```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 P3 {
    private static HashMap<String, Double> map;

    //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
68              //query string for top student
69              String topQuery = queryTopStudent();
70
71              start = System.nanoTime();
72
73              update(statement);
74              statement.executeBatch();
75
76              rsTop = statement.executeQuery(topQuery);
77              processTopStudents(rsTop);
78
79              start = System.nanoTime() - start;
80              if(DEBUG)
81                  System.out.printf("(Update + Query + Processing) Time: %s"
                    ,start/1000000000.0);
82
83              //cleans up
84              statement.close();
85              rsTop.close();
86              conn.close();
87
88          } catch (SQLException e) {
89              System.out.println("SQLException: " + e.getMessage());
90              System.out.println("SQLState: " + e.getSQLState());
91              System.out.println("VendorError: " + e.getErrorCode());
92          }
93      }
94
95
96
97
98      private static void processTopStudents(ResultSet rs) throws SQLException {
99          String out = "";
100         out+= String.format("%-10s| %-10s| %-10s\n", "StudentID", "MentorID", "GPA");
101         double gpa = 0;
102         while(rs.next()) {
103             gpa = round(rs.getDouble(3));
104             out+= String.format("%-10s| %-10s| %.2f\n", rs.getString(1),
                    rs.getString(2), gpa);
105         }
106
107         if(DEBUG)
108             System.out.println(out);
109
110         try {
111             File f = new File("P3Output.txt");
112             FileWriter fw = new FileWriter(f);
113             fw.write(out);
114             fw.close();
115         } catch (IOException e) {
116             e.printStackTrace();
117         }
118
119
120     }
121
122     public static double round(double x) {
123         return Math.round(x*100.0)/100.0;
124     }
125
126     private static void reset(Statement s) throws SQLException {
127         dropTables(s);
128         createTables(s);
129         loadTables(s);
130     }
131     private static void update(Statement s) throws SQLException {
132         s.addBatch(updateGPAs());
133         s.addBatch(updateCreditHours());
```

```
134             s.addBatch(updateClassification());
135         }
136 
137         /*
138          * The limit 4,1 takes the first record after the 4th row.
139          * Guarantees all returned
140          * records have a GPA >= the 5th best GPA
141          */
142         private static String queryTopStudent() {
143             return "select distinct s.StudentID, s.MentorID, s.GPA from \r\n" +
144                     "Student s where s.GPA >=\r\n" +
145                     "(select p.GPA \r\n" +
146                     "from Student p\r\n" +
147                     "order by p.GPA desc\r\n" +
148                     "limit 4, 1) order by s.GPA desc";
149         }
150 
151         private static String updateCreditHours() {
152             return "update Student s \r\n" +
153                     "set s.CreditHours = s.CreditHours + 3 *(\r\n" +
154                     "select count(e.StudentID) from Enrollment e\r\n" +
155                     "where e.StudentID = s.StudentID)";
156         }
157 
158         private static String updateClassification() {
159             return "update Student s \r\n" +
160                     "set s.Classification = CASE\r\n" +
161                     "   when s.CreditHours < 30  then 'Freshman'\r\n" +
162                     "    when s.CreditHours >29 \r\n" +
163                     "       and s.CreditHours < 60  then 'Sophomore'\r\n" +
164                     "   when s.CreditHours >59 \r\n" +
165                     "       and s.CreditHours <90  then 'Junior'\r\n" +
166                     "   else 'Senior'\r\n" +
167                     "end";
168         }
169 
170         /*
171          * It's a bit of a mess but it works.
172          * The case statement coverts the letter grades into number.
173          * runs much faster than the best java solution I could come up with.
174          * probably not best sql solution
175          */
176         private static String updateGPAs() {
177             return "update Student q set q.GPA = \r\n" +
178                     "(select newGPA from \r\n" +
179                     "( select StudentID, (SumGrade + OldGrade)/(CreditHours +3*CountGrade)
                        as newGPA from\r\n" +
180                     "( select StudentID , sum(NumGrade) as SumGrade,  count(NumGrade) as
                        CountGrade   from \r\n" +
181                     "( select StudentID , \r\n" +
182                     "3* (CASE when Grade = 'A' then 4 \r\n" +
183                     "   when Grade = 'A-' then 3.66 \r\n" +
184                     "   when Grade = 'B+' then 3.33 \r\n" +
185                     "   when Grade = 'B' then 3.00 \r\n" +
186                     "   when Grade = 'B-' then 2.66 \r\n" +
187                     "   when Grade = 'C+' then 2.33 \r\n" +
188                     "   when Grade = 'C' then 2.00 \r\n" +
189                     "   when Grade = 'C-' then 1.66 \r\n" +
190                     "   when Grade = 'D+' then 1.33 \r\n" +
191                     "   when Grade = 'D' then 1.00 \r\n" +
192                     "   else 0 end) as NumGrade from \r\n" +
193                     "( select e.StudentID , e.Grade from Enrollment e) \r\n" +
194                     "as b ) \r\n" +
195                     "as c group by StudentID) \r\n" +
196                     "as d,  (select s.StudentID as StdID, s.GPA*s.CreditHours  as OldGrade,
                        CreditHours   from Student \r\n" +
197                     "s) as f where StudentID = StdID ) as p where p.StudentID =
                        q.StudentID)";
198         }
```

```
199
200     private static void loadTables(Statement s) throws SQLException {
201         s.addBatch(String.format("load xml local infile '%s' " +
202                 "into table Person " +
203                 "rows identified by '<Person>' ", person));
204
205         s.addBatch(String.format("load xml local infile '%s' " +
206                 "into table Course " +
207                 "rows identified by '<Course>' " , course));
208
209         s.addBatch(String.format("load xml local infile '%s' " +
210                 "into table Instructor " +
211                 "rows identified by '<Instructor>'  " , instructor));
212
213         s.addBatch(String.format("load xml local infile '%s' " +
214                 "into table Student " +
215                 "rows identified by '<Student>'  ", student)) ;
216
217         s.addBatch(String.format("load xml local infile '%s' " +
218                 "into table Offering " +
219                 "rows identified by '<Offering>' ", offering));
220
221         s.addBatch(String.format("load xml local infile '%s' " +
222                 "into table Enrollment " +
223                 "rows identified by '<Enrollment>' ", enrollment));
224
225     }
226     private static void createTables(Statement s) throws SQLException {
227         s.addBatch("create table Person ( "+
228                 "Name char (20), " +
229                 "ID char (9) not null, " +
230                 "Address char (30), " +
231                 "DOB date, " +
232                 "Primary key (ID)) ");
233
234         s.addBatch("create table Instructor ( " +
235                 "InstructorID char(9) not null references Person(ID), " +
236                 "Rank char(12), " +
237                 "Salary int, " +
238                 "primary key (InstructorID) " +
239                 ") ");
240         s.addBatch(" create table Student ( " +
241                 "StudentID char(9) not null references Person(ID), " +
242                 "Classification char(10), " +
243                 "GPA double, " +
244                 "MentorID char(9) references Instructor(InstructorID) , " +
245                 "CreditHours int, " +
246                 "primary key (StudentID) " +
247                 ") " );
248         s.addBatch("create table Course ( " +
249                 "CourseCode char(6) not null, " +
250                 "CourseName char(50), " +
251                 "PreReq char(6), " +
252                 "primary key (CourseCode, PreReq) " +
253                 ") ");
254         s.addBatch( "create table Offering ( " +
255                 "CourseCode char(6) not null, " +
256                 "SectionNo int not null, " +
257                 "InstructorID char(9) not null references Instructor(InstructorID) , " +
258                 "primary key (CourseCode, SectionNo) " +
259                 ") ");
260         s.addBatch("create table Enrollment ( " +
261                 "CourseCode char(6) NOT NULL, " +
262                 "SectionNo int NOT NULL, " +
263                 "StudentID char(9) NOT NULL references Student, " +
264                 "Grade char(4) NOT NULL, " +
265                 "primary key (CourseCode, StudentID), " +
266                 "foreign key (CourseCode, SectionNo) references Offering(CourseCode,
                    SectionNo)) " );
```

```
267             ;
268         }
269         private static void dropTables(Statement s) throws SQLException {
270             s.addBatch("drop table Enrollment ");
271             s.addBatch("drop table Offering ");
272             s.addBatch("drop table Course ");
273             s.addBatch("drop table Student ");
274             s.addBatch("drop table Instructor ");
275             s.addBatch("drop table Person ");
276         }
277     }
278
279
```