```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 , James Eenhuis

 */
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);

            //query string for top student
```

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

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

```
                "   else 'Senior'\r\n" +
                "end";
    }

    private static void loadTables(Statement s) throws SQLException {
        s.addBatch(String.format("load xml local infile '%s' " +
                "into table Person " +
                "rows identified by '<Person>' ", person));

        s.addBatch(String.format("load xml local infile '%s' " +
                "into table Course " +
                "rows identified by '<Course>' " , course));

        s.addBatch(String.format("load xml local infile '%s' " +
                "into table Instructor " +
                "rows identified by '<Instructor>'  " , instructor));

        s.addBatch(String.format("load xml local infile '%s' " +
                "into table Student " +
                "rows identified by '<Student>'  ", student)) ;

        s.addBatch(String.format("load xml local infile '%s' " +
                "into table Offering " +
                "rows identified by '<Offering>' ", offering));

        s.addBatch(String.format("load xml local infile '%s' " +
                "into table Enrollment " +
                "rows identified by '<Enrollment>' ", enrollment));

    }
    private static void createTables(Statement s) throws SQLException {
        s.addBatch("create table Person ( "+
                "Name char (20), " +
                "ID char (9) not null, " +
                "Address char (30), " +
                "DOB date, " +
                "Primary key (ID)) ");

        s.addBatch("create table Instructor ( " +
                "InstructorID char(9) not null references Person(ID), " +
                "Rank char(12), " +
                "Salary int, " +
                "primary key (InstructorID) " +
                ") ");

        s.addBatch(" create table Student ( " +
                "StudentID char(9) not null references Person(ID), " +
                "Classification char(10), " +
                "GPA double, " +
                "MentorID char(9) references Instructor(InstructorID) , " +
                "CreditHours int, " +
                "primary key (StudentID) " +
                ") " );

        s.addBatch("create table Course ( " +
                "CourseCode char(6) not null, " +
                "CourseName char(50), " +
                "PreReq char(6), " +
                "primary key (CourseCode, PreReq) " +
                ") ");

        s.addBatch( "create table Offering ( " +
                "CourseCode char(6) not null, " +
                "SectionNo int not null, " +
                "InstructorID char(9) not null references Instructor(InstructorID) , " +
                "primary key (CourseCode, SectionNo) " +
                ") ");
        s.addBatch("create table Enrollment ( " +
                "CourseCode char(6) NOT NULL, " +
```

```
                "SectionNo int NOT NULL, " +
                "StudentID char(9) NOT NULL references Student, " +
                "Grade char(4) NOT NULL, " +
                "primary key (CourseCode, StudentID), " +
                "foreign key (CourseCode, SectionNo) references Offering(CourseCode,
                SectionNo)) " );
        ;
    }
    private static void dropTables(Statement s) throws SQLException {
        s.addBatch("drop table Enrollment ");
        s.addBatch("drop table Offering ");
        s.addBatch("drop table Course ");
        s.addBatch("drop table Student ");
        s.addBatch("drop table Instructor ");
        s.addBatch("drop table Person ");
    }
}
```