Zi Xuan Li

Professor Auda

CSC 22100

May 17th, 2023

Assignment #4

Statement of the problem:

The purpose of this assignment is to work with a database schema and complete various tasks using a Relational Database Management System (RDBMS) and Java. The tasks include creating and populating tables based on provided data, calculating, and outputting the number of students for each letter grade in a specific course and semester, building a Java application that connects to the database and performs database operations, and displaying a pie chart showing the proportion of students for each letter grade. The report should include input tables, an output table with aggregated grades, and a corresponding pie chart. The Java application should use JavaFX graphics with the code being adaptable to canvases of variable dimensions.

Solution methods:

1. Consider the database schema below:

Students (empID, firstName, lastName, email, gender)

Courses (courseID, courseTitle, department)

Classes (courseID, studentID, sectionNo, year, semester, grade)

The underlined attributes are the primary keys of their corresponding tables. The value of attribute gender may only be F, M, or U, referring, respectively, for female, male, or unidentified. The only letter grades allowed in the database are A, B, C, D, F, or W.

Further you are provided the class schedule for the Fall 2023 semester in file scheduleFall2023.txt. The key to the data in scheduleFall2023 is (courseID, sectionNo).

2. Using a Relational Database Management System (RDBMS) of your choice, your tasks are to:

A. Create and populate a **Schedule** table using the data provided in file scheduleSpring2022.txt.

B. Create and populate **Courses** and **Classes** tables using the data in table **Schedule**.

C. Create and populate **Students** and **Classes** tables using data of your own together with the data in table Schedule.

D. Using GROUP BY, calculate and output the number of students for each letter grade in CSc 22100 [Introduction to Database Systems] in the Spring 2022 semester.

3. Build and test a Java application that [1] connects to the database, [2] creates, [3] populates, and [4] updates the **Students**, **Courses**, **Classes**, and **AggregateGrades** tables. The application should utilize PreparedStatement objects for the execution of DDL statements and SQL queries.

A. The Java application utilizes a class **Database** which includes inner classes **Schedule**, **Students**, **Courses**, **Classes**, and **AggregateGrades**, corresponding, respectively, to database Tables **Schedule**, **Students**, **Classes**, and **AggregateGrades**. The constructor of class **Database** is utilized to establish a connection to the RDBMS, while the constructors of the inner classes are utilized to create and populate the corresponding database Tables.

B. Classes **Schedule** and **Classes** also include update methods that update, respectively, the instructor of a class and grade of a student.

C. Class **Database** implements interfaces **StudentsDatabaseInterface** and **TableInterface** which include constants, and abstract, and static methods that define the DDL and SQL expressions used for creating, populating, and querying the database tables.

D. Utilize the classes in Assignment 3, including **HistogramAlphaBet** and **MyPieChart**, to build and display a pie chart showing the proportion of students for each letter grade. In the pie chart:

a. Each segment has a different color.

b. Each segment has a legend showing the corresponding grades and number of students.

c. The segments for the grades are displayed in alphabetical order.

4. The report should show [1] sample input tables, [2] output table for the aggregated grades and corresponding pie chart for a sufficient amount of input data, and [3] example[s] of the use of the update function.

5. You may only use JavaFX graphics and your own classes and methods for the operations included. Further,

a. The code is applicable to canvases of variable height and width.

b. The size of the pie chart is proportional to the smallest dimension of the canvas.

c. The segments of the pie chart are filled with different colors of your choice, specified through a **MyColor** enum reference type.

6. Explicitly specify all the classes imported and used in your Java application.

All classes that are imported:

```
import javafx.application.Application
import javafx.scene.Group
import javafx.scene.Scene
import javafx.scene.canvas.Canvas
import javafx.scene.canvas.GraphicsContext
import javafx.stage.Stage
import java.awt.*
import java.sql.Connection
import java.sql.DriverManager
import java.sql.SQLException
import java.sql.ResultSet
import java.util.HashMap
import java.util.Map
import java.sql.PreparedStatement
import java.util.stream.Collectors
```

Java code:

StudentsDatabaseApplication.java

```
package com.example.assignment1;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.stage.Stage;
import javafx.scene.text.Font;

import java.awt.*;
import java.sql.SQLException;
import java.util.Map;

public class StudentsDatabaseApplication extends Application {
        @Override
        public void start(Stage stage) {
                stage.setTitle("Assignment 4");
                double width = 900;
                double height = 600;

                double radius = 300;

                Group root = new Group();
                Canvas canvas = new Canvas(width, height);
                GraphicsContext GC = canvas.getGraphicsContext2D();
                root.getChildren().add(canvas);
                stage.setScene(new Scene(root));
                stage.show();

                MyPoint center = new MyPoint(width / 2, height / 2,
MyColor.BLACK);

                String server = "jdbc:mysql://localhost:3306/assignment4";
                String username = "root";
                String password = "password";

                try {
                        StudentsDatabase DataBase = new
StudentsDatabase(server, username, password);
                        String TextFile =
"C:/Users/zixua/OneDrive/Desktop/ScheduleFall2023.txt";

                        String sqlTable =
StudentsDatabaseInterface.TableSchedule;
                        String populateTable =
StudentsDatabaseInterface.FillTableSchedule(TextFile);
                        StudentsDatabase.Schedule schedule = DataBase.new
Schedule(sqlTable, populateTable);

                        sqlTable = StudentsDatabaseInterface.TableCourses;
                        populateTable =
StudentsDatabaseInterface.FillTableCourses();
```

```
                        StudentsDatabase.Courses courses = DataBase.new
Courses(sqlTable, populateTable);

                        sqlTable = StudentsDatabaseInterface.TableStudents;
                        StudentsDatabase.Students students = DataBase.new
Students(sqlTable);

                        students.insertStudents("378965124", "Emma", "Lee",
"emma.lee3789@gmail.com", "F");
                        students.insertStudents("576814329", "Aiden",
"Garcia", "aiden.garcia5768@gmail.com", "M");
                        students.insertStudents("912473086", "Chloe",
"Harris", "chloe.harris9124@gmail.com", "F");
                        students.insertStudents("235609187", "Landon",
"Wang", "landon.wang2356@gmail.com", "M");
                        students.insertStudents("789201536", "Avery",
"Gonzalez", "avery.gonzalez7892@gmail.com", "F");
                        students.insertStudents("401285937", "Noah",
"Rodriguez", "noah.rodriguez4012@gmail.com", "M");
                        students.insertStudents("857392614", "Ella", "Scott",
"ella.scott8573@gmail.com", "F");
                        students.insertStudents("142859736", "Cameron",
"Nguyen", "cameron.nguyen1428@gmail.com", "M");
                        students.insertStudents("936254710", "Lila",
"Rivera", "lila.rivera9362@gmail.com", "F");
                        students.insertStudents("510276349", "Logan",
"Hernandez", "logan.hernandez5102@gmail.com", "M");
                        students.insertStudents("267319048", "Mia", "Gomez",
"mia.gomez2673@gmail.com", "F");
                        students.insertStudents("803562149", "Ethan", "Cruz",
"ethan.cruz8035@gmail.com", "M");
                        students.insertStudents("459827613", "Aria", "Perez",
"aria.perez4598@gmail.com", "F");
                        students.insertStudents("641275098", "Daniel",
"Campbell", "daniel.campbell6412@gmail.com", "M");
                        students.insertStudents("298710345", "Sophia",
"Reed", "sophia.reed2987@gmail.com", "F");
                        students.insertStudents("736429081", "Adam", "Allen",
"adam.allen7364@gmail.com", "M");
                        students.insertStudents("184753629", "Arianna",
"Davis", "arianna.davis1847@gmail.com", "F");
                        students.insertStudents("512649307", "Jackson",
"Parker", "jackson.parker5126@gmail.com", "M");
                        students.insertStudents("897013246", "Mila", "Evans",
"mila.evans8970@gmail.com", "F");
                        students.insertStudents("325978164", "William",
"Moore", "william.moore3259@gmail.com", "M");
                        students.insertStudents("670413258", "Lily",
"Taylor", "lily.taylor6704@gmail.com", "F");
                        students.insertStudents("983142657", "Owen", "Green",
"owen.green9831@gmail.com", "M");
                        students.insertStudents("176429835", "Hailey",
"Baker", "hailey.baker1764@gmail.com", "F");
                        students.insertStudents("542098361", "Caleb",
"Gutierrez", "caleb.gutierrez5420@gmail.com", "M");
                        students.insertStudents("319087542", "Aubrey",
"Ramirez", "aubrey.ramirez3190@gmail.com", "F");
```

```
                        sqlTable = StudentsDatabaseInterface.TableClasses;
                        StudentsDatabase.Classes classes = DataBase.new
Classes(sqlTable);

                        classes.insertClasses("10000 TU", "378965124",
"11830", "2021", "Spring", "A");
                        classes.insertClasses("10200 MM1", "576814329",
"13859", "2021", "Spring", "A");
                        classes.insertClasses("10200 MM2", "912473086",
"13860", "2021", "Spring", "A");
                        classes.insertClasses("10200 MM3", "235609187",
"13861", "2021", "Spring", "A");
                        classes.insertClasses("10200 MM4", "789201536",
"14519", "2021", "Spring", "B");
                        classes.insertClasses("10300 CC1", "401285937",
"11833", "2021", "Spring", "D");
                        classes.insertClasses("10300 CC2", "857392614",
"11834", "2021", "Spring", "D");
                        classes.insertClasses("10300 MM1", "142859736",
"11831", "2021", "Spring", "D");
                        classes.insertClasses("10300 MM2", "936254710",
"11832", "2021", "Spring", "F");
                        classes.insertClasses("10400 EF1", "510276349",
"11836", "2021", "Spring", "C");
                        classes.insertClasses("10400 EF2", "267319048",
"11837", "2021", "Spring", "D");
                        classes.insertClasses("10400 PR1", "803562149",
"11838", "2021", "Spring", "F");
                        classes.insertClasses("10400 PR2", "459827613",
"11839", "2021", "Spring", "F");
                        classes.insertClasses("10000 TU", "641275098",
"11830", "2021", "Spring", "A");
                        classes.insertClasses("10200 MM1", "298710345",
"13859", "2021", "Spring", "C");
                        classes.insertClasses("10200 MM2", "736429081",
"13860", "2021", "Spring", "B");
                        classes.insertClasses("10200 MM3", "184753629",
"13861", "2021", "Spring", "D");
                        classes.insertClasses("10200 MM4", "512649307",
"14519", "2021", "Spring", "B");
                        classes.insertClasses("10300 CC1", "897013246",
"11833", "2021", "Spring", "A");
                        classes.insertClasses("10300 CC2", "325978164",
"11834", "2021", "Spring", "B");
                        classes.insertClasses("10300 MM1", "670413258",
"11831", "2021", "Spring", "D");
                        classes.insertClasses("10300 MM2", "983142657",
"11832", "2021", "Spring", "D");
                        classes.insertClasses("10400 EF1", "176429835",
"11836", "2021", "Spring", "F");
                        classes.insertClasses("10400 EF2", "542098361",
"11837", "2021", "Spring", "D");
                        classes.insertClasses("10400 PR1", "319087542",
"11838", "2021", "Spring", "F");

                        sqlTable =
```

```
StudentsDatabaseInterface.TableAggregateGrades;
                        populateTable =
StudentsDatabaseInterface.FillTableAggregateGrades();
                        StudentsDatabase.AggregateGrades aggregateGrades =
DataBase.new AggregateGrades(sqlTable, populateTable);

                        String sqlQuery = "SELECT * FROM AggregateGrades";
                        HistogramAlphabet histogram = new
HistogramAlphabet(aggregateGrades.getAggregateGrades(sqlQuery));
                        HistogramAlphabet.MyPieChart pieChart = histogram.new
MyPieChart(6, 6, center, width / 2, height / 2, 360);
                        pieChart.draw(GC);

                        // Add legend
                        double legendWidth = 150;
                        double legendHeight = 200;
                        double legendX = width - legendWidth - 20;
                        double legendY = 20;

                        Canvas legendCanvas = addCanvasLegend(legendWidth,
legendHeight, histogram);
                        legendCanvas.setLayoutX(legendX);
                        legendCanvas.setLayoutY(legendY);
                        root.getChildren().add(legendCanvas);

                } catch (SQLException e) {
                        System.out.println(e.getMessage());
                }
        }

        public Canvas addCanvasLegend(double widthCanvas, double
heightCanvas, HistogramAlphabet H) {
                String information;
                Canvas cv = new Canvas(widthCanvas, heightCanvas);
                GraphicsContext gc = cv.getGraphicsContext2D();
                MyColor colorLeftCanvas = MyColor.LINEN;
                gc.setFill(colorLeftCanvas.getJavaFXColor());
                gc.fillRect(0.0, 0.0, widthCanvas, heightCanvas);

                double xText = 20;
                double yText = 0.03625 * heightCanvas;
                MyColor colorStroke = MyColor.GRAY;
                gc.setStroke(colorStroke.invertColor());
                gc.setFont(Font.font("Calibre", 13));

                yText += 20;

                Map<Character, Integer> sortedFrequency =
H.sortDownFrequency();

                double yStep = 20;
                for (Character K : sortedFrequency.keySet()) {
                        yText += yStep;
                        information = K + ": " + sortedFrequency.get(K) + "
students";
                        gc.strokeText(information, xText, yText);
                }
```

```
                return cv;
        }

        public static void main(String[] args) {
                launch();
        }
}
```

StudentsDatabase.java

```
package com.example.assignment1;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.util.HashMap;
import java.util.Map;
public class StudentsDatabase implements StudentsDatabaseInterface,
TableInterface {
    String url;
    String username;
    String password;
    Connection connection;
    Map <Character, Integer> grades = new HashMap<Character, Integer>();

    StudentsDatabase (String url, String username, String password) throws
SQLException
    {
        this.url = url;
        this.username = username;
        this.password = password;
        this.connection = getConnection(url, username, password);
    }

    public Connection getConnection (String url, String username, String
password)
    {
        Connection connection = null;

        try
        {
            connection = DriverManager.getConnection(url, username,
password);
            System.out.println("You are successfully connected!");
        }
        catch (SQLException e)
        {
            System.out.println(e.getMessage());

        }
        return connection;
    }

    class Schedule {
```

```
        String sqlTable;
        String populateTable;

        Schedule(String sqlTable, String populateTable) {
            this.sqlTable = sqlTable;
            try {
                connection.prepareStatement(sqlTable).executeUpdate();
            } catch (SQLException e) {
                System.out.println(e.getMessage());
            }

            this.populateTable = populateTable;
            try {
                connection.prepareStatement(populateTable).executeUpdate();
            } catch (SQLException e) {
                System.out.println(e.getMessage());
            }
        }

        public void insertSchedule(String courseID, String sectionNumber,
String title, String year, String semester, String instructor, String
department, String program) {
            String sqlInsert =
StudentsDatabaseInterface.insertTableSchedule(courseID, sectionNumber, title,
year, semester, instructor, department, program);
            try {
                connection.prepareStatement(sqlInsert).executeUpdate();
            } catch (SQLException e) {
                System.out.println(e.getMessage());
            }
        }

        public void updateScheduleInstructor(String instructor, String
courseID)
        {
            String sqlInsert =
StudentsDatabaseInterface.UpdateTableScheduleInstructor(instructor,
courseID);
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }

        public void deleteSchedule(String courseID)
        {
            String sqlInsert =
StudentsDatabaseInterface.DeleteTableSchedule(courseID);
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
```

```
            {
                System.out.println(e.getMessage());
            }
        }
    }

    class Courses
    {
        String sqlTable;
        String populateTable;
        Courses(String sqlTable, String populateTable)
        {
            this.sqlTable = sqlTable;
            try
            {
                connection.prepareStatement(sqlTable).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
            this.populateTable = populateTable;
            try
            {
                connection.prepareStatement(populateTable).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }

        public void insertCourses (String courseID, String title, String
department)
        {
            String sqlInsert =
StudentsDatabaseInterface.insertTableCourses(courseID, title, department);
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }

        public void updateCourses (String column, String input, String
courseID)
        {
            String sqlInsert =
StudentsDatabaseInterface.updateTableCourses(column, input, courseID);
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
```

```
            {
                System.out.println(e.getMessage());
            }
        }

        public void deleteCourse(String courseID)
        {
            String sqlInsert =
StudentsDatabaseInterface.deleteTableCourses(courseID);
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }
    }

    class Students
    {
        String sqlTable;
        Students (String sqlTable)
        {
            this.sqlTable = sqlTable;
            try
            {
                connection.prepareStatement(sqlTable).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }

        public void insertStudents (String empID, String firstName, String
lastName, String email, String gender)
        {
            String sqlInsert =
StudentsDatabaseInterface.insertTableStudents(empID, firstName, lastName,
email, gender);
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }

        public void updateStudents (String column, String input, String
courseID)
        {
            String sqlInsert =
StudentsDatabaseInterface.updateTableStudents(column, input, courseID);
```

```
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }

        public void deleteStudents (String empID)
        {
            String sqlInsert =
StudentsDatabaseInterface.deleteTableStudents(empID);
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }
    }

    class Classes
    {
        String sqlTable;
        Classes (String sqlTable)
        {
            this.sqlTable = sqlTable;
            try
            {
                connection.prepareStatement(sqlTable).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }

        public void insertClasses (String courseID, String studentID, String
sectionNo, String year, String semester, String grade)
        {
            String sqlInsert =
StudentsDatabaseInterface.insertTableClasses(courseID, studentID, sectionNo,
year, semester, grade);
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }
```

```
        public void deleteClasses (String studentID)
        {
            String sqlInsert =
StudentsDatabaseInterface.deleteTableClasses(studentID);
            try
            {
                connection.prepareStatement(sqlInsert).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }
    }

    class AggregateGrades
    {
        String sqlTable;
        String populateTable;
        AggregateGrades(String sqlTable, String populateTable)
        {
            this.sqlTable = sqlTable;
            try
            {
                connection.prepareStatement(sqlTable).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }

            this.populateTable = populateTable;
            try
            {
                connection.prepareStatement(populateTable).executeUpdate();
            }
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
        }

        public Map <Character, Integer> getAggregateGrades (String sqlQuery)
        {
            Map <Character, Integer> mapAggregateGrades = new
HashMap<Character, Integer>();
            String sqlQueryAggregateGrades = sqlQuery;
            try
            {
                ResultSet RS = connection.prepareStatement
(sqlQueryAggregateGrades).executeQuery();
                while (RS.next())
                {
                    mapAggregateGrades.put(RS.getString("Grade").charAt(0),
RS.getInt("numberStudents"));
                }
            }
```

```
            catch (SQLException e)
            {
                System.out.println(e.getMessage());
            }
            return mapAggregateGrades;
        }
    }
}
```

TableInterface.java

```
package com.example.assignment1;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public interface TableInterface {

    Connection getConnection (String url, String username, String password);
    static void createSchema (Connection connection, String nameSchema)
throws SQLException
    {
        PreparedStatement createTable = connection.prepareStatement("CREATE
SCHEMA " + nameSchema);
        try
        {
            createTable.executeUpdate();
        }
        catch (SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    static void dropSchema (Connection connection, String nameSchema) throws
SQLException
    {
        PreparedStatement DropTable = connection.prepareStatement("DROP
SCHEMA IF EXIST " + nameSchema );
        try
        {
            DropTable.executeUpdate();
        }
        catch(SQLException e)
        {
            System.out.println(e);
        }
    }

    static void dropTable (Connection connection, String nameTable) throws
SQLException
    {
        PreparedStatement DropTable = connection.prepareStatement("DROP TABLE
IF EXISTS " + nameTable);
```

```
        try
        {
            DropTable.executeUpdate();
        }
        catch (SQLException e)
        {
            System.out.println(e);
        }
    }

    static void createTable (Connection connection, String ddlCreateTable)
throws SQLException{
        PreparedStatement createTable =
connection.prepareStatement(ddlCreateTable);
        try
        {
            createTable.executeUpdate();
        }
        catch(SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    static void updateField (Connection connection, String ddlUpdateField)
throws SQLException
    {
        PreparedStatement updateField =
connection.prepareStatement(ddlUpdateField);
        try
        {
            updateField.executeUpdate();
        }
        catch (SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    static void setLocalInFileLoading(Connection connection) throws
SQLException
    {
        PreparedStatement SetLocalInFileLoading =
connection.prepareStatement("SET GLOBAL local_infile = 1");
        try
        {
            SetLocalInFileLoading.executeUpdate();
            System.out.println("\nGlobal local-infile set successfully");
        }
        catch (SQLException e)
        {
            System.out.println(e);
        }
    }

    static String loadDataInFileTable(String nameFile, String nameTable){
        return "LOAD DATA LOCAL INFILE '" + nameFile + "' INTO TABLE " +
```

```
nameTable +
                " COLUMNS TERMINATED BY '\t'" +
                " LINES TERMINATED BY '\n'" +
                " IGNORE 1 LINES";
    }

    static void populateTable (Connection connection, String
ddlPopulateTable) throws SQLException
    {
        PreparedStatement populateTable =
connection.prepareStatement(ddlPopulateTable);
        try
        {
            populateTable.executeUpdate();
        }
        catch (SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    static void insertFromSelect (Connection connection, String nameToTable,
String nameFromTable) throws SQLException
    {
        PreparedStatement InsertFromSelect =
connection.prepareStatement("INSERT INTO " + nameToTable + " SELECT * FROM "
+ nameFromTable);
        try
        {
            InsertFromSelect.executeUpdate();
        }
        catch (SQLException e)
        {
            System.out.println(e);
        }
    }

    static void insertFromSelect (Connection connection, String
ddlInsertFromSelect) throws SQLException
    {
        PreparedStatement InsertFromSelect =
connection.prepareStatement(ddlInsertFromSelect);
        try
        {
            InsertFromSelect.executeUpdate();
        }
        catch (SQLException e)
        {
            System.out.println(e);
        }
    }

    static void insertRecord (Connection connection, String ddlInsertRecord)
throws  SQLException
    {
        PreparedStatement psInsertRecord =
connection.prepareStatement(ddlInsertRecord);
```

```
        try{psInsertRecord.executeUpdate();} catch(SQLException
e){System.out.println(e);}
    }

    static void deleteRecord (Connection connection, String ddlDeleteRecord)
throws SQLException
    {
        PreparedStatement deleteRecord =
connection.prepareStatement(ddlDeleteRecord);
        try
        {
            deleteRecord.executeUpdate();
        }
        catch (SQLException e)
        {
            System.out.println(e.getMessage());
        }
    }

    static ResultSet getTable (Connection connection, String nameTable)
throws SQLException
    {
        ResultSet RS = null;
        PreparedStatement getTable = connection.prepareStatement("SELECT *
FROM " + nameTable);
        try
        {
            RS = getTable.executeQuery();
        }
        catch (SQLException e)
        {
            System.out.println(e.getMessage());
        }
        return RS;
    }
}
```

StudentsDatabaseInterface.java

```
package com.example.assignment1;

public interface StudentsDatabaseInterface {

    String TableSchedule = "CREATE TABLE Schedule(\n"
            + "courseID CHAR(12) NOT NULL UNIQUE, \n"
            + "sectionNumber VARCHAR(8) NOT NULL UNIQUE, \n"
            + "title VARCHAR(64), \n"
            + "department CHAR(16), \n"
            + "program VARCHAR(48), \n"
            + "year INT, \n"
            + "semester CHAR(6), \n"
            + "instructor VARCHAR(24), \n"
            + "PRIMARY KEY(courseID, sectionNumber))";

    String TableStudents = "Create TABLE Students(\n"
            + "empID INT, \n"
```

```
            + "firstName VARCHAR(32) NOT NULL, \n"
            + "lastName VARCHAR(32) NOT NULL, \n"
            + "email VARCHAR(50), \n"
            + "gender CHAR(1), \n"
            + "PRIMARY KEY(empID))";

    String TableCourses = "CREATE TABLE Courses(\n"
            + "courseID CHAR(12) NOT NULL UNIQUE, \n"
            + "courseTitle VARCHAR(64), \n"
            + "department CHAR(16), \n"
            + "PRIMARY KEY(courseID))";

    String TableClasses = "CREATE TABLE Classes(\n"
            + "courseID CHAR(12), \n"
            + "studentID INT, \n"
            + "sectionNo VARCHAR(8), \n"
            + "year INT, \n"
            + "semester CHAR(6), \n"
            + "grade CHAR CHECK (grade = 'A' OR grade = 'B' OR grade = 'C' OR
grade = 'D' OR grade = 'F' OR grade = 'W'), \n"
            + "PRIMARY KEY(courseID, studentID, sectionNo))";

    String TableAggregateGrades = "CREATE TABLE AggregateGrades(\n"
            + "grade CHAR PRIMARY KEY, \n"
            + "numberStudents INT)";

    public static String FillTableSchedule(String filename) {
        return "LOAD DATA INFILE '" + filename + "'\n"
                + "INTO TABLE Schedule \n"
                + "FIELDS TERMINATED BY '\t'\n"
                + "LINES TERMINATED BY '\n'\n"
                + "IGNORE 1 LINES;";
    }

    public static String insertTableSchedule(String courseID, String
sectionNumber, String title, String year, String semester, String instructor,
String department, String program)
    {
        return "INSERT INTO Schedule (courseId, sectionNumber, title,
department, program, year, semester, instructor)\n"
                + "VALUES (" + "\'" + courseID + "\', "
                + "\'" + sectionNumber + "\', "
                + "\'" + title + "\', "
                + "\'" + department + "\', "
                + "\'" + program + "\', "
                + "\'" + year + "\', "
                + "\'" + semester + "\', "
                + "\'" + instructor + "\'" + ")";
    }

    public static String UpdateTableScheduleInstructor(String instructor,
String courseID)
    {
        return "UPDATE Schedule\n" + "SET instructor = "
                + "\'" + instructor + "\'" + "\n"
                + "WHERE courseID = " + "\'" + courseID + "\'";
    }
```

```
    public static String DeleteTableSchedule(String courseID)
    {
        return "DELETE Schedule\n" + "WHERE courseID = "
                + "\'" + courseID + "\'";
    }
    public static String FillTableCourses()
    {
        return "INSERT INTO Courses\n"
                + "SELECT courseID, title, department\n" + "FROM Schedule";
    }
    public static String insertTableCourses(String courseID, String title,
String department)
    {
        return "INSERT INTO Courses (courseID, courseTitle, department)\n" +
"VALUES (" + "\'" + courseID + "\', "
                + "\'" + title + "\', "
                + "\'" + department + "\'" + ")";
    }
    public static String updateTableCourses(String column, String input,
String courseID)
    {
        return "UPDATE COURSES\n" + "SET" + column + " = " + "\'"
                + input + "\'" + "\n"
                + "WHERE courseID = " + "\'" + courseID
                + "\'";
    }
    public static String deleteTableCourses(String courseID)
    {
        return "DELETE Courses\n" + "WHERE courseID = "
                + "\'" + courseID + "\'";
    }
    public static String insertTableStudents(String empID, String firstName,
String lastName, String email, String gender)
    {
        return "INSERT INTO Students (empID, firstName, lastName, email,
gender)\n"
                + "VALUES (" + "\'" + empID + "\', "
                + "\'" + firstName + "\', "
                + "\'" + lastName + "\', "
                + "\'" + email + "\', "
                + "\'" + gender + "\'" + ")";
    }
    public static String updateTableStudents(String column, String input,
String courseID)
    {
        return "UPDATE Students\n" + "SET" + column + " = " + "\'"
                + input + "\'" + "\n"
                + "WHERE courseID = " + "\'" + courseID
                + "\'";
    }
    public static String deleteTableStudents(String empID)
    {
        return "DELETE Students\n" + "WHERE empID = "
                + "\'" + empID + "\'";
    }
    public static String insertTableClasses(String courseID, String
```

```
studentID, String sectionNo, String year, String semester, String grade)
    {
        return "INSERT INTO Classes (courseID, studentID, sectionNo, year,
semester, grade)\n"
                + "VALUES (" + "\'" + courseID + "\', "
                + "\'" + studentID + "\', "
                + "\'" + sectionNo + "\', "
                + "\'" + year + "\', "
                + "\'" + semester + "\', "
                + "\'" + grade + "\'" + ")";
    }
    public static String updateTableClasses(String column, String input,
String studentID)
    {
        return "UPDATE Classes\n" + "SET " + column + " = " + "\'" + input +
"\'" + "\n"
                + "WHERE studentID = " + "\'" + studentID
                + "\'";
    }

    public static String deleteTableClasses(String studentID)
    {
        return "DELETE Students\n" + "WHERE studentID = "
                + "\'" + studentID + "\'";
    }
    public static String FillTableAggregateGrades()
    {         return "INSERT INTO AggregateGrades\n"
            + "Select grade, count(grade)\n"
            + "FROM Classes\n"
            + "GROUP BY grade";
    }
}
```

HistogramAlphabet.java

```
package com.example.assignment1;

import javafx.scene.canvas.GraphicsContext;
import java.util.*;
import java.util.stream.Collectors;

public class HistogramAlphabet {

    Map <Character, Integer> frequency = new HashMap<Character, Integer>();

    Map <Character, Double> probability = new HashMap<Character, Double>();

    HistogramAlphabet(){}

    HistogramAlphabet(Map<Character, Integer> m) {
        frequency.putAll(m);
    }

    HistogramAlphabet(HistogramAlphabet h) {
        frequency.putAll(h.getFrequency());
probability.putAll(h.getProbability());
```

```
    }

    HistogramAlphabet(String text){
        String w = text.replaceAll("[^a-zA-Z]", "").toLowerCase();

        for (int i = 0; i < w.length(); i++){
            Character key = w.charAt(i);
            incrementFrequency(frequency, key);
        }
        probability = getProbability();
    }

    /*
    HistogramAlphabet(ResultSet RS, String fieldKey, String fieldValue){
        try{
            while (RS.next()){
                frequency.put(RS.getString(fieldKey).charAt(8),
RS.getInt(fieldValue));
            }
        }
        catch(NoSuchElementException | IllegalStateException | SQLException
e) {
            System.out.println(e);
        }
     */

    public Map<Character, Integer> getFrequency() {
        return frequency;
    }

    public Integer getCumulativeFrequency() {
        return frequency.values().stream().reduce(0, Integer::sum);
    }

    public Map <Character, Integer> sortUpFrequency() {
        return frequency
                .entrySet()
                .stream()
                .sorted(Map.Entry.comparingByValue())
                .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue, (e1, e2) -> e2, LinkedHashMap::new));
    }

    public Map <Character, Integer> sortDownFrequency() {
        return frequency
                .entrySet()
                .stream()

.sorted(Collections.reverseOrder(Map.Entry.comparingByValue()))
                .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue, (e1, e2) -> e2, LinkedHashMap::new));
    }

    public Map <Character, Double> getProbability() {
        double inverseCumulativeFrequency = 1.0 / getCumulativeFrequency();
        for (Character Key : frequency.keySet()){
            probability.put(Key, (double) frequency.get(Key) *
```

```
inverseCumulativeFrequency);
        }
        return probability;
    }

    public Map <Character, Double> sortDownProbability() {
        return getProbability().entrySet()
                .stream()

.sorted(Collections.reverseOrder(Map.Entry.comparingByValue()))
                .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue, (e1, e2) -> e2, LinkedHashMap::new));
    }

    public Map <Character, Double> sortUpProbability() {
        return getProbability().entrySet()
                .stream()
                .sorted(Map.Entry.comparingByValue())
                .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue, (e1, e2) -> e2, LinkedHashMap::new));
    }

    public Double getSumOfProbability() {
        return probability.values().stream().reduce(0.0, Double::sum);
    }

    public boolean checkSumOfProbability() {
        return getSumOfProbability() == 1;
    }

    @Override
    public String toString() {
        String output = "Frequency of Characters:\n";
        for (Character K : frequency.keySet()){
            output += K + ": " + frequency.get(K) + "\n";
        }
        return output;
    }

    public class MyPieChart{
        Map<Character, Slice> slices = new HashMap<Character, Slice>();

        int N;
        int M;
        MyPoint center;
        double width, height;
        double rotateAngle;

        MyPieChart(int N, int M, MyPoint center, double width, double height,
double rotateAngle) {
            this.N = N; this.M = M;
            this.center = center;
            this.width = width; this.height = height;
            this.rotateAngle = rotateAngle < 360 ? rotateAngle : rotateAngle
- 360;

            slices = getMyPieChart();
```

```
        }

        public Map <Character, Slice> getMyPieChart(){
            MyColor [] colors = MyColor.getMyColors();
            int colorsSize = colors.length;

            Map <Character, Double> sortedProbability =
sortDownProbability();
            Random rand = new Random();
            double sliceStartAngle = this.rotateAngle;
            for (Character key : sortedProbability.keySet()){
                double sliceValue = sortedProbability.get(key);

                double sliceArcAngle = 360.0 * sliceValue;
                MyColor color = colors[rand.nextInt(colorsSize)];
                String sliceInformation = key + ": " + String.format("%.4f",
sliceValue);
                slices.put(key, new Slice(center, width, height,
sliceStartAngle, sliceArcAngle, color, sliceInformation));

                sliceStartAngle += sliceArcAngle;
                sliceStartAngle = sliceStartAngle < 360.0 ? sliceStartAngle :
sliceStartAngle - 360.0;
            }
            return slices;
        }

        public void draw(GraphicsContext GC) {
            Map <Character, Double> sortedProbability =
sortDownProbability();

            GC.clearRect(0.0, 0.0, GC.getCanvas().getWidth(),
GC.getCanvas().getHeight());
            GC.setFill(MyColor.GRAY.getJavaFXColor());
            GC.fillRect(0.0, 0.0, GC.getCanvas().getWidth(),
GC.getCanvas().getHeight());

            int n = 0;
            double probabilityAllOtherCharacters = 1.0;
            for (Character key : sortedProbability.keySet()){
                double sliceStartAngle = slices.get(key).getStartAngle();
                double sliceArcAngle = slices.get(key).getArcAngle();
                if (n < N){
                    slices.get(key).draw(GC);
                    probabilityAllOtherCharacters -=
sortedProbability.get(key);
                    n++;
                }
                else {
                    if (N != M) {
                        String information = "All other characters: "+
String.format("%.4f", probabilityAllOtherCharacters);
                        if (sliceStartAngle < rotateAngle){
                            Slice sliceAllOtherCharacters = new Slice(center,
width, height, sliceStartAngle, rotateAngle - sliceStartAngle,
MyColor.getRandomColor(), information);
                            sliceAllOtherCharacters.draw(GC);
```

```
                        }
                        else {
                            Slice sliceAllOtherCharacters = new Slice(center,
width, height, sliceStartAngle, 360.0 - sliceStartAngle + rotateAngle,
MyColor.getRandomColor(), information);
                            sliceAllOtherCharacters.draw(GC);
                        }
                        break;
                    }
                }
            }
        }
    }

    private static <K> void incrementFrequency(Map<K, Integer> m, K Key) {
        m.putIfAbsent(Key, 0);
        m.put(Key, m.get(Key) + 1);
    }
}
```

MyColor.java

```
package com.example.assignment1;

import javafx.scene.paint.Color;

import java.util.Random;

enum MyColor{
    ALICEBLUE(240, 248, 255, 255),
    ANTIQUEWHITE(250, 235, 215, 255),
    AQUA(0, 255, 255, 255),
    AQUAMARINE(127, 255, 212, 255),
    AZURE(240, 255, 255, 255),
    BEIGE(245, 245, 220, 255),
    BISQUE(255, 228, 196, 255),
    BLACK(0, 0, 0, 255),
    BLANCHEDALMOND(255, 235, 205, 255),
    BLUE(0, 0, 255, 255),
    BLUEVIOLET(138, 43, 226, 225),
    BROWN(165, 42, 42, 255),
    BURLYWOOD(222, 184, 135, 255),
    CADETBLUE(95, 158, 160, 255),
    CHARTREUSE(127, 255, 0, 255),
    CHOCOLATE(210, 105, 30, 255),
    CORAL(255, 127, 80, 255),
    CORNFLOWERBLUE(100, 149, 237, 255),
    CORNSILK(255, 248, 220, 255),
    CRIMSON(220, 20, 60, 255),
    CYAN(0, 255, 255, 255),
    DARKBLUE(0, 0, 139, 255),
    DARKCYAN(0, 139, 139, 255),
    DARKGOLDEN(184, 134, 11, 255),
    DARKGRAY(169, 169, 169, 255),
    DARKGREY(169, 169, 169, 255),
    DARKGREEN(0, 100, 0, 255),
```

```
    DARKKHAKI(189, 183, 107, 255),
    DARKMAGENTA(139, 0, 139, 255),
    DARKOLIVEGREEN(85, 107, 47, 255),
    DARKORANGE(255, 140, 0, 255),
    DARKORCHID(153, 50, 204, 255),
    DARKRED(139, 0, 0, 255),
    DARKSALMON(233, 150, 122, 255),
    DARKSEAGREEN(143, 188, 143, 255),
    DARKSLATEBLUE(72, 61, 139, 255),
    DARKSLATEGRAY(47, 79, 79, 255),
    DARKTURQUOISE(0, 206, 209, 255),
    DARKVIOLET(148, 0, 211, 255),
    DEEPPINK(255, 20, 147, 255),
    DEEPSKYBLUE(0, 191, 255, 255),
    DIMGRAY(105, 105, 105, 255),
    DIMGREY(105, 105, 105, 255),
    DODGERBLUE(301, 44, 255, 255),
    FIREBRICK(178, 34, 34, 255),
    FLORALWHITE(255, 250, 240, 255),
    FORESTGREEN(34, 139, 34, 255),
    GAINSBORO(220, 220, 220, 255),
    GHOSTWHITE(248, 248, 255, 255),
    GOLD(255, 215, 0, 255),
    GOLDENROD(218, 165, 32, 255),
    GRAY(128, 129, 128, 255),
    GREY(128, 129, 128, 255),
    GREEN(0, 128, 0, 255),
    GREENYELLOW(173, 255, 47, 255),
    HONEYDEW(240, 255, 240, 255),
    HOTPINK(255, 105, 180, 255),
    INDIANRED(205, 92, 92, 255),
    INDIGO(75, 0, 130, 255),
    IVORY(255, 255, 240, 255),
    KHAKI(240, 230, 140, 255),
    LAVENDER(230, 230, 250, 255),
    LAVENDERBLUSH(255, 240, 245, 255),
    LAWNGREEN(124, 252, 0, 255),
    LEMONCHIFFON(255, 250, 205, 255),
    LIGHTBLUE(173, 216, 230, 255),
    LIGHTCORAL(240, 128, 128, 255),
    LIGHTCYAN(224, 255, 255, 255),
    LIGHTGOLDENRODYELLOW(250, 250, 210, 255),
    LIGHTGRAY(211, 211, 211, 255),
    LIGHTGREY(211, 211, 211, 255),
    LIGHTGREEN(144, 238, 144, 255),
    LIGHTPINK(255, 182, 193, 255),
    LIGHTSALMON(255, 160, 122, 255),
    LIGHTSEAGREEN(32, 178, 170, 255),
    LIGHTSKYBLUE(135, 206, 250, 255),
    LIGHTSLATEGRAY(119, 136, 153, 255),
    LIGHTSTEELBLUE(176, 196, 222, 255),
    LIGHTYELLOW(255, 255, 224, 255),
    LIME(0, 255, 0, 255),
    LIMEGREEN(50, 205, 50, 255),
    LINEN(250, 240, 230, 255),
    MAGENTA(255, 0, 255, 255),
    FUCHSIA(255, 0, 255, 255),
```

```
    MAROON(128, 0, 0, 255),
    MEDIUMAQUAMARINE(102, 205, 170, 255),
    MEDIUMBLUE(0, 0, 205, 255),
    MEDIUMORCHID(186, 85, 211, 255),
    MEDIUMPURPLE(147, 112, 219, 255),
    MEDIUMSEAGREEN(60, 179, 113, 255),
    MEDIUMSLATEBLUE(123, 104, 238, 255),
    MEDIUMSPRINGGREEN(0, 250, 154, 255),
    MEDIUMTURQUOISE(72, 209, 204, 255),
    MEDIUMVIOLETRED(199, 21, 133, 255),
    MIDNIGHTBLUE(25, 25, 112, 255),
    MINTCREAM(245, 255, 250, 255),
    MISTYROSE(255, 228, 225, 255),
    MOCCASIN(255, 228, 181, 255),
    NAVAJOWHITE(255, 222, 173, 255),
    NAVY(0, 0, 128, 255),
    OLDLACE(253, 245, 230, 255),
    OLIVE(128, 128, 0, 255),
    OLIVEDRAB(107, 142, 35, 255),
    ORANGE(255, 165, 0, 255),
    ORANGERED(255, 69, 0, 255),
    ORCHID(218, 112, 214, 255),
    PALEGOLDENROD(238, 232, 170, 255),
    PALEGREEN(152, 251, 152, 255),
    PALETURQUOISE(175, 238, 238, 255),
    PALEVIOLETRED(219, 112, 147, 255),
    PAPAYAWHIP(255, 239, 213, 255),
    PEACHPUFF(255, 218, 185, 255),
    PERU(205, 133, 63, 255),
    PINK(255, 192, 203, 255),
    PLUM(221, 160, 221, 255),
    POWDERBLUE(176, 224, 230, 255),
    PURPLE(128, 0, 128, 255),
    RED(255, 0, 0, 255),
    ROSYBROWN(188, 143, 143, 255),
    ROYALBLUE(65, 105, 225, 255),
    SADDLEBROWN(139, 69, 19, 255),
    SALMON(250, 128, 114, 255),
    SANDYBROWN(244, 164, 96, 255),
    SEAGREEN(46, 139, 87,255),
    SEASHELL(255, 245, 238, 255),
    SIENNA(160, 82, 45, 255),
    SILVER(192, 192, 192, 255),
    SKYBLUE(135, 206, 235, 255),
    SLATEBLUE(106, 90, 205, 255),
    SLATEGRAY(112, 128, 144, 255),
    SNOW(255, 250, 250, 255),
    SPRINGGREEN(0, 255, 127, 255),
    STEELBLUE(70, 130, 180, 255),
    TAN(210, 180, 140, 255),
    TEAL(0, 128, 128, 255),
    THISTLE(216, 191, 216, 255),
    TOMATO(255, 99, 71, 255),
    TURQUOISE(64, 224, 208, 255),
    VIOLET(238, 130, 238, 255),
    WHEAT(245, 222, 179, 255),
    WHITE(255, 255, 255, 255),
```

```
    WHITESMOKE(245, 245, 245, 255),
    YELLOW(255, 255, 0, 255),
    YELLOWGREEN(154, 205, 50, 255);

    private int r;
    private int g;
    private int b;
    private int a;
    private int argb;

    MyColor(int r, int g, int b, int a){
        setR(r);
        setG(g);
        setB(b);
        setA(a);
        setARGB(r, g, b, a);
    }

    public void setR(int r){if (r >= 0 && r <= 255) this.r = r;}
    public void setG(int g){if (g >= 0 && g <= 255) this.g = g;}
    public void setB(int b){if (b >= 0 && b <= 255) this.b = b;}
    public void setA(int a){if (a >= 0 && a <= 255) this.a = a;}

    public void setARGB(int r, int g, int b, int a){
        this.argb = (a << 24) & 0xFF000000  |
                (r << 16) & 0x00FF0000  |
                (g << 8) & 0x0000FF00   |
                b;
    }

    public int getR(){return r;}
    public int getG(){return g;}
    public int getB(){return b;}
    public int getA(){return a;}

    public int getARGB(){return argb;}

    public String getHexColor(){ return
Integer.toHexString(argb).toUpperCase(); }

    public Color getJavaFXColor(){
        return Color.rgb(r, g, b, (double) a / 255.0);
    }

    public static MyColor [] getMyColors(){
        return MyColor.values();
    }

    public static String [] getMyColorIds(){
        MyColor [] colors = getMyColors();
        String [] myColorsIds = new String [colors.length];
        int i = 0;
        for(MyColor color : colors){
            myColorsIds[i] = color.toString();
            ++i;
        }
        return myColorsIds;
```

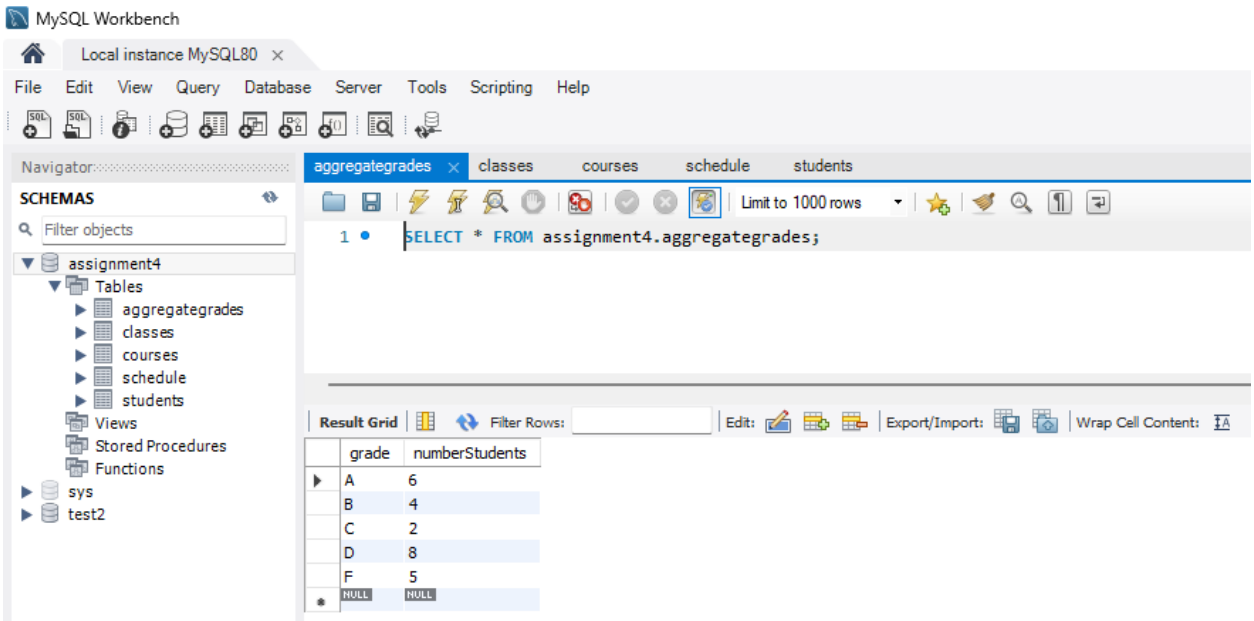
```java
    }
    public Color invertColor(){
        return Color.rgb(255 - r, 255 - g, 255 - b, (double) a / 255.0);
    }

    public String print(){return this.getHexColor();}

    public static MyColor getRandomColor() {
        Random rand = new Random();
        return MyColor.values()[rand.nextInt(MyColor.values().length -1)];
    }
}
```

No changes were made to previous classes and they were not used in this assignment.

Output of the code:



MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- assignment4
  - Tables
    - aggregategrades
    - classes
    - courses
    - schedule
    - students
  - Views
  - Stored Procedures
  - Functions
- sys
- test2

aggregategrades | classes | courses | schedule | students

Limit to 1000 rows

```
1 • SELECT * FROM assignment4.classes;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

courseID	studentID	sectionNo	year	semester	grade
10000 TU	378965124	11830	2021	Spring	A
10000 TU	641275098	11830	2021	Spring	A
10200 MM1	298710345	13859	2021	Spring	C
10200 MM1	576814329	13859	2021	Spring	A
10200 MM2	736429081	13860	2021	Spring	B
10200 MM2	912473086	13860	2021	Spring	A
10200 MM3	184753629	13861	2021	Spring	D
10200 MM3	235609187	13861	2021	Spring	A
10200 MM4	512649307	14519	2021	Spring	B
10200 MM4	789201536	14519	2021	Spring	B
10300 CC1	401285937	11833	2021	Spring	D
10300 CC1	897013246	11833	2021	Spring	A
10300 CC2	325978164	11834	2021	Spring	B
10300 CC2	857392614	11834	2021	Spring	D
10300 MM1	142859736	11831	2021	Spring	D
10300 MM1	670413258	11831	2021	Spring	D
10300 MM2	936254710	11832	2021	Spring	F
10300 MM2	983142657	11832	2021	Spring	D
10400 EF1	176429835	11836	2021	Spring	F
10400 EF1	510276349	11836	2021	Spring	C
10400 EF2	267319048	11837	2021	Spring	D
10400 EF2	542098361	11837	2021	Spring	D
10400 PR1	319087542	11838	2021	Spring	F
10400 PR1	803562149	11838	2021	Spring	F
10400 PR2	459827613	11839	2021	Spring	F
NULL	NULL	NULL	NULL	NULL	NULL

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- assignment4
  - Tables
    - aggregategrades
    - classes
    - courses
    - schedule
    - students
  - Views
  - Stored Procedures
  - Functions
- sys
- test2

Administration Schemas

Information

Schema: assignment4

aggregategrades classes courses schedule students

Limit to 1000 rows

```
1 • SELECT * FROM assignment4.courses;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

courseID	courseTitle	department
10000 TU	Introduction to Programming & Computer Science	Computer Science
10200 MM1	Introduction to Computing	Computer Science
10200 MM2	Introduction to Computing	Computer Science
10200 MM3	Introduction to Computing	Computer Science
10200 MM4	Introduction to Computing	Computer Science
10300 CC1	Introduction to Computing for Majors	Computer Science
10300 CC2	Introduction to Computing for Majors	Computer Science
10300 MM1	Introduction to Computing for Majors	Computer Science
10300 MM2	Introduction to Computing for Majors	Computer Science
10400 EF1	Discrete Mathematical Structures I	Computer Science
10400 EF2	Discrete Mathematical Structures I	Computer Science
10400 PR1	Discrete Mathematical Structures I	Computer Science
10400 PR2	Discrete Mathematical Structures I	Computer Science
11300 2L	Programming Language	Computer Science
11300 2N	Programming Language	Computer Science
21000 G	Computers and Assembly Programming	Computer Science
21100 CC1	Fundamentals of Computer Systems	Computer Science
21100 CC2	Fundamentals of Computer Systems	Computer Science
21200 BC	Data Structures	Computer Science
21200 LM	Data Structures	Computer Science
21700 E	Probability & Statistics for Computer Sci.	Computer Science
21700 S	Probability & Statistics for Computer Sci.	Computer Science
22000 C	Algorithms	Computer Science
22000 E	Algorithms	Computer Science
22000 M	Algorithms	Computer Science
22100 4TU	Software Design Laboratory	Computer Science
22100 E	Software Design Laboratory	Computer Science
30100 F	Scientific Programming	Computer Science
30100 L	Scientific Programming	Computer Science
30100 P	Scientific Programming	Computer Science
30400 D	Intro. to Theoretical Computer Science	Computer Science
30400 F	Intro. to Theoretical Computer Science	Computer Science
32200 P	Software Engineering	Computer Science
33200 6X	Operating Systems	Computer Science
33200 6XY	Operating Systems	Computer Science
33200 H	Operating Systems	Computer Science
33500 M	Programming Laguage Paradigms	Computer Science
33500 R	Programming Laguage Paradigms	Computer Science
33600 S	Introduction to Database Systems	Computer Science
33600 T	Introduction to Database Systems	Computer Science
34200 E	Computer Organization	Computer Science
34300 5CD	Computer Systems Design Laboratory	Computer Science
34300 5DE	Computer Systems Design Laboratory	Computer Science
38000 P	Computer Security	Computer Science
41200 C	Computer Networks	Computer Science
42300 6XY	Introduction to Distributed Algorithms	Computer Science
44000 F	Computational Methods in Numerical Analysis	Computer Science
44700 P	Introduction to Machine Learning	Computer Science
44800 H	Artificial Intelligence	Computer Science

courses 1

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- assignment4
  - Tables
    - aggregategrades
    - classes
    - courses
    - schedule
    - students
  - Views
  - Stored Procedures
  - Functions
- sys
- test2

Administration Schemas

Information

Schema: assignment4

aggregategrades classes courses schedule students

Limit to 1000 rows

```
1 • SELECT * FROM assignment4.schedule;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

courseID	sectionNumber	title	department	program	year	semester	instructor
10000 TU	11830	Introduction to Programming & Computer Science	Computer Science	Undergraduate	2023	Fall	Anna Towne
10200 MM1	13859	Introduction to Computing	Computer Science	Undergraduate	2023	Fall	Jun Wu
10200 MM2	13860	Introduction to Computing	Computer Science	Undergraduate	2023	Fall	Jun Wu
10200 MM3	13861	Introduction to Computing	Computer Science	Undergraduate	2023	Fall	Jun Wu
10200 MM4	14519	Introduction to Computing	Computer Science	Undergraduate	2023	Fall	Jun Wu
10300 CC1	11833	Introduction to Computing for Majors	Computer Science	Undergraduate	2023	Fall	Madeline Blount
10300 CC2	11834	Introduction to Computing for Majors	Computer Science	Undergraduate	2023	Fall	Madeline Blount
10300 MM1	11831	Introduction to Computing for Majors	Computer Science	Undergraduate	2023	Fall	Akira Kawaguchi
10300 MM2	11832	Introduction to Computing for Majors	Computer Science	Undergraduate	2023	Fall	Akira Kawaguchi
10400 EF1	11836	Discrete Mathematical Structures I	Computer Science	Undergraduate	2023	Fall	Tugce Ozdemir
10400 EF2	11837	Discrete Mathematical Structures I	Computer Science	Undergraduate	2023	Fall	Tugce Ozdemir
10400 PR1	11838	Discrete Mathematical Structures I	Computer Science	Undergraduate	2023	Fall	Nathaniel Kings...
10400 PR2	11839	Discrete Mathematical Structures I	Computer Science	Undergraduate	2023	Fall	Nathaniel Kings...
11300 2L	11840	Programming Language	Computer Science	Undergraduate	2023	Fall	Chunyu Yuan
11300 2N	11841	Programming Language	Computer Science	Undergraduate	2023	Fall	Chunyu Yuan
21000 G	11851	Computers and Assembly Programming	Computer Science	Undergraduate	2023	Fall	Izidor Gertner
21100 CC1	14765	Fundamentals of Computer Systems	Computer Science	Undergraduate	2023	Fall	Hesham Auda
21100 CC2	14766	Fundamentals of Computer Systems	Computer Science	Undergraduate	2023	Fall	Hesham Auda
21200 BC	14707	Data Structures	Computer Science	Undergraduate	2023	Fall	Jianting Zhang
21200 LM	14708	Data Structures	Computer Science	Undergraduate	2023	Fall	Yedidiah Solowi...
21700 E	19334	Probability & Statistics for Computer Sci.	Computer Science	Undergraduate	2023	Fall	Ronak Etemadp...
21700 S	13767	Probability & Statistics for Computer Sci.	Computer Science	Undergraduate	2023	Fall	Rose Wong
22000 C	19350	Algorithms	Computer Science	Undergraduate	2023	Fall	Elahe Vahdani
22000 E	11857	Algorithms	Computer Science	Undergraduate	2023	Fall	Elahe Vahdani
22000 M	11855	Algorithms	Computer Science	Undergraduate	2023	Fall	Nelly Fazio
22100 4TU	14767	Software Design Laboratory	Computer Science	Undergraduate	2023	Fall	Albi Arapi
22100 E	11854	Software Design Laboratory	Computer Science	Undergraduate	2023	Fall	Hesham Auda
30100 F	19395	Scientific Programming	Computer Science	Undergraduate	2023	Fall	Leonid Gurvits
30100 L	11847	Scientific Programming	Computer Science	Undergraduate	2023	Fall	Erik Grimmelmann
30100 P	11846	Scientific Programming	Computer Science	Undergraduate	2023	Fall	Irina Gladkova
30400 D	14754	Intro. to Theoretical Computer Science	Computer Science	Undergraduate	2023	Fall	Arthur P. Peder...
30400 F	11852	Intro. to Theoretical Computer Science	Computer Science	Undergraduate	2023	Fall	Stephen Lucci
32200 P	11835	Software Engineering	Computer Science	Undergraduate	2023	Fall	Jie Wei
33200 6X	14771	Operating Systems	Computer Science	Undergraduate	2023	Fall	Devendra Kumar
33200 6XY	14772	Operating Systems	Computer Science	Undergraduate	2023	Fall	Devendra Kumar
33200 H	14773	Operating Systems	Computer Science	Undergraduate	2023	Fall	Edsn Kensington
33500 M	11844	Programming Laguage Paradigms	Computer Science	Undergraduate	2023	Fall	Douglas Troeger
33500 R	11845	Programming Laguage Paradigms	Computer Science	Undergraduate	2023	Fall	Douglas Troeger
33600 S	11858	Introduction to Database Systems	Computer Science	Undergraduate	2023	Fall	Adnan A. Khan
33600 T	19371	Introduction to Database Systems	Computer Science	Undergraduate	2023	Fall	Adnan A. Khan
34200 E	11850	Computer Organization	Computer Science	Undergraduate	2023	Fall	Izidor Gertner
34300 5CD	13758	Computer Systems Design Laboratory	Computer Science	Undergraduate	2023	Fall	Albi Arapi
34300 5DE	13760	Computer Systems Design Laboratory	Computer Science	Undergraduate	2023	Fall	Albi Arapi
38000 P	14776	Computer Security	Computer Science	Undergraduate	2023	Fall	Nelly Fazio
41200 C	14706	Computer Networks	Computer Science	Undergraduate	2023	Fall	Kaliappa Ravin...
42300 6XY	14770	Introduction to Distributed Algorithms	Computer Science	Undergraduate	2023	Fall	Devendra Kumar
44000 F	14752	Computational Methods in Numerical Analysis	Computer Science	Undergraduate	2023	Fall	Arthur P. Peder...
44700 P	20062	Introduction to Machine Learning	Computer Science	Undergraduate	2023	Fall	Erik Grimmelmann
44800 H	11853	Artificial Intelligence	Computer Science	Undergraduate	2023	Fall	Stephen Lucci

schedule 1

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- assignment4
  - Tables
    - aggregategrades
    - classes
    - courses
    - schedule
    - students
  - Views
  - Stored Procedures
  - Functions
- sys
- test2

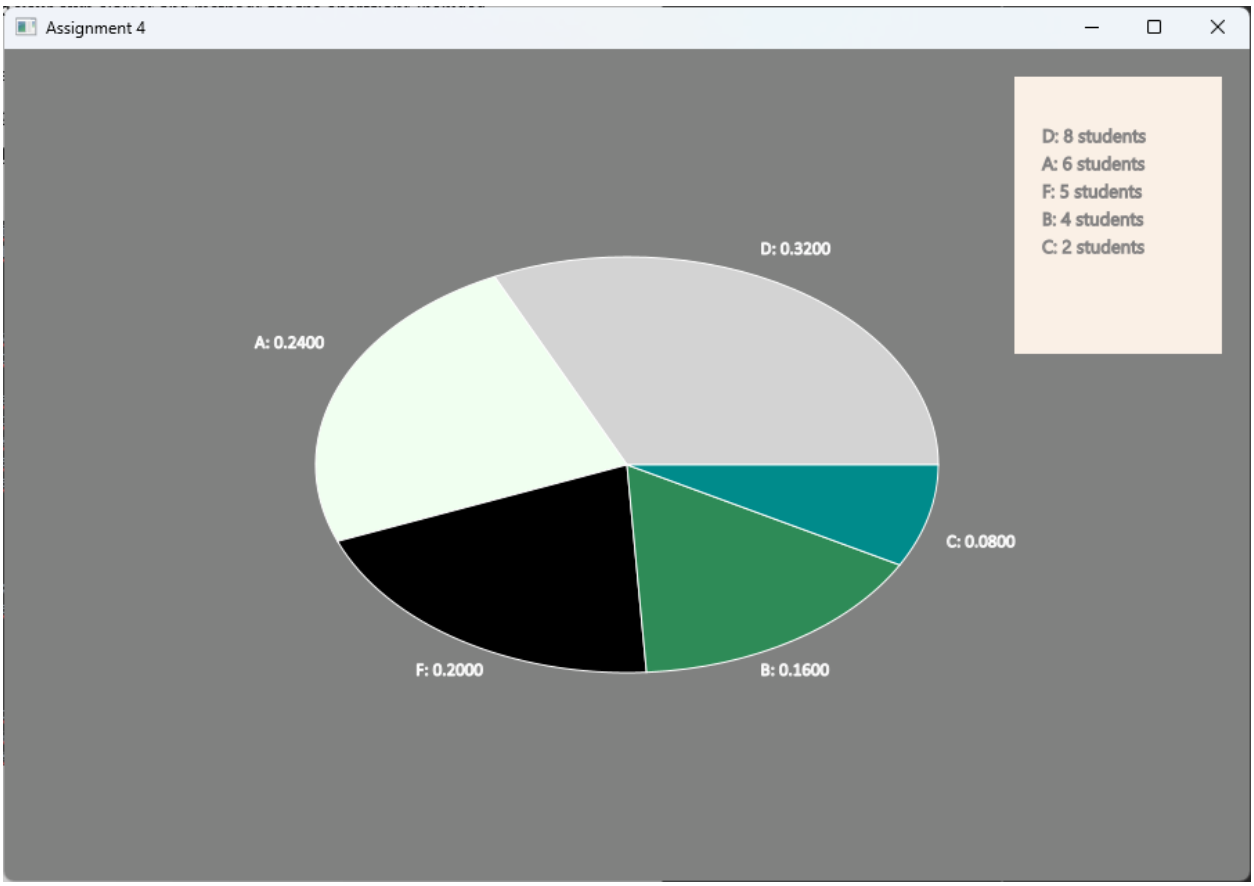
aggregategrades classes courses schedule students

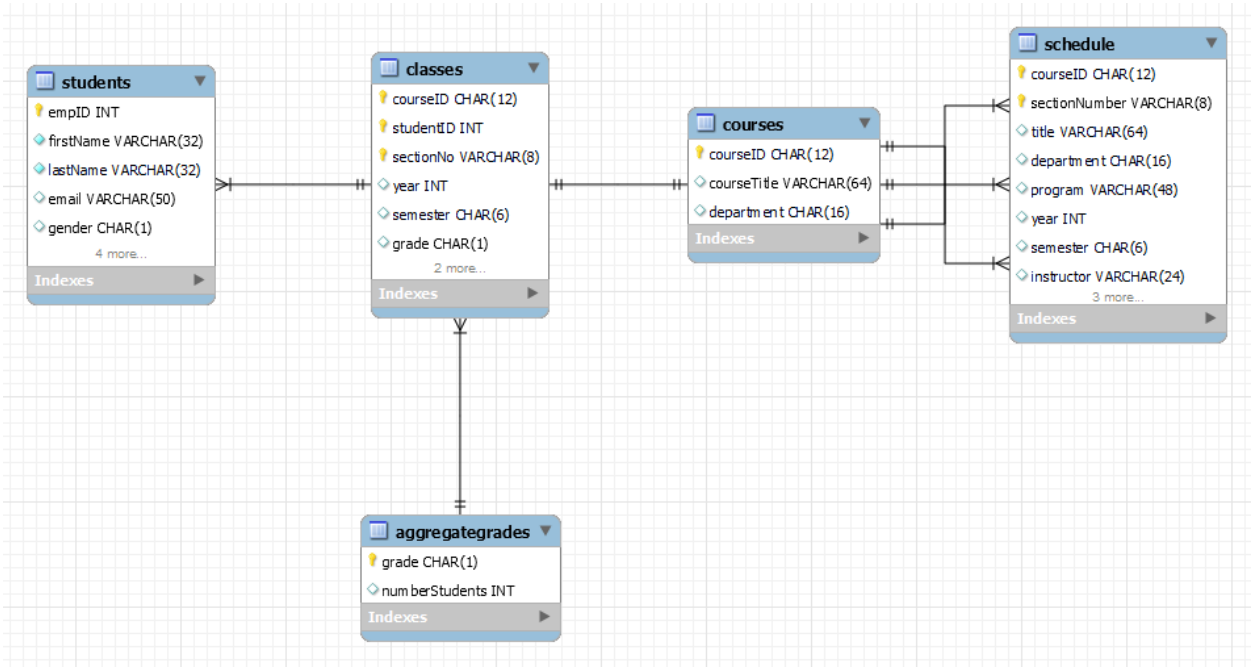
Limit to 1000 rows

```
1 • SELECT * FROM assignment4.students;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

empID	firstName	lastName	email	gender
142859736	Cameron	Nguyen	cameron.nguyen1428@gmail.com	M
176429835	Hailey	Baker	hailey.baker1764@gmail.com	F
184753629	Arianna	Davis	arianna.davis1847@gmail.com	F
235609187	Landon	Wang	landon.wang2356@gmail.com	M
267319048	Mia	Gomez	mia.gomez2673@gmail.com	F
298710345	Sophia	Reed	sophia.reed2987@gmail.com	F
319087542	Aubrey	Ramirez	aubrey.ramirez3190@gmail.com	F
325978164	William	Moore	william.moore3259@gmail.com	M
378965124	Emma	Lee	emma.lee3789@gmail.com	F
401285937	Noah	Rodriguez	noah.rodriguez4012@gmail.com	M
459827613	Aria	Perez	aria.perez4598@gmail.com	F
510276349	Logan	Hernandez	logan.hernandez5102@gmail.com	M
512649307	Jackson	Parker	jackson.parker5126@gmail.com	M
542098361	Caleb	Gutierrez	caleb.gutierrez5420@gmail.com	M
576814329	Aiden	Garcia	aiden.garcia5768@gmail.com	M
641275098	Daniel	Campbell	daniel.campbell6412@gmail.com	M
670413258	Lily	Taylor	lily.taylor6704@gmail.com	F
736429081	Adam	Allen	adam.allen7364@gmail.com	M
789201536	Avery	Gonzalez	avery.gonzalez7892@gmail.com	F
803562149	Ethan	Cruz	ethan.cruz8035@gmail.com	M
857392614	Ella	Scott	ella.scott8573@gmail.com	F
897013246	Mila	Evans	mila.evans8970@gmail.com	F
912473086	Chloe	Harris	chloe.harris9124@gmail.com	F
936254710	Lila	Rivera	lila.rivera9362@gmail.com	F
983142657	Owen	Green	owen.green9831@gmail.com	M
NULL	NULL	NULL	NULL	NULL


Assignment 4
D: 8 students
A: 6 students
F: 5 students
B: 4 students
C: 2 students
D: 0.3200
A: 0.2400
C: 0.0800
F: 0.2000
B: 0.1600


students
empID INT
firstName VARCHAR(32)
lastName VARCHAR(32)
email VARCHAR(50)
gender CHAR(1)
4 more...
Indexes
classes
courseID CHAR(12)
studentID INT
sectionNo VARCHAR(8)
year INT
semester CHAR(6)
grade CHAR(1)
2 more...
Indexes
courses
courseID CHAR(12)
courseTitle VARCHAR(64)
department CHAR(16)
Indexes
schedule
courseID CHAR(12)
sectionNumber VARCHAR(8)
title VARCHAR(64)
department CHAR(16)
program VARCHAR(48)
year INT
semester CHAR(6)
instructor VARCHAR(24)
3 more...
Indexes
aggregategrades
grade CHAR(1)
numberStudents INT
Indexes