```java
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.sql.*;
import java.util.HashMap;

/**
 * Project #2
 * @author Tumsa Musa
 * @email tmusa@iastate.edu
 *
 */
public class JDBC_Students {

    //set to true to see how fast this runs
    private static final boolean DEBUG = true;

    //optional database reset
    private static final boolean RESET = true;

    //set to appropriate filepaths
    private static final String person = "C:/Users/tmusa/Desktop/cs363//Person.xml";
    private static final String course = "C:/Users/tmusa/Desktop/cs363//Course.xml";
    private static final String instructor =
    "C:/Users/tmusa/Desktop/cs363//Instructor.xml";
    private static final String student = "C:/Users/tmusa/Desktop/cs363//Student.xml";
    private static final String offering = "C:/Users/tmusa/Desktop/cs363//Offering.xml";
    private static final String enrollment =
    "C:/Users/tmusa/Desktop/cs363//Enrollment.xml";


    public static void main(String[] args) {
        /*
         * Solution is mostly SQL
         */

        try {
            // Load the driver (registers itself)
            Class.forName("com.mysql.jdbc.Driver");
        } catch (Exception E) {
            System.err.println("Unable to load driver.");
            E.printStackTrace();
        }
        try {
            // Connect to the database
            Connection conn;
            String dbUrl = "jdbc:mysql://csdb.cs.iastate.edu:3306/db363tmusa";
            String user = "dbu363tmusa";
            String password = "LxlU5136";
            conn = DriverManager.getConnection(dbUrl, user, password);
            System.out.println("*** Connected to the database ***");

            // Create Statement and ResultSet variables to use throughout the project
            Statement statement = conn.createStatement();
            ResultSet rsTop;

            //resets the database to origin state if flag is set
            long start = System.nanoTime();

            if(RESET) {
                reset(statement);
                statement.executeBatch();
            }

            start = (System.nanoTime() - start);
            if(DEBUG)
                System.out.printf("\nReset the database in %s seconds!\n\n",
                ""+start/1000000000.0);

```

```
67              //query string for top student
68              String topQuery = queryTopStudent();
69
70              start = System.nanoTime();
71
72              statement.executeUpdate(update());
73
74              rsTop = statement.executeQuery(topQuery);
75              processTopStudents(rsTop);
76
77              start = System.nanoTime() - start;
78              if(DEBUG)
79                  System.out.printf("(Update + Query + Processing) Time: %s"
                    ,start/1000000000.0);
80
81              //cleans up
82              statement.close();
83              rsTop.close();
84              conn.close();
85
86          } catch (SQLException e) {
87              System.out.println("SQLException: " + e.getMessage());
88              System.out.println("SQLState: " + e.getSQLState());
89              System.out.println("VendorError: " + e.getErrorCode());
90          }
91      }
92
93
94
95
96      private static void processTopStudents(ResultSet rs) throws SQLException {
97          String out = "";
98          out+= String.format("%-20s| %-20s| %-20s\n", "Student", "Mentor", "GPA");
99          double gpa = 0;
100         while(rs.next()) {
101             gpa = round(rs.getDouble(3));
102             out+= String.format("%-20s| %-20s| %.2f\n", rs.getString(1),
                    rs.getString(2), gpa);
103         }
104
105         if(DEBUG)
106             System.out.println(out);
107
108         try {
109             File f = new File("JDBC_StudentsOutput.txt");
110             FileWriter fw = new FileWriter(f);
111             fw.write(out);
112             fw.close();
113         } catch (IOException e) {
114             e.printStackTrace();
115         }
116
117
118     }
119
120     public static double round(double x) {
121         return Math.round(x*100.0)/100.0;
122     }
123
124     private static void reset(Statement s) throws SQLException {
125         dropTables(s);
126         createTables(s);
127         loadTables(s);
128     }
129
130     /*
131      * The limit 4,1 takes the first record after the 4th row.
132      * Guarantees all returned
133      * records have a GPA >= the 5th best GPA
```

```
134         */
135        private static String queryTopStudent() {
136            return "select q.stdName, w.Name, q.GPA from (\r\n" +
137                    "select  p.Name as stdName, s.MentorID as mentorID, s.GPA as GPA\r\n" +
138                    "from \r\n" +
139                    "Student s , Person p \r\n" +
140                    "where p.ID = s.StudentID \r\n" +
141                    "and s.GPA >=\r\n" +
142                    "(select p.GPA \r\n" +
143                    "from Student p\r\n" +
144                    "order by p.GPA desc\r\n" +
145                    "limit 4, 1) ) as q , Person w\r\n" +
146                    "where w.ID = q.mentorID\r\n" +
147                    "order by q.GPA desc ";
148        }
149
150        /*
151         * It's a bit of a mess but it works.
152         * The case statement coverts the letter grades into number.
153         * runs much faster than the best java solution I could come up with.
154         * probably not best sql solution
155         *
156         * It updates the Student table's gpa, credit hours, and
157         * classificaiton in one statement
158         */
159        private static String update() {
160            return "update Student q "
161
162                    //GPA update
163                    + "set q.GPA = \r\n" +
164                    "(select newGPA from \r\n" +
165                    "( select StudentID, (SumGrade + OldGrade)/(CreditHours +3*CountGrade)
                    as newGPA from\r\n" +
166                    "( select StudentID , sum(NumGrade) as SumGrade,  count(NumGrade) as
                    CountGrade   from \r\n" +
167                    "( select StudentID , \r\n" +
168                    "3* (CASE when Grade = 'A' then 4 \r\n" +
169                    "   when Grade = 'A-' then 3.66 \r\n" +
170                    "   when Grade = 'B+' then 3.33 \r\n" +
171                    "   when Grade = 'B' then 3.00 \r\n" +
172                    "   when Grade = 'B-' then 2.66 \r\n" +
173                    "   when Grade = 'C+' then 2.33 \r\n" +
174                    "   when Grade = 'C' then 2.00 \r\n" +
175                    "   when Grade = 'C-' then 1.66 \r\n" +
176                    "   when Grade = 'D+' then 1.33 \r\n" +
177                    "   when Grade = 'D' then 1.00 \r\n" +
178                    "   else 0 end) as NumGrade from \r\n" +
179                    "( select e.StudentID , e.Grade from Enrollment e) \r\n" +
180                    "as b ) \r\n" +
181                    "as c group by StudentID) \r\n" +
182                    "as d,  (select s.StudentID as StdID, s.GPA*s.CreditHours  as OldGrade,
                    CreditHours   from Student \r\n" +
183                    "s) as f where StudentID = StdID ) as p where p.StudentID =
                    q.StudentID)"+
184
185                    //credit hours update
186                    ", q.CreditHours = q.CreditHours + 3 *(\r\n" +
187                    "select count(e.StudentID) from Enrollment e\r\n" +
188                    "where e.StudentID = q.StudentID)" +
189
190                    //classification update
191                    ", q.Classification = CASE\r\n" +
192                    "   when q.CreditHours < 30  then 'Freshman'\r\n" +
193                    "    when q.CreditHours >29 \r\n" +
194                    "       and q.CreditHours < 60  then 'Sophomore'\r\n" +
195                    "   when q.CreditHours >59 \r\n" +
196                    "       and q.CreditHours <90  then 'Junior'\r\n" +
197                    "   else 'Senior'\r\n" +
198                    "end";
```

```
199         }
200
201         private static void loadTables(Statement s) throws SQLException {
202             s.addBatch(String.format("load xml local infile '%s' " +
203                     "into table Person " +
204                     "rows identified by '<Person>' ", person));
205
206             s.addBatch(String.format("load xml local infile '%s' " +
207                     "into table Course " +
208                     "rows identified by '<Course>' " , course));
209
210             s.addBatch(String.format("load xml local infile '%s' " +
211                     "into table Instructor " +
212                     "rows identified by '<Instructor>'  " , instructor));
213
214             s.addBatch(String.format("load xml local infile '%s' " +
215                     "into table Student " +
216                     "rows identified by '<Student>'  ", student)) ;
217
218             s.addBatch(String.format("load xml local infile '%s' " +
219                     "into table Offering " +
220                     "rows identified by '<Offering>' ", offering));
221
222             s.addBatch(String.format("load xml local infile '%s' " +
223                     "into table Enrollment " +
224                     "rows identified by '<Enrollment>' ", enrollment));
225
226         }
227         private static void createTables(Statement s) throws SQLException {
228             s.addBatch("create table Person ( "+
229                     "Name char (20), " +
230                     "ID char (9) not null, " +
231                     "Address char (30), " +
232                     "DOB date, " +
233                     "Primary key (ID)) ");
234
235             s.addBatch("create table Instructor ( " +
236                     "InstructorID char(9) not null references Person(ID), " +
237                     "Rank char(12), " +
238                     "Salary int, " +
239                     "primary key (InstructorID) " +
240                     ") ");
241
242             s.addBatch(" create table Student ( " +
243                     "StudentID char(9) not null references Person(ID), " +
244                     "Classification char(10), " +
245                     "GPA double, " +
246                     "MentorID char(9) references Instructor(InstructorID) , " +
247                     "CreditHours int, " +
248                     "primary key (StudentID) " +
249                     ") " );
250
251             s.addBatch("create table Course ( " +
252                     "CourseCode char(6) not null, " +
253                     "CourseName char(50), " +
254                     "PreReq char(6), " +
255                     "primary key (CourseCode, PreReq) " +
256                     ") ");
257
258             s.addBatch( "create table Offering ( " +
259                     "CourseCode char(6) not null, " +
260                     "SectionNo int not null, " +
261                     "InstructorID char(9) not null references Instructor(InstructorID) , " +
262                     "primary key (CourseCode, SectionNo) " +
263                     ") ");
264             s.addBatch("create table Enrollment ( " +
265                     "CourseCode char(6) NOT NULL, " +
266                     "SectionNo int NOT NULL, " +
267                     "StudentID char(9) NOT NULL references Student, " +
```

```
268                 "Grade char(4) NOT NULL, " +
269                 "primary key (CourseCode, StudentID), " +
270                 "foreign key (CourseCode, SectionNo) references Offering(CourseCode,
                    SectionNo)) " );
271         ;
272     }
273     private static void dropTables(Statement s) throws SQLException {
274         s.addBatch("drop table Enrollment ");
275         s.addBatch("drop table Offering ");
276         s.addBatch("drop table Course ");
277         s.addBatch("drop table Student ");
278         s.addBatch("drop table Instructor ");
279         s.addBatch("drop table Person ");
280     }
281 }
282
283
```