Closing Your JDBC Resources Cleanly

It is important that you close your JDBC connections properly. Otherwise, you risk getting locked out of the database server. Even if you kill your Java program, it is possible that the database server thinks that the client program is still around. It may keep its end of the connection open, waiting for data. This could eventually lock you out of the database if you end up creating too many instances of open connections.

Here is a real example of an unfortunate situation posted by a student:

"So I've been having this same problem, but I locked myself over 24 hours ago and I still don't have access. I'm not sure what to do since ITS isn't open and if CAEN can't help I'm skeptical anyway. I created this problem by my SQL queries were running on forever (can you even get into an infinite loop in SQL? I think I was) and when you Ctrl+C out it closes SQL improperly. I've learned from my mistake now, but it's too late and I can't get back in."

Here are a few tips to help reduce the likelihood of the above type of problem:

1. First, use SQL*Plus to debug your queries rather than using Java. Also make sure you quit your SQL*Plus sessions, otherwise you can still get locked out. It may help to design your queries on paper first and avoid too much nesting of queries. Nested queries tend to run slower as query optimizers have difficulty handling them. The above student was smart enough to do that but still ran into trouble. So, let's look at one more thing to do (Step 2) that may help.
2. Make sure you close all `Connection`s, `Statement`s, and `ResultSet`s in your code. This is tricky to do. Read this for some of the nuances:

   http://blog.shinetech.com/2007/08/04/howtoclosejdbcresourcesproperlyeverytime/

   The problem is, even closing a connection can lead to an exception in rare cases.

   We suggest using try-with-resources statements in your Java code to automatically close any JDBC objects that you create. This is a newer feature introduced in Java SE 7 that makes closing resources easier and more reliable. The next page shows an example of how this can be done for JDBC connections.

```
// Example use of a try-with-resources statement
public static void viewTable(Connection con) throws SQLException {

    String query = "SELECT COF_NAME, SUP_ID, PRICE, SALES, TOTAL FROM
COFFEES";

    try (Statement stmt = con.createStatement()) {
        ResultSet rs = stmt.executeQuery(query);

        while (rs.next()) {
            String coffeeName = rs.getString("COF_NAME");
            int supplierID = rs.getInt("SUP_ID");
            float price = rs.getFloat("PRICE");
            int sales = rs.getInt("SALES");
            int total = rs.getInt("TOTAL");

            System.out.println(coffeeName + ", " + supplierID + ", " +
                               price + ", " + sales + ", " + total);
        }
    } catch (SQLException e) {
        System.err.println(e.getMessage());
    }
}
```

This code is adapted from an example at:

https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html

Feel free to take the code and adapt it to your needs. The above code assumes that you only need to use a single `Statement` object, for example. That may not always be appropriate. For example, you may sometimes want to use multiple `Statement` objects to execute different fixed queries, in which case you could have one try-with-resources block for each `Statement` object that you create. Since a `Connection` object is provided to you in the Project 2 code, you do not need to worry about creating a try-with-resources block for the `Connection` object that you use in your query implementations.

If you think about the problem, it is actually pretty hard for a database server to distinguish between a slow client at the other end and a dead client. Remember that the communication between the client and the server occurs over a network. Doing all you can within Java to close the connections in all possible situations (including when queries fail) will help the server greatly.