CS1003 W07 Practical 170025298 14 Mar 2018

Overview:

This essay design for get data use jdbc from csv file and use query to get specific data from database. This program contains 3 classes, which are: JDBC setting (read and get data from csv file, set the jdbc and read and set data use prepared statement), List_of_csv class(store data which is setted in JDBCsetting class), and main class (W07 practical class with only a few lines of command).

Design:

1. W07Practical (main class)

```
import java.io.IOException;
import java.sql.SQLException;

public class W07Practical {
    public static void main(String[] args) throws SQLException, IOException{
        if(args.length == 3){
            JDBCsetting csetting = new JDBCsetting(args[0],args[1],args[2]);
        }
        else {
            if(args.length == 2){
                JDBCsetting csetting = new JDBCsetting(args[0], args[1]);
            }
            else {
                System.out.println("Usage: java -cp sqlite-jdbc.jar:. W07Practical <db_file> <action> [input_file]");
                System.exit( status: -1);
            }
        }
    }
}
```

This class is used to get data using args 1-3 and use if to prevent indexOutOfBounds exception. If there are two argument, there will be two arguments being inputted into the constructor and while the length of argument is 3, there will be 3 argument being inputted into another constructor in JDBC setting class.

2. JDBCsetting

There are several method available in JDBCsetting class:

Constructor:

There are two constructors available in the program: The first one has two parameters which is used for input two arguments in W07 practical class, another constructor has three parameters which is used for input three arguments in W07Practical class. I put this.variable = parameter inside of the constructor and then use the dBconnection() to access to next method.

```java
/**
 * Consturctor for query 1--4
 * Prevent the array index out of boundary
 * @param DBpath
 * @param action
 * @throws SQLException
 */
public JDBCsetting(String DBpath, String action) throws SQLException, IOException{
    this.DBpath = DBpath;
    this.action = action;
    dBconnection();
}

/**
 * Constructor is used to find the path of database, define action and read the CSV file.
 * Only available for creat
 * @param DBpath
 * @param action
 * @param readFile
 */
public JDBCsetting(String DBpath, String action,String readFile) throws SQLException, IOException{
    this.DBpath = DBpath;
    this.action = action;
    this.readFile = readFile;
    dBconnection();
}
```

dBconnection() - void:

This method is used to connect to the database which is shown as url in DriverManager.getConnection() via the database driver. If the process finished, action will be took and direct to the action method. If not, error message will be shown to alert where is wrong. After the action, the connection will close.

```java
/**
 * dbConnection is used to connect to Database.
 * @throws SQLException
 */
private void dBconnection() throws SQLException, IOException{
    Connection connection = null;
    try{
        //connect to database manage system
        connection = DriverManager.getConnection( url: "jdbc:sqlite:" + DBpath);
        action(connection);
    }
    catch (SQLException e){
        System.out.println(e.getMessage());
    }
    finally {
        if(connection != null) connection.close();
    }
}
```

action(Connection connection) - void:

This method use the switch command to judge the second argument that which action will be took. If something was wrong with the sql and the getMessage() will return the wrong message.

Four query and one create action are defined inside of the action() and each action refer to one method. An default has also been definded which will return the usage while input the wrong action.

```java
private void action(Connection connection) throws SQLException, IOException {
    try {
        switch (action) {
            case "create": {
                creatTable(connection);
                System.out.println("OK");
                break;
            }
            case "query1": {
                printAllData(connection);
                break;
            }
            case "query2": {
                totalNumberOfTitanic(connection);
                break;
            }
            case "query3": {
                query3(connection);
                break;
            }
            case "query4": {
                query4(connection);
                break;
            }
            default: {
                System.out.println("Usage: java -cp sqlite-jdbc.jar:. W07Practical <db_file> <action> [input_file]");
                break;
            }
        }
    }
    catch (SQLException e) {
        System.out.println("SQL error: " + e.getMessage());

    }
}
```

External file changes sync may be slow

createTable(Connection connection) throws IOException, SQLException– void

Create table method refers to the first action – create. This method will drop a table while the table with same dbpath name exist in the database file and then create the table with several tables with several types of variables.

After create the table, I created one List<List_of_CSV> and make it equal to the return value from read file, which is the method returns a list of data read from .csv file. Then I use for loop to inset data via insertData() method.

```java
/**
 * Creat table is to creat the table in create command
 * If the table exist , program will drop the table and then create a new one
 * @param connection
 * @throws SQLException
 */
private void createTable(Connection connection) throws SQLException, IOException{
    Statement statement = connection.createStatement();

    //Create a table includes name, sex, tickets ......

    statement.executeUpdate( s: "DROP TABLE IF EXISTS person");
    statement.executeUpdate( s: "CREATE TABLE person (passengerID INT, survived INT, pClass INT, name VARCHAR(100), "
            "sex VARCHAR(100), age FLOAT, sibSp INT, parch INT, ticket VARCHAR (100), fare FLOAT, cabin VARCHAR(100),
            "embarked VARCHAR(100))");

    //read the CSV file and INSERT all data into DB
    List<List_of_CSV> list_of_csv = readFile();
      System.out.println(list_of_csv);
    for(List_of_CSV s : list_of_csv) {
        insertData(s.getPassengerId(),s.getSurvived(),s.getpClass(),s.getName(),s.getSex(),s.getAge(),s.getSibSp(),
                s.getParch(),s.getTicket(),s.getFare(), s.getCabin(),s.getEmbarked(),connection);
         System.out.println(s.getAge());
    }

}
```

InsertData(parameters of value) throws SQLException – void

This method uses prepared statement to insert the data. Because the age can be empty, I use setNull() while the age is 0 (which is convert into 0 in List_of_CSV class). While all elements are defined, the statement update will be save and will close the statement.

```
private void insertData(int passengerID,int survived, int pClass, String name, String sex, Double age,
                        int sibSp, int parch, String ticket, double fare, String cabin, String embarked,
                       Connection connection) throws SQLException {
    PreparedStatement statement = connection.prepareStatement( s: "INSERT INTO person VALUES (?, ?, ?, ?, ?, ?, ?, ?,

    statement.setInt( i: 1, passengerID);
    statement.setInt( i: 2, survived);
    statement.setInt( i: 3, pClass);
    statement.setString( i: 4, name);
    statement.setString( i: 5, sex);
    if(age == 0){
        statement.setNull( i: 6, Types.FLOAT);
          System.out.println(age);
    }
    else {
        statement.setDouble( i: 6, age);
    }
    statement.setInt( i: 7, sibSp);
    statement.setInt( i: 8, parch);
    statement.setString( i: 9, ticket);
    statement.setDouble( i: 10, fare);
    statement.setString( i: 11, cabin);
    statement.setString( i: 12, embarked);
    statement.executeUpdate();
    statement.close();
}
```

ReadFile() throws IOException – List<List_of_CSV>

This method uses buffered reader to read the data from CSV file and use line.split to split the data as an array named elements and save it into the ArrayList which is named as by for elements.

While all the data is saved into the list, this list will be added into the list_of_CSV file. If something is wrong with finding .csv or some other error messages, the data will return the related error messages and finallt the reader will be closed.

```
private List<List_of_CSV> readFile() throws IOException{
    BufferedReader reader = null;
    List<List_of_CSV> list_of_csv = new ArrayList<>();

    try {
        reader = new BufferedReader(new FileReader(readFile));
        String title = reader.readLine();

        String line = "";
        while((line = reader.readLine()) != null){
            ArrayList<String> fields = new ArrayList();
            String[] elements = line.split( regex: ",");

            for(String s : elements){
                fields.add(s);
            }

            list_of_csv.add(new List_of_CSV(fields));
        }
          System.out.println(1);
    }
    catch (FileNotFoundException e){
        System.out.println("File not found: " + e.getMessage());
    }
    catch (IOException e){
        System.out.println("IO Exception: " + e.getMessage());
    }
    finally {
        reader.close();
    }
    return list_of_csv;
}
```

External file changes sync n

TotalNumberOfTitanic(Connection connection) throws SQLException– void

This method is used to calculate the total number of survivors available in titanic and use System.out to print the data which refers to the action “query2” . While is used to calculate to total number

```
/**
 *This method is for query 2 and the total number of Titanic will be calculated by using while command
 * query2: print out the total number of survivors of the Titanic
 * @param connection
 * @throws SQLException
 */
private void totalNumberOfTitanic(Connection connection) throws SQLException{
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery( s: "SELECT survived FROM person");

    int survived = 0;
    while (resultSet.next()){
        survived += resultSet.getInt( s: "survived");
    }
    System.out.println("Number of Survivors");
    System.out.println(survived);
}
```

printAllData(Connection connection) throws SQLException – void

This method is used for print all data from the database, which refers to the action “query1” in action() method. This method use Resultset to select all data from all database and println as the format. Because age has the default number of double which is 0, while the age is 0 and null will be replace the 0 as null, otherwise the system will print it normally.

```
/**
 * This method is used to print all data
 * All the data will be shown by System.out.println
 * list all the records in the database
 * @param connection
 * @throws SQLException
 */
private void printAllData(Connection connection) throws SQLException{
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery( s: "SELECT * FROM person");
    String title = "passengerId, survived, pClass, name, sex, age, sibSp, parch, ticket, fare, cabin, embarked";

    //System.out.println(resultSet.getInt("pClass"));
    System.out.println(title);
    while (resultSet.next()){
        int pClass = resultSet.getInt( s: "pClass");
        int survived = resultSet.getInt( s: "survived");
        int passengerID = resultSet.getInt( s: "passengerId");
        String name = resultSet.getString( s: "name");
        String sex = resultSet.getString( s: "sex");
        String ticket = resultSet.getString( s: "ticket");
        String cabin = resultSet.getString( s: "cabin");
        String embarked = resultSet.getString( s: "embarked");
        Double age = resultSet.getDouble( s: "age");
        double fare = resultSet.getDouble( s: "fare");
        int sibSp = resultSet.getInt( s: "sibSp");
        int parch = resultSet.getInt( s: "parch");

        System.out.println(passengerID + ", " + survived + ", " + pClass + ", " + name + ", " +
                sex + ", " + (age == 0.0 ? "null" : age) + ", " + sibSp + ", " + parch + ", " + ticket + ", " + fare
                + embarked);
    }
}
```

External file changes sync may be slow

Project files cannot be watched (are they under network mount?)

query3(Connection connection) throws SQLException – void

This method refers to query3 in action method and will print count as the category of pClass and survivals by using the database command line:

```
SELECT pClass, survived, COUNT (pClass||survived) as counts
FROM person GROUP by pClass, survived
```

Then use System.out.println() to print the data as the format

```java
/**
 * Some issues occurred in this method<- seems solved wed 16.04
 * Group By and COUNT to print the data
 * query3: list pClass, survived, and a count of passengers in each class and survival category
 * @param connection
 * @throws SQLException
 */
private void query3(Connection connection) throws SQLException{
    //Issue 1 <- 20.59 Sun
    Statement statement = connection.createStatement();

    String query = "SELECT pClass, survived, COUNT (pClass||survived) as counts FROM person GROUP by pClass, survived";

    ResultSet resultSet;
    resultSet = statement.executeQuery(query);
    System.out.println("pClass, survived, count");

    while (resultSet.next()){

        int counts = resultSet.getInt( s: "counts");
        int pClass = resultSet.getInt( s: "pClass");
        int survived = resultSet.getInt( s: "survived");

        System.out.println(pClass + ", " + survived + ", " + counts);
    }
}
```

query4(Connection connection) throws SQLException – void

This method is similar to query3 and I use the sql command line to get the data directly:

`SELECT sex, survived, MIN(age) as min_age FROM person GROUP BY sex , survived`

Then I use while() to print all the data line by line.

```java
/**
 * list sex, survived, and minimum age of passengers in each sex and survival category
 * Issue3 happens in Tue 23:40<-
 * @throws SQLException
 */
private void query4(Connection connection) throws SQLException {
    Statement statement = connection.createStatement();
    String query = "SELECT sex, survived, MIN(age) as min_age FROM person GROUP BY sex , survived";
    ResultSet resultSet = statement.executeQuery(query);
    System.out.println("sex, survived, minimum age");

    while(resultSet.next()){
        String sex = resultSet.getString( s: "sex");
        int survived = resultSet.getInt( s: "survived");
        double age = resultSet.getDouble( s: "min_age");

        System.out.println(sex + ", " + survived + ", "+ age);
    }
}

/**
```

List_of_CSV class:

List of CSV class contains a constructor and several getters. To prevent the indexOutofBounds exception, I set while size of field is 12, then I get the embark. If not, I won't set the embarked.

To prevent the empty element, I uses the null check which is as follows:

```java
public void nullCheck(String age, String cabin){
    if(!age.isEmpty()){
        this.age = Double.parseDouble(age);
    }

    if(!cabin.isEmpty()){
        this.cabin = cabin;
    }
}
```

And make sure that the data will keep null if there is no elements inside the data rather than convert the data type and cause Exception.

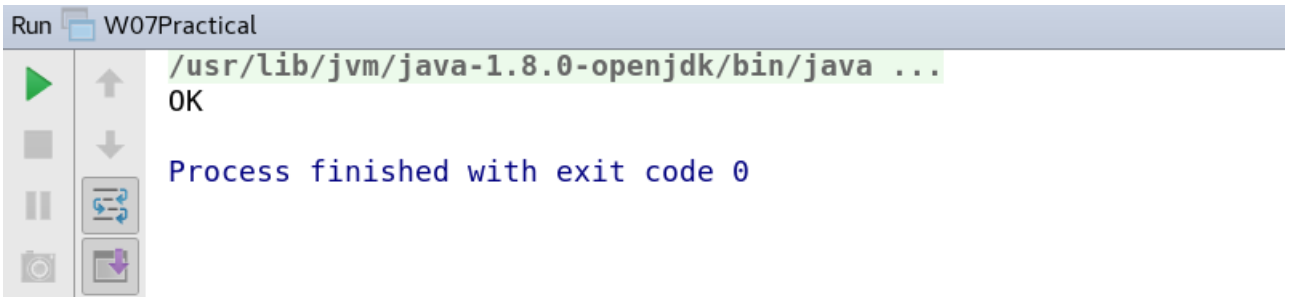
```
public class List_of_CSV {
    private int passengerId, survived, pClass, sibSp, parch;
    private String name, sex, ticket, cabin, embarked = null;
    private double fare;
    private Double age = null;

    public List_of_CSV(ArrayList<String> fields){
        if(fields.size() == 12){
            this.embarked = fields.get(11);
        }
        this.passengerId = Integer.parseInt(fields.get(0));
        this.sibSp = Integer.parseInt(fields.get(6));
        this.parch = Integer.parseInt(fields.get(7));
        this.survived = Integer.parseInt(fields.get(1));
        this.pClass = Integer.parseInt(fields.get(2));
        this.fare = Double.parseDouble(fields.get(9));
        this.name = fields.get(3);
        this.sex = fields.get(4);
        this.ticket = fields.get(8);
        nullCheck(fields.get(5), fields.get(10));
    }
```

Testing and Examples:
All checks have passed in stacscheck and is now shown as follow and I have been checked using titanic small data and the result will be shown after the screen shot:

```
pc2-143-l:~/Documents/cs1003/W07-Practical hl74$ stacscheck /cs/studres/CS1003/Practicals/W07
Testing CS1003 Week 7 Practical
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - basic/build : pass
* COMPARISON TEST - basic/Test01_no_arguments/progRun-expected.out : pass
* COMPARISON TEST - basic/Test02_titanic-small_create/progRun-expected.out : pass
* COMPARISON TEST - basic/Test03_titanic-small_query1/progRun-expected.out : pass
* COMPARISON TEST - basic/Test04_titanic-small_query2/progRun-expected.out : pass
* COMPARISON TEST - basic/Test05_titanic-small_query3/progRun-expected.out : pass
* COMPARISON TEST - basic/Test06_titanic-small_query4/progRun-expected.out : pass
* COMPARISON TEST - basic/Test07_titanic-large_create/progRun-expected.out : pass
* COMPARISON TEST - basic/Test08_titanic-large_query1/progRun-expected.out : pass
* COMPARISON TEST - basic/Test09_titanic-large_query2/progRun-expected.out : pass
* COMPARISON TEST - basic/Test10_titanic-large_query3/progRun-expected.out : pass
* COMPARISON TEST - basic/Test11_titanic-large_query4/progRun-expected.out : pass
* INFO - basic/TestQ_CheckStyle/infoCheckStyle : pass
--- submission output ---
13 out of 13 tests passed
```

test.db create titanic-small.csv:



test.db query1:

```
/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
passengerId, survived, pClass, name, sex, age, sibSp, parch, ticket, fare, cabin, embarked
1, 0, 3, "Braund  Mr. Owen Harris", male, 22.0, 1, 0, A/5 21171, 7.25, null, S
2, 1, 1, "Cumings  Mrs. John Bradley (Florence Briggs Thayer)", female, 38.0, 1, 0, PC 17599, 71.2833, C85, C
3, 1, 3, "Heikkinen  Miss. Laina", female, 26.0, 0, 0, STON/O2. 3101282, 7.925, null, S
4, 1, 1, "Futrelle  Mrs. Jacques Heath (Lily May Peel)", female, 35.0, 1, 0, 113803, 53.1, C123, S
5, 0, 3, "Allen  Mr. William Henry", male, 35.0, 0, 0, 373450, 8.05, null, S
6, 0, 3, "Moran  Mr. James", male, null, 0, 0, 330877, 8.4583, null, Q
7, 0, 1, "McCarthy  Mr. Timothy J", male, 54.0, 0, 0, 17463, 51.8625, E46, S
8, 0, 3, "Palsson  Master. Gosta Leonard", male, 2.0, 3, 1, 349909, 21.075, null, S
9, 1, 3, "Johnson  Mrs. Oscar W (Elisabeth Vilhelmina Berg)", female, 27.0, 0, 2, 347742, 11.1333, null, S
```

test.db query2:

```
/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
passengerId, survived, pClass, name, sex, age, sibSp, parch, ticket, fare, cabin, embarked
1, 0, 3, "Braund  Mr. Owen Harris", male, 22.0, 1, 0, A/5 21171, 7.25, null, S
2, 1, 1, "Cumings  Mrs. John Bradley (Florence Briggs Thayer)", female, 38.0, 1, 0, PC 17599, 71.2833, C85, C
3, 1, 3, "Heikkinen  Miss. Laina", female, 26.0, 0, 0, STON/O2. 3101282, 7.925, null, S
4, 1, 1, "Futrelle  Mrs. Jacques Heath (Lily May Peel)", female, 35.0, 1, 0, 113803, 53.1, C123, S
5, 0, 3, "Allen  Mr. William Henry", male, 35.0, 0, 0, 373450, 8.05, null, S
6, 0, 3, "Moran  Mr. James", male, null, 0, 0, 330877, 8.4583, null, Q
7, 0, 1, "McCarthy  Mr. Timothy J", male, 54.0, 0, 0, 17463, 51.8625, E46, S
8, 0, 3, "Palsson  Master. Gosta Leonard", male, 2.0, 3, 1, 349909, 21.075, null, S
9, 1, 3, "Johnson  Mrs. Oscar W (Elisabeth Vilhelmina Berg)", female, 27.0, 0, 2, 347742, 11.1333, null, S
```

test.db query3:

```
/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
pClass, survived, count
1, 0, 1
1, 1, 2
2, 0, 1
2, 1, 2
3, 0, 5
3, 1, 2
```

test.db query4:

```
/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
sex, survived, minimum age
female, 0, 31.0
female, 1, 14.0
male, 0, 2.0
male, 1, 34.0

Process finished with exit code 0
```

no argument:

```
/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
Usage: java -cp sqlite-jdbc.jar:. W07Practical <db_file> <action> [input_file]

Process finished with exit code 255
```

As is shown, it is the same as the result shown in pdf.

Evaluation:

There are several code style issues available in my program such as the name as method. Some commands is too complex such as convert the number as null. Next time I will try to solve the problem of the database instead of change the ways of printing message. The third part I should

improve is the name of method, I should make the name be more regular instead of name method as its function.

Conclusion:

As a result, there are some of the issues available to my program but there is no doubt that my program works fine. In this practical, I have learned the use of JDBC and how to insert data or update data to the database. I have also reviewed the use of list and with the help of teacher I have learned the use of Arraylist and how to prevent the missing vlaue. If I have more time I will try to re-name the method and try to find a way to make the program shows null as numbers instead of convert the showing result into “null”.

Extensions:

Extension 1: additional action
Users are able to select the specific column as they wish and what they select will be shown as the order they have inputted.
This method is designed using default methods and if there is no such column, the program will return the input example:

```
catch (SQLException e) {
    System.out.println("SQL error: " + e.getMessage());
    System.out.print("Input example: ");
    System.out.println("create / query(1-4) / \"name of column\"(Please split the column with ',')");
```

```
        Statement statement = connection.createStatement();
        String query = "SELECT " + action + " FROM person";
        ResultSet resultSet = statement.executeQuery(query);
        ResultSetMetaData resultSetMetaData = resultSet.getMetaData();

        int count = resultSetMetaData.getColumnCount();
        String type = null;
//        System.out.println(count);

        while (resultSet.next()){
            String result = "";
            for(int i=1;i<=count;i++) {
                type = resultSetMetaData.getColumnTypeName(i);
//                System.out.println(type);
                switch (type) {
                    case "VARCHAR": {
                        String value = resultSet.getString(i);
                        result += " " + value;
                        break;
                    }
                    case "FLOAT": {
                        double value = resultSet.getDouble(i);
                        result += " " + (value == 0.0 ? "null" : value);
                        break;
                    }
                    case "INT": {
                        int value = resultSet.getInt(i);
                        result += " " + (value == 0.0 ? "null" : value);
                        break;
                    }
                    default: {
                        break;
                    }
                }
            }
            System.out.println(result);
```

Test:

no column:

```
/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
SQL error: [SQLITE_ERROR] SQL error or missing database (no such column: asdd)
Input example: create / query(1-4) / "name of column"(Please split the column with ',')

Process finished with exit code 0
```

“age, sex”:

```
/usr/lib/jvm/java-1.8.0-openjdk/bin/java ...
 22.0 male
 38.0 female
 26.0 female
 35.0 female
 35.0 male
 null male
 54.0 male
 2.0 male
 27.0 female
 14.0 female
```

Extension 2: View created

To create a view, I created a method which is used for create a general database first:

```
private void createView(Connection connection, String queryName, String columnName) throws SQLException{
    Statement statement = connection.createStatement();
    String drop = "DROP VIEW IF EXISTS " + queryName;
    String query = "CREATE VIEW " + queryName + " AS SELECT " + columnName + " FROM person";
    statement.executeUpdate(drop);
    statement.executeUpdate(query);
}
```

This is used for create the view with three parameters. Then I putted a queryName to the place I need to create a view:e.g. query 3

```
Statement statement = connection.createStatement();
String queryName = "query3";
createView(connection, queryName,  columnName: "pClass, survived");

  String query = "SELECT pClass, survived, COUNT (pClass||survived) as counts FROM person GROUP by pClass, surv
String query = "SELECT *, COUNT (pClass||survived) as counts FROM " + queryName + " GROUP by pClass, survived";
ResultSet resultSet;
resultSet = statement.executeQuery(query);
System.out.println("pClass, survived, count");

while (resultSet.next()){

    int counts = resultSet.getInt( s: "counts");
    int pClass = resultSet.getInt( s: "pClass");
    int survived = resultSet.getInt( s: "survived");

    System.out.println(pClass + ", " + survived + ", " + counts);
}
```

and I run this program:

```
pClass, survived, count
1, 0, 1
1, 1, 2
2, 0, 1
2, 1, 2
3, 0, 5
3, 1, 2
```

These are all my extensions and main program.