Creating arrays of objects from database tables using SQLite

SQL NOTES:

SELECT fields

FROM tbls

JOIN other tables

WHERE condition

GROUP BY grouping for aggregate functions

HAVING filter for groups

ORDER BY sort order

SELECT

FROM tbl, tbl, tbl etc

WHERE (**describe the join here**), conditions

GROUP BY

HAVING

ORDER BY

DELETE FROM tbl

WHERE condition

UPDATE tbl

SET field = value

WHERE condition

INSERT INTO tbl (field, field, field field etc)

VALUES (value, 'value', 'value', value etc)

(The primary key, if auto-number, is not included. Strings go into inverted commas)

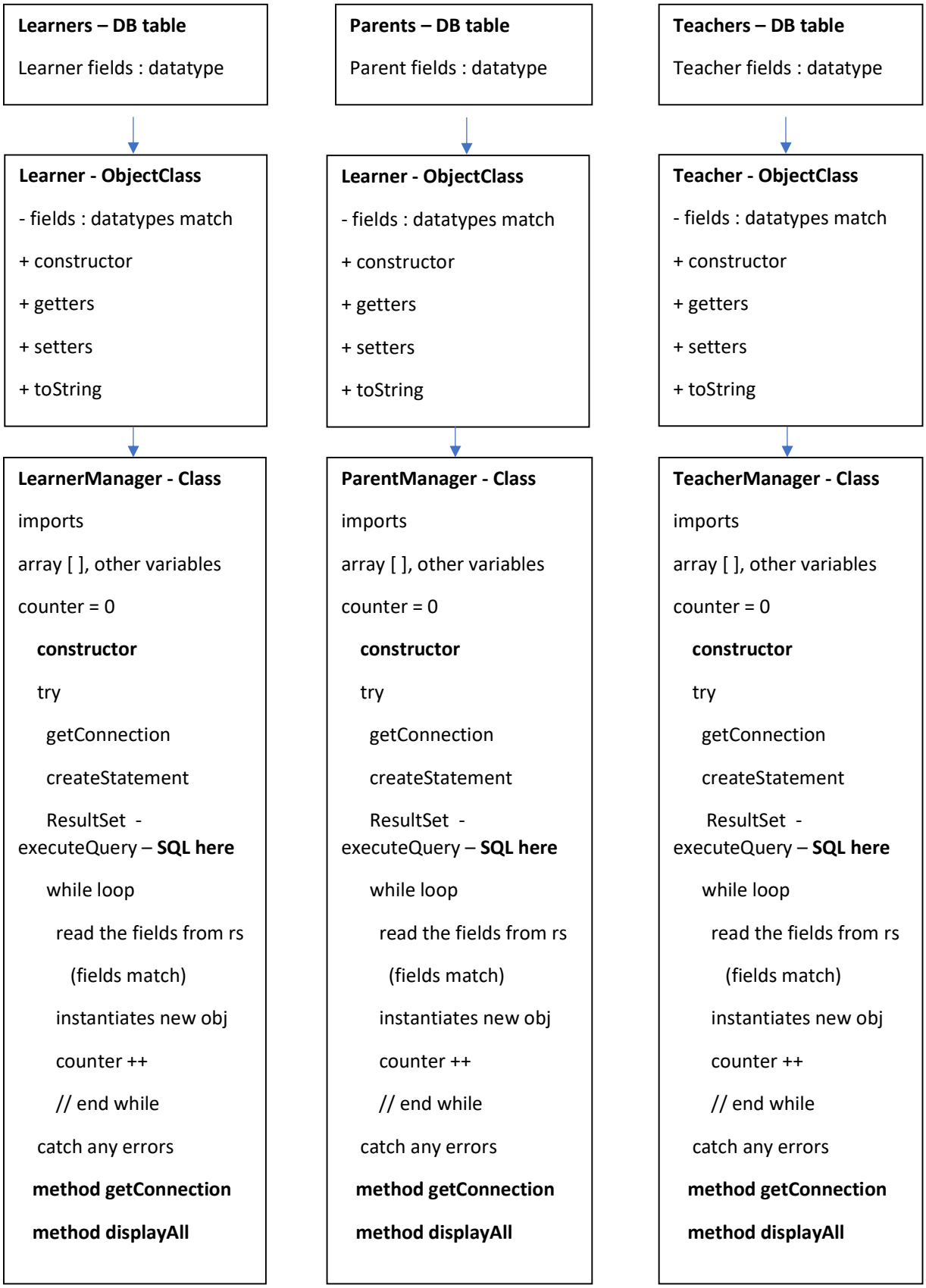
NOTES:

1. Create a new object class for each table in the database.
2. Create a new Manager class for each object class.
3. The "ResultSet" is a dataset (sub-set of the database) that is returned from the database, based on the query; it has the same field names as the original database table. The dataset is based the structure of the SQL statement . . .
   - SELECT * FROM table – Here the dataset is the same as the original table.
   - SELECT firstName, dateOfBirth FROM table WHERE firstName = "Joe" – Here the dataset is perhaps only one record and only features the two selected fields.
   - Not all SQL queries return a resultSet e.g. DELETE and UPDATE.

The Process

- Connect to the database.
- Create the statement
- Execute the statement with its SQL code
  - The results (if any) get stored in the resultSet
  - `resultSet = state.executeQuery("SELECT * FROM tbl");`
- Do something with the resultSet (display, sort, search etc)
- Write back to the database using the INSERT, UPDATE or DELETE SQL statements

---


Learners – DB table
Learner fields : datatype
Learner - ObjectClass
- fields : datatypes match
+ constructor
+ getters
+ setters
+ toString
LearnerManager - Class
imports
array [ ], other variables
counter = 0
constructor
try
getConnection
createStatement
ResultSet - executeQuery – SQL here
while loop
read the fields from rs
(fields match)
instantiates new obj
counter ++
// end while
catch any errors
method getConnection
method displayAll
Parents – DB table
Parent fields : datatype
Learner - ObjectClass
- fields : datatypes match
+ constructor
+ getters
+ setters
+ toString
ParentManager - Class
imports
array [ ], other variables
counter = 0
constructor
try
getConnection
createStatement
ResultSet - executeQuery – SQL here
while loop
read the fields from rs
(fields match)
instantiates new obj
counter ++
// end while
catch any errors
method getConnection
method displayAll
Teachers – DB table
Teacher fields : datatype
Teacher - ObjectClass
- fields : datatypes match
+ constructor
+ getters
+ setters
+ toString
TeacherManager - Class
imports
array [ ], other variables
counter = 0
constructor
try
getConnection
createStatement
ResultSet - executeQuery – SQL here
while loop
read the fields from rs
(fields match)
instantiates new obj
counter ++
// end while
catch any errors
method getConnection
method displayAll

```
 1 /* Connecting of a SQLite database and creating an array
of Learner objects
 2  */
 4 // package sqlitedemo3;
 5
 6 import java.sql.Connection;
 7 import java.sql.DriverManager;
 8 import java.sql.ResultSet;
 9 import java.sql.SQLException;
10 import java.sql.Statement;
11 import java.util.logging.Level;
12 import java.util.logging.Logger;
13
14 public class LearnerManager {


15 // Global variables belonging to the whole class.
16 private static Connection con = null;
17 private static Statement state = null;
18 private static ResultSet rs = null;
19 private static Learners[] ln = new Learners[200];
20 private static int size = 0;
21 ============================================================
```

```
// Constructor
22 public LearnerManager() {
23
24  getConnection();
25  rs = null;
26
27  try {
28    state = con.createStatement();
29    rs = state.executeQuery("SELECT * FROM tblLeaners");
30
31// Read in the field names from the table into temporary
variables. NB some field names have been shorten for printing
purposes
32     while (rs.next()) {
33        int learnerID = rs.getInt("LearnerID");
34        int dutyID = rs.getInt("DutyID");
35        String nickName = rs.getString("Nickname");
36        String firstName  = rs.getString("FirstName");;
37        String surname = rs.getString("Surname");;
38        int grade = rs.getInt("Grade");;
39        String dob  = rs.getString("DOB");;
40        String cellNumber  = rs.getString("CellNumber");;
41        String emailAddress = rs.getString("Email");;
42        String preference = rs.getString("Preference");;
43        boolean Licence = rs.getBoolean("Drivers");;
44        double AmountPaid = rs.getDouble("AmountPaid");;
45
46 // Copy the contends of the temporary variables into a
unique object in the array
47    ln[size] = new Learners(learnerID, dutyID, nickName,
firstName, surname, grade, dob, cellNumber, emailAddress,
preference, Licence, AmountPaid);
48
49     System.out.println("Success" + size);
50     size++;
51
52     } // end while
53  } catch (SQLException ex) {
54      System.out.println("Error");
55      System.out.println(ex.getMessage());
56      Logger.getLogger
(SQLiteManager.class.getName()).log(Level.SEVERE, null, ex);
57          }
60 } // end constructor LearnerManager
61 ======================================================
```

```
62 // Establish a connection with the exact database in the
NetBeans project folder. The name of the database is
"LearnersDonations.db".(same place you would put your text
file.)

65 private void getConnection(){
66     try {
67         Class.forName("org.sqlite.JDBC");
68         con = DriverManager.getConnection
               ("jdbc:sqlite:LearnersDonations.db");
69         System.out.println("Connection success");

70     } catch (ClassNotFoundException | SQLException e) {
71         System.out.println("Error%");
72         System.out.println(e.getMessage());
73     }
74 } // end getConnection
75 ============================================================

76 // Terminate connection to LearnersDonations.db database
77 public void closeConnection(){
78     try {
79         if (con != null) {
80             con.close();
81             state.close();
82             rs.close();
83             System.out.println("Connection closed");
84         }
85     } catch (SQLException ex) {
86         System.out.println(ex.getMessage());
87     }
88 } // end close connection
87 ============================================================
```

```
 88 // Display all the learners in the database. Only a few
fields
 89 public void displayAll() {
 90     if (con == null) {
 91         getConnection();
 92         System.out.println("connection null");
 93     }
 94     try {
 95         state = con.createStatement();
 96         rs = state.executeQuery("SELECT firstName,
Surname, Grade FROM tblLeaners");


 97         while (rs.next()){
 98            System.out.println(rs.getString("firstName") +
" " + rs.getString("Surname") + " " +  rs.getString("Grade"));
 99         }
100
101     } catch (SQLException ex) {
102         System.out.println("error#");
103     }
104
105 } // end displayAll
106 ============================================================


107 // Displays all the learners in the array of objects. All
the fields as per the toString method as found in the Learners
object class.
109 public String displayAll2() {
110       String output = "";
111       for(int i = 0; i < size; i++) {
112          output = output + ln[i].toString();
113       }  // end for
114
115       return output;
116
117 } // end displayAll2
118 ============================================================
119 } // end class
```

<u>SQL examples:</u> Build the SQL statement by joining hard code to the various field values from the GUI.

txfFirstName.getText( ) – get the text that is stored in the text field called “txfFirstName”. Therefore it is important to name the fields on the GUI consistently and fully.

Code behind an insert button

```
String myString = “INSERT INTO USER (id, name) VALUES (“ + txfId.getText( ) + “ ‘ “ +
txfName.getText( ) + “ ‘ ) “ ;
System.out.println(myString);                    // Prints for troubleshooting purposes
s.execute.Update(myString);                      // s is the created statement object
r = s.executeQuery(“SELECT * FROM tblUser”);     // Execute and assign to the resultSet
txfId.setText (“ “);                             // Clears the ID text field in the GUI
txfName.setText(“ “);                            // Clears the name text field in the GUI
```

Code behind an update button – Build the SQL statement by joining hard code to the various field values from the GUI.

```
String s2 = “UPDATE USER SET NAME = ‘ “ + txfText.getText( ) + “ ‘ WHERE ID = “+
txfMoreText.getText( ) ;
System.out.println( s2);                         // Prints for troubleshooting purposes
s.executeUpdate(s2)                              // Execute only. No resultSet for UPDATE
```

Code behind the delete button - Build the SQL statement by joining hard code to the various field values from the GUI.

```
String s3 = “DELETE FROM USER WHERE ID = “ + txfFirst.getText( ) ;
System.out.println( s3);                         // Prints for troubleshooting purposes
s.executeUpdate(s3);                             // Execute only. No resultSet for DELETE
r=s.executeQuery(“ SELECT * FROM USER”);         // Displays to view the new entry
txfFirst.getText(“ “);                           // Clears the text fields in the form
```

Code behind the Exit button

```
conn.close ( );                                  // closes the relevant connection object
st.dispose ( );                                  // disposes of the relevant created object
```