SCC201 Databases: Assessed Coursework

Write a program (in Java and using JDBC) that takes a database and produces a textual "backup" of the contents.

This textual backup should consist of SQL statements that will recreate the contents of the original database i.e. `CREATE TABLE` and `INSERT .. INTO` instructions. You should be able to recreate the original database by using the text files your program produces as input to SQLite.

Say there is an SQLite database called Original.db and the main method of your coursework solution is in Coursework.class. On Linux:

```
java -cp "sqlite-jdbc-3.7.2.jar:."  Coursework Original.db >
          statements.sql
```

(on Windows change the : to ;)

You should run your output through sqlite3 to ensure that you have generated a correct textual backup.

```
sqlite3 Copied.db < statements.sql
```

Remember that when running sqlite3 on Linux, you must run it in a local folder (not on your "h:" drive). `Copied.db` should be identical in content to `Original.db`.

Your program must NOT use the system schema tables found in SQLite; any access to schema information must use the appropriate JDBC methods. The JDBC documentation is linked from the course Moodle page, but another good starting place to find out how to access metadata via JDBC is to google "JDBC metadata".

We provide skeletal code which must be used.
You should ONLY change the `DatabaseDumper201.java` file.
`DatabaseDumper201.java` MUST extend the `DatabaseDumper` abstract class found in `DatabaseDumper.java`. You must NOT modify `DatabaseDumper.java`. Javadoc for `DatabaseDumper` is included in the downloads. A `Coursework.java` file is included to demonstrate how your code may be used with a connection to a sqlite3 database.

Milestones

Here we suggest a series of milestones your program should reach. You do not have to develop your code in this way. (For example, we suggest you deal with INSERT statements before CREATE TABLE statements. But you could do this the other way around if you prefer). However, your work will be marked according to these milestones and how well you achieved the required outcome.

a) A single text file containing all the INSERT..INTO statements required.

```
INSERT INTO projects VALUES( COMIC, COMIC, ESPRIT, 100000 );
```

This will not work, as the text field values are not quoted in primes'.

b) A single text file containing all the correct INSERT..INTO statements required.

```
INSERT INTO projects VALUES( 'COMIC', 'COMIC', 'ESPRIT', 100000 );
```

To test this, we will provide you (on the Moodle page) with the `CREATE TABLE` statements required to create the tables that your text file will document.

c) A single text file as in (b) above, but also contains at the start the `CREATE TABLE` statements that create the tables that your text file will document. (But without the primary and foreign keys being indicated).

```
CREATE TABLE give_course (
      s_id VARCHAR(4),  c_id
      VARCHAR(3)
);
```

d) A single text file as in (c) above, but the CREATE TABLE statements include indicators of primary keys.

```
CREATE TABLE give_course (  s_id
      VARCHAR(4),
      c_id VARCHAR(3),
      PRIMARY KEY(s_id,c_id)
);
```

e) A single text file as in (d) above, but including foreign keys.

```
CREATE TABLE give_course (  s_id
      VARCHAR(4),
      c_id VARCHAR(3),
      PRIMARY KEY(s_id,c_id),
      FOREIGN KEY (s_id) REFERENCES staff(s_id),
      FOREIGN KEY (c_id) REFERENCES courses(c_id)  );
```

f) The database metadata also includes details of the indexes present in the database. Add code to establish what indexes are present and include CREATE INDEX statements in your backup to recreate these. You should have a set of statements of the form (what follows is just an example):

```
CREATE INDEX planets_id ON planets (planet_id);
```

Indexes can be specified as `ASC` (ascending) or `DESC` (descending). However, there is a bug in SQLITE which means when you try to find out if an index is `ASC` or `DESC` it always returns null. So in your solution you should have the code which attempts to find out if an index is `ASC` or `DESC` and deal with the possibility that `ASC, DESC` or null has been returned by your enquiry. (So your executing code will always see "null" but your code should include statements to deal with all three possible outcomes : `ASC, DESC` or null).

g) If the database structure is modified after creation it is possible for tables to be returned in an order that would break key constraints – i.e. a table depends on a yet to be created table. Tables need sorting based on foreign keys.

As with (e) but with code to ensure the `CREATE TABLE` statements are in the 'correct' order.

Note You must not create a set of classes where each class achieves one milestone. Rather, as you move from one milestone to the next, you should enhance your class to achieve the next milestone.

Your solution will be tested against the databases you have been given **and at least one other database** – the expectation is that your solution is complete/ correct enough, at the stage you have reached, to **work with any database**. Marks will be awarded for how far you got with your solution, how well it addresses the issues for each stage, correct and well-presented indented output and, for your approach. Some additional marks may be available for additional features – a simple example might be inclusion of `DROP TABLE` statements that prepare the database for creating new tables if some already exist, additional flags like `NOT NULL` on key attributes (there are a few possible), including comments indicating driver and database version information, etc. Marks may be deducted for any concerns relating to output or code quality… structure, efficiency, commenting, etc.

NB

If you want to add additional classes to support DatabaseDumper201.java. You should add the extra classes to the end of DatabaseDumper201.java as static classes. E.g.

```
public class DatabaseDumper201 extends DatabaseDumper {



    . . .
    private static class TableModel{
        private String name;
        public TableModel(ResultSet rs,String tn){
            name=tn;
            . . .
        }
        public String getTableName(){return name;}
        . . .
    }
}
```

Submission

If you get past milestone (a), you must create a batch/shell file containing all the instructions necessary for running the files created as output by your program to create and populate the database copy.

Submission (to Moodle) Checklist (for the milestone you have reached)

1. All your Java source files. If you have developed your code in an IDE, do **not** submit the entire project. Just the Java source files you created. (You should make sure they compile and run outside of the IDE).
2. All the output files created by your code (for the milestone you have reached).
3. The batch/shell files required to create the database copies.

4. FINALLY and not least: Paste the java code for `DatabaseDumper.java` into the coderunner quiz for this assignment.

To gain marks you must submit to Moodle and demonstrate your solution as submitted to Moodle. Failure to submit, to demo, or demoing code different to that submitted will result in a mark of zero.