Project 2 Report (DBMS)

Programming 2

This Document contains a brief Description for the application and Design overview also a description for some important classes and methods, UML diagram for the application and some implementation for some methods, JDBC description, and a Fine User Manual for the usage of the application.

Application Description

A Computer Database is a structured collection of records or data that is stored in a computer system. This application is simply a system that handles the organization, storage, management, and retrieval of data in a database, It's accepts an SQL Query from the user and Execute this query using DBMS which stores the DATABASE in XML Format. It also handles any type of DATABASE using JDBC(Java DataBase connectivity).

Features:

This application supports the following features:-

- Create a DATABASE in the form of an XML-File.
- Insert Data into a table.
- Delete Data from a table.
- Select data from a table.
- Drop a table.
- Handles any External DATABASE (DBMS) by connecting to this DATABASE using (JDBC).
- Handles the Validation of each XML-Data file.
- Very easy to use.
- Command user interface.
- Parsing an XML Document using **STAX** Parser.

Description of some important functions:

- ***The Validation Factory(Interpreter):-***

This class contains some function that are responsible for the validation of the entered statements, also it's responsible for parsing the user statements and getting the required data from it, like "Table_Name , Table_Columns , Delete_Conditions , Select_Conditions" , It works like an interpreter.

- **Create Table (Parser Class DBMS):-**

boolean createTable(String tableName, String[] columnNames, String[] dataTypes)

This function is responsible for creating a new table in the form of an XML File, It also creates a XSD File with the XML file which is responsible for the Validation of the XML File, It takes a String Table_Name , array of String that contains Columns, array of Strings that contains DATA TYPE as a Parameters, the created files are "Table_Name.xml" and "Table_Name.xsd".
The function uses Stax Parser to write the XMl file, XMLStreamWriter class that writes event like (Start Element, End Element, Character).

- **Validate(Parser Class DBMS):-**

boolean validate(File xml, File xsd)

This function takes two Files as a parameters, the first one is the XML File the second one is the XSD file, It's validates the XML Files according to the XSD File.

- **GetLocation(Parser Class DBMS):-**

Multimap GetLocation(String table_name, String[] condition)

This function takes as a Parameters 'String Table_name' 'Array of String that contains a certain condition', this function searches the Table for this condition, and determine the Rows/Lines that match the condition, It returns a MultiMap that contains Row and columns number which match the condition.

- ### ValidateStructureQuery(Interpreter Class DBMS):-

boolean ValidateStructureQuery(String query)

It takes as a Parameter String Query which was entered by the user, it validate this query using Regex, It compares the query with a Certain Pattern and then check if this query matches the Pattern or not.

Design Overview:-

- ### The DBMS:-

Consists of 4 main classes and DBMS Interface (Interpter.java,DBMS.java,Parser.java,XSDParser.java).

- **Interpreter.java**: It deals with the REGEX and Queries entered by the user, Validate them and extract needed Data, It has a connection the Parser Class, as it send the needed Data from the Queries.

- **Parser.java**: It deals with the XML Files, Create-Drop-Delete-Insert-Select-Drop , It accept Data required from the interpreter class, also uses the XSDParser class to validate the XML file.

- **XSDParser.java**: Handles the Validation of the XML file according to it's XSD File, it accepts the FILE from the Parser class then Sends the confirmation to the parser class.

- **DBMS.java**: It implements the DBMS Interface, it's considered as the main class of the application.

- ### The JDBC:-

5 main Classes (Connection.java,Driver.java,ResultSet.java,ResultsetMetadata.java,Statements.java)

- **Connection.java**: Connection Class contains a getStatement() method and close() one .

- **Driver.java:** Driver class which implements " Java.sql.Driver " has a single method to be modified which was the "connect" , there we cancelled the properties part and set it with "null" as it's useless in current Application , but the "URL" part was set to take a Path , the path of your XML files.

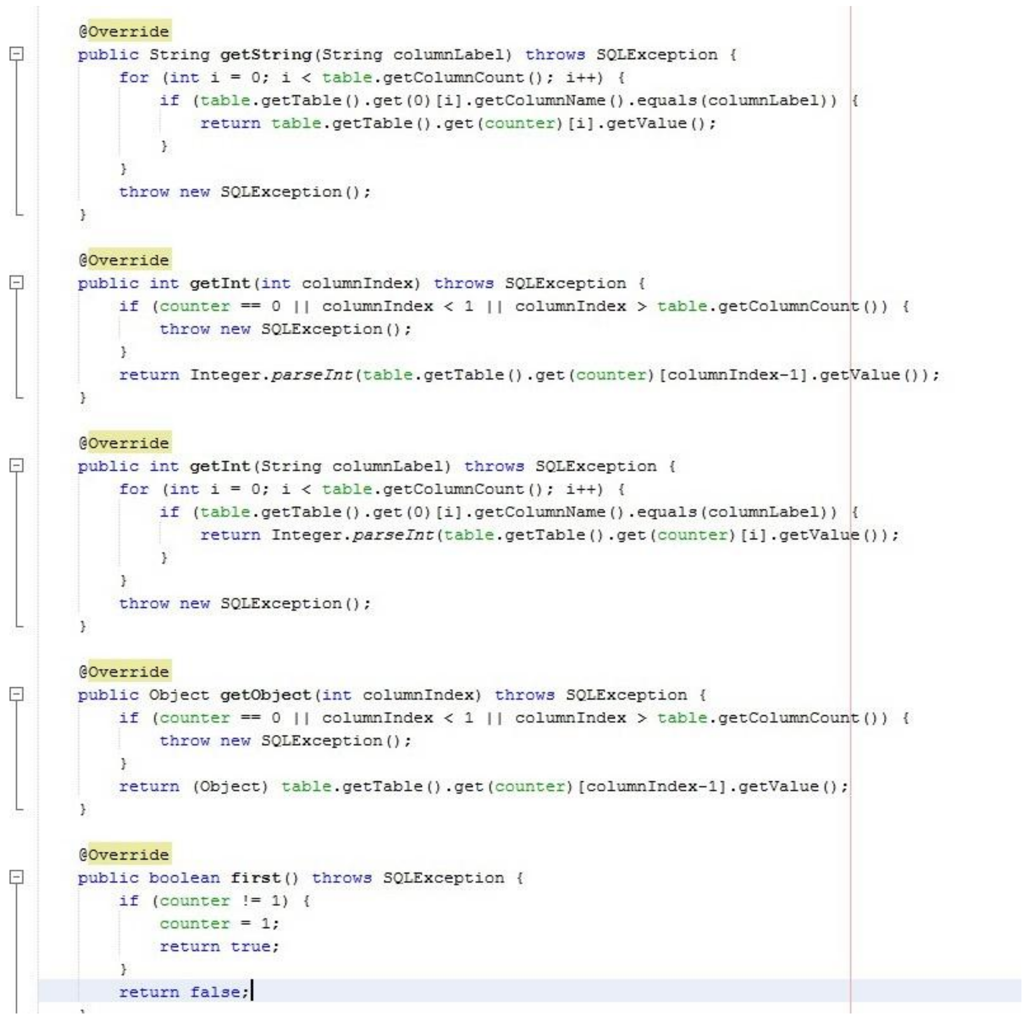
- **Resultset.java:** The one who controls where the cursor is, and returns what is in the cell whatever it was "Varchar" or "Int" We overloaded the constructor to take a table object of our Table class from the "Parser" class in DBMS and a Statement object.

- **ResultSetMetaDAta.java:** The class that responsible of gathering Basic data about the table like tableName , columnCount , columnName and Label which are in this DBMS return the same as the 'AS' operator is not supported .

- **Statements.java**: Considered as the main class in the JDBC, as it's responsible for the execution methods.

Some design patterns are used in this application like:-

1- Singleton which is used in all classes.



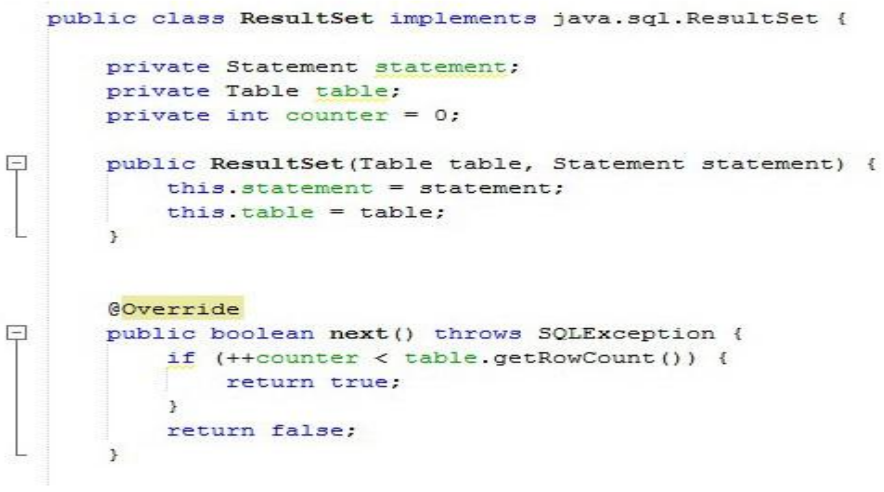
```java
@Override
public String getString(String columnLabel) throws SQLException {
    for (int i = 0; i < table.getColumnCount(); i++) {
        if (table.getTable().get(0)[i].getColumnName().equals(columnLabel)) {
            return table.getTable().get(counter)[i].getValue();
        }
    }
    throw new SQLException();
}

@Override
public int getInt(int columnIndex) throws SQLException {
    if (counter == 0 || columnIndex < 1 || columnIndex > table.getColumnCount()) {
        throw new SQLException();
    }
    return Integer.parseInt(table.getTable().get(counter)[columnIndex-1].getValue());
}

@Override
public int getInt(String columnLabel) throws SQLException {
    for (int i = 0; i < table.getColumnCount(); i++) {
        if (table.getTable().get(0)[i].getColumnName().equals(columnLabel)) {
            return Integer.parseInt(table.getTable().get(counter)[i].getValue());
        }
    }
    throw new SQLException();
}

@Override
public Object getObject(int columnIndex) throws SQLException {
    if (counter == 0 || columnIndex < 1 || columnIndex > table.getColumnCount()) {
        throw new SQLException();
    }
    return (Object) table.getTable().get(counter)[columnIndex-1].getValue();
}

@Override
public boolean first() throws SQLException {
    if (counter != 1) {
        counter = 1;
        return true;
    }
    return false;
```



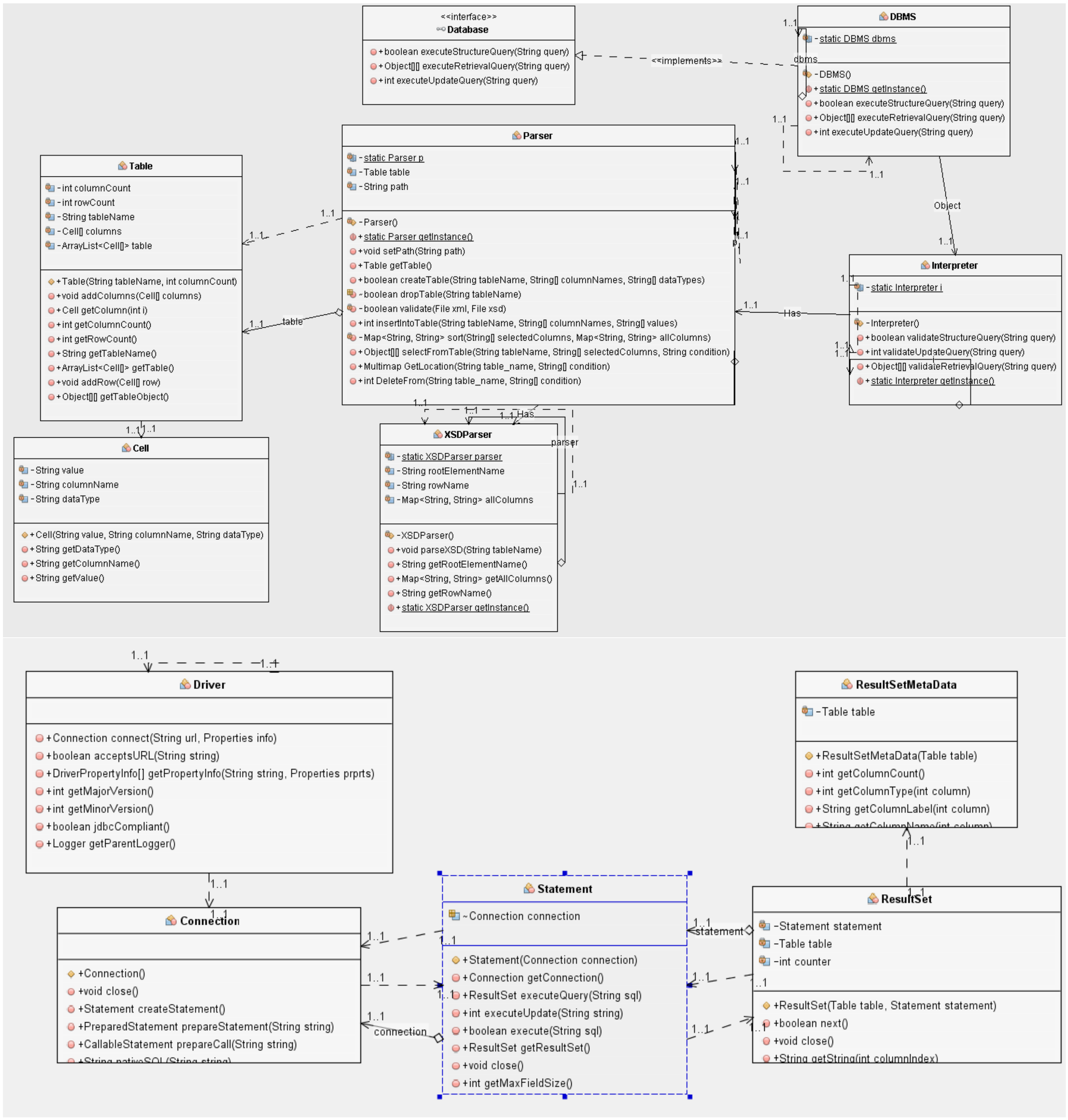
```java
public class ResultSet implements java.sql.ResultSet {

    private Statement statement;
    private Table table;
    private int counter = 0;

    public ResultSet(Table table, Statement statement) {
        this.statement = statement;
        this.table = table;
    }


    @Override
    public boolean next() throws SQLException {
        if (++counter < table.getRowCount()) {
            return true;
        }
        return false;
    }
```

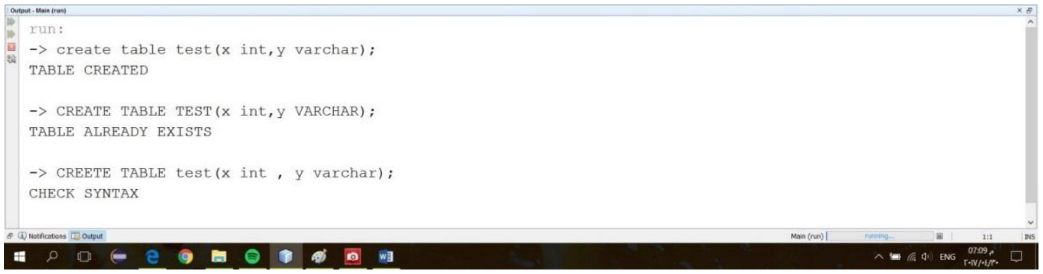
❖ UML Diagram for the Design:-



<<interface>>
Database
+boolean executeStructureQuery(String query)
+Object[][] executeRetrievalQuery(String query)
+int executeUpdateQuery(String query)
<<implements>>
DBMS
-static DBMS dbms
dbms
-DBMS()
+static DBMS getInstance()
+boolean executeStructureQuery(String query)
+Object[][] executeRetrievalQuery(String query)
+int executeUpdateQuery(String query)
Object
Parser
-static Parser p
-Table table
-String path
-Parser()
+static Parser getInstance()
+void setPath(String path)
+Table getTable()
+boolean createTable(String tableName, String[] columnNames, String[] dataTypes)
-boolean dropTable(String tableName)
-boolean validate(File xml, File xsd)
+int insertIntoTable(String tableName, String[] columnNames, String[] values)
-Map<String, String> sort(String[] selectedColumns, Map<String, String> allColumns)
+Object[][] selectFromTable(String tableName, String[] selectedColumns, String condition)
+Multimap GetLocation(String table_name, String[] condition)
+int DeleteFrom(String table_name, String[] condition)
Table
-int columnCount
-int rowCount
-String tableName
-Cell[] columns
-ArrayList<Cell[]> table
+Table(String tableName, int columnCount)
+void addColumns(Cell[] columns)
+Cell getColumn(int i)
+int getColumnCount()
+int getRowCount()
+String getTableName()
+ArrayList<Cell[]> getTable()
+void addRow(Cell[] row)
+Object[][] getTableObject()
table
Interpreter
-static Interpreter i
-Interpreter()
+boolean validateStructureQuery(String query)
+int validateUpdateQuery(String query)
+Object[][] validateRetrievalQuery(String query)
+static Interpreter getInstance()
Has
Cell
-String value
-String columnName
-String dataType
+Cell(String value, String columnName, String dataType)
+String getDataType()
+String getColumnName()
+String getValue()
XSDParser
-static XSDParser parser
-String rootElementName
-String rowName
-Map<String, String> allColumns
-XSDParser()
+void parseXSD(String tableName)
+String getRootElementName()
+Map<String, String> getAllColumns()
+String getRowName()
+static XSDParser getInstance()
parser
Has
1..1
Driver
+Connection connect(String url, Properties info)
+boolean acceptsURL(String string)
+DriverPropertyInfo[] getPropertyInfo(String string, Properties prprts)
+int getMajorVersion()
+int getMinorVersion()
+boolean jdbcCompliant()
+Logger getParentLogger()
ResultSetMetaData
-Table table
+ResultSetMetaData(Table table)
+int getColumnCount()
+int getColumnType(int column)
+String getColumnLabel(int column)
+String getColumnName(int column)
Connection
+Connection()
+void close()
+Statement createStatement()
+PreparedStatement prepareStatement(String string)
+CallableStatement prepareCall(String string)
+String nativeSQL(String string)
connection
Statement
~Connection connection
+Statement(Connection connection)
+Connection getConnection()
+ResultSet executeQuery(String sql)
+int executeUpdate(String string)
+boolean execute(String sql)
+ResultSet getResultSet()
+void close()
+int getMaxFieldSize()
statement
ResultSet
-Statement statement
-Table table
-int counter
+ResultSet(Table table, Statement statement)
+boolean next()
+void close()
+String getString(int columnIndex)

➢ UserManual:-

CREATE AND DROP :

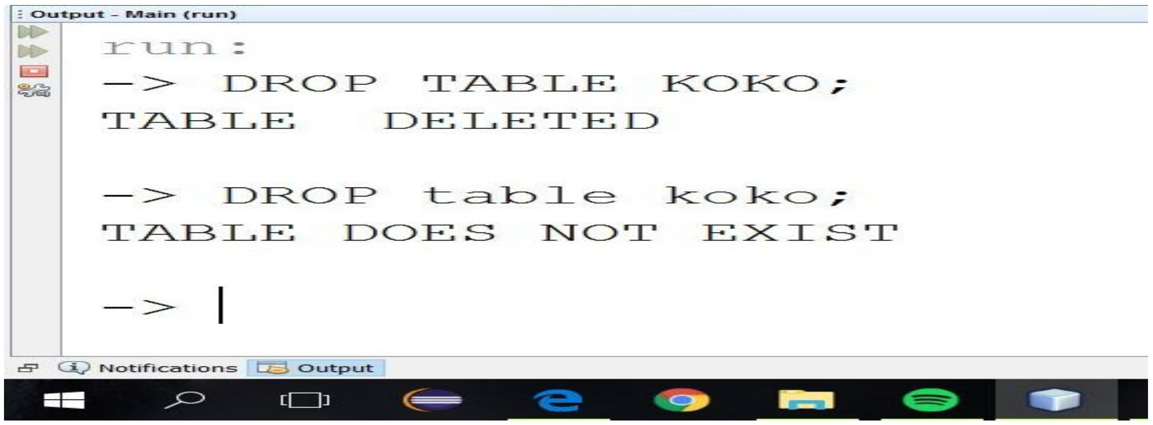


```
run:
-> create table test(x int,y varchar);
TABLE CREATED

-> CREATE TABLE TEST(x int,y VARCHAR);
TABLE ALREADY EXISTS

-> CREETE TABLE test(x int , y varchar);
CHECK SYNTAX
```

- You're Free to type the create command with upper or lower case as you see. Whole DMBS is case ***INSENSITIVE*** as required in pdf.
- If table exists , it's not created and a feedback is given that the table already exists.
- If the command has a Syntax error , a message is shown.



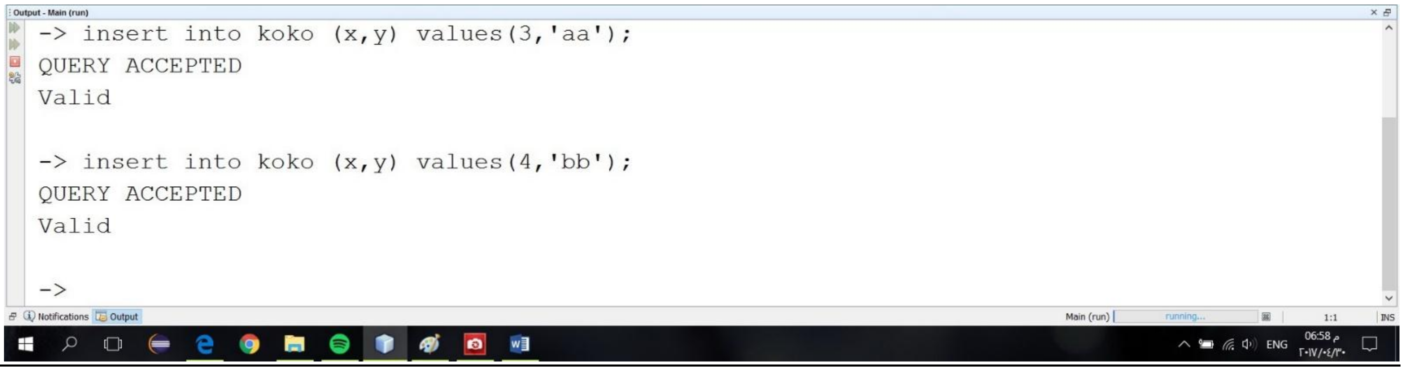
```
run:
-> DROP TABLE KOKO;
TABLE DELETED

-> DROP table koko;
TABLE DOES NOT EXIST

-> |
```

- The Drop Opeation is close to the Create one as you see in example.
- It also gives feedback if table if table deleted before or not created at all.
- If wrong columns entered m Warning message is shown and operation Is cancelled.

INSERT:

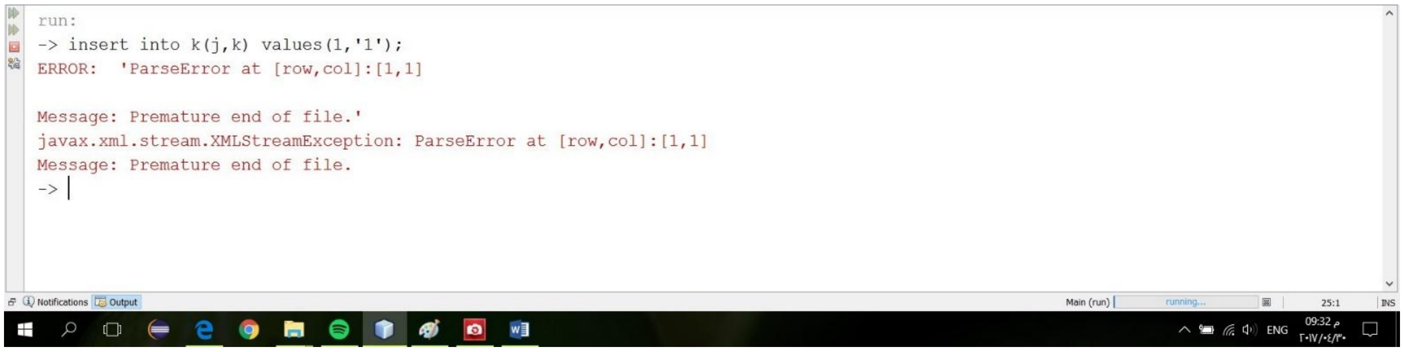


```
-> insert into koko (x,y) values(3,'aa');
QUERY ACCEPTED
Valid

-> insert into koko (x,y) values(4,'bb');
QUERY ACCEPTED
Valid

->
```

- The Syntax is as shown and it's also case insensitive.
- String is given between two single quotes (' ').
- if wrong columns are entered a warning message appears and operation is cancelled.

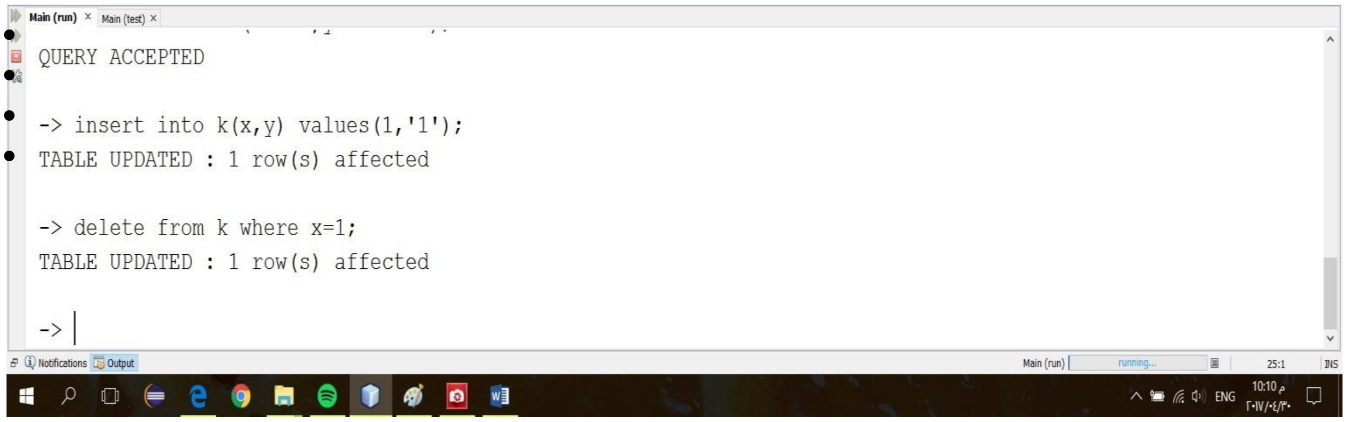


```
run:
-> insert into k(j,k) values(1,'1');
ERROR:  'ParseError at [row,col]:[1,1]

Message: Premature end of file.'
javax.xml.stream.XMLStreamException: ParseError at [row,col]:[1,1]
Message: Premature end of file.
->
```

DELETE:

- **The Delete syntax is as shown.**
- **As well as the syntax is correct , no check Syntax messages will appear.**



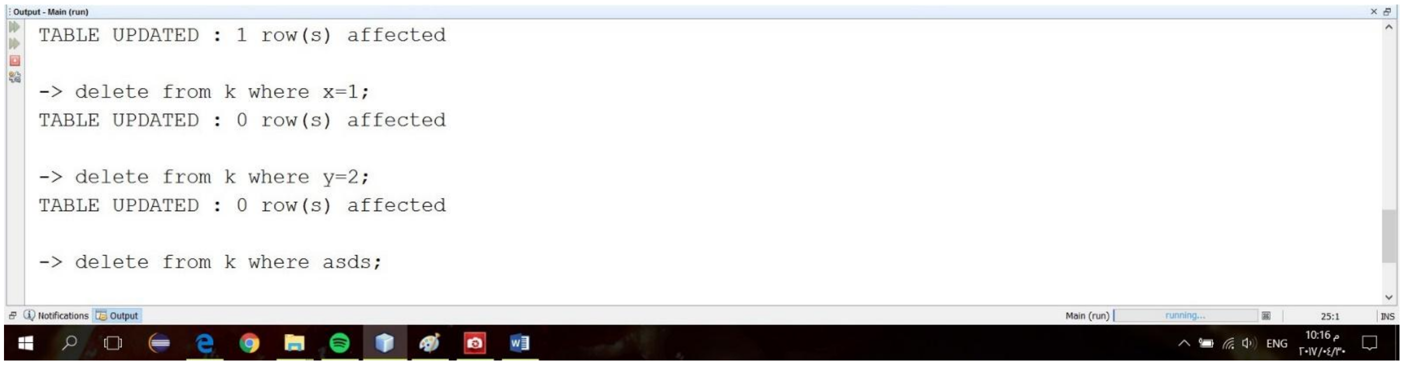
```
QUERY ACCEPTED

-> insert into k(x,y) values(1,'1');
TABLE UPDATED : 1 row(s) affected

-> delete from k where x=1;
TABLE UPDATED : 1 row(s) affected

->
```

- **If table is empty and syntax is still valid, it's accepted but you will see that 0 row(s) affected as follows.**
- **If you delete from wrong table the following message appears**.

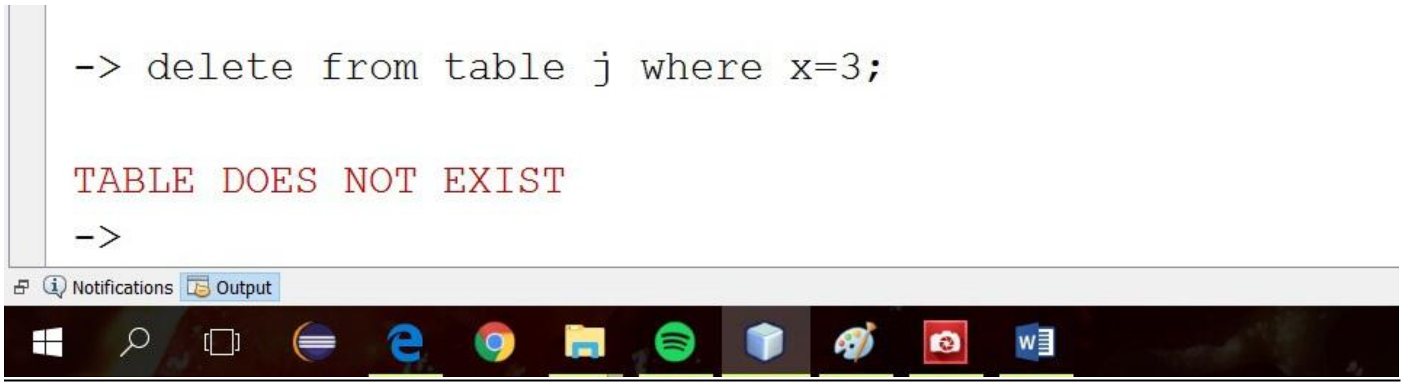


```
TABLE UPDATED : 1 row(s) affected

-> delete from k where x=1;
TABLE UPDATED : 0 row(s) affected

-> delete from k where y=2;
TABLE UPDATED : 0 row(s) affected

-> delete from k where asds;
```

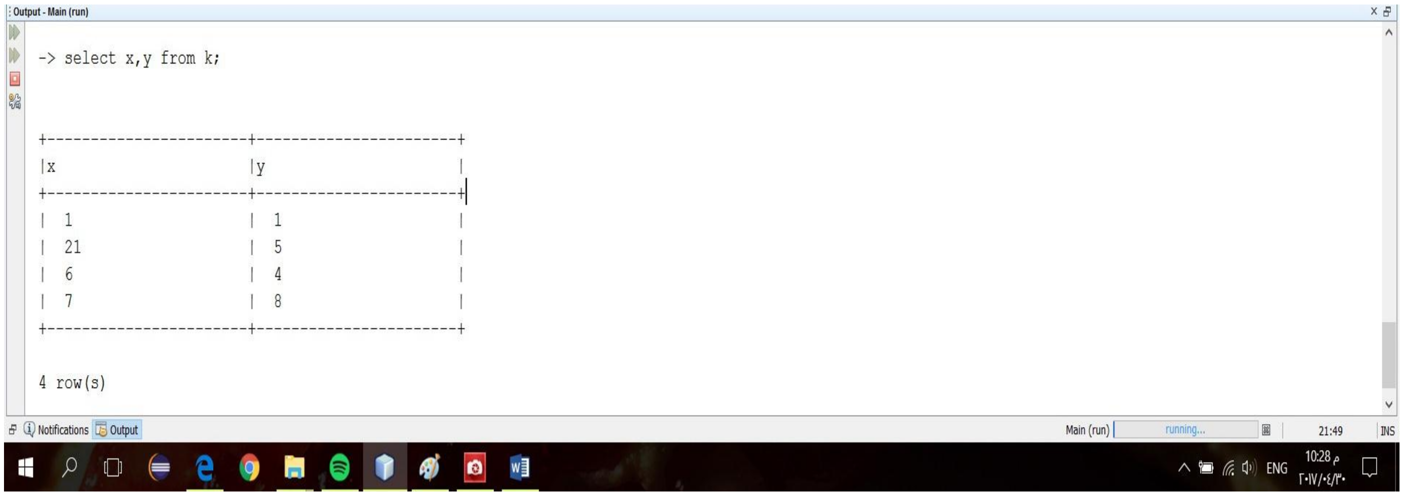


```
-> delete from table j where x=3;

TABLE DOES NOT EXIST
->
```

SELECT:

- **Select Syntax is as follows.**
- **A table is shown with given parameters carrying all data in table.**

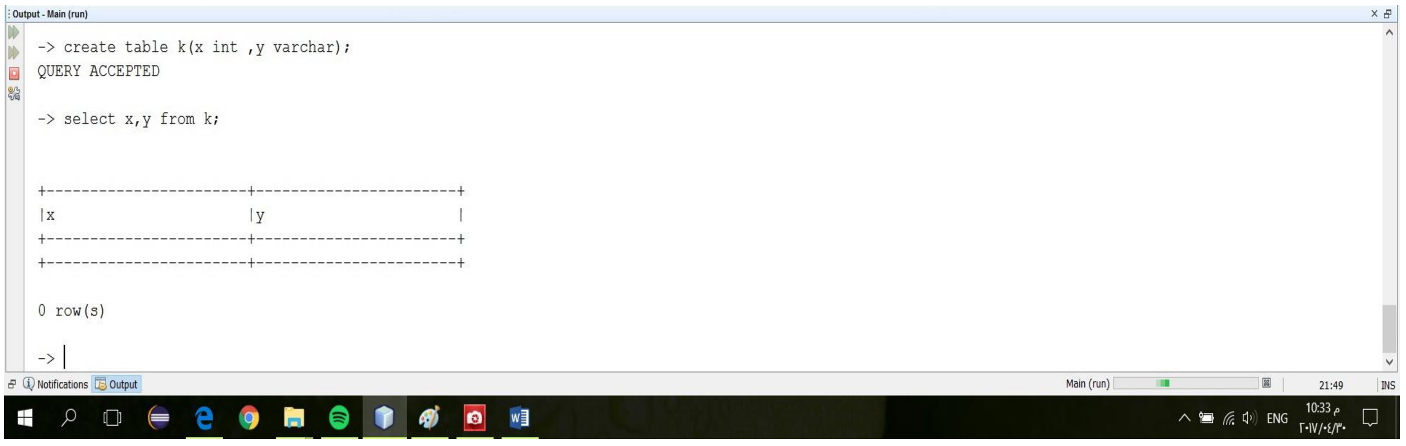


```
-> select x,y from k;


+-----------------------+-----------------------+
|x                      |y                      |
+-----------------------+-----------------------+
|  1                    |  1                    |
|  21                   |  5                    |
|  6                    |  4                    |
|  7                    |  8                    |
+-----------------------+-----------------------+

4 row(s)
```

- **You can select even one column.**
- **Syntax Err message is shown if Syntax is wrong.**
- **If the table is empty , same table is drawn but with empty data in each column.**



```
-> create table k(x int ,y varchar);
QUERY ACCEPTED

-> select x,y from k;


+-----------------------+-----------------------+
|x                      |y                      |
+-----------------------+-----------------------+
+-----------------------+-----------------------+

0 row(s)

->
```

Exit:-

<u>All you need is to Type ‘Exit’ in the Command Window.</u>