CS1002 W07 Practical 160026526 Tutor: Ruth Letham 8 March 2018

• Overview:
The aim of the practical was to create a database based on the survivors of the Titanic and the data should be manipulated in accordance to the specifications. The program should be first able to choose, manipulate and store data from a csv file into a database. The program should then be able to manipulate the database to provide the specified outputs from 4 specific queries statements provided in the specification. The JDBC API is used so appropriate methods and classes should be created to manipulate the data in a SQLite database. Errors should also be taken into consideration to ensure that the program runs smoothly. The program should also take in 3 arguments according to the specification.

• Design:
Creating the database
To begin the practical, it is first noted that a database must be created which will later store the values of the people who were on the Titanic. To do this, a method called tryToAccessDB was created which requires the database file name and a csv. For most of the methods which involves the database, an SQLException should be thrown to handle the exception. A try-catch statement is created which will first create a String called dbUrl which contains the database filename with the jdbc:sqlite prefix. This is then used when making a connection to the DriverManager. Method to create the table, read the csv and print "OK" are written which will be explained later on. It is good practice to close the connection and, after catching the exception, writing a finally statement to close the connection to the database.

Writing the createTable method
Since the table will be created in the database, the method should take a connection. A statement is first created to connect with the database. The specification states to drop any table for person if it exists so the statement then executes an update using these parameters. The second execute update then creates a person table which holds values for the 12 values (passengerId, survived, pClass, name, sex, age, sibSp, parch, ticket, fare, cabin and embarked). The statement is then closed.

Writing the readCSV method
The readCSV method should take in a csv file and connect with the database so it requires them as its parameters. To read the csv, a try with resources statement is used to first create a new buffered reader which will be used to read the csv file. The first line contains the headers so those are read off first using reader.readLine(). A while loop is then created to read the lines of the csv and each line is then split by commas into elements that will be stored in an array. Now it is required to transfer the elements from the string array into the table. At first this was done by creating multiple array lists to store the different elements buy individually inserting into the array lists and then making an inserPersonUsingPreparedStatment method. It was noticed that this method is inefficient as it took much more space to make the increased number of array lists so this idea was scrapped and commented out. Validity check methods were then created to insert the data into the database and change empty data to null. It should be noted here that a String line is set empty at the start and 12 of the table hears were initialised at the start from 0 to 11 using the private final prefixes.

Validity check for string/int/float
At this point of time, a method is required to check if the string is valid and if it contains any empty data. The empty data is required to be set to null and the data should later be inserted into the person table in the database. To do this, an if statement is first used to check the individual string arrays have any empty data and then statement.setString is used to set the string to null if there is empty

data and the else is used to just add the data which are not null to the database. It should be noted that (index + 1) is used as the database parameter index starts from 1 instead of 0. For the int and float validity checks, it basically uses similar logic but instead uses the statement.setNull to if the data is empty. The else statements for both of them requires to parse the element value into the appropriate data type before setting the perimeter index.

## Writing the readCSV method (cont.)

Going back to the readCSV method, the validity checks will require a PreparedStatement statement which will be used to call the validity checks and insert the data into the database. The validity checks are then called upon using the appropriate conditions for all the values except the embarked value. When checking the large Titanic csv file, it is noted that there was a passenger which did not embark therefore the embarked was empty for the person. An if else statement was manually written for the embarked value which takes into account the passenger which did not board. Finally the statement is updated and closed and the catch statements were written.

Now that the readCSV method works, an action method is created and this should hold the individual queries and the initial create statement.

## Writing the action method

The action method should be called when writing the main method so it will take in an array that contains the arguments. A connection is first used similar to the one in the tryToAccessDB method in which the only difference being that it takes args[0] which will be the name of the database file. Inside the try-catch statement, a switch statement is used which takes 6 cases (create, query1, query2, query3, query4 and a default for good practice). The create action requires for the database to be created while reading the csv file and adding the values into the table so the tryToAccessDB method is called with args[0] and args[2] which will take in the database file name and csv file respectively.

## Starting query1

Query 1 requires a list of all records in the database so, to begin, a printTable method was written up. The statement is used to connect to the database so and the resultSet is used to select all the data from the person table. The headers for all the values are first manually printed out and then a while loop is used to set first temporarily store each value from the resultSet to its corresponding value and they are then printed out which loops through all the lines. The statement is finally closed. Going back to the action method, the printTable method is written in the query1 case. Thus case one prints all records.

## Starting query2

Query 2 requires for the total number of survivors to be counted so a totalSurvivors method is created. In this method, counting is first initialised as 0 and this will be used to count the values. After connecting to the database, a resultSet is used to locate the values from person in where survived = 1. Counting is increased by one using a while loop around the resultSet.next which will only get survivors. So the method will then print “Number of Survivors “ and print the final value of counting which will now be the total number of survivors and thus the statement is closed. This method is then called in the query2 case in the action method.

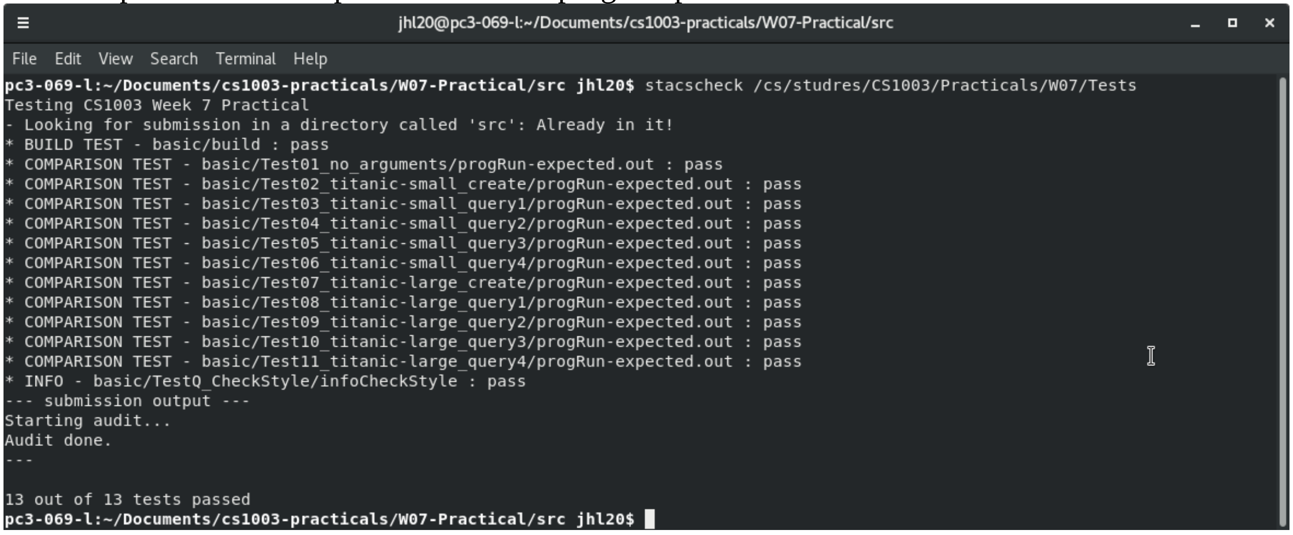
## Starting query3

Query 3 requires the count of people in different classes depending if they survived or not. Since there are a total of 3 classes and a person can only be alive or dead, we can say that there should be a total of 6 groups (1st class survived, 1st class didn’t survive, 2nd class survived, 2nd class didn’t survive, 3rd class survived, 3rd class didn't survive) which will incorporate the class and survived status. A method is created called countingSurvivorByClass and 6 counting's are initialised as 0 to represent the count for the 6 groups. After connecting to the database, the headers are first printed and  6 resultSet statements are created to hold the different groups of people and individual while loops are loop for all 6 statements, incrementing the corresponding counting values.  After each while loop, the counting is printed along with the corresponding class and survived value. This method is then called in the query3 case in the action method.

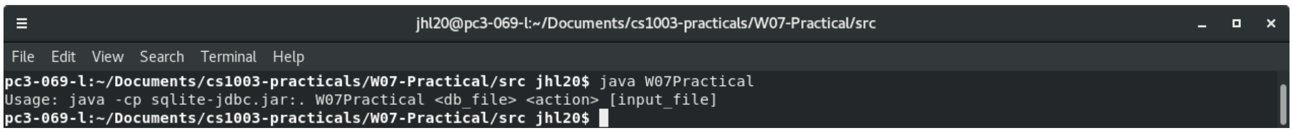
<u>Starting query4</u>
Query 4 requires to find the minimum age of females who survived and did not survive and makes who survived and did not survive. Therefore there are four groups of people here (females who didn't survive, females who survived, males who didn't survive and males who survived). A minimum age method is created and 4 floats are initialized as 0 with the names min, min2, min3, min4 which will later hold the minimum age of the groups respectively. After connecting to the database and printing the headers,  4 resultSet statements are created which will select the min age from people depending on sex and if they survived. Each of the min's are then set to the resultSet as they hold the min values depending on the group and then the min is printed along with the sex and if they survived. The statement is then closed and the method is called in the query4 case in the action method. The action method is then catches the exceptions and is finally closed.

<u>Writing the main method</u>
After importing the relevant packages, the main method is written and an if statement is used to set the perimeter and ensure that the usage is printed if the argument length is less than 1. Else the program will create a new practical and run the arguments into the action method and thus completes the program.

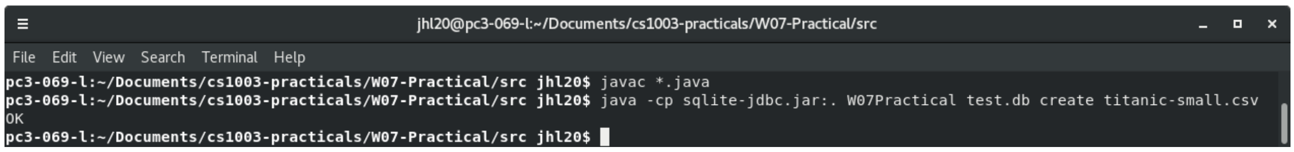
• Testing:
The first test to ensure that the program works according to the specification is the autochecker which is provided in the specification. The program passed all 13 of the autochecker tests.


```
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$ stacscheck /cs/studres/CS1003/Practicals/W07/Tests
Testing CS1003 Week 7 Practical
- Looking for submission in a directory called 'src': Already in it!
* 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 ---
Starting audit...
Audit done.
---

13 out of 13 tests passed
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$
```

The program is then manually tested. The first test runs without arguments to check the usage if it is included in the program.

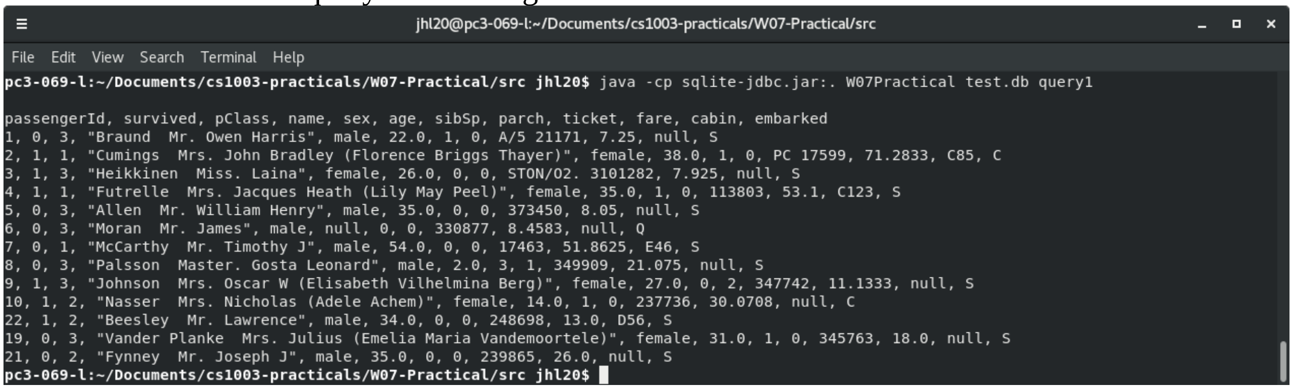

```
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$ java W07Practical
Usage: java -cp sqlite-jdbc.jar:. W07Practical <db_file> <action> [input_file]
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$
```

The second test is to create the database for titanic-small.csv.


jhl20@pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src
File Edit View Search Terminal Help

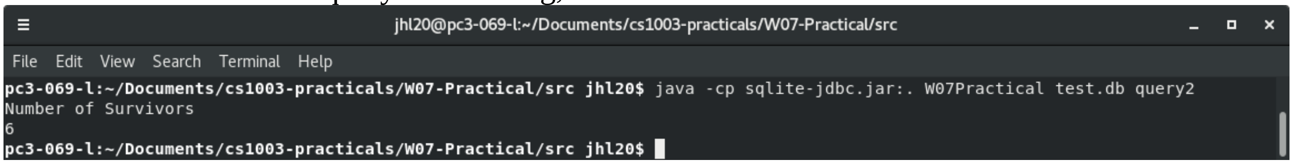
```
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$ javac *.java
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$ java -cp sqlite-jdbc.jar:. W07Practical test.db create titanic-small.csv
OK
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$
```

The third test checks if query1 is working.


jhl20@pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src
File Edit View Search Terminal Help

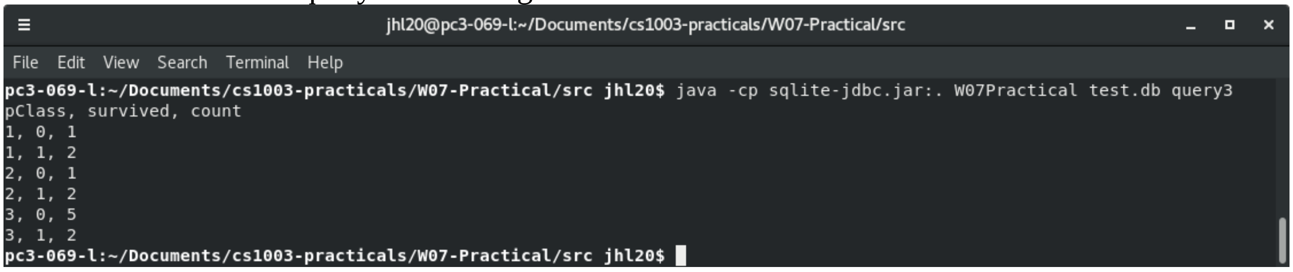
```
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$ java -cp sqlite-jdbc.jar:. W07Practical test.db query1

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
10, 1, 2, "Nasser  Mrs. Nicholas (Adele Achem)", female, 14.0, 1, 0, 237736, 30.0708, null, C
22, 1, 2, "Beesley  Mr. Lawrence", male, 34.0, 0, 0, 248698, 13.0, D56, S
19, 0, 3, "Vander Planke  Mrs. Julius (Emelia Maria Vandemoortele)", female, 31.0, 1, 0, 345763, 18.0, null, S
21, 0, 2, "Fynney  Mr. Joseph J", male, 35.0, 0, 0, 239865, 26.0, null, S
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$
```

The fourth test checks if query2 is working,


jhl20@pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src
File Edit View Search Terminal Help

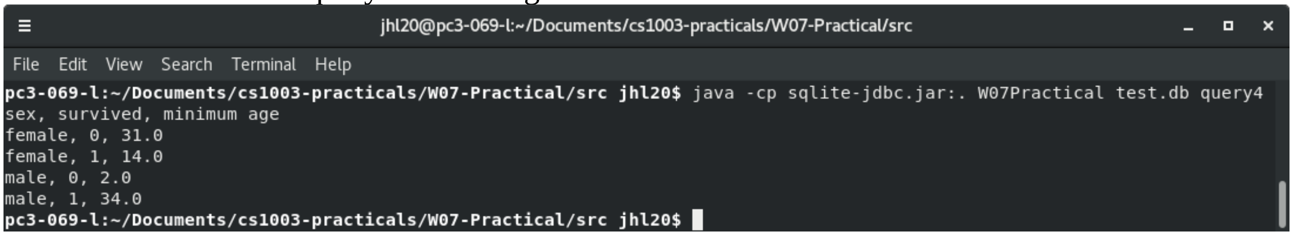
```
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$ java -cp sqlite-jdbc.jar:. W07Practical test.db query2
Number of Survivors
6
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$
```

The fifth test checks if query3 is working.


jhl20@pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src
File Edit View Search Terminal Help

```
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$ java -cp sqlite-jdbc.jar:. W07Practical test.db query3
pClass, survived, count
1, 0, 1
1, 1, 2
2, 0, 1
2, 1, 2
3, 0, 5
3, 1, 2
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$
```

The sixth test checks if query4 is working


jhl20@pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src
File Edit View Search Terminal Help

```
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$ java -cp sqlite-jdbc.jar:. W07Practical test.db query4
sex, survived, minimum age
female, 0, 31.0
female, 1, 14.0
male, 0, 2.0
male, 1, 34.0
pc3-069-l:~/Documents/cs1003-practicals/W07-Practical/src jhl20$
```

• Evaluation:
In this practical, the program was required to be able to create a database which is able to hold values from a csv dataset of the survivors of the Titanic. The program should then be able to use queries to perform specific tasks. These queries should be able to print a list of all records in the database, print out the total survivors of the Titanic, list the count of the number of people in a certain group of people depending on class and if they survived and list the minimum age of a certain group of people depending on sex and if they survived. The program should also run all the autochecker's tests without flaws. The program was able to successfully carry out the requirements set by the practical therefore is successful in completing the task provided.

• Conclusion:
In conclusion, the program was successful in creating a database and using queries to perform specific tasks in regards to the regulations placed by the practical.  Difficulties were encountered when first trying to set up the database. Other than that, when trying to transfer the data from the csv into the database, it took too long as many array lists were initially created which were scrapped as they were deemed to be inefficient. The code was also all written in one java file so it should also be noted that more java classes should be created in the future to ensure the code is cleaner and more readable. The practical was challenging to an extent as this was the first time using SQLite in a practical and more practice is required to fluently implement SQLite in the future. None of the extensions were carried out due to the lack of time and knowledge of the usage of outside packages. Given more time, the first extensions would have been attempted as it seems to be the simplest to implement when considering all other extensions.