Class Registration – Final Paper
GE5: Daniel Boos, Nick Starr, Donathan Tuck

General Description:

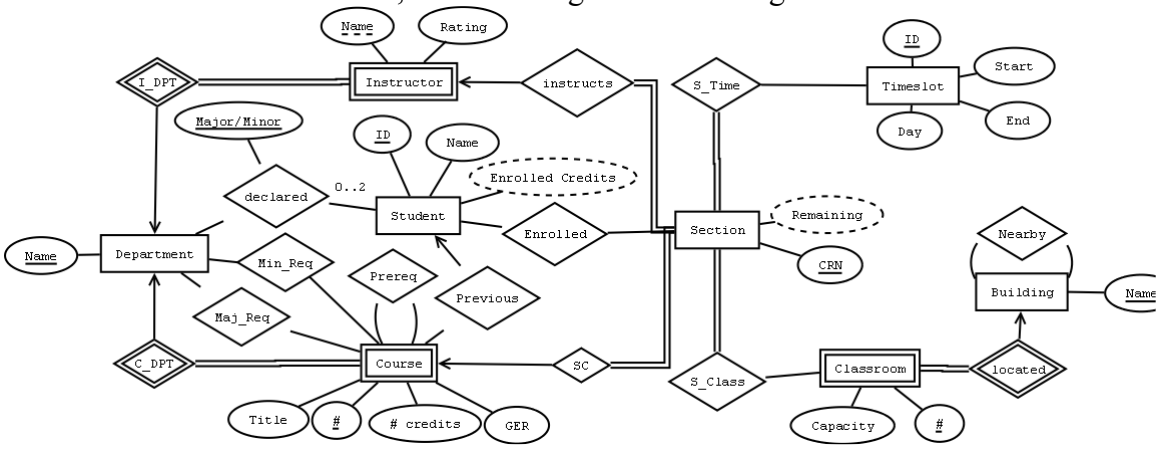
When registering for classes at the college of William and Mary, coming up with a schedule can be a pretty daunting task. What you may or may not take the next semester can severely affect what you end up graduating with. Therefore to make this process easier, we've designed a database that contains all of the necessary information required to create a schedule for class registration. This includes teacher's ratings from RateMyProfessor.com, major requirements, minor requirements, and prerequisites from the departments' websites, and everything related to the actual classes (courses and sections) themselves.

With our application, the student can create an account in the database by logging in their studentID. Then the student can choose to declare/edit their majors or minor, add to the database previous classes they have already taken, browse all the classes by their preference, and ultimately register for classes with the section's CRN. Browsing classes in particular allows the student to search classes by Department, Course Number, if it is near a specific building, and letting the program find what classes the student should take before graduation. This is calculated by finding all the GERs the student has not taken already and all the required classes left for the student's declarations, and removing classes they have already taken and classes with unfulfilled prerequisites. Also for any query the student makes, it can include ordering the classes by the teacher's RateMyProfessor rating. When signing up for classes, the program checks the database for any time-conflicts as well as any remaining seats.

With this program, students can easily manage and register their classes for the next semester, removing most of the worries of class registration.

Database Design:

As seen on our website, the following is our E/R diagram.



From this we derived our entire ideal relational schema:

Course

Department | Title | Number | Number of Credits

GER
Department | Course Number | GER Attribute

Required_Major
Department | Course Number

Required_Minor
Department | Course Number

Prerequisites
Department | Course Number | Prerequisites

Student
ID | Name | Enrolled Credits

Declared
Student ID | Major/Minor | Department

Enrolled
Student ID | CRN

Previous
Student ID | CRN

Instructor
Department | Name | Rating

Section
Department | Course Number | CRN | Remaining | Instructor

Sect_Time
CRN | Time ID

Timeslot
ID | Start | End | Day

Sect_Room
CRN | Room Number

Classroom
Building Name | Room Number | Capacity

Nearby
Building #1 | Building #2

These are all in BCNF due to the fact that the tables cannot be decomposed into smaller tables without removing important information.

Database Implementation:

Unfortunately, importing the information from Banner into the database required us to change the design we had planned. In an ideal world with more time, we might have been able to work around the hurdles of the Banner data but I am satisfied with the less normalized version of the design. To do this we used a Ruby script to parse an HTML of all the classes for this semester on Banner and on RateMyProfessor and convert them to csv files. Unfortunately, we couldn't simply import this csv into our tables and had to use Excel to have the data match our schema and export them as smaller csv files, which were then copied into our database through postresql as seen in makefile.sql.

```
create table Course
        (Department varchar(20),
        Course_Number varchar(5),
        Title varchar(100),
        Credits numeric(2,1),
        primary key (Department, Course_Number));
create table GER
        (Department varchar(20),
        Course_Number varchar(5),
        GER_Attribute varchar(200),
        primary key (Department, Course_Number));
create table Required_Major
        (Department varchar(20),
        Course_Number varchar(5),
        primary key (Department, Course_Number));
create table Required_Minor
        (Department varchar(20),
        Course_Number varchar(5),
        primary key (Department, Course_Number));
create table Prerequisites
        (Department varchar(20),
        Course_Number varchar(5),
        Prerequisite varchar(5));
create table Student
        (Student_ID integer,
        Name varchar(20),
        Credits_Enrolled integer,
        primary key (Student_ID));
create table Declared
        (Student_ID integer,
        Declaration char(5),
        Department varchar(20));
```

```
create table Section
        (Department varchar(20),
        Course_Number varchar(5),
        CRN integer,
        Remaining integer,
        Instructor varchar(50),
        days varchar(6),
        starttime varchar(12),
        endtime varchar(12),
        building varchar(12),
        room varchar(12),
        primary key (CRN));
create table Enrolled
        (Student_ID integer,
        CRN integer);
create table Previous
        (Student_ID integer,
        Department varchar(20),
        Course_Number varchar(5),
        GER_Attribute varchar(200),
        primary key (Student_ID, Department, Course_Number));
create table Instructor
        (Instructor varchar(50),
        Rating float(1),
        primary key (Instructor));
create table Classroom
        (Building_Name varchar(15),
        Room varchar(15),
        Capacity integer,
        primary key (Building_Name, Room));
create table Nearby
        (Building1 varchar(15),
        Building2 varchar(15));

\copy course from 'course.csv' with csv
\copy ger from 'ger.csv' with csv
\copy required_major from 'required_maj.csv' with csv
\copy required_minor from 'required_min.csv' with csv
\copy prerequisites from 'prereq.csv' with csv
\copy section from 'section4.csv' with csv
\copy instructor from 'instructor4.csv' with csv
\copy classroom from 'classroom.csv' with csv
\copy nearby from 'nearby.csv' with csv
```

Required majors, required minors, and prerequisites were manually created for the database and only relate to computer science department because it allows you to see how

the database works. Also, nearby was manually created and obviously based on a subjective view on how buildings were ‘nearby’ each other.

Application Interface:

The interface generally changes values regarding each student while reading information regarding classes. When the student logs in, they type in their studentID. If the studentID is not in the database, it asks for the student’s name and it adds the information to the student relation.

For changing the student’s declarations, the database checks to see what declarations the student has already made, and gives them the options based on that information so that the student can only double major, major and minor, major, minor (for freshmen/sophomores), or simply be undeclared. Whenever a student makes a declaration, it is added to the table, declared, the studentID, what they declared, and in what department.

For adding previous classes, the database displays a list what the student already included in their previous classes. The student must then type the department and number of the course and may include a GER attribute. This is then added to the previous table with the student id. If no GER attribute was included, the GER attribute in the table is empty.

When browsing classes, the application only filters information through one giant query. If the student selects submit, all the classes from section inner joined with several other tables is shown. If the student selects department, the query adds a where clause filtering by department. Similarly, it does the same for course number as well. For near building, it gets the original query that would have been for submit and then looks at the nearby table for the building given. Then it only prints if there is a match on building2 to the sections building. Teacher rating was done by simply adding an order by cause to the end of the query string. Finally, finding the classes left to graduate was calculated by first looking up the classes for the student’s majors/minor, subtracting all the gers that the student has taken (using pattern matching) from GERs, and union the two together. Using that, we removed all the classes with unfulfilled prerequisites and the student has already taken from that relation using the tables prerequisites and previous. This would get us the recommended classes for graduation.

Finally, when the student registers for a class, it checks for time conflicts by comparing the times of the already enrolled class with the new class. It also doesn’t add if the number of remaining seats for a section is zero. We can use this because whenever a student registers for a class, the number of remaining seats for the section is decreased by one.

Notables:

The most notable features of this product are finding the student’s recommended classes for graduation, including the ratemyprofessor ratings, and using the nearby function. While the ratings and the nearby functions were not hard to implement, it would definitely make the registration process a lot easier. I do know people that always take

these factors seriously when considering which courses to take. Therefore, I feel most people would appreciate those functional aspects of the database.

However, the most notable feature and hard to implement is finding the recommended classes for gradation. Every semester, most students worry about what classes they will need to take before graduation and upon supplying their information, they can easily find out. Not only can they find out, they can find out but they can also find out from that selection which ones are nearby where and which teachers are preferred. Therefore, I believe that these functions can greatly help each student pick out a schedule for every semester.

Appendix A

```
import java.sql.*;
import java.util.ArrayList;
import java.awt.BorderLayout;
import java.awt.CardLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.BoxLayout;
import javax.swing.JCheckBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

/**
 * The purpose of the ClassRegistration class is to access the registration database. It contains methods necessary for the function to
 * query and manipulate the database. It also contains methods that assist with problems such as time overlap.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
public class ClassRegistration {
```

```
	Class driverClass = loadDriver( "org.postgresql.Driver" );

	static Connection con =
connectURL("jdbc:postgresql:g5_registration?user=dpboos&password=password");

	/**
	 * This function checks to determine if an id is in the database.
	 * @param id
	 * @return
	 */
	public boolean inDatabase(int id){
		ResultSet students = execQuery( con, "SELECT student_id FROM
student;" );
	try {
			while(students.next()){
				if(id == students.getInt(1)){
					return true;
				}
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}
		return false;

	}

	/**
	 * A function to add a student to the database.
	 * @param id student id
	 * @param name student name
	 */
	public void addToDatabase(int id, String name){
		execQuery( con, "INSERT into student values('" + id + "', '" + name + "',
'0');");
	}

	/**
	 * Returns the number of majors that the student has enroled in.
	 * @param id student id
	 * @return
	 */
	public int getNumMaj(int id){
		ResultSet smajors = execQuery( con, "SELECT count(*) FROM declared
WHERE student_id = "
				+ id + " AND declaration = 'major';");
		int nummajor = 0;
```

```
		try {
			smajors.next();
			return smajors.getInt(1);
		} catch (SQLException e) {
			e.printStackTrace();
			return -1;
		}
	}

	/**
	 * Returns a string containing the students majors.
	 * @param id student id
	 * @return
	 */
	public String getMajList(int id){
		String tempString = new String();
		ResultSet dpts = execQuery( con, "SELECT DISTINCT department
FROM declared WHERE student_id="
				+ id + " AND declaration = 'major';");
		try {
			while(dpts.next()){
				tempString += (dpts.getString(1) + "\n");
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}return tempString;
	}

	/**
	 * Returns an arraylist containing the students majors.
	 * @param id student id
	 * @return
	 */
	public ArrayList<String> getMajListArray(int id){
		ArrayList<String> tempArray = new ArrayList<String>();
		ResultSet dpts = execQuery( con, "SELECT DISTINCT department
FROM declared WHERE student_id="
				+ id + " AND declaration = 'major';");
		try {
			while(dpts.next()){
				tempArray.add(dpts.getString(1));
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}
		return tempArray;
```

```
	}

	/**
	 * Returns an array list of the studensts minors.
	 * @param id student id
	 * @return
	 */
	public ArrayList<String> getMinListArray(int id){
		ArrayList<String> tempArray = new ArrayList<String>();
		ResultSet dpts = execQuery( con, "SELECT DISTINCT department
FROM declared WHERE student_id="
				+ id + " AND declaration = 'minor';");
		try {
			while(dpts.next()){
				tempArray.add(dpts.getString(1));
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}
		return tempArray;
	}

	/**
	 * Adds a major to the declared relation for a specific id.
	 * @param department
	 * @param id student id
	 */
	public void addMajor(String department, int id){
		execQuery( con, "INSERT into declared values('" + id + "', 'major','" +
department + "');");
	}

	public void removeMajor(String department, int id){
		execQuery( con, "DELETE from declared where department = '" +
department + "' AND declaration = 'major';");
	}

	public void removeMinor(String department, int id){
		execQuery( con, "DELETE from declared where department = '" +
department + "' AND declaration = 'minor';");
	}

	/**
	 * Returns the number of minors a student has declared.
	 * @param id student id
	 * @return
```

```
	 */
	public int getNumMin(int id){
		ResultSet sminors = execQuery( con, "SELECT count(*) FROM declared
WHERE student_id = "
				+ id + " AND declaration = 'minor';");
		try {
			sminors.next();
			return sminors.getInt(1);
		} catch (SQLException e) {
			e.printStackTrace();
			return -1;
		}
	}

	/**
	 * Returns a string containing the students minors.
	 * @param id student id
	 * @return
	 */
	public String getMinList(int id){
		String tempString = new String();
		ResultSet dpts = execQuery( con, "SELECT DISTINCT department
FROM declared WHERE student_id="
				+ id + " AND declaration = 'minor';");
		try {
			while(dpts.next()){
				tempString += (dpts.getString(1) + "\n");
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}return tempString;
	}

	/**
	 * Adds a minor to the declared relation for a specific id.
	 * @param department
	 * @param id student id
	 */
	public void addMinor(String department, int id){
		execQuery( con, "INSERT into declared values('" + id + "', 'minor','" +
department + "');");
	}

	/**
	 * Returns a list of all departments. Used to assist the user when choosing a major
or minor.
```

```
	 * @return
	 */
	public String getDepList(){
		String tempString = new String();
		ResultSet dpts = execQuery( con, "SELECT DISTINCT department
FROM course;");
		try {
			while(dpts.next()){
				tempString += (dpts.getString(1) + "\n");
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}return tempString;
	}

	/**
	 * This function checks a given department against the database.
	 * @param department
	 * @return
	 */
	public Boolean isValidDep(String department){
		ResultSet dpts = execQuery( con, "SELECT DISTINCT department
FROM course;");
		try {
			while(dpts.next()){
				if (department.equals(dpts.getString(1))){
					return true;
				}
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}return false;
	}

	/**
	 * Determines if the user input for crn is correct.
	 * @param selection
	 * @return
	 */
	public Boolean isValidCRN(String selection){
		ResultSet dpts = execQuery( con, "SELECT DISTINCT crn FROM
section;");
		try {
			while(dpts.next()){
				if (selection.equals(dpts.getString(1))){
					return true;
```

```
                        }
                    }
                } catch (SQLException e) {
                    e.printStackTrace();
                }return false;
        }

        /**
         * Determins if the users choice of building is correct
         * @param building
         * @return
         */
        public Boolean isValidBuilding(String building){
                ResultSet dpts = execQuery( con, "SELECT DISTINCT building1 FROM
nearby ORDER BY building1 asc;");
                try {
                    while(dpts.next()){
                        if (building.equals(dpts.getString(1))){
                            return true;
                        }
                    }
                } catch (SQLException e) {
                    e.printStackTrace();
                }return false;
        }

        /**
         * Returns a string containing all previous classes the user has taken
         * @param id
         * @return
         */
        public String getPrevClasses(int id){
                String tempString = String.format("%-45s", "ClassName") +
String.format("%5s","Class") + String.format("%10s","#") + "\n";
                tempString += "-----------------------------------------------\n";
                ResultSet dpts = execQuery(con, "SELECT department, course_number,
ger_attribute FROM previous WHERE student_id = '" + id + "';");
                try {
                    while(dpts.next()){
                        if (isValidClass(new String[]{dpts.getString(1),
dpts.getString(2)})){
                            ResultSet className = execQuery(con, "SELECT title
FROM course WHERE department = '" + dpts.getString(1)
                                    + "' and course_number = '" +
dpts.getString(2) +"';");
                            className.next();
```

```
                                        tempString += String.format("%-
40s",className.getString(1)) + " " + String.format("%5s",dpts.getString(1))
                                        + " " + String.format("%5s",dpts.getString(2)) + " " +
dpts.getString(3) + "\n\n";
                                        } else {
                                                tempString += String.format("%-40s","") + " " +
String.format("%5s",dpts.getString(1))
                                        + " " + String.format("%5s",dpts.getString(2)) + " " +
dpts.getString(3) + "\n\n";
                                        }

                                }
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }return tempString;
                }

                /**
                 * Returns a string containing all classes that the student has registered for on this
semester.
                 * @param id
                 * @return
                 */
                public String getRegClasses(int id){
                        String tempString = new String();
                        ResultSet dpts = execQuery(con, "SELECT crn FROM enrolled WHERE
student_id = '" + id + "';");
                        try {
                                while(dpts.next()){
                                        ResultSet classInfo = execQuery(con, "SELECT
department,course_number,instructor,days,starttime,endtime FROM section WHERE crn
= '"
                                                        + dpts.getString(1) + "' ;");
                                        classInfo.next();
                                        ResultSet className = execQuery(con, "SELECT title
FROM course WHERE department = '"
                                                        + classInfo.getString(1) + "' and
course_number = '" + classInfo.getString(2) +"';");
                                        className.next();
                                        tempString += dpts.getString(1) + " " +
className.getString(1) + " " + classInfo.getString(1) + " " + classInfo.getString(2)
                                        + " " + classInfo.getString(3) + " " + classInfo.getString(4)
+ " " + classInfo.getString(5) + " " + classInfo.getString(6) +"\n\n";
                                }
                        } catch (SQLException e) {
                                e.printStackTrace();
```

```
		}return tempString;
	}

	/**
	 * Adds a class to the list of classes the student has previously enrolled in.
	 * @param info
	 * @param id
	 */
	public void addClass(String[] info, int id){
		execQuery(con, "INSERT into previous values('" + id + "', '" + info[0] +
"','" + info[1] + "','" + info[2] + "');");
	}

	/**
	 * Removes a class from the list of classes the student has previously taken.
	 * @param info
	 * @param id
	 */
	public void removeClass(String[] info, int id){
		execQuery( con, "DELETE from previous where department = '" + info[0]
+ "' AND course_number = '" + info[1] + "' AND student_id = '" + id + "';");
	}

	/**
	 * Used to check if the users class input is valid.
	 * @param info
	 * @return
	 */
	public boolean isValidClass(String[] info){
		ResultSet dpts = execQuery(con, "SELECT department, course_number
FROM course;");
		try {
			while(dpts.next()){
				if (info[0].equals(dpts.getString(1)) &&
info[1].equals(dpts.getString(2))){
					return true;
				}
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}return false;
	}

	/**
	 * Takes a custom query built by the program and converts the result to a string
showing the results.
```

```
	 * @param query Built by the BrowseClass class
	 * @return
	 */
	public String classQuery(String query){
		String tempString = new String();
		tempString +=  String.format("%-40s", "Class Name") +
String.format("%-40s", "Professor")
		+ " " + String.format("%-10s", "#") + " " + String.format("%-7s", "Class")
+ " " + String.format("%-7s", "CRN")
		+ " " + String.format("%-10s", "Seats") + String.format("%-12s", "Day")
+ String.format("%-17s", "Begins")
		+ String.format("%-12s", "Ends")+ String.format("%-12s", "Building") +
String.format("%-7s", "Room") + String.format("%-10s", "Credits") + String.format("%-
7s", "Prof Rating") +"\n";
		tempString += "----------------------------------------------------------------------
------------" +
					"----------------------------------------------------------------------
\n";
		ResultSet dpts = execQuery(con, query + ";");
		try {
			while(dpts.next()){
				tempString +=   String.format("%-40s", dpts.getString(11))
+ String.format("%-40s", dpts.getString(1))
				+ " " + String.format("%-5s",dpts.getString(2)) + " " +
String.format("%-7s", dpts.getString(3)) + " " +
				String.format("%-7s", dpts.getString(4)) + " " +
String.format("%-7s", dpts.getString(5))
				+ String.format("%-7s", dpts.getString(6)) +
String.format("%-7s", dpts.getString(7)) + String.format("%-15s", dpts.getString(8)) +
				String.format("%-10s", dpts.getString(9)) +
String.format("%-7s", dpts.getString(10)) + String.format("%-7s", dpts.getString(12)) +
				String.format("%-7s", dpts.getString(13)) + "\n\n";
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}return tempString;
	}

	/**
	 * Registers a user in the class in the enrolled relation
	 * @param id
	 * @param crsn
	 * @return
	 */
	public int registerClass(String id, String crsn){
			if (timeconflicts(id, crsn)){
```

```
                        return 0;
                }
                else if (!remainingseats(crsn)){
                        return 1;
                }
                else{
                        execQuery( con, "INSERT into enrolled values('"  + id + "', '"
+crsn + "');");
                        execQuery( con, "UPDATE section set remaining = remaining -
1 where crn = '" + crsn + "';");
                        return 2;
                }
        }

        /**
         * Removes a students class from the enrolled database
         * @param id
         * @param crsn
         */
        public void unregisterClass(int id, String crsn){
                execQuery( con, "DELETE from enrolled where student_id = '" + id + "'
AND crn = '" + crsn + "';");
        }

        /**
         * Called when program is ran. Calls GUIManager to build the GUI.
         * @param args
         */
         public static void main(String[] args) {
      /* Use an appropriate Look and Feel */
      try {

//UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFee
l");
         UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
      } catch (UnsupportedLookAndFeelException ex) {
         ex.printStackTrace();
      } catch (IllegalAccessException ex) {
         ex.printStackTrace();
      } catch (InstantiationException ex) {
         ex.printStackTrace();
      } catch (ClassNotFoundException ex) {
         ex.printStackTrace();
      }
      /* Turn off metal's use of bold fonts */
      UIManager.put("swing.boldMetal", Boolean.FALSE);
```

```
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                GUIManager.createAndShowGUI();
            }
        });
            }

             /**
              * Helper method to check classes that the user wishes to enroll in and compares
it to classes that they have
              * already taken. Returns whether or not there is a time conflict.
              * @param sid
              * @param crn
              * @return
              */
             @SuppressWarnings({ "rawtypes", "unchecked" })
             private static boolean timeconflicts(String sid, String crn){
                                ResultSet scurrent = execQuery( con, "SELECT days, starttime,
endtime FROM enrolled " +

"inner join section using (crn)" +

"WHERE student_id = '" + sid + "';");
                                ResultSet snew = execQuery( con, "SELECT days, starttime,
endtime FROM section " +

"WHERE crn = '" + crn + "';");
                        try {
                            ArrayList schedule[] = new ArrayList[5];
                            schedule[0] = new ArrayList();
                            schedule[1] = new ArrayList();
                            schedule[2] = new ArrayList();
                            schedule[3] = new ArrayList();
                            schedule[4] = new ArrayList();
                                    while (scurrent.next()){
                                        String days = scurrent.getString(1);
                                        int starttime = convertTime(scurrent.getString(2));
                                        int endtime = convertTime(scurrent.getString(3));
                                        if(days.contains("M")){
                                                schedule[0].add(starttime);
                                                schedule[0].add(endtime);
                                        }
                                        if(days.contains("T")){
                                                schedule[1].add(starttime);
                                                schedule[1].add(endtime);
```

```
                    }
                    if(days.contains("W")){
                        schedule[2].add(starttime);
                        schedule[2].add(endtime);
                    }
                    if(days.contains("T")){
                        schedule[3].add(starttime);
                        schedule[3].add(endtime);
                    }
                    if(days.contains("F")){
                        schedule[4].add(starttime);
                        schedule[4].add(endtime);
                    }
                }
                while (snew.next()){
                    String days = snew.getString(1);
                    int starttime = convertTime(snew.getString(2));
                    int endtime = convertTime(snew.getString(3));
                    if(days.contains("M")){
                        for(int i=0; i<schedule[0].size(); i+=2){
                            boolean ends = endtime >
(Integer)schedule[0].get(i) && endtime <= (Integer)schedule[0].get(i+1);
                            boolean starts = starttime >=
(Integer)schedule[0].get(i) && starttime < (Integer)schedule[0].get(i+1);
                            if(starts || ends)return true;
                        }
                    }
                    if(days.contains("T")){
                        for(int i=0; i<schedule[1].size()-1; i+=2){
                            boolean ends = endtime >
(Integer)schedule[1].get(i) && endtime <= (Integer)schedule[1].get(i+1);
                            boolean starts = starttime >=
(Integer)schedule[1].get(i) && starttime < (Integer)schedule[1].get(i+1);
                            if(starts || ends)return true;
                        }
                    }
                    if(days.contains("W")){
                        for(int i=0; i<schedule[2].size(); i+=2){
                            boolean ends = endtime >
(Integer)schedule[2].get(i) && endtime <= (Integer)schedule[2].get(i+1);
                            boolean starts = starttime >=
(Integer)schedule[2].get(i) && starttime < (Integer)schedule[2].get(i+1);
                            if(starts || ends)return true;
                        }
                    }
                    if(days.contains("T")){
```

```
                                for(int i=0; i<schedule[3].size(); i+=2){
                                        boolean ends = endtime >
(Integer)schedule[3].get(i) && endtime <= (Integer)schedule[3].get(i+1);
                                        boolean starts = starttime >=
(Integer)schedule[3].get(i) && starttime < (Integer)schedule[3].get(i+1);
                                        if(starts || ends)return true;
                                }
                        }
                        if(days.contains("F")){
                                for(int i=0; i<schedule[4].size(); i+=2){
                                        boolean ends = endtime >
(Integer)schedule[4].get(i) && endtime <= (Integer)schedule[4].get(i+1);
                                        boolean starts = starttime >=
(Integer)schedule[4].get(i) && starttime < (Integer)schedule[4].get(i+1);
                                        if(starts || ends)return true;
                                }
                        }
                }

                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                        return false;
        }

        /**
         * Helper method that converts a string time into a corresponding integer.
         * @param time
         * @return
         */
        private static int convertTime(String time){
                String stime = time.replaceAll(":", "");
        boolean pm = false;
        if (stime.contains("PM")) pm = true;
        int itime = Integer.parseInt(stime.substring(0, 6));
        if (pm && itime < 120000) itime += 120000;
        return itime;
        }

        /**
         * Helper method that checks classes the user wants to register for against how
many seats are remaining in the class.
         * @param crn
         * @return
         */
        private static boolean remainingseats(String crn){
```

```
		boolean remaining=true;
		ResultSet sremaining = execQuery( con, "SELECT count(remaining)
FROM section WHERE crn = '" + crn + "';");
	try {
		sremaining.next();
				if(sremaining.getInt(1)<=0) remaining = false;
		} catch (SQLException e) {
			e.printStackTrace();
		}
		return remaining;
	}

	static Class loadDriver( String driverName ){
		try {
			return( Class.forName( driverName ));
		}
		catch( ClassNotFoundException e ) {
			System.err.println( "Can't load driver - " + e.getMessage());
			return( null );
		}
	}

	static Connection connectURL( String URL ){
		try {
			return( DriverManager.getConnection( URL ));
		}
		catch( SQLException e ) {
			System.err.println( "Can't connect - " + e.getMessage());
			return( null );
		}
	}

	static void execQueries (Connection con, String query){
		String subquery = "";
		for (int i=0; i<query.length();){
			int old = i;
			i=query.indexOf(';', i)+1;
			subquery = query.substring(old, i);
			execQuery( con, subquery );
		}
	}

	static ResultSet execQuery( Connection con, String query ){
		try {
			Statement stmt = con.createStatement();
			System.out.println( query );
```

```
                return( stmt.executeQuery( query ));
            }
            catch( SQLException e ) {
                System.err.println( "Query failed - " + e.getMessage());
                return( null );
            }
        }
}

/**
 * This class serves as the main element for the GUI. It creates the window and switches between different
 * panels using a cardlayout to offer functionality to the user.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class GUIManager{

        private static final long serialVersionUID = 1L;
        private JPanel cards;
        private static ClassRegistration database = new ClassRegistration();
        private int id;
        private JTextArea queryField = new JTextArea(15, 30);

        /**
         * This method adds initial cards to the card layout. These cards are either called immediately, or
         * do not have any changing elements.
         */
        public void addComponentToPane(Container pane) {
      //Create the panel that contains the "cards".
      cards = new JPanel(new CardLayout());
      cards.add(new Login(this), "login");
      cards.add(new Menu(this), "menu");
      cards.add(new EnterName(this), "name");
      cards.add(new MajorMinorChoice(this), "declare");
      cards.add(new Declare(this, 0), "major");
      cards.add(new Declare(this, 2), "minor");
      cards.add(new Declare(this, 4), "doubledecide");
      pane.add(cards, BorderLayout.CENTER);
   }

        /**
         * Called by ClassRegistration to create a JFrame that contains the GUI.
         */
        public static void createAndShowGUI() {
```

```
    //Create and set up the window.
    JFrame frame = new JFrame("Schedule Manager");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400,400);
    //Create and set up the content pane.
    GUIManager title = new GUIManager();
    title.addComponentToPane(frame.getContentPane());

    //Display the window.
    //frame.pack();
    frame.setVisible(true);
}

    /**
     * Stores the student id after the user sets it in the login menu
     * @param id
     */
    public void setID(int id){
            this.id = id;
    }

    /**
     * Returns the id.
     * @return
     */
    public int getID(){
            return id;
    }

    /**
     * Sets the text in the Query Field text area.
     * @param field
     */
    public void setQueryField(String field){
            queryField.setText(field);
    }

    /**
     * Used to call the database to run queries in the dabase.
     * @return
     */
    public ClassRegistration getDatabase(){
            return database;
    }

    /**
```

```
	 * Called to change the curren JPanel set to show. Creates new JPanels if the
information in them changes.
	 * @param screen
	 */
	public void setScreen(String screen){
		if (screen.equals("majorRemove")){
			cards.add(new Declare(this, 1), "majorRemove");
		}else if(screen.equals("minorRemove")){
		cards.add(new Declare(this, 3), "minorRemove");
		}else if(screen.equals("addclass")){
			cards.add(new AddClass(this), "addclass");
		}else if(screen.equals("classBrowse")){
			cards.add(new ClassBrowser(this, queryField), "classBrowse");
		}else if(screen.equals("registerClass")){
			cards.add(new RegisterClass(this), "registerClass");
		}else if(screen.equals("login")){
			cards.add(new Login(this), "login");
		}
		CardLayout cl = (CardLayout)(cards.getLayout());
		cl.show(cards, screen);
	}
}

/**
 * The Menu class creates a panel that serves as the main menu.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class Menu extends JPanel implements ActionListener {

	private static final long serialVersionUID = 1L;
	private GUIManager context;
	private JButton declare;
	private JButton logout;
	private JButton addClass;
	private JButton browseClass;
	private JButton createSchedule;

	/**
	 * When called menu creates the buttons it displays.
	 * @param context
	 */
	public Menu(GUIManager context){
		setLayout(new GridLayout(6, 0));
		this.context = context;
		JLabel title = new JLabel("Schedule Helper", SwingConstants.CENTER);
```

```
                title.setBackground(Color.LIGHT_GRAY);
                add(title);
                declare = new JButton("Edit Major/Minors");
                declare.addActionListener(this);
                add(declare);
                addClass = new JButton("Add Classes");
                addClass.addActionListener(this);
                add(addClass);
                browseClass = new JButton("Browse Classes");
                browseClass.addActionListener(this);
                add(browseClass);
                createSchedule = new JButton("Create Schedule");
                createSchedule.addActionListener(this);
                add(createSchedule);
                logout = new JButton("Logout");
                logout.addActionListener(this);
                add(logout);
        }

        /**
         * Uses the button press to determine which JPanel to switch to.
         */
        public void actionPerformed(ActionEvent arg0) {
                if (logout == arg0.getSource()){
                        context.setScreen("login");
                }else if (declare == arg0.getSource()){
                        context.setScreen("declare");
                }else if (addClass == arg0.getSource()){
                        context.setScreen("addclass");
                }else if (browseClass == arg0.getSource()){
                        context.setScreen("classBrowse");
                }else if (createSchedule == arg0.getSource()){
                        context.setScreen("registerClass");
                }
        }
}

/**
 * The Login Class creates a panel that allows a user to log in.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class Login extends JPanel implements ActionListener{

        private static final long serialVersionUID = 1L;
        private JTextField id;
```

```
	private GUIManager context;

	/**
	 * Creates the text filed and button necessary to login.
	 * @param context
	 */
	public Login(GUIManager context){
		setLayout(new GridLayout(10,0));
		this.context = context;
		JButton enter = new JButton("Login");
		id = new JTextField(9);
		enter.addActionListener(this);
		JPanel format = new JPanel();
		format.add(id);
		format.add(enter);
		add(new JPanel());
		add(new JPanel());
		add(new JPanel());
		JLabel title = new JLabel("Type in your ID to login.",
SwingConstants.CENTER);
		title.setBackground(Color.LIGHT_GRAY);
		add(title);
		add(format);
	}

	/**
	 * When login is presses, if the id number is of valid length it is compared to ids
already registerd.
	 * If no matches are found the user is taken to a screen and given the option to
register.
	 */
	public void actionPerformed(ActionEvent e) {
		if (id.getText().length() == 9){
			int idNumber = Integer.parseInt(id.getText());
			context.setID(idNumber);
			if(context.getDatabase().inDatabase(idNumber)){
				context.setScreen("menu");
			}else{
				context.setScreen("name");
			}
		}else{
			ErrorMessage error = new ErrorMessage("Invalid ID");
		}
	}

}
```

```
/**
 * If the users id is invalid, the EnterName class creates a JPanel that allows the user to be added to the students database.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class EnterName extends JPanel implements ActionListener{

	private static final long serialVersionUID = 1L;
	private JTextField name;
	private GUIManager context;
	private JButton back;

	/**
	 * Creates the text field and button. Also contains a cancel button.
	 * @param context
	 */
	public EnterName (GUIManager context){
		setLayout(new GridLayout(0,1));
		this.context = context;
		JButton submit = new JButton("Submit");
		submit.addActionListener(this);
		back = new JButton("Back");
		back.addActionListener(this);
		name = new JTextField(15);
		JPanel line = new JPanel();
		line.add(name);
		line.add(submit);
		line.add(back);
		JTextArea title = new JTextArea("ID not found.\nIf you have reached this screen " +
				"by mistake, press the back button\nIf you would like to be added to the database, " +
				"put in your name\n and press the add button.",10,30);
		title.setLineWrap(true);
		title.setEditable(false);
		title.setBackground(Color.LIGHT_GRAY);
		add(title);
		add(line);
	}

	/**
	 * Adds the user to the database if login is pressed. Cancel returns to the main page.
	 */
```

```
	public void actionPerformed(ActionEvent arg0) {
		if (arg0.getSource() == back){
			context.setScreen("login");
		}else{
			context.getDatabase().addToDatabase(context.getID(),
name.getText());
			context.setScreen("menu");
		}
	}
}

/**
 * Class that builds a panel to allow a user to choose between major and minor.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class MajorMinorChoice extends JPanel implements ActionListener{

	private static final long serialVersionUID = 1L;
	private JButton major;
	private JButton minor;
	private GUIManager context;

	/**
	 * Creates two buttons.
	 * @param context
	 */
	public MajorMinorChoice( GUIManager context){
		setLayout(new GridLayout(0,1));
		JPanel format = new JPanel();
		this.context = context;
		major = new JButton("Major");
		major.addActionListener(this);
		format.add(major);
		minor = new JButton("Minor");
		minor.addActionListener(this);
		format.add(minor);
		add(new JPanel());
		add(format);
	}

	/**
	 * Based on the button pressed, the user is taken to different versions of the
Declare panel.
	 */
	public void actionPerformed(ActionEvent arg0) {
```

```
                if (arg0.getSource() == major){
                        int numMajor =
context.getDatabase().getNumMaj(context.getID());
                        if (numMajor == 0){
                                context.setScreen("major");
                        }else if(numMajor == 1){
                                int numMinor =
context.getDatabase().getNumMin(context.getID());
                                if (numMinor == 0){
                                        context.setScreen("doubledecide");
                                }else {
                                        context.setScreen("majorRemove");
                                }
                        }else{
                                context.setScreen("majorRemove");
                        }
                }else{
                        int numMinor =
context.getDatabase().getNumMin(context.getID());
                        if (numMinor == 0){
                                int numMajor =
context.getDatabase().getNumMaj(context.getID());
                                if (numMajor < 2){
                                        context.setScreen("minor");
                                }else{
                                        context.setScreen("majorRemove");
                                }
                        }else{
                                context.setScreen("minorRemove");
                        }
                }
        }
}

/**
 * Creates a panel that lets the user add/remove majors and minors. There are strict
 * reinforcements in place to keep the user from having more than two majors or one
major
 * and one minor.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class Declare extends JPanel implements ActionListener{

        private static final long serialVersionUID = 1L;
        private static final int ADD_MAJOR = 0;
```

```
private static final int REMOVE_MAJOR = 1;
private static final int ADD_MINOR = 2;
private static final int REMOVE_MINOR = 3;
private JTextField department;
private JButton cancel;
private JButton submit;
private JButton remove;
private JButton doubleDeclare;
private JButton unDeclare;
private GUIManager context;
private int majorMinor;

/**
 * Depending on the users choice in declare and their number of majors/minors,
 * the constructor either calls a helper method to build a GUI or creaes a GUI
 * for the choice between double declaring and removing a major.
 * @param context the GUIManager
 * @param majorMinor an int that is used to determine the screen built
 */
public Declare(GUIManager context, int majorMinor){
        this.majorMinor = majorMinor;
        this.context = context;
        if (majorMinor == ADD_MAJOR){
                createAdd();
        }else if (majorMinor == REMOVE_MAJOR){
                createRemove(REMOVE_MAJOR);
        }else if (majorMinor == ADD_MINOR){
                createAdd();
        }else if (majorMinor == REMOVE_MINOR){
                createRemove(REMOVE_MINOR);
        }else{
                setLayout(new GridLayout(0,1));
                JPanel format = new JPanel();
                doubleDeclare = new JButton("Double Declare");
                doubleDeclare.addActionListener(this);
                unDeclare = new JButton("remove");
                unDeclare.addActionListener(this);
                format.add(doubleDeclare);
                format.add(unDeclare);
                add(new JPanel());
                add(format);
        }
}

/**
 * This screen will be created if the user decides to change major and
```

```
         * has two majors, has one major and one minor, or has one major and has
decided to
         * remove a major. It can be used to remove a minor if the user selects minor and
already has
         * one minor declared.
         * @param majorMinor an int which determines if they are removing a major or
minor
         */
        private void createRemove(int majorMinor) {
                JTextArea declaredMajMin;
                if (majorMinor == REMOVE_MAJOR){
                        declaredMajMin = new
JTextArea(context.getDatabase().getMajList(context.getID()), 15, 30);
                }else{
                        declaredMajMin = new
JTextArea(context.getDatabase().getMinList(context.getID()), 15, 30);
                }
                declaredMajMin.setEditable(false);
                JTextArea title = new JTextArea("Here you can remove majors or
minors.\n" +
                                " Note that you can only have two majors or one major and
one\nminor." +
                                "\nIf you want to add a major or minor,\n you may need to
remove a major or minor first",6,5);
                title.setLineWrap(true);
                title.setEditable(false);
                title.setBackground(Color.LIGHT_GRAY);
                add(title);
                JScrollPane scroll = new JScrollPane(declaredMajMin);

scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
        add(scroll);
                department = new JTextField(10);
                cancel = new JButton("cancel");
                cancel.addActionListener(this);
                remove = new JButton("remove");
                remove.addActionListener(this);
                add(department);
                add(remove);
                add(cancel);
        }

        /**
         * Creates a screen that lets the user add a new major or minor.
         */
        private void createAdd() {
```

```
		JTextArea allDep = new JTextArea(context.getDatabase().getDepList(),
15, 30);
		JTextArea title = new JTextArea("Here you can add majors or minors.\n
Note that you" +
					" can only have two majors or one major and one\n
minor.\nIf you want to add a major or minor" +
					",\n you may need to remove a major or minor first",6,5);
		title.setLineWrap(true);
		title.setEditable(false);
		title.setBackground(Color.LIGHT_GRAY);
		add(title);
		allDep.setEditable(false);
		JScrollPane scroll = new JScrollPane(allDep);

scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
	add(scroll);
		department = new JTextField(10);
		cancel = new JButton("done");
		cancel.addActionListener(this);
		submit = new JButton("submit");
		submit.addActionListener(this);
		add(department);
		add(submit);
		add(cancel);
	}

	/**
	 * Action listener that listens for button presses and calls appropriate queries after
checking for
	 * valid input.
	 */
	public void actionPerformed(ActionEvent e) {
		if (e.getSource() == cancel){
			context.setScreen("menu");
		}else if(e.getSource() == doubleDeclare){
			context.setScreen("major");
		}else if(e.getSource() == unDeclare){
			context.setScreen("majorRemove");
		}else if(e.getSource() == submit){
			if (context.getDatabase().isValidDep(department.getText())){
				if (majorMinor == ADD_MAJOR){

	context.getDatabase().addMajor(department.getText(), context.getID());
				}else{

	context.getDatabase().addMinor(department.getText(), context.getID());
```

```
                        }
                        context.setScreen("menu");
                    }else{
                        ErrorMessage warning = new ErrorMessage("Invalid
Department");
                    }
                }else if(e.getSource() == remove){
                    if (context.getDatabase().isValidDep(department.getText())){
                        if (majorMinor == REMOVE_MAJOR){

        context.getDatabase().removeMajor(department.getText(), context.getID());
                        }else{

        context.getDatabase().removeMinor(department.getText(), context.getID());
                        }
                        context.setScreen("menu");
                    }else{
                        ErrorMessage warning = new ErrorMessage("Invalid
Department");
                    }
                }
        }
}

/**
 * The AddClass class creates a panel that shows the classes that the user has previously
taken
 * and allows them to add more to the list or remove items from the lis.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class AddClass extends JPanel implements ActionListener{

        private static final long serialVersionUID = 1L;
        private GUIManager context;
        private JButton submit;
        private JButton done;
        private JButton remove;
        private JTextField classes;

        /**
         * The constructor builds the main screen with a text area and some buttons.
         * @param context GUIManager
         */
        public AddClass(GUIManager context){
                setLayout(new BoxLayout(this,1));
```

```
		this.context = context;
		JTextArea prevClasses = new
JTextArea(context.getDatabase().getPrevClasses(context.getID()), 15, 30);
		prevClasses.setEditable(false);
		JScrollPane scroll = new JScrollPane(prevClasses);

scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
	add(scroll);
	JTextField classFormat = new JTextField("Department Course# GER");
	classFormat.setEditable(false);
	classFormat.setBackground(Color.LIGHT_GRAY);
	add(classFormat);

	JPanel line = new JPanel();
	classes = new JTextField(15);
		line.add(classes);
		submit = new JButton("add");
		submit.addActionListener(this);
		line.add(submit);
		remove = new JButton("remove");
		remove.addActionListener(this);
		line.add(remove);
		done = new JButton("done");
		done.addActionListener(this);
		line.add(done);

		add(line);

	}

	/**
	 * Action listener that listens for button presses and will either call the query for
adding or removing information
	 * or will take the user back to the menu.
	 */
	public void actionPerformed(ActionEvent arg0) {
		if (arg0.getSource() == done){
			context.setScreen("menu");
		}else if (arg0.getSource() == submit){
			String[] info = classes.getText().split(" ");
			if (info.length == 2){
				info = new String[]{info[0], info[1], " "};
			}
			context.getDatabase().addClass(info, context.getID());
			context.setScreen("addclass");
		}else{
```

```
                        String[] info = classes.getText().split(" ");
                        if (context.getDatabase().isValidClass(info)){
                                context.getDatabase().removeClass(info, context.getID());
                                context.setScreen("addclass");
                        }else{
                                ErrorMessage warning = new ErrorMessage("Invalid
Class");
                        }
                }
        }
}

/**
 * The ClassBrowser class creates a panel that allows the user to pose controlled queries
to the database
 * using the GUI interface and prints the results to a text area.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class ClassBrowser extends JPanel implements ActionListener{

        private static final long serialVersionUID = 1L;
        private GUIManager context;
        private JTextField department;
        private JTextField classes;
        private JTextField build;
        private JButton done;
        private JButton submit;
        private JCheckBox dep;
        private JCheckBox classID;
        private JCheckBox teacherRate;
        private JCheckBox graduate;
        private JCheckBox building;

        /**
         * The constructor creates checkboxes and text fields to allow the user to specify a
query.
         * @param context GUIManager
         * @param queryField
         */
        public ClassBrowser(GUIManager context, JTextArea queryField){
                this.context = context;
                setLayout(new BoxLayout(this,1));
                queryField.setEditable(false);
                //queryField.setLineWrap(true);
                JScrollPane scroll = new JScrollPane(queryField);
```

```
scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
                add(scroll);

                JPanel line1 = new JPanel();
                department = new JTextField(10);
                dep = new JCheckBox("Department");
                line1.add(department);
                line1.add(dep);
                add(line1);

                JPanel line2 = new JPanel();
                classes = new JTextField(10);
                classID = new JCheckBox("Course Number");
                line2.add(classes);
                line2.add(classID);
                add(line2);

                JPanel line3 = new JPanel();
                build = new JTextField(10);
                building = new JCheckBox("Near Building");
                line3.add(build);
                line3.add(building);
                add(line3);

                JPanel line4 = new JPanel();
                graduate = new JCheckBox("Left to Graduate");
                teacherRate = new JCheckBox("By Teacher Rating");
                line4.add(graduate);
                line4.add(teacherRate);
                add (line4);

                JPanel line5 = new JPanel();
                done = new JButton("Done");
                done.addActionListener(this);
                submit = new JButton("Submit");
                submit.addActionListener(this);
                line5.add(done);
                line5.add(submit);
                add(line5);
                }

        /**
         * If done is pressed the user is returned to the menu.
         * If submit is pressed the checkboxes are checked and a query is
         * built based on the them.
```

```
	 */
	public void actionPerformed(ActionEvent arg0) {
		if (arg0.getSource() == done){
			context.setScreen("menu");
		}else{
			boolean fail = false;
			String query = "";
			//If left to graduate is selected a query is built to build a relation on
the classes the user needs
			if (graduate.isSelected()){
				//check what majors the student has
			ArrayList<String> majors = new ArrayList<String>();
			ArrayList<String> minors = new ArrayList<String>();
			majors = context.getDatabase().getMajListArray(context.getID());
			minors = context.getDatabase().getMinListArray(context.getID());

				String declared = "(";
				String qmajors = "";
				if(majors.size() > 0){
					qmajors = "select * " +
					"from required_major " +
					"where department = '" + majors.get(0) + "' ";
					if(majors.size() > 1) qmajors += "or department =
'" + majors.get(1) + "' ";
				}
				String qminors = "";
				if(minors.size() > 0){
					qmajors = "select * " +
					"from required_minor " +
					"where department = '" + minors.get(0) + "' ";
				}
				if (qmajors.compareTo("")==0) declared += qminors;
				else if (qminors.compareTo("")==0) declared += qmajors;
				else declared += qmajors + "union " + qminors;

				declared+=") ";

				//GERS
				String gersleft = "(select department, course_number from
ger) EXCEPT (select g.department, g.course_number " +
					"from ger as g, previous as p " +
					"where g.ger_attribute LIKE ('%' ||
p.ger_attribute || '%') and p.student_id = '"
					+ context.getID() + "' and p.ger_attribute <>
' ') ";
```

```
                                String dgers="(";
                                if (declared.compareTo("() ")==0) dgers = gersleft;
                                else dgers = gersleft + "union " + declared;
                                dgers+=") ";

                                String prereqs="select department, course_number from
prerequisites as prep " +
                                "where prerequisite not in " +
                                        "(select course_number from previous as prev " +
                                                "where student_id='" + context.getID() +"'
and prev.department = prep.department) ";

                                dgers += " EXCEPT ((select department, course_number
from previous where student_id = '"
                                        + context.getID() + "') union (" + prereqs + ")) ";
                                query = "select distinct * from ((" + dgers + ") as f " +
                                                "inner join section using (course_number,
department) " +
                                                "inner join course using (course_number,
department) " +
                                                "inner join instructor using (instructor) " ;
                        }else{ //If graduate is not selected a query selects every class from
the database
                                query = "select * " +
                                "from section " +
                                "inner join course using (course_number, department) " +
                                "inner join instructor using (instructor) ";
                        }
                        // If building is selected nearby is added to the query and it selects
building close to the given building
                        if (building.isSelected()){
                                if
(context.getDatabase().isValidBuilding(build.getText())){
                                        query += "join nearby on section.building =
nearby.building1 " +
                                        "WHERE building2 = '" + build.getText() + "' ";
                                }else{
                                        fail = true;
                                        ErrorMessage error = new ErrorMessage("Invalid
building");
                                }
                        }
                        //If department is selected a where statements is added based on
department.
                        if (dep.isSelected()){
                                String selection = department.getText();
```

```
                                        if (building.isSelected()){
                                                query += "and ";
                                        }
                                        else{
                                                query += "WHERE ";
                                        }
                                        if (context.getDatabase().isValidDep(selection)){
                                                //If department and course number are selected a
where stated based on department and course number is added
                                                if (classID.isSelected()){
                                                        String classChoice = classes.getText();
                                                        query += "department = '" + selection + "'
and course_number = '" + classChoice + "' ";
                                                }else{
                                                        query += "department = '" + selection + "' ";
                                                }
                                        }else{
                                                fail = true;
                                                ErrorMessage warning = new
ErrorMessage("Invalid Department");
                                        }
                                }else{
                                        if (classID.isSelected()){
                                                fail = true;
                                                ErrorMessage warning = new
ErrorMessage("Cannot use course number without department.");
                                        }
                                }
                                // If by rating is selected a sort is added to the end of the query.
                                if (teacherRate.isSelected()){
                                        query += "order by rating desc ";
                                }
                                if (!fail){

        context.setQueryField(context.getDatabase().classQuery(query));
                                }
                                context.setScreen("classBrowse");
                        }
                }
}

/**
 * The RegisterClass class creates a panel that allows the user to register for classes on
 * the current semester. Checks the amount of space in the class and the users schedule
for
 * time conflicts.
```

```
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class RegisterClass extends JPanel implements ActionListener{

	private static final long serialVersionUID = 1L;
	private GUIManager context;
	private JButton register;
	private JButton remove;
	private JButton done;
	private JTextField CRN;

	/**
	 * Constructs a GUI for the user to register or remove classes.
	 * @param context GUIManager
	 */
	public RegisterClass(GUIManager context){
		this.context = context;
		JTextArea regClasses = new
JTextArea(context.getDatabase().getRegClasses(context.getID()), 15, 30);
		regClasses.setEditable(false);
		JScrollPane scroll = new JScrollPane(regClasses);

scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
	add(scroll);
	CRN = new JTextField(10);
	add(CRN);
	register = new JButton("Register");
	register.addActionListener(this);
	add(register);
	remove = new JButton("Remove");
	remove.addActionListener(this);
	add(remove);
	done = new JButton("Done");
	done.addActionListener(this);
	add(done);
	   }

	/**
	 * If the input is valid the users class choice will be registered or removed.
	 */
	public void actionPerformed(ActionEvent arg0) {
		if (arg0.getSource() == done){
			System.out.println("done");
			context.setScreen("menu");
		}else if (arg0.getSource() == register){
```

```
                        System.out.println("register");
                        // ... Time conflicts + Remaining Seats check
                        String selection = CRN.getText();
                        if(context.getDatabase().isValidCRN(selection)){
                                int error =
context.getDatabase().registerClass(Integer.toString(context.getID()), selection);
                                if (error == 0){
                                        ErrorMessage warning = new
ErrorMessage("Cannot register class: Time Conflict");
                                }else if (error == 1){
                                        ErrorMessage warning = new
ErrorMessage("Cannot register class: Unfulfilled Prerequisite");
                                }
                                context.setScreen("registerClass");
                        }else{
                                ErrorMessage warning = new ErrorMessage("Invalid
CRN");
                        }
                }else if (arg0.getSource() == remove){
                        System.out.println("remove");
                        String selection = CRN.getText();
                        if(context.getDatabase().isValidCRN(selection)){
                                context.getDatabase().unregisterClass(context.getID(),
selection);
                        }else{
                                ErrorMessage warning = new ErrorMessage("Invalid
CRN");
                        }
                        context.setScreen("registerClass");
                }
        }
}

/**
 * This is a class used to create error messages in a pop-up window.
 * @author Daniel Boos, Nick Starr, Donathan Tuck
 *
 */
class ErrorMessage extends JFrame implements ActionListener{

        private static final long serialVersionUID = 1L;
        private JButton ok;

        /**
         * Creates and shows an error message.
         * @param errorMessage
```

```
	 */
	public ErrorMessage(String errorMessage){
		setLayout(new GridLayout(0,1));
		JLabel error = new JLabel(errorMessage);
		add(error);
		ok = new JButton("OK");
		ok.addActionListener(this);
		add(ok);
		pack();
		setVisible(true);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}

	/**
	 * Removes the error message.
	 */
	public void actionPerformed(ActionEvent arg0) {
		this.setVisible(false);
	}
}
```

Appendix B

Note: due to the large size of some of these tables, we only selected at most twenty per table.

Classroom

building_name	room	capacity
ADAIR	1	16
ADAIR	11	12
ADAIR	108	20
ADAIR	203	10
ADAIR	308	30
ADAIR	407	18
ANDREW	101	110
ANDREW	118	16
ANDREW	120	16
ANDREW	201	17
ANDREW	203	15
ANDREW	207	20
ANDREW	215	16
ANDREW	216	16
ANDREW	220	4
ANDRH	326	25
ANDRH	423	20
ANDRH	424	10
APTS	5	15
APTS	9	14

(20 rows)

Course

department	course_number	title	credits
AFST	150W	FR SEM:August Wilson's10 Plays	4.0
AFST	302	The Idea of Race	3.0
AFST	303	AfrAmer Hist since Emncptn	3.0
AFST	306	African Religions, Afrcn Lives	3.0
AFST	309	African Econ Development	3.0
AFST	331	Jazz	4.0
AFST	334	History Amer Vernacular Dance	3.0
AFST	341	African Ritual/Relig Practice	3.0
AFST	344	Politics in Africa	3.0
AFST	365	Early African American Lit	3.0
AFST	406	Sem on Hip-Hop Culture&History	3.0
AFST	417	Harlem in Vogue	3.0
AMST	150W	FR SM:Other Am:Caribbean Lit	4.0
AMST	208	Intro to Dis/ability Studies	4.0
AMST	208D	Intro Dis/ability Studies Disc	0.0
AMST	209	Interracialism:Race/Lit/Law	4.0
AMST	209D	Interracialism Discussion	0.0
AMST	241	History Amer Vernacular Dance	3.0
AMST	273	Jazz	4.0
AMST	350	Nubia in American Thought	3.0

(20 rows)

Declared

student_id	declaration	department
930697382	major	FILM
930634706	major	CSCI
930634706	major	PHIL

(3 rows)

Enrolled

student_id	crn
930697382	18122

(1 row)

GER

department	course_number	ger_attribute
AFST	150W	Freshmen Seminar Req and GE5 Lit/Hist of the Arts and Lower Divsn Writing Req
AFST	303	GE4A Hist/Cultr Euro Tradition
AFST	331	GE4A Hist/Cultr Euro Tradition and GE5 Lit/Hist of the Arts
AFST	334	GE5 Lit/Hist of the Arts
AFST	341	GE4B Hist/Cult outside EurTrad
AFST	344	GE4B Hist/Cult outside EurTrad
AMST	241	GE5 Lit/Hist of the Arts
AMST	273	GE4A Hist/Cultr Euro Tradition and GE5 Lit/Hist of the Arts
AMST	350	GE4A Hist/Cultr Euro Tradition
ANTH	201	GE4B Hist/Cult outside EurTrad
ANTH	202	GE4C Cross-Cultural Issues
ANTH	203	GE2B Natural Sci-Biological
ANTH	204	GE3 Social Sciences
ANTH	241	GE4B Hist/Cult outside EurTrad
ANTH	312	GE4C Cross-Cultural Issues
ANTH	315	GE3 Social Sciences
ANTH	324	GE4B Hist/Cult outside EurTrad
ANTH	337	GE4B Hist/Cult outside EurTrad
ANTH	349	GE4B Hist/Cult outside EurTrad
ANTH	363	GE4C Cross-Cultural Issues

(20 rows)

Instructor

instructor	rating
A M. van der Veen (P)	-1
Aaron Beck (P)	-1
Aaron P. Blossom (P)	-1
Adam M. Gershowitz (P)	-1
Adam S. Potkay (P)	4.2
Admasu Shiferaw Seyoum (P)	-1
Aiko Kitamura (P)	4.2
Akiko Fujimoto (P)	4.4
Alan H. Goldman (P)	3.6
Alan J. Meese (P)	-1
Alan Wallach (P)	3.8
Alastair M. Connell (P)	5
Alemante Gebre-Selassie (P)	-1
Alexander B. Angelov (P)	-1
Alexander V. Prokhorov (P)	4.7
Alexander Woods (P)	-1
Alfredo M. Pereira (P)	4.1
Alison Scott (P)	-1
Allison O. Larsen (P)	-1
Amy A. Quark (P)	3.5

(20 rows)

Nearby

building1	building2
ADAIR	ANDREW
ADAIR	ISC
ADAIR	JONES
ADAIR	MILLER
ADAIR	MLLNTN
ADAIR	MORTON
ADAIR	PBK
ADAIR	SMALL
ADAIR	SWEM
ANDREW	ADAIR
ANDREW	ISC
ANDREW	JONES
ANDREW	MLLNTN
ANDREW	MORTON
ANDREW	PBK
ANDREW	SMALL
ANDREW	SWEM
BLOW	EWELL
BLOW	JBLAIR
BLOW	MCGLTH

(20 rows)

Prerequisites

department	course_number	prerequisite
CSCI	241	141
CSCI	243	141
CSCI	321	241
CSCI	321	243
CSCI	312	241
CSCI	312	243
CSCI	301	241
CSCI	303	241
CSCI	303	243
CSCI	304	241
CSCI	412	321
CSCI	412	312
CSCI	412	301
CSCI	435	312
CSCI	435	301
CSCI	426	301
CSCI	426	303
CSCI	427	301
CSCI	427	303
CSCI	423	303

(20 rows)

Previous

student_id	department	course_number	ger_attribute
930634706	MUSC	101	GE6

(1 row)

Required_Major

department	course_number
CSCI	141
CSCI	241
CSCI	243
CSCI	312
CSCI	301
CSCI	303
CSCI	304
CSCI	423

(8 rows)

Required_Minor

department	course_number
CSCI	141
CSCI	241
CSCI	243

(3 rows)

Section

department	course_number	crn	remaining	instructor	days	starttime	endtime	building	room
AFST	150W	17044	5	Artisia V. Green (P)	MW	02:00:00 PM	03:20:00 PM	PBK	221
AFST	302	17638	4	Michael L. Blakey (P)	TR	11:00:00 AM	12:20:00 PM	WREN	204
AFST	303	17157	3	TBA	MWF	11:00:00 AM	11:50:00 AM	JBLAIR	223
AFST	306	16361	15	Mei Mei E. Sanford (P)	T	03:30:00 PM	05:50:00 PM	MORTON	238
AFST	306	16363	3	Beverly C. Peterson (P)	MWF	09:00:00 AM	09:50:00 AM	TYLER	301
AFST	306	16364	3	Beverly C. Peterson (P)	MWF	11:00:00 AM	11:50:00 AM	TYLER	336
AFST	306	17154	5	Hermine D. Pinson (P)	MW	03:30:00 PM	04:50:00 PM	TYLER	301
AFST	306	17639	3	Joanne M. Braxton (P)	TR	02:00:00 PM	03:20:00 PM	TYLER	201
AFST	306	16367	5	Jacquelyn Y. McLendon (P)	TR	12:30:00 PM	01:50:00 PM	TYLER	201
AFST	306	16368	3	James D. La Fleur (P)	TR	11:00:00 AM	12:20:00 PM	BLOW	332
AFST	306	17641	3	Jeremy W. Pope (P)	TR	09:30:00 AM	10:50:00 AM	BLOW	332
AFST	306	17710	5	Melvin P. Ely (P)	TR	12:30:00 PM	01:50:00 PM	JBLAIR	223
AFST	309	18443	20	Admasu Shiferaw Seyoum (P)	TR	03:30:00 PM	04:50:00 PM	MORTON	302
AFST	331	17045	4	TBA	TR	12:30:00 PM	01:50:00 PM	EWELL	151
AFST	334	17042	3	Leah F. Glenn (P)	TR	09:30:00 AM	10:50:00 AM	PBK	221
AFST	341	18356	5	Brad L. Weiss (P)	W	03:30:00 PM	06:20:00 PM	WSHGTN	301
AFST	344	17041	3	Philip G. Roessler (P)	TR	12:30:00 PM	01:50:00 PM	MORTON	38
AFST	344	17714	3	Philip G. Roessler (P)	TR	02:00:00 PM	03:20:00 PM	MORTON	20
AFST	365	17115	5	Mary Lynn Weiss (P)	MW	02:00:00 PM	03:20:00 PM	TYLER	102
AFST	406	18456	20	Chinua A. Thelwell (P)	MW	03:30:00 PM	04:50:00 PM	MLLNTN	123

(20 rows)

Student

student_id	name	credits_enrolled
930634706	Daniel Boos	0
930697382	Donathan Tuck	0

(2 rows)

Appendix C

Click the application link on our website and save it in a directory (it's best if you save it in an empty directory). Assuming the driver and java has been set up on the computer (we used the command, "export CLASSPATH=/usr/local/pgsql/share/java/postgresql-9.1-901.jdbc4.jar:.", on the bg6 computer) and you have ClassRegistration.java in your current directory, simply compile with "javac -g ClassRegistration.java". This will create all the class files to make the program run. Once finished, enter "java ClassRegistration" and be on your way. There a window will pop-up asking you to type your ID to logon. Here you can write any 9-digit number and hit logon. It will prompt you to type in your name. Type it in and hit submit. From now on, whenever you type that id into that login again, it will bring you to the main menu.

To add/remove a declaration, hit ‘Edit Majors/Minors.’ It will ask you to declare a major or minor. To declare a minor, hit minor and from the list, type in the abbreviated department you wish to minor in and hit submit. This is the same for majors except that you also have the ability to double declare.

To edit previous classes, hit ‘Add Classes’ on the main menu. Here you must type the department and course number you want to add/remove. You may also add a GER attribute to a course to help calculate which classes are recommended for graduation by including a space and the GER attribute (such as GE1, GE2, …).

To browse courses, hit ‘Browse Classes’ on the main menu. If the you click submit immediately, all the classes will be shown. If you select ‘Department’ and type in the abbreviated department name and hit submit, all the classes in that department will be shown. Similarly, you can do the same for ‘Course Number’ with the department as well. If you want to see all the classes nearby a building, simply check ‘Near Building’ and type in the abbreviated name of the building and hit submit. If you want the classes to be

ordered by the teacher rating, simply check ‘By Teacher Rating’ and hit submit. If you want it to recommend some classes for you to graduate, simply check ‘Left to Graduate’ and hit submit.

When you finally want to register for classes, hit ‘Create Schedule’ on the main menu. Then in the box below, type in the CRN of the section you would like to register in and hit register. If you want to remove one from your current schedule, type it in and hit remove.

Finally, if you want to logout so someone else can use it, hit ‘logout’ on the main screen.