COP2805C Module 10 Practice Exercise

In this practice exercise we will use JDBC to store birthday application users in a relational database.

The solution for this exercise uses Microsoft SQL Server; all necessary driver files are included in the repository with the solution project. If you prefer to use another database you will need to modify the driver files and connection URL as necessary.

If you are not familiar with Microsoft SQL Server or cannot load it on your personal system you will need to use the Horizon Academic IT system, where everything is installed and configured to run with no modifications.

The following instructions only apply if you are using a personal system; if you are using Horizon, they are not necessary.

- A free developer version of SQL Server can be downloaded using the following link: https://go.microsoft.com/fwlink/p/?linkid=2215158&clcid=0x409&culture=en-us&country=us
- You will also need to install SQL Server Management Studio using the following link: https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?redirectedfrom=MSDN&view=sql-server-ver16
- After installing the software you will need to use the SQL Server Configuration Manager tool to enable TCP/IP (port 1433); the service must then be restarted.
- There are two files specific to Microsoft SQL server which are included in the solution project, they are available using the following link: https://learn.microsoft.com/en-us/sql/connect/jdbc/download-microsoft-jdbc-driver-for-sql-server?view=sql-server-ver16
- The files you need from this download are:
    - mssql-jdbc-12.2.0.jre11.jar
        - this is the most recent JDBC driver downloaded from Microsoft
    - mssql-jdbc_auth-12.2.0.x64.dll
        - this is a dynamic library required to use Windows authentication, also downloaded from Microsoft

For this exercise a new UserDB class has been created, this class encapsulates the database operations using static methods which create the database, read the data, and delete the database. Normally we would allow the data to persist in between execution sessions and would not delete it, but for testing purposes we are removing it every time we exit the application.

The user object file operations have been disabled, we now write users and read users using the database. The log file functionality is still active.

Here are a few lines of the UserDB class. The class is declared final and the default constructor is declared private since it is not intended to be instantiated or extended, we are only using the static methods.

All necessary objects relating to using the database are declared as static members. We also need a few boolean variables to check the state of the database connection.

```
public final class UserDB {

    private UserDB() {}

    private static Connection con = null;
    private static Statement stmt = null;
    private static PreparedStatement pStatement = null;
    private static ResultSet rSet = null;
    private static boolean connected = false;
    private static boolean dbCreated = false;
```

There are quite a few constants declared which represent the database information and SQL statements used to manipulate the data:

```
private static final String DB_NAME = "BirthdayGreetings";
private static final String USER_TABLE_NAME = "Users";

// connection URLs: one for no specified DB, other for DB name
private static final String CONN_URL = "jdbc:sqlserver://localhost:1433;" +
        "integratedSecurity=true;" +
        "dataBaseName=" + DB_NAME + ";" +
        "loginTimeout=2;" +
        "trustServerCertificate=true";
private static final String CONN_NODB_URL = "jdbc:sqlserver://localhost:1433;"
+
        "integratedSecurity=true;" +
        "loginTimeout=2;" +
        "trustServerCertificate=true";
private static final String DB_CREATE = "CREATE DATABASE " + DB_NAME + ";";
private static final String TABLE_CREATE = "USE " + DB_NAME + ";" +
        "CREATE TABLE " + USER_TABLE_NAME +
        " (ID smallint PRIMARY KEY NOT NULL," +
        "FNAME varchar(80) NOT NULL," +
        "LNAME varchar(80) NOT NULL," +
        "EMAIL varchar(80) NOT NULL," +
        "LOCALE varchar(80) NOT NULL," +
        "BIRTHDAY varchar(80) NOT NULL);";
private static final String TABLE_INSERT = "USE " + DB_NAME + ";" +
        "INSERT INTO " + USER_TABLE_NAME +
        "(ID, FNAME, LNAME, EMAIL, LOCALE, BIRTHDAY)" +
        "VALUES(?, ?, ?, ?, ?, ?)";
private static final String TABLE_SELECT = "SELECT * FROM " + USER_TABLE_NAME +
";";
private static final String TABLE_DROP = "DROP TABLE " + USER_TABLE_NAME + ";";
private static final String DB_DROP = "DROP DATABASE " + DB_NAME + ";";
```

The createDB method in this class makes the connection to the database, creates the database if necessary, creates the user data table, and inserts test users (same data from previous versions) using the following algorithm:

start with a database-specific connection URL
while not connected
    get a connection to the user database
        if this fails, it is probably because the DB does not exist, try again with a generic connection
        if the second attempt fails, exit the loop, we will be exiting the application

if all connect attempts failed, exit the application
        (is SQL Server service running? Is TCP/IP enabled?)
if we connected without a DB, create the DB
create the table
create and insert the data

For some reason SQL Server frequently refuses to cleanly drop databases from JDBC, the above logic deals with this by not attempting to recreate the user DB; it just connects to the existing database, creates the table and inserts the data (dropping the table works reliably).

The readUserDB method executes a query against the user table to extract all of the data into a ResultSet, which is used to create our user list for the application.

The deleteDB method drops the table, attempts to drop the DB (again, this frequently fails) and cleans up by closing all of the opened objects. The failure to drop the DB is not an issue on Horizon since that system will eventually close the session and reset everything.

User Class Modifications

The User class has been modified to provide an integer which serves as a unique primary key for each User record:

```
public class User implements Serializable {
    private static Integer idStatic = 0;
    private Integer id; // primary key
```

The key value is set when a User is instantiated using the static idStatic variable which is a User object counter (to insure unique values). If the database already exists when we connect, the existing id values from the table are used. The constructor has been modified to accept an id passed as a parameter; if this value is 0 we assign a new value to the id, otherwise we use the parameter's value (if we weren't doing the workaround for the DB drop failure, we would not do this; we would likely design the table so that SQL Server assigned and managed the ID values).

```
// create new user
// if id is 0, set it based on static value
public User(Integer id, String fName, String lName, String email,
                    Locale locale, ZonedDateTime birthday) {
    if (id == 0) {
        idStatic++;
        this.id = idStatic;
```

```
} else {
    this.id = id;
}
```

In the HappyBirthdayApp class, you can see that the data creation responsibilities have now been delegated to the UserDB class. The logic is otherwise unchanged.

One final change needed, I noticed that as the application has grown in size we may encounter a race condition where the eventlog server takes a bit longer to open the server socket; the processor thread was subsequently unable to open a client connection. I inserted a sleep in the BirthdayCardProcessor constructor. This is a hack (but it works), a clean solution would have the constructor loop until the connection succeeded, or have it disable logging if it was ultimately unable to connect to the server thread.