint(String):Savepoint
rollback(Savepoint):void
releaseSavepoint(Savepoint):void
createStatement(int,int,int):Statement
prepareStatement(String,int,int,int):PreparedStatement
prepareCall(String,int,int,int):CallableStatement
prepareStatement(String,int):PreparedStatement
prepareStatement(String,int[]):PreparedStatement
prepareStatement(String,String[]):PreparedStatement
createClob():Clob
createBlob():Blob
createNClob():NClob
createSQLXML():SQLXML
isValid(int):boolean
setClientInfo(String,String):void
setClientInfo(Properties):void
getClientInfo(String):String
getClientInfo():Properties
createArrayOf(String,Object[]):Array
createStruct(String,Object[]):Struct
setSchema(String):void
getSchema():String
abort(Executor):void
setNetworkTimeout(Executor,int):void
getNetworkTimeout():int
<<Java Class>>
AbstractResultSetMetaData
eg.edu.alexu.csd.oop.jdbc.abstract_classes
AbstractResultSetMetaData()
isAutoIncrement(int):boolean
isCaseSensitive(int):boolean
isSearchable(int):boolean
isCurrency(int):boolean
isNullable(int):int
isSigned(int):boolean
getColumnDisplaySize(int):int
getSchemaName(int):String
getPrecision(int):int
getScale(int):int
getCatalogName(int):String
getColumnTypeName(int):String
isReadOnly(int):boolean
isWritable(int):boolean
isDefinitelyWritable(int):boolean
getColumnClassName(int):String
unwrap(Class<T>)
isWrapperFor(Class<?>):boolean
<<Java Class>>
DBResultSet
eg.edu.alexu.csd.oop.jdbc
entries: DatabaseEntries
currentCursorIndex: int
statement: Statement
resultSetMetaData: ResultSetMetaData
numberOfEntries: int
DBResultSet(Statement,DatabaseEntries)
getString(int):String
getString(String):String
getInt(int):int
getInt(String):int
getObject(int):Object
findColumn(String):int
beforeFirst():void
afterLast():void
first():boolean
last():boolean
absolute(int):boolean
isBeforeFirst():boolean
isAfterLast():boolean
isFirst():boolean
isLast():boolean
getStatement():Statement
isClosed():boolean
next():boolean
previous():boolean
close():void
getMetaData():ResultSetMetaData
getCurrentEntry():DatabaseRow
throwExceptionIfNullEntries(String):void
<<Java Class>>
SQLConnection
eg.edu.alexu.csd.oop.jdbc
databaseName: String
closed: boolean
SQLConnection(String)
createStatement():Statement
close():void
isClosed():boolean
getName(String):String
<<Java Class>>
DBResultSetMetaData
eg.edu.alexu.csd.oop.jdbc
tableName: String
tableColumnsTypes: List<DatabaseColumn>
DBResultSetMetaData(DatabaseEntries)
getColumnCount():int
getColumnLabel(int):String
getColumnName(int):String
getTableName(int):String
getColumnType(int):int
<<Java Class>>
AbstractStatement
eg.edu.alexu.csd.oop.jdbc.abstract_classes
AbstractStatement()
unwrap(Class<T>)
isWrapperFor(Class<?>):boolean
executeQuery(String):ResultSet
executeUpdate(String):int
close():void
getMaxFieldSize():int
setMaxFieldSize(int):void
getMaxRows():int
setMaxRows(int):void
setEscapeProcessing(boolean):void
getQueryTimeout():int
setQueryTimeout(int):void
cancel():void
getWarnings():SQLWarning
clearWarnings():void
setCursorName(String):void
execute(String):boolean
getResultSet():ResultSet
getUpdateCount():int
getMoreResults():boolean
setFetchDirection(int):void
getFetchDirection():int
setFetchSize(int):void
getFetchSize():int
getResultSetConcurrency():int
getResultSetType():int
addBatch(String):void
clearBatch():void
executeBatch():int[]
getConnection():Connection
getMoreResults(int):boolean
getGeneratedKeys():ResultSet
executeUpdate(String,int):int
executeUpdate(String,int[]):int
executeUpdate(String,String[]):int
execute(String,int):boolean
execute(String,int[]):boolean
execute(String,String[]):boolean
getResultSetHoldability():int
isClosed():boolean
setPoolable(boolean):void
isPoolable():boolean
closeOnCompletion():void
isCloseOnCompletion():boolean
~resultSet
0..1
~statement
0..1
<<Java Class>>
SQLStatement
eg.edu.alexu.csd.oop.jdbc
parser: SQLparser
db: DatabaseImplement
batch: List<String>
dbName: String
timeOut: int
connection: Connection
closed: boolean
SQLStatement(Connection,String)
executeQuery(String):ResultSet
executeUpdate(String):int
close():void
isClosed():boolean
getQueryTimeout():int
setQueryTimeout(int):void
addBatch(String):void
clearBatch():void
execute(String):boolean
executeStructure(String):void
executeBatch():int[]
getConnection():Connection
<<Java Class>>
AbstractResultSet
eg.edu.alexu.csd.oop.jdbc.abstract_classes
AbstractResultSet()
wasNull():boolean
getBoolean(int):boolean
getByte(int):byte
getShort(int):short
getLong(int):long
getFloat(int):float
getDouble(int):double
getBigDecimal(int,int):BigDecimal
getBytes(int):byte[]
getDate(int):Date
getTime(int):Time
getTimestamp(int):Timestamp
getAsciiStream(int):InputStream
getUnicodeStream(int):InputStream
getBinaryStream(int):InputStream
getBoolean(String):boolean
getByte(String):byte
getShort(String):short
getLong(String):long
getFloat(String):float
getDouble(String):double
getBigDecimal(String,int):BigDecimal
getBytes(String):byte[]
getDate(String):Date
getTime(String):Time
getTimestamp(String):Timestamp
getAsciiStream(String):InputStream
getUnicodeStream(String):InputStream
getBinaryStream(String):InputStream
getWarnings():SQLWarning
clearWarnings():void
getCursorName():String
getObject(String):Object
getCharacterStream(int):Reader
getCharacterStream(String):Reader
getBigDecimal(int):BigDecimal

- Sequence Diagram

  Link : https://i.imgur.com/PIL2TtJ.png


sqlDriver
SQLDriver
eg.edu.alexu.csd.oop.jdbc
sqlConnection
SQLConnection
eg.edu.alexu.csd.oop.jdbc
sqlStatement
SQLStatement
eg.edu.alexu.csd.oop.jdbc
dbResultSet
DBResultSet
eg.edu.alexu.csd.oop.jdbc
dbResultSetMetaData
DBResultSetMetaData
eg.edu.alexu.csd.oop.jdbc
acceptsURL(String):boolean
new SQLStatement(Connectic
execute(String):boolean
new SQLConnection(String)
executeQuery(String):ResultSet
new DBResultSet(Statement,DatabaseEntries)
findColumn(String):int
executeUpdate(String):int
getMetaData():ResultSetMetaData
executeStructure(String):void
new DBResultSetMetaData(DatabaseEntries)
getColumnCount():int
addBatch(String):void
getStatement():Statement
executeBatch():int[]
getColumnLabel(int):String
isFirst():boolean
clearBatch():void
getColumnName(int):String
isBeforeFirst():boolean
getColumnType(int):int
isLast():boolean
getTableName(int):String

5. User Interface:

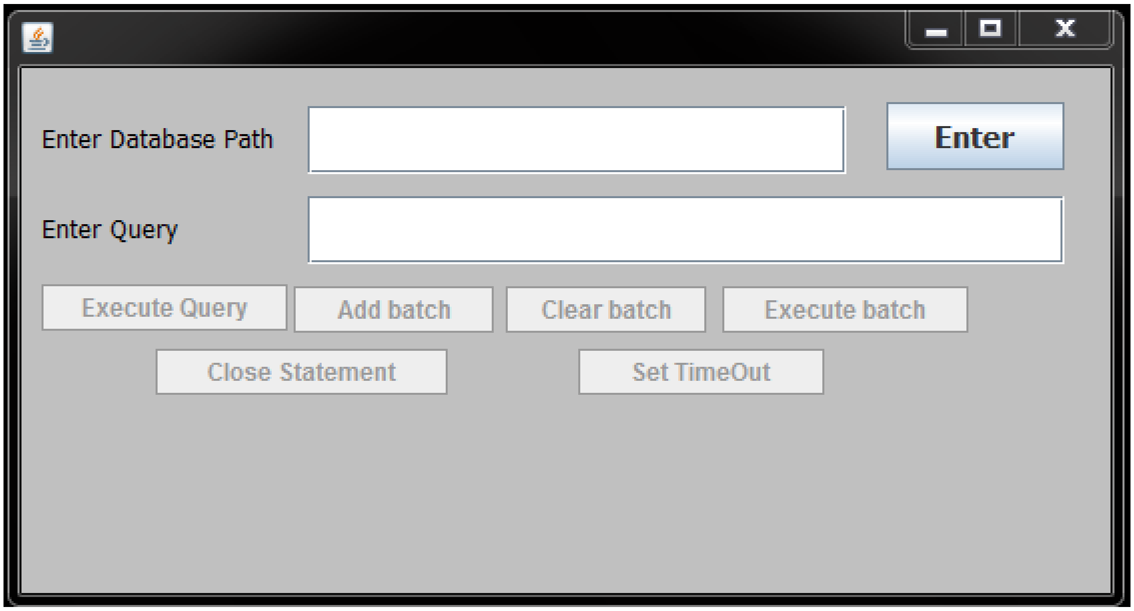
Enter your query in the command line specified area.

If query Syntax is incorrect an error appears notifying the user.

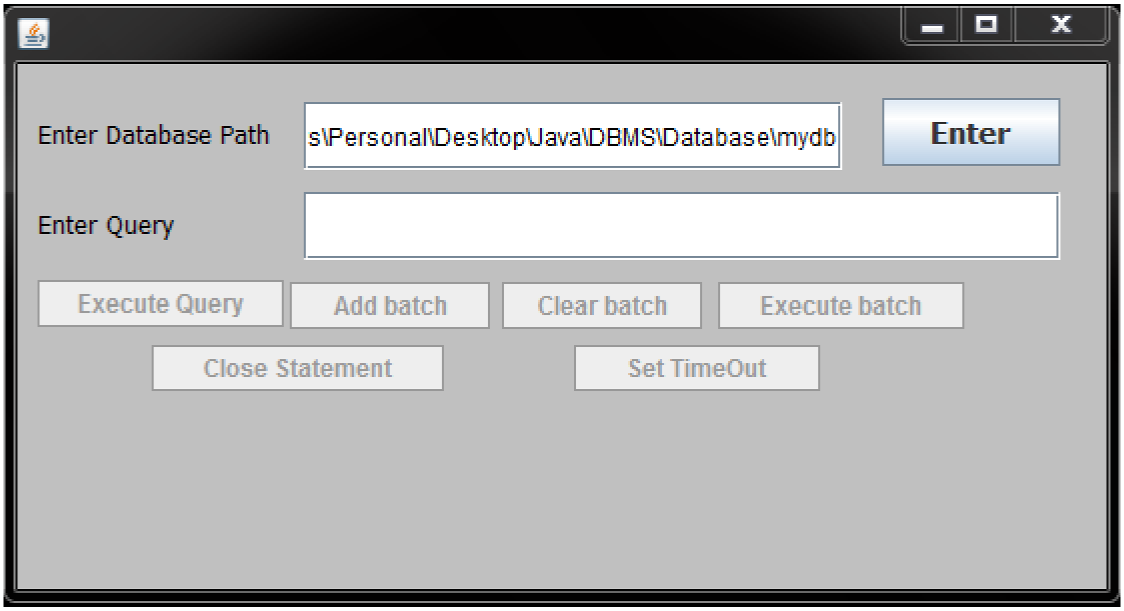
A database a table must be first created or opened to start writing queries.

For Every table two files will be created in it, one is an XML file that contains the table columns and elements and the other file is the DTD file that contains the basic information about the table with which could be used for table validation.

a) User first insert the database Path in ‘Enter Database Path’ section. As seen below all Buttons are disabled, until the user enters a valid path.

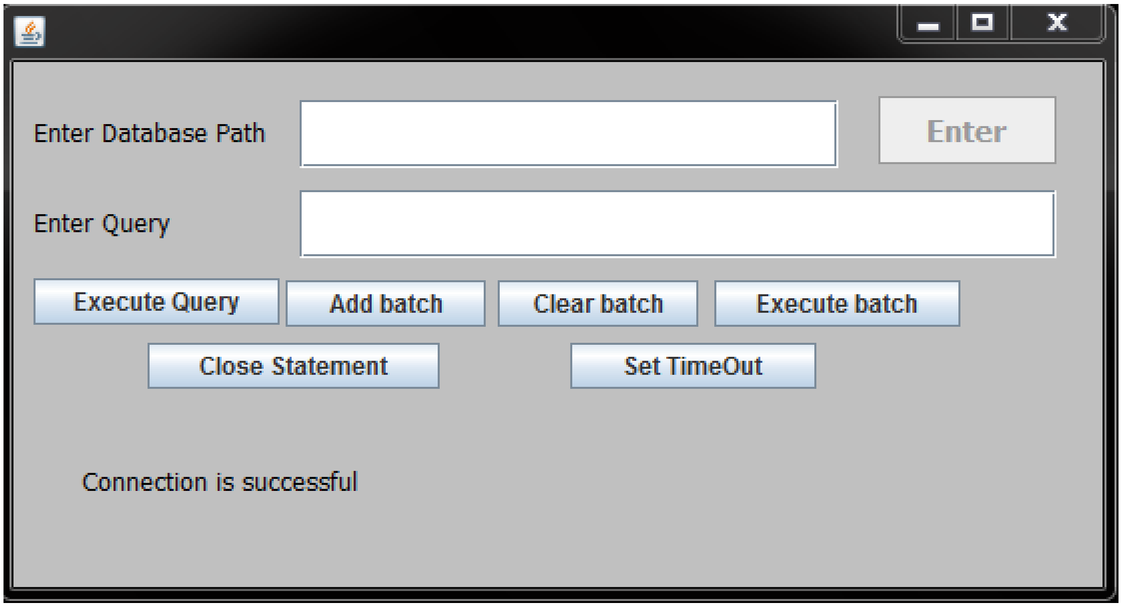


b) The path is entered as shown below.



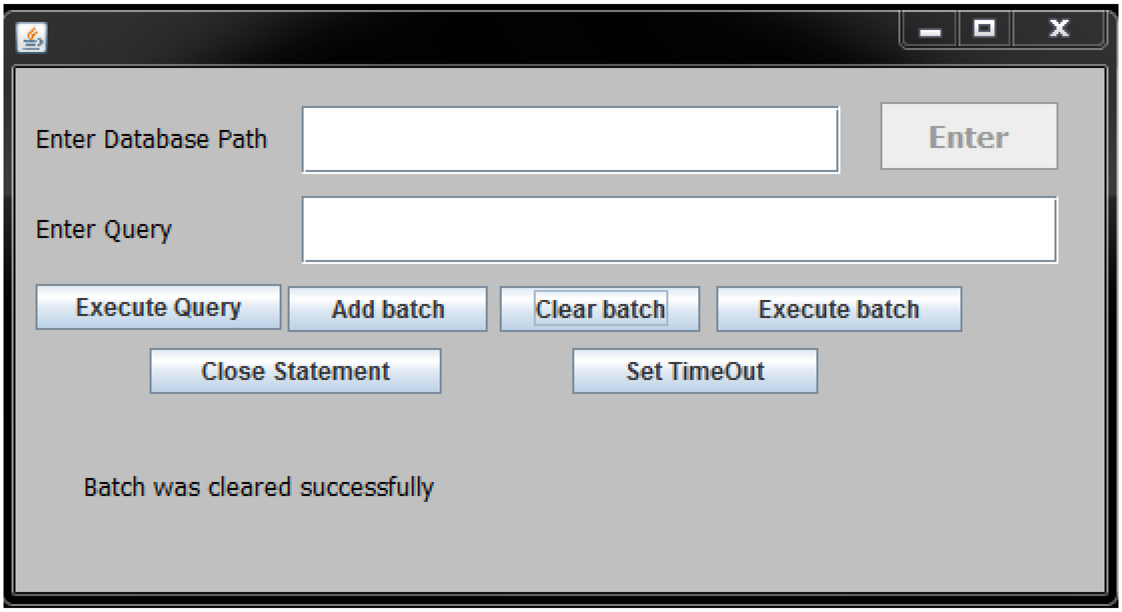
Enter Database Path: s\Personal\Desktop\Java\DBMS\Database\mydb

c) On pressing enter the Connection is established and a database is either used or created and Enter is disabled.



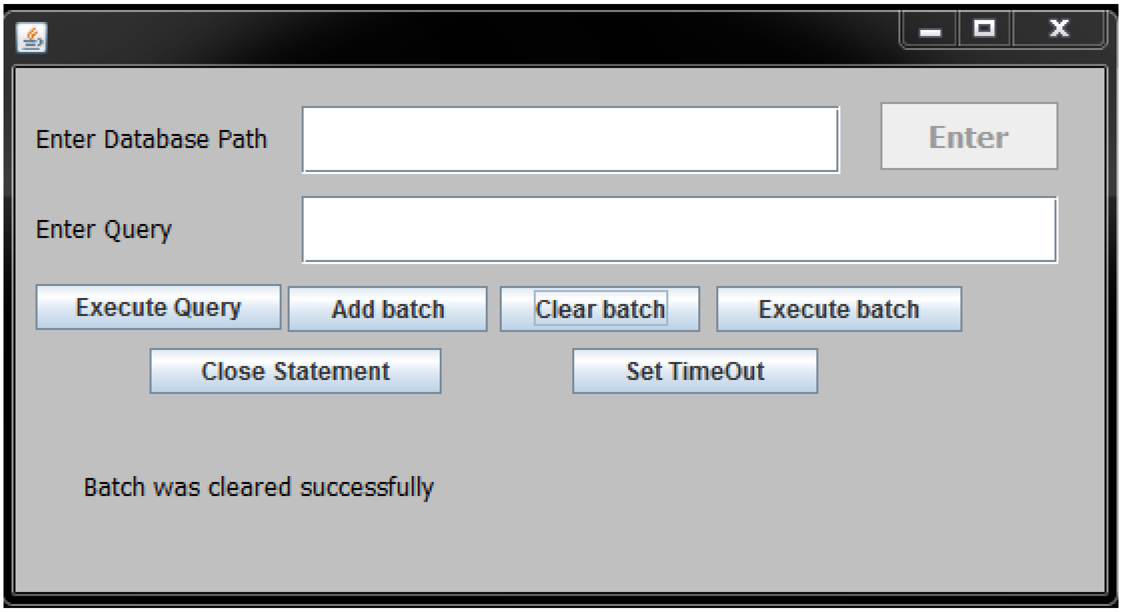
Connection is successful

d) The Text Area next to ‘Enter Query’ is where the user enters any query to be executes.



Batch was cleared successfully

e) The available features for the user are adding , clearing and executing a batch. Also setting Time out of queries in seconds.



Batch was cleared successfully