The University of Texas at Dallas

School System Database Project

Members:

Kyle Bolin - kjb19000

Derik Flores - frf180000

Sein Oh - sho180001

Luke Wells - law190001

Jake Yoo - jxy190013

Gabe Vogel - gjv180000

CS 4347.502

Database Systems

Prof. Jalal Omer

12/5/2022

Table of Contents

Introduction **3**

System Requirements **3**

System Description 3

Context Diagram 3

Functional Requirements 4

Non-Functional Requirements 4

Conceptual Design of the Database **6**

Entity-Relationship Diagram 6

Business Rules and Integrity Constraints 7

Logical Database Schema **10**

Database Schema 10

SQL Statements for Construction 11

Functional Dependencies and Database Normalization **14**

The Database System **15**

Suggestions on Database Tuning **18**

Additional Queries and Views **19**

User Application Interface **22**

Student View 22

Faculty View 24

Staff View 26

Conclusions and Future Work **27**

References **28**

Appendix **29**

Introduction

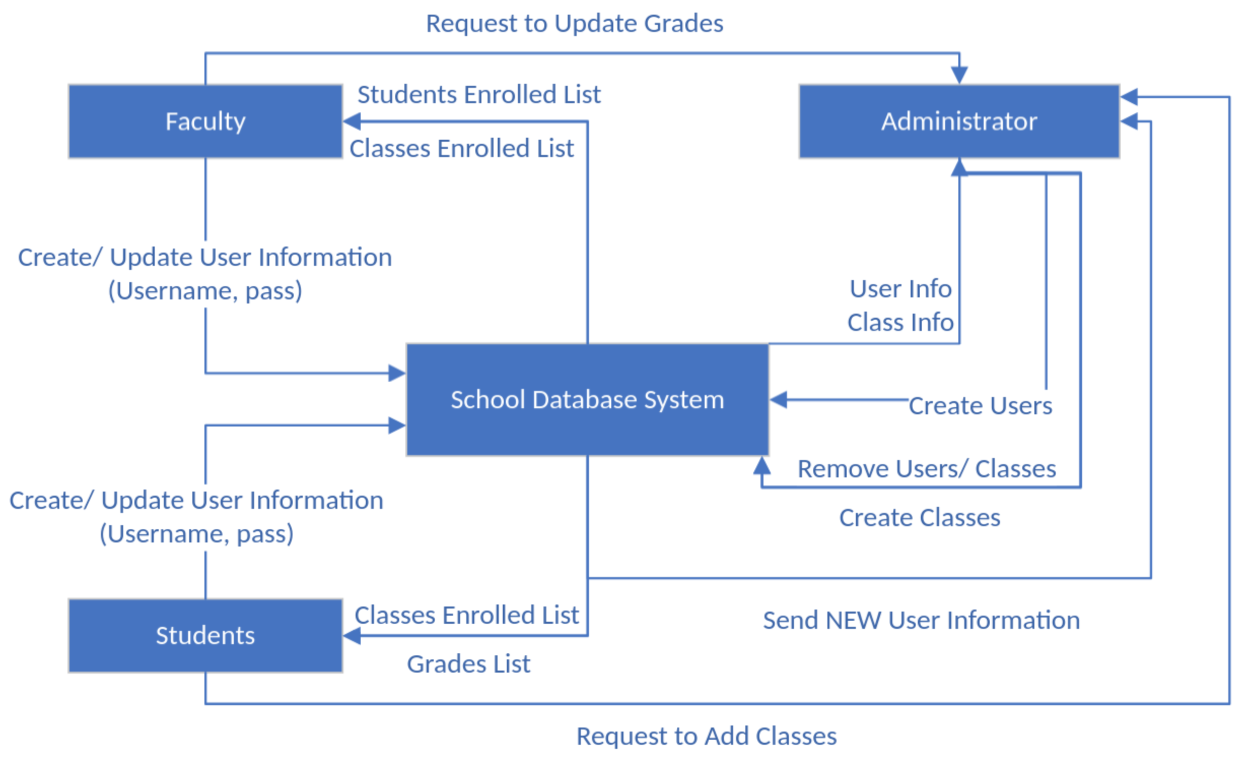
This report outlines the design and implementation of a school system database made for the use of students and faculty alike. Users will be given permissions to utilize the database based on their role, and have access to tools that suit their needs. Major sections will be outlined in the report, along with supporting subsections.

System Requirements

System Description

The system is a school portal that supports users of various levels in a school system. Students can register for classes and view their grades, faculty can see students in their classes and update grades. Staff are administrators that have more robust access to the database

Context Diagram



Functional Requirements

- User Requirements
  - Login and Logout
  - Edit contact information and address
- Student Requirements
  - Look at available classes
  - Check grades for classes
- Faculty Requirements
  - Look at assigned classes
  - Assign grades for each student in a class
  - View student list for a class
- Staff Requirements
  - Staff are database administrators who have full access to the database
  - Staff should be able to enroll students into classes
- System Requirements
  - Provide a GUI for users to log in and provide the appropriate functions for the user based on their type

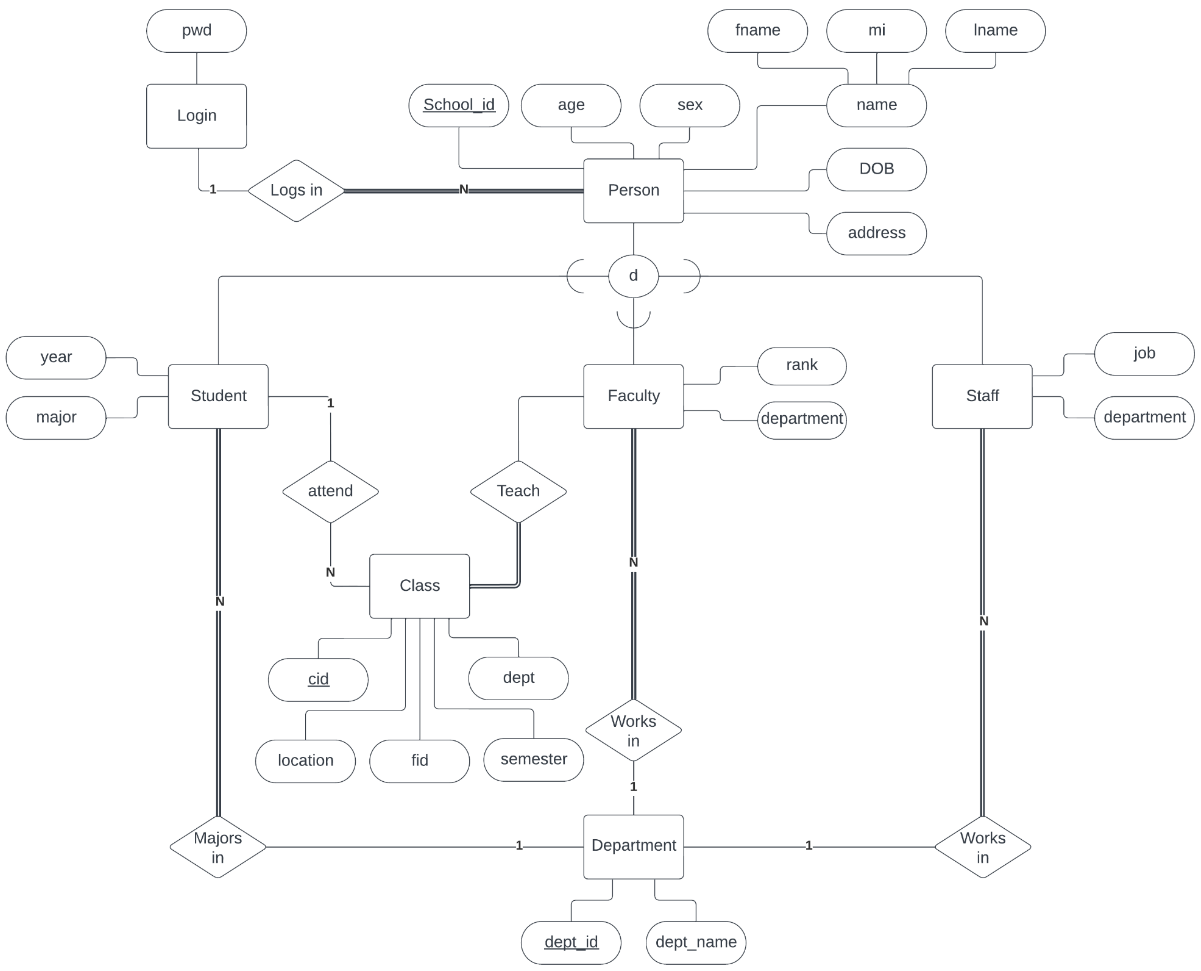
Non-Functional Requirements

- The system must be written in JAVA
- Runs on popular user systems like Mac or Windows
- Must have a super-user account to maintain database and software
- Uses PostgreSQL
- Not vulnerable to SQL injection

- Has a response time faster than 10 seconds
- Able to support up to 200 students

Conceptual Design of the Database

Entity-Relationship Diagram



Business Rules and Integrity Constraints

- Person
  - School_id should be an integer representing a person’s unique ID
  - Age should be an integer with a max length of 3
  - Sex should be a character value M or F
  - DOB should be a string of max length 10 with the format mm/dd/yyyy
  - Address should be a string of max length 50, formatted in standard US addressing (building number, street name, city, state, zip code)
- Name
  - Fname should be a string of characters with max length 20, with the first character uppercase
  - Lname should be a string of characters with max length 20, with the first character uppercase
  - Mi should be a string of characters with max length 20
- Student
  - School_id should be a foreign key of Person (student_id), cascading changes both on delete and on update
  - Year should be an integer to list the year that the student is in
  - Major should be a string of max length 30 representing a student’s major
  - GPA is a floating point number that represents a student’s GPA
- Faculty
  - School_id should be a foreign key of Person (student_id), cascading changes both on delete and on update

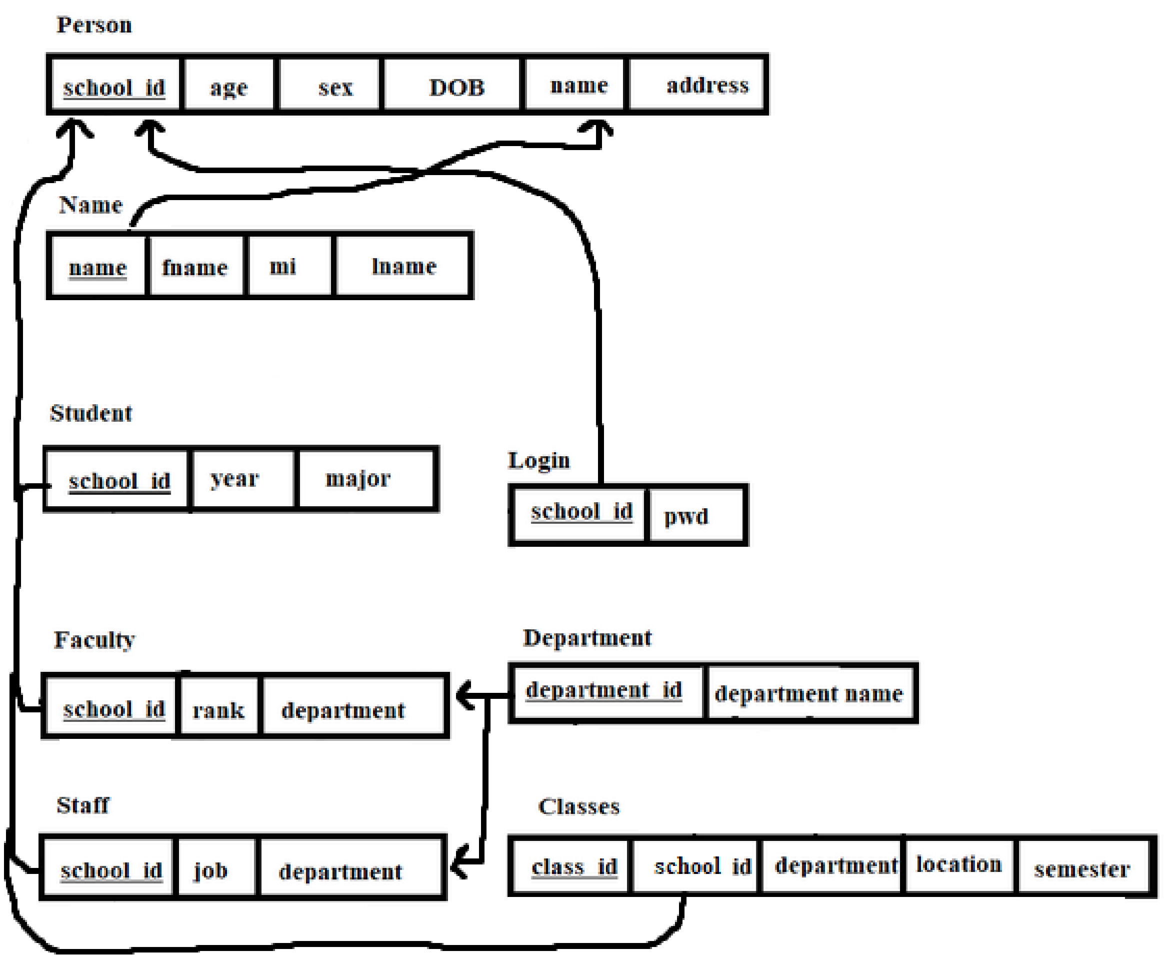
    - Rank should be a string of max length 20 showing a faculty member’s rank
    - Department_id should be a foreign key of Person (student_id), restricting deletion changes, but cascading on update
- Staff
    - School_id should be a foreign key of Person (student_id), cascading changes both on delete and on update
    - Department_id should be a foreign key of Department (department_id), restricting deletion changes, but cascading on update
    - Job should be a string of max length 20 that describes a staff member’s job
- Department
    - Department_id should be an integer representing the department’s ID
    - Department_name is a string of max length 30 that describes a department’s name
- Login
    - School_id should be a foreign key of Person (student_id), cascading changes both on delete and on update
    - Password should be a string of encrypted characters with max length 20
- Classes
    - Class_id should be an integer representing a course’s unique ID
    - faculty _id is a foreign key of Faculty (faculty_id), set to NULL on delete and cascading on update.
    - Department_id should be a foreign key of Department (department_id), restricting deletion changes, but cascading on update
    - Location should be a string with max length 20, containing the classroom number

that the class will be held in

- Semester should be a string of max length 3, with the first character representing the Winter, Spring, Summer, or Fall semesters followed by the last two digits of the year.

Logical Database Schema

Database Schema



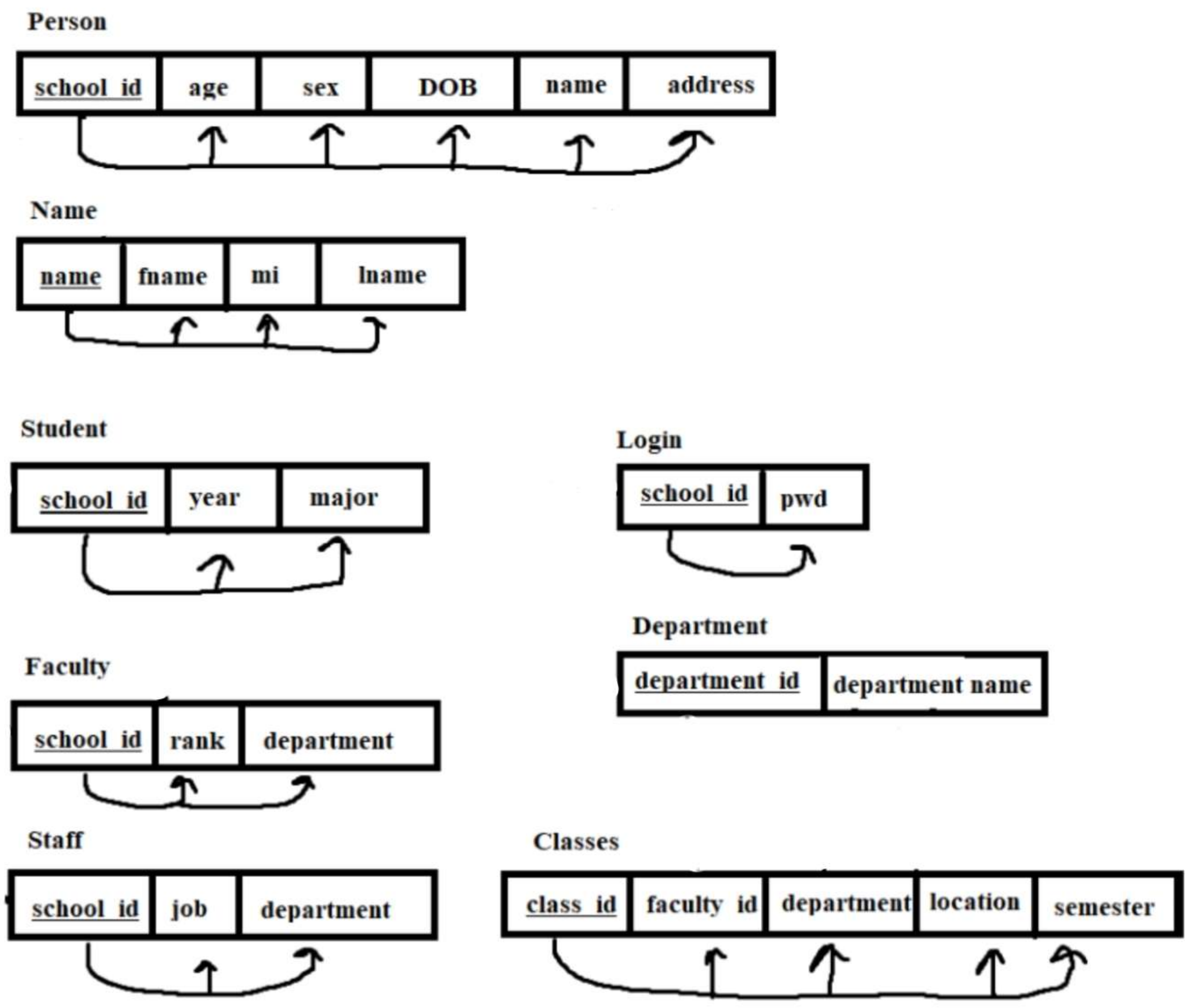
SQL Statements for Construction

```
DROP TABLE if exists Classes;

DROP TABLE if exists Login;

DROP TABLE if exists Staff;

DROP TABLE if exists Faculty;

DROP TABLE if exists Student;

DROP TABLE if exists Name;

DROP TABLE if exists Department;

DROP TABLE if exists Person;


CREATE TABLE Person (

      school_id int not null,

      age int null,

      sex varchar(1) null,

      DOB varchar(10) null,

      address varchar(50) null,


      PRIMARY KEY(school_id)

);


CREATE TABLE Department (

      department_id int not null,

      department_name varchar(30),
      PRIMARY KEY(department_id)

);


CREATE TABLE Name (

      school_id int not null,

      fname varchar(20) not null,

      mi varchar(20) null,

      lname varchar(20) not null,


      FOREIGN KEY(school_id) REFERENCES Person (school_id)

      ON DELETE CASCADE ON UPDATE CASCADE

);


CREATE TABLE Student (

      school_id int not null,

      year int not null,

      major varchar(30) not null,

      GPA int not null,


      FOREIGN KEY(school_id) REFERENCES Person (school_id)

      ON DELETE CASCADE ON UPDATE CASCADE

);


CREATE TABLE Faculty (
        school_id int not null,

        rank varchar(20) null,

        department int not null,


        FOREIGN KEY(school_id)

        REFERENCES Person (school_id)

        ON DELETE CASCADE ON UPDATE CASCADE,


        FOREIGN KEY(department)

        REFERENCES Department (department_id)

        ON DELETE restrict ON UPDATE CASCADE

);


CREATE TABLE Staff (

        school_id int not null,

        job varchar(20) not null,

        department int not null,


        FOREIGN KEY(school_id)

        REFERENCES Person (school_id)

        ON DELETE CASCADE ON UPDATE CASCADE,


        FOREIGN KEY(department)

        REFERENCES Department (department_id)
```

```
      ON DELETE restrict ON UPDATE CASCADE
);

CREATE TABLE Login (
      school_id int not null,
      pwd varchar(20) not null,

      FOREIGN KEY(school_id)
      REFERENCES Person (school_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE classes (
      class_id int not null,
      faculty_id int not null,
      department int not null,
      location varchar(20) not null,
      semester varchar(20) not null,

      PRIMARY KEY(class_id),

      FOREIGN KEY(department)
      REFERENCES Department (department_id)
      ON DELETE restrict ON UPDATE CASCADE,
        FOREIGN KEY(faculty_id)

        REFERENCES Person (school_id)

        ON DELETE restrict ON UPDATE CASCADE

);
```

Functional Dependencies and Database Normalization

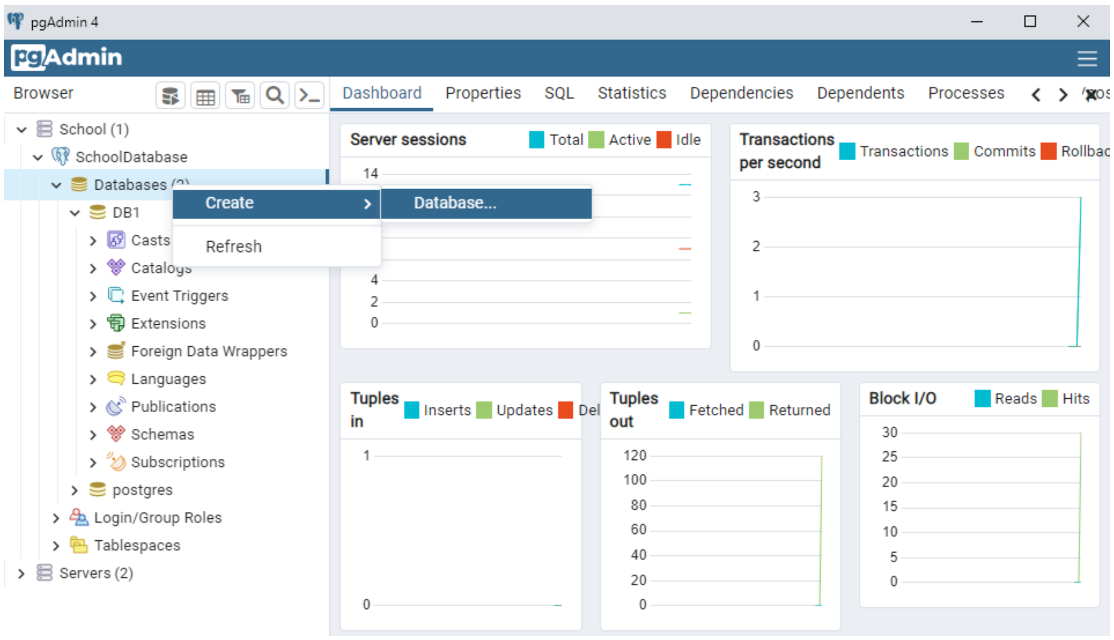
All the tables are already in 3NF, so no further normalizations are required.



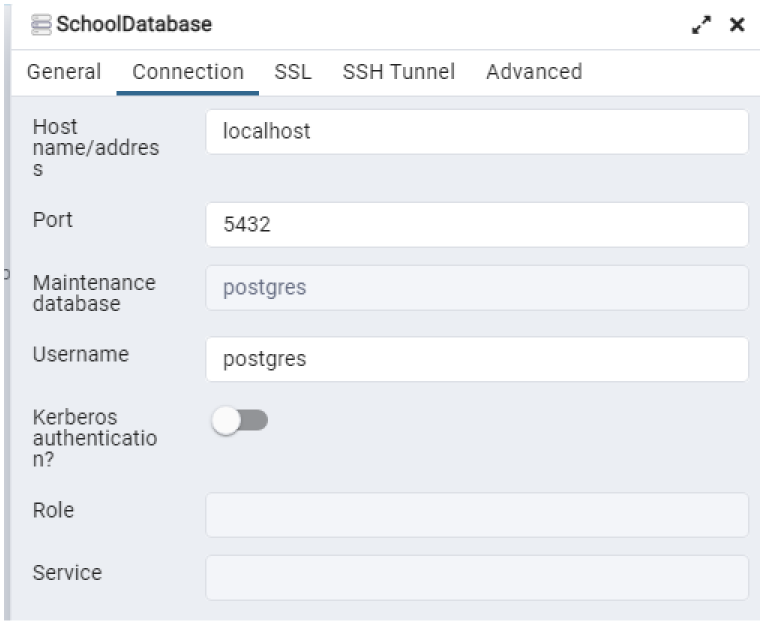
The Database System

Our system uses PostGRESQL, along with pgAdmin to start the database.

A new database can be initialized as follows.



Note the connection properties that the server has, in this instance of PostGRE/pgAdmin, the default port is 5432 and the default password has been set to 1234.

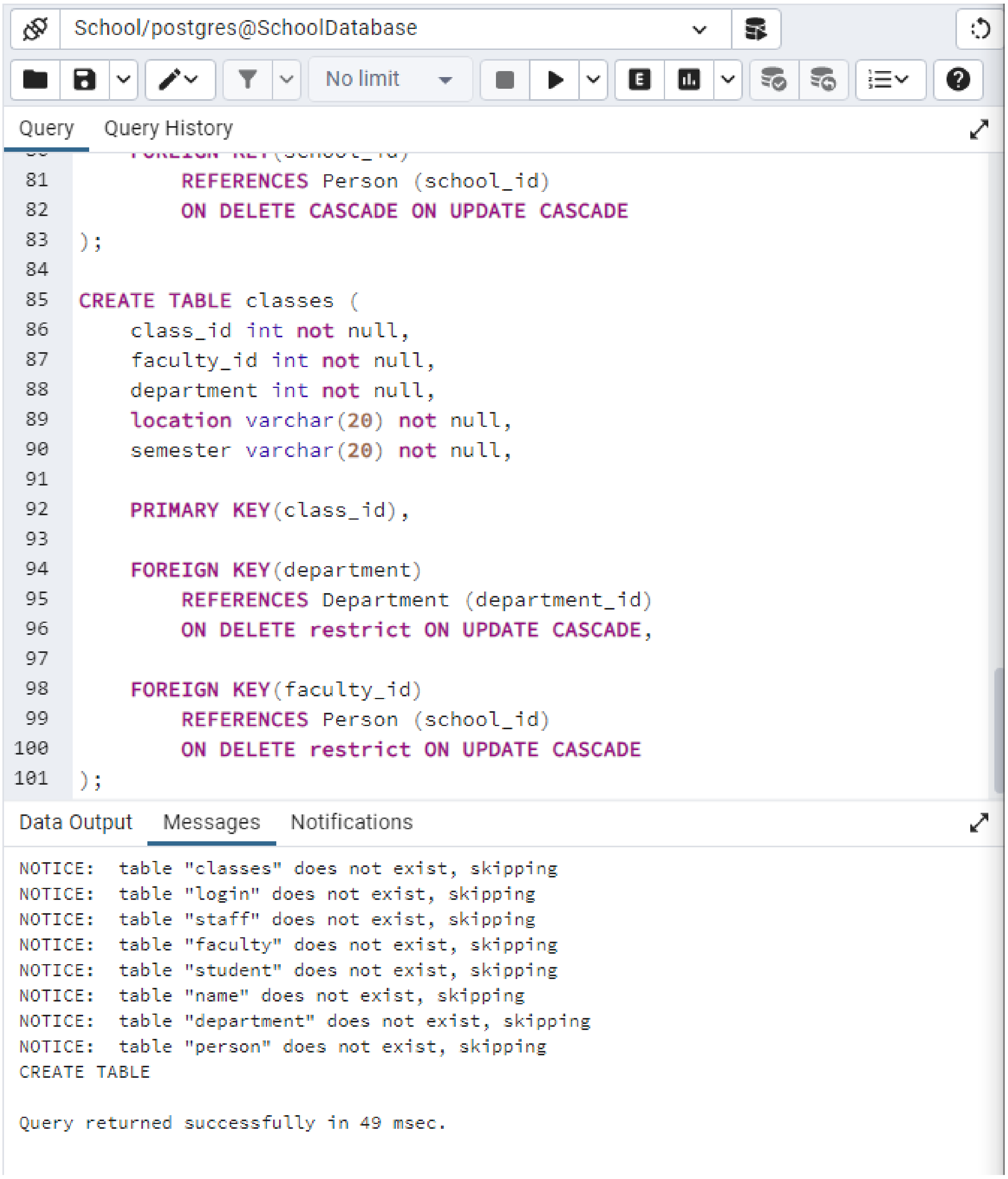


SchoolDatabase

General | Connection | SSL | SSH Tunnel | Advanced

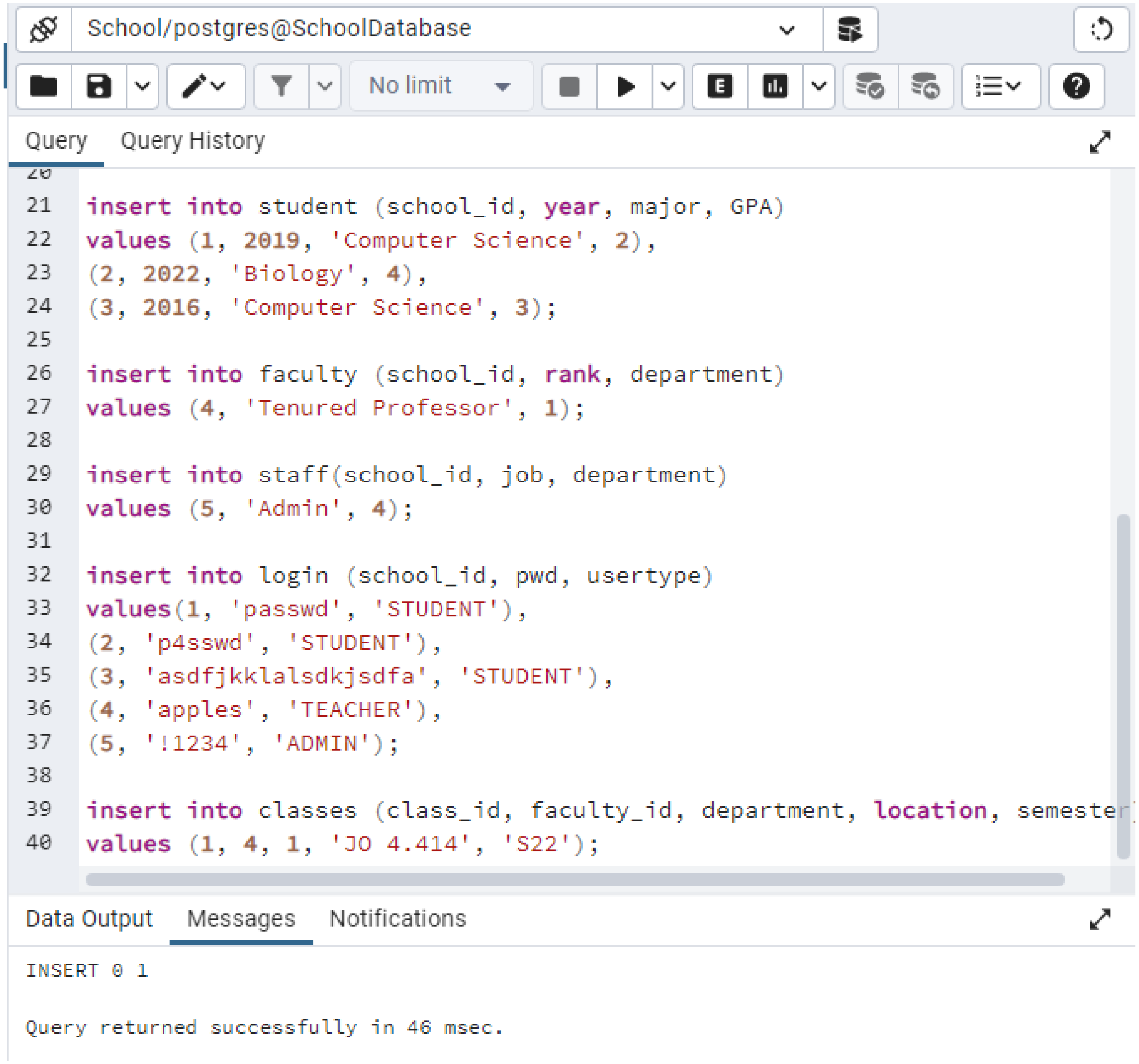
Host name/address	localhost
Port	5432
Maintenance database	postgres
Username	postgres
Kerberos authentication?	
Role	
Service	

Once the database has been initialized, we can use sql statements to create and populate the tables. These files can be found in the appendix and are attached in the zip file.


School/postgres@SchoolDatabase

```
        REFERENCES Person (school_id)
        ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE classes (
    class_id int not null,
    faculty_id int not null,
    department int not null,
    location varchar(20) not null,
    semester varchar(20) not null,

    PRIMARY KEY(class_id),

    FOREIGN KEY(department)
        REFERENCES Department (department_id)
        ON DELETE restrict ON UPDATE CASCADE,

    FOREIGN KEY(faculty_id)
        REFERENCES Person (school_id)
        ON DELETE restrict ON UPDATE CASCADE
);
```

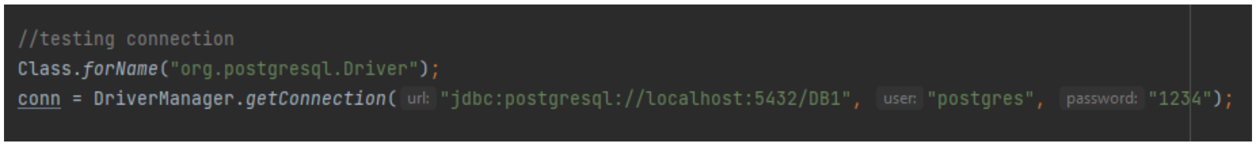
```
NOTICE:  table "classes" does not exist, skipping
NOTICE:  table "login" does not exist, skipping
NOTICE:  table "staff" does not exist, skipping
NOTICE:  table "faculty" does not exist, skipping
NOTICE:  table "student" does not exist, skipping
NOTICE:  table "name" does not exist, skipping
NOTICE:  table "department" does not exist, skipping
NOTICE:  table "person" does not exist, skipping
CREATE TABLE

Query returned successfully in 49 msec.
```


School/postgres@SchoolDatabase

```sql
insert into student (school_id, year, major, GPA)
values (1, 2019, 'Computer Science', 2),
(2, 2022, 'Biology', 4),
(3, 2016, 'Computer Science', 3);

insert into faculty (school_id, rank, department)
values (4, 'Tenured Professor', 1);

insert into staff(school_id, job, department)
values (5, 'Admin', 4);

insert into login (school_id, pwd, usertype)
values(1, 'passwd', 'STUDENT'),
(2, 'p4sswd', 'STUDENT'),
(3, 'asdfjkklalsdkjsdfa', 'STUDENT'),
(4, 'apples', 'TEACHER'),
(5, '!1234', 'ADMIN');

insert into classes (class_id, faculty_id, department, location, semeste
values (1, 4, 1, 'JO 4.414', 'S22');
```

```
INSERT 0 1

Query returned successfully in 46 msec.
```

The database is online, and we can now connect it to our login system. The following line of code connects the application to the database. For Java applications such as this, please note that postgresql-42.5.1.jar must be installed and put in your build path, which can be installed from https://jdbc.postgresql.org/download/. The program is now ready to compile and run



```java
//testing connection
Class.forName("org.postgresql.Driver");
conn = DriverManager.getConnection( url: "jdbc:postgresql://localhost:5432/DB1",  user: "postgres",  password: "1234");
```

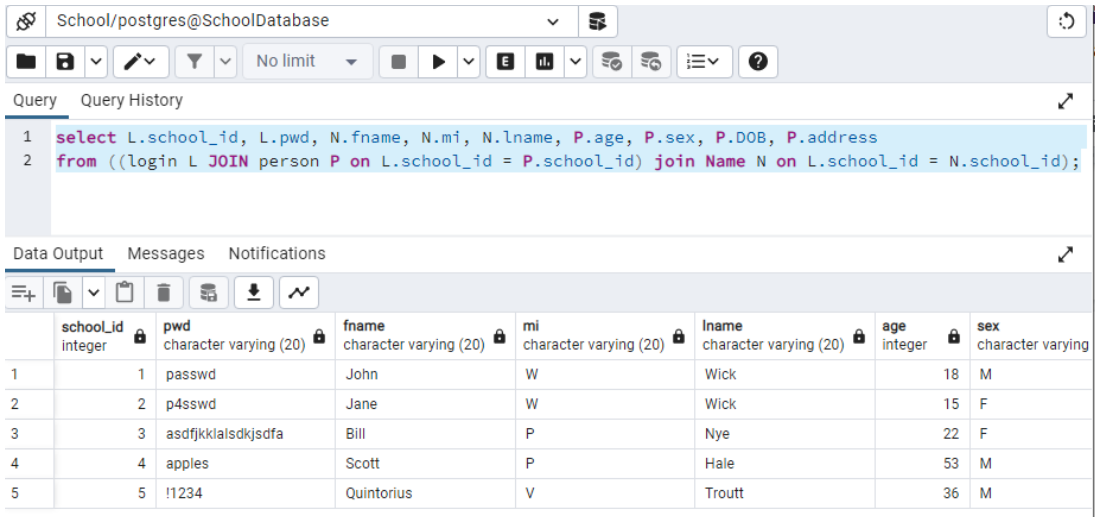
Suggestions on Database Tuning

For sufficiently large tables, indexing values can be used to speed up record retrieval. This can be especially useful for the “person” table, which will have many different values stored within it. Implementing tighter constraints on our columns can also help to speed up queries, although it will come with a slight cost to memory.

Additional Queries and Views

The following query returns a table containing all the personal information about a person, with their school id, password, full name, age, sex, date of birth, and address via a 3-way join of the login, person, and name tables.

```
select L.school_id, L.pwd, N.fname, N.mi, N.lname, P.age, P.sex,
P.DOB, P.address
from ((login L JOIN person P on L.school_id = P.school_id) join
Name N on L.school_id = N.school_id);
```



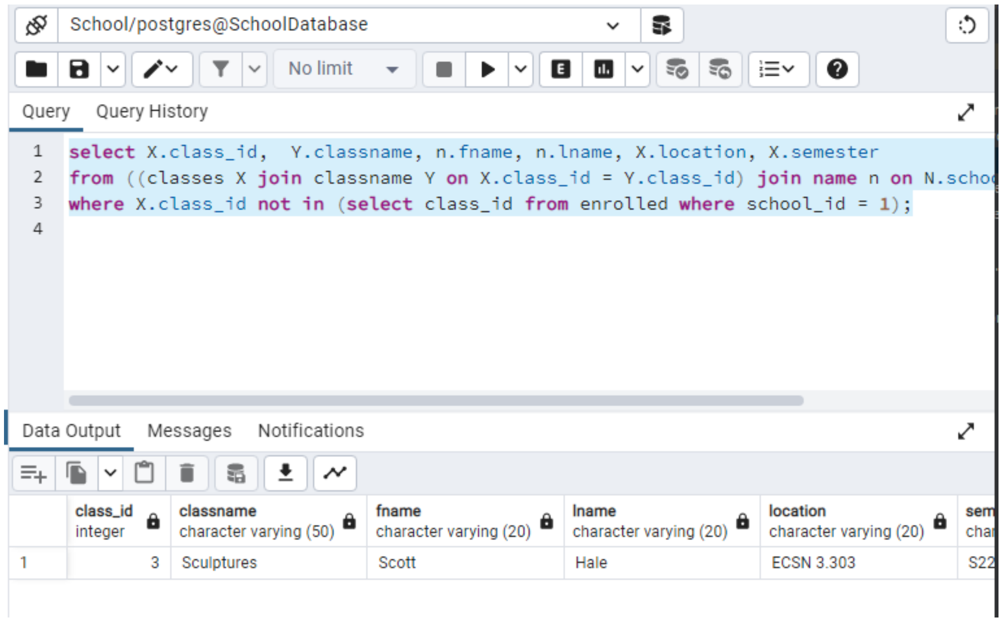
School/postgres@SchoolDatabase

```
select L.school_id, L.pwd, N.fname, N.mi, N.lname, P.age, P.sex, P.DOB, P.address
from ((login L JOIN person P on L.school_id = P.school_id) join Name N on L.school_id = N.school_id);
```

	school_id integer	pwd character varying (20)	fname character varying (20)	mi character varying (20)	lname character varying (20)	age integer	sex character varying
1	1	passwd	John	W	Wick	18	M
2	2	p4sswd	Jane	W	Wick	15	F
3	3	asdfjkklalsdkjsdfa	Bill	P	Nye	22	F
4	4	apples	Scott	P	Hale	53	M
5	5	!1234	Quintorius	V	Troutt	36	M

The following query is used by students to display classes they are eligible to enroll for. Since This user is already enrolled in class IDs 1 and 2, only class 3 shows eligible enrollment. This is done via a 3-way join of the classes, classname, and name tables, and then exclude values found in classes that the current user is enrolled

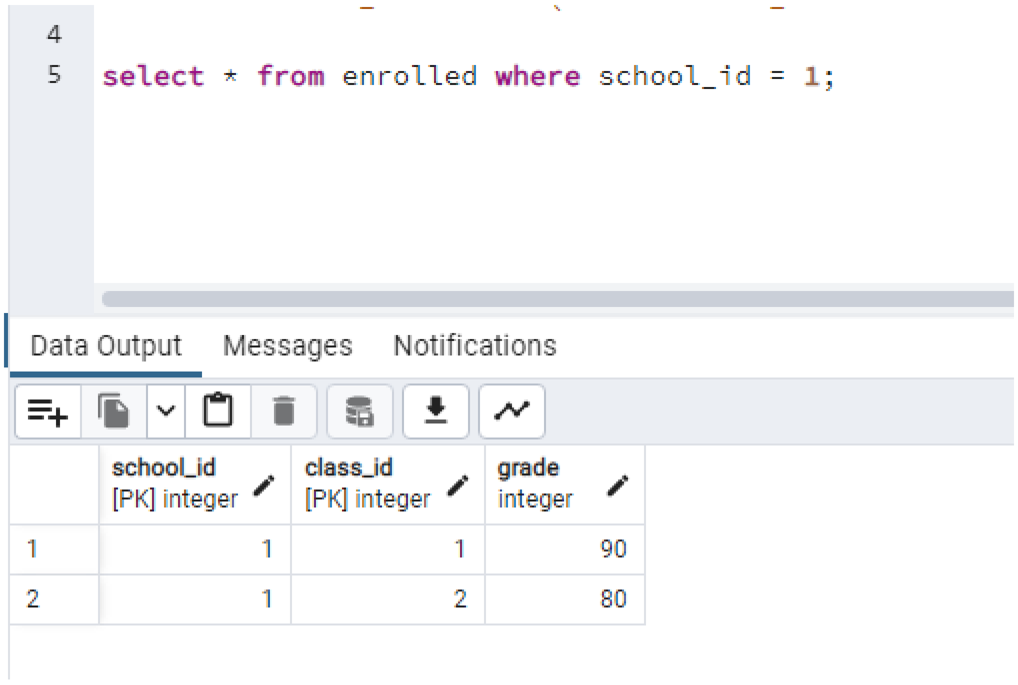
```
select X.class_id,  Y.classname, n.fname, n.lname, X.location, X.semester
from ((classes X join classname Y on X.class_id = Y.class_id) join name n
on N.school_id = X.faculty_id)
where X.class_id not in (select class_id from enrolled where school_id =
1);
```



```
select X.class_id,  Y.classname, n.fname, n.lname, X.location, X.semester
from ((classes X join classname Y on X.class_id = Y.class_id) join name n on N.scho
where X.class_id not in (select class_id from enrolled where school_id = 1);
```

	class_id integer	classname character varying (50)	fname character varying (20)	lname character varying (20)	location character varying (20)	sem char
1	3	Sculptures	Scott	Hale	ECSN 3.303	S22

As shown, the user with school ID 1 is already enrolled in class IDs 1 and 2.

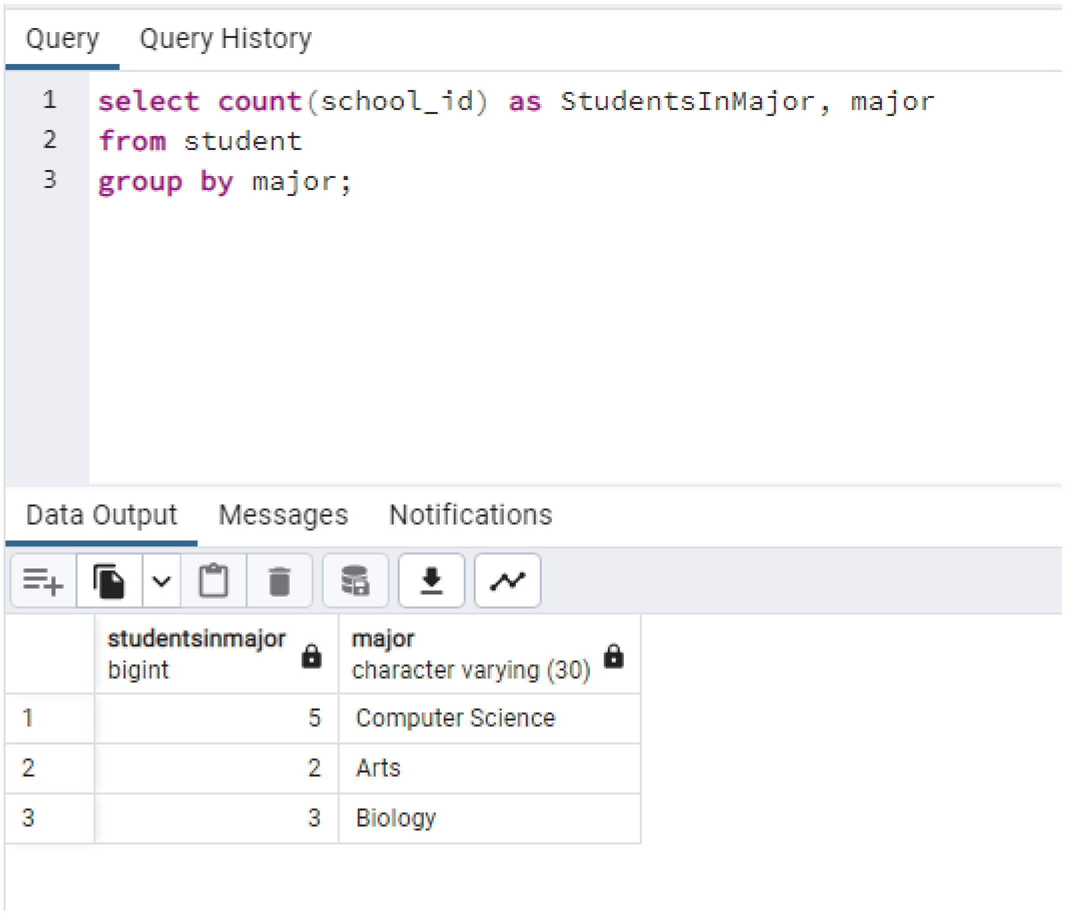


```
select * from enrolled where school_id = 1;
```

	school_id [PK] integer	class_id [PK] integer	grade integer
1	1	1	90
2	1	2	80

The following query returns a table of students in each major.

```
select count(school_id) as StudentsInMajor, major

from student

group by major;
```

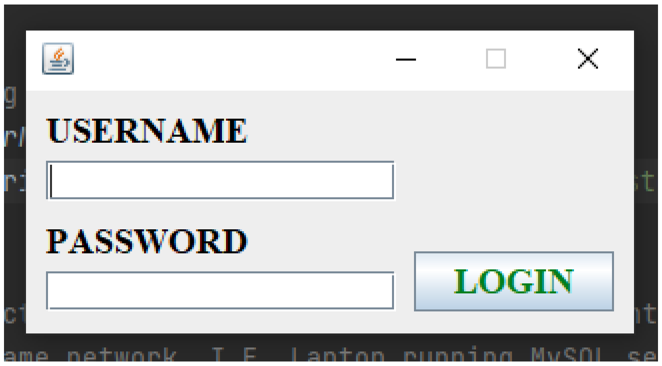


```
select count(school_id) as StudentsInMajor, major
from student
group by major;
```

	studentsinmajor bigint	major character varying (30)
1	5	Computer Science
2	2	Arts
3	3	Biology

User Application Interface

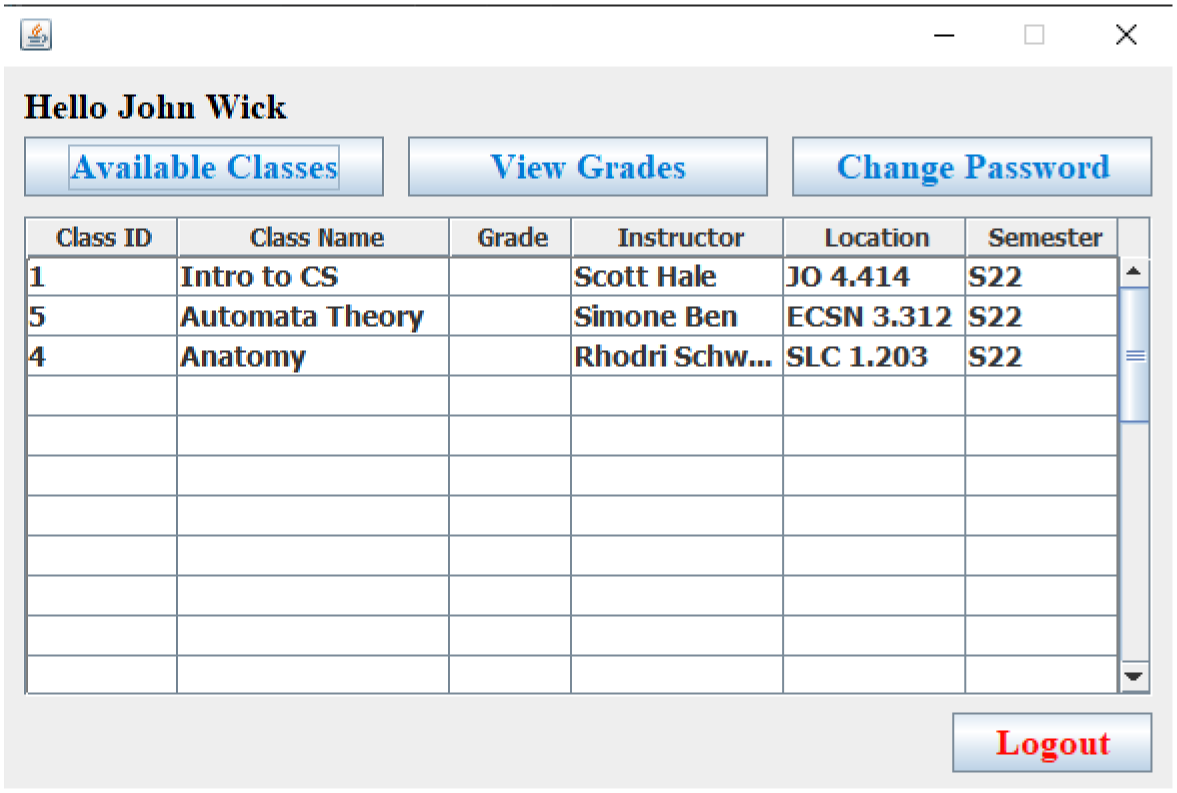
Compiling and running the program will bring up a login prompt



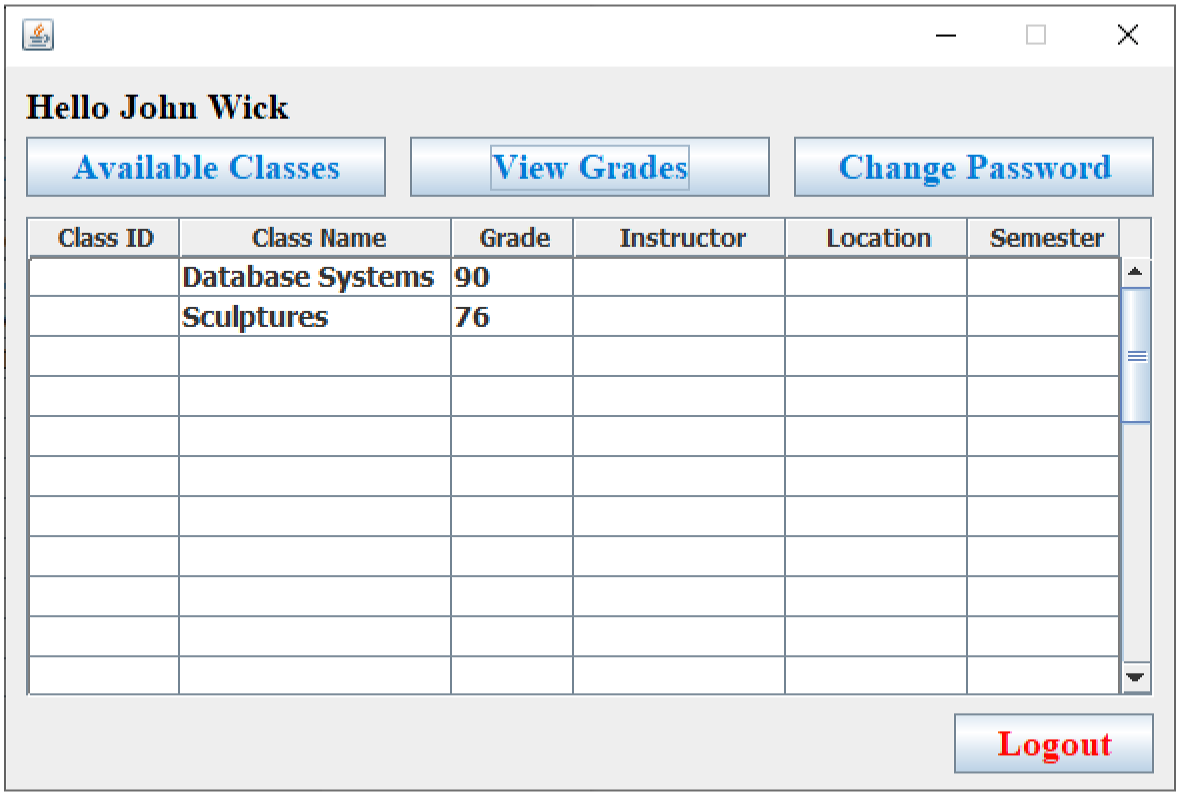
Depending on the user accessing the system, it will bring up a different window based on their user type, as shown below.

Student View

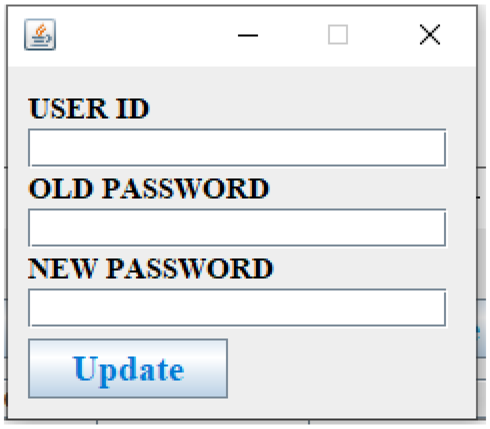
The Student UI allows students to view available classes for enrollment, view their class grades, and change their password. If they wish to register for a new class, a Staff member must be contacted via external means. The available class list will not list classes that a student is already registered in. The Grades section will show grades for classes that the student is attending. The login prompt will allow any user to edit his or her password. The logout



Class ID	Class Name	Grade	Instructor	Location	Semester
1	Intro to CS		Scott Hale	JO 4.414	S22
5	Automata Theory		Simone Ben	ECSN 3.312	S22
4	Anatomy		Rhodri Schw...	SLC 1.203	S22



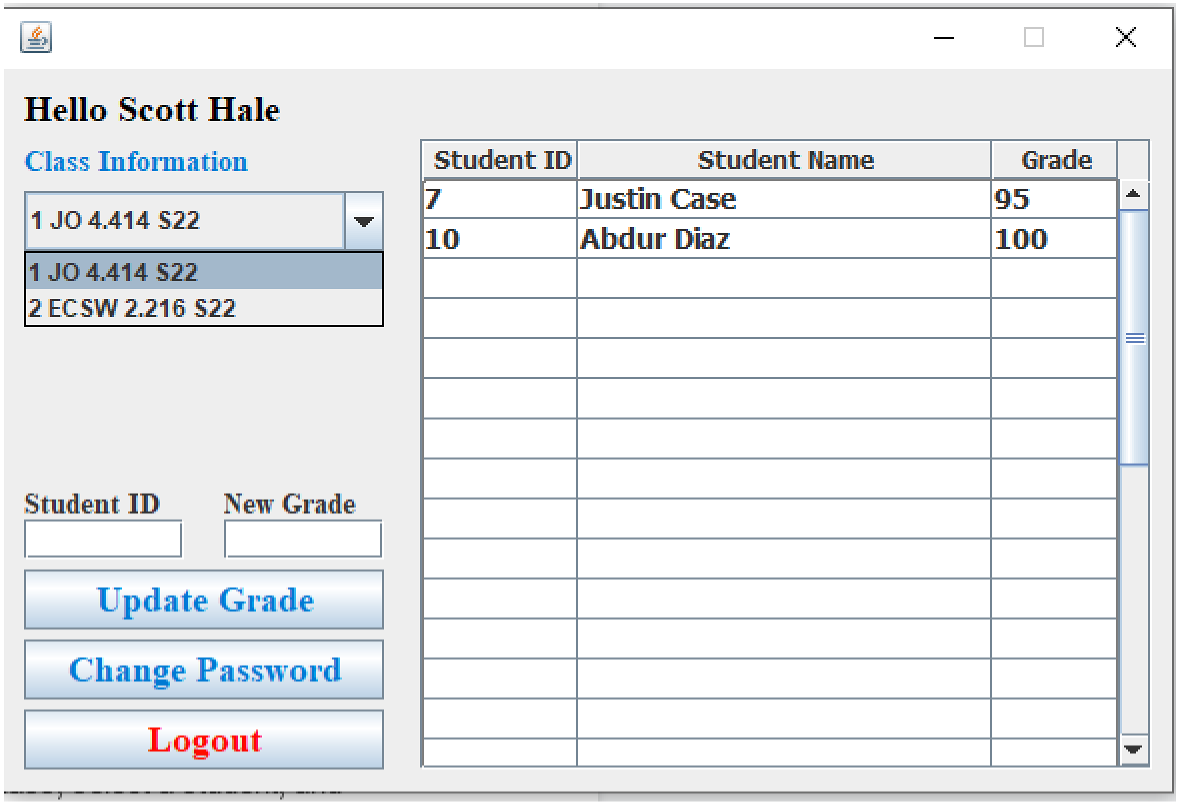
Class ID	Class Name	Grade	Instructor	Location	Semester
	Database Systems	90			
	Sculptures	76			



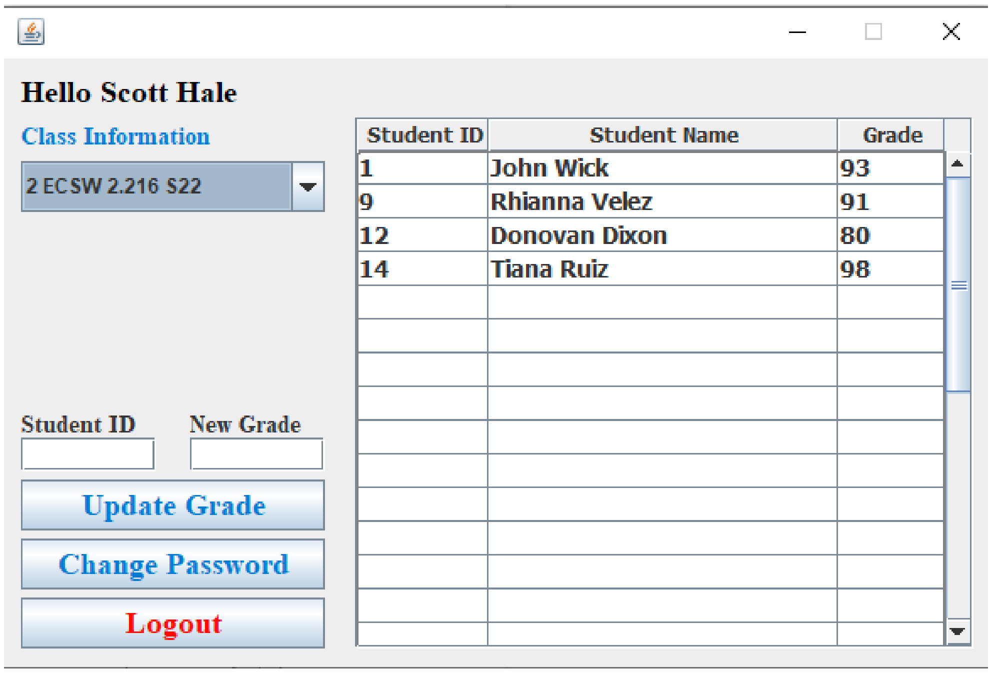
Faculty View

Logging in as a faculty member will allow you to view students in your classes, update their grades, change password, and logout.

When you view students, you will see a list of classes via a drop-down menu and be able to select a class to see the students enrolled in that class.



Student ID	Student Name	Grade
7	Justin Case	95
10	Abdur Diaz	100



Hello Scott Hale

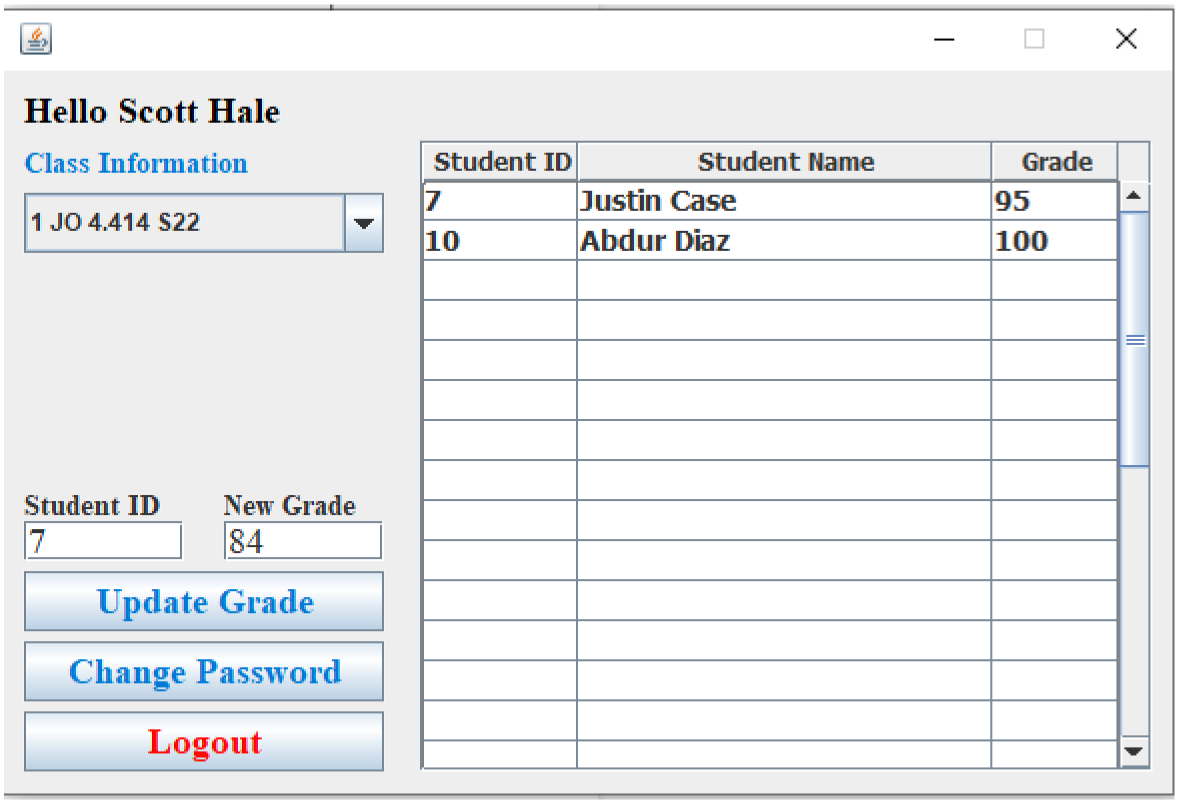
Class Information

2 ECSW 2.216 S22

Student ID	Student Name	Grade
1	John Wick	93
9	Rhianna Velez	91
12	Donovan Dixon	80
14	Tiana Ruiz	98

Student ID | New Grade

When you update grades, you will see a list of classes, select a class, select a student, and update their grades.



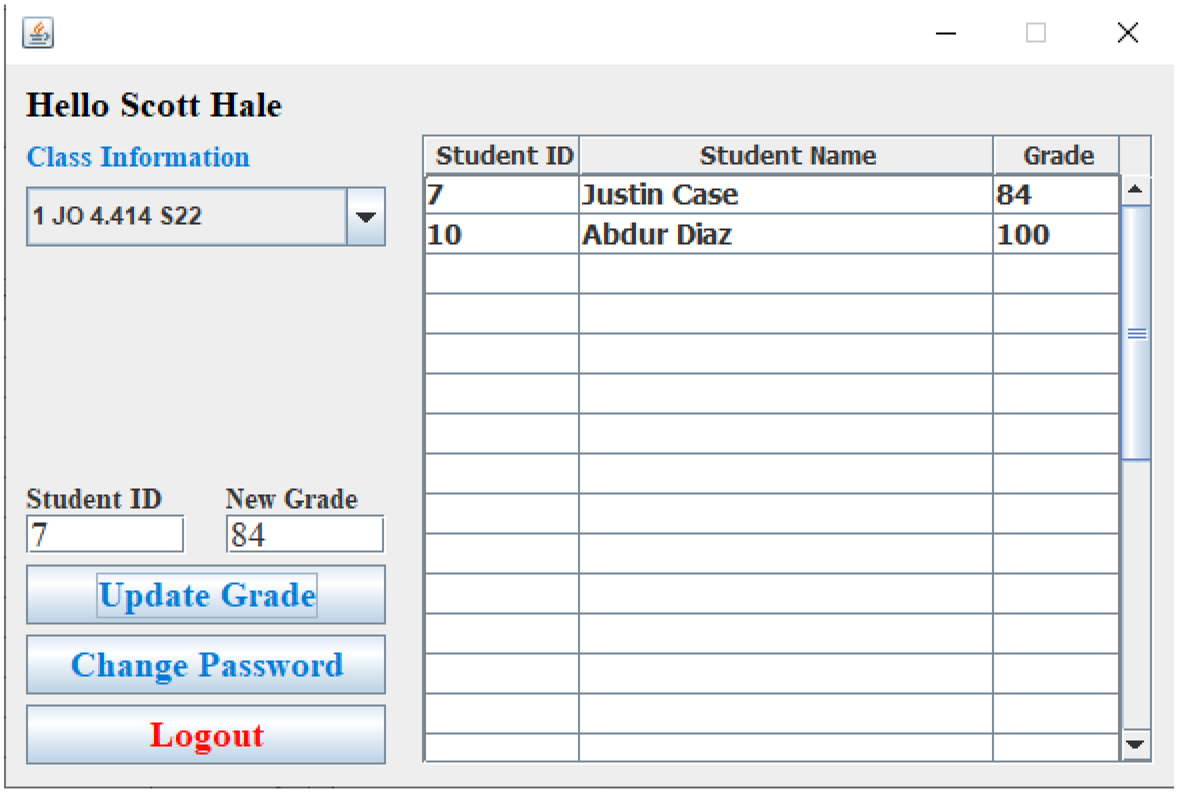
Hello Scott Hale

Class Information

1 JO 4.414 S22

Student ID	Student Name	Grade
7	Justin Case	95
10	Abdur Diaz	100

Student ID: 7
New Grade: 84


Hello Scott Hale

Class Information

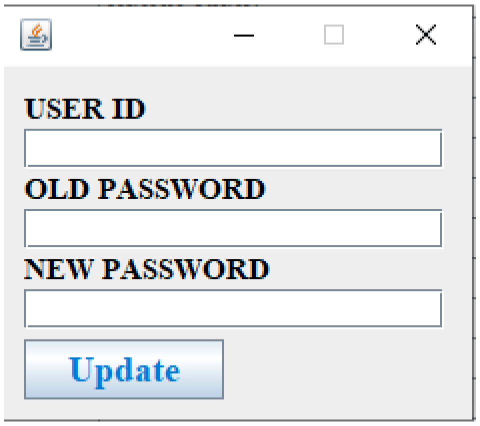
1 JO 4.414 S22

Student ID	Student Name	Grade
7	Justin Case	84
10	Abdur Diaz	100

Student ID: 7

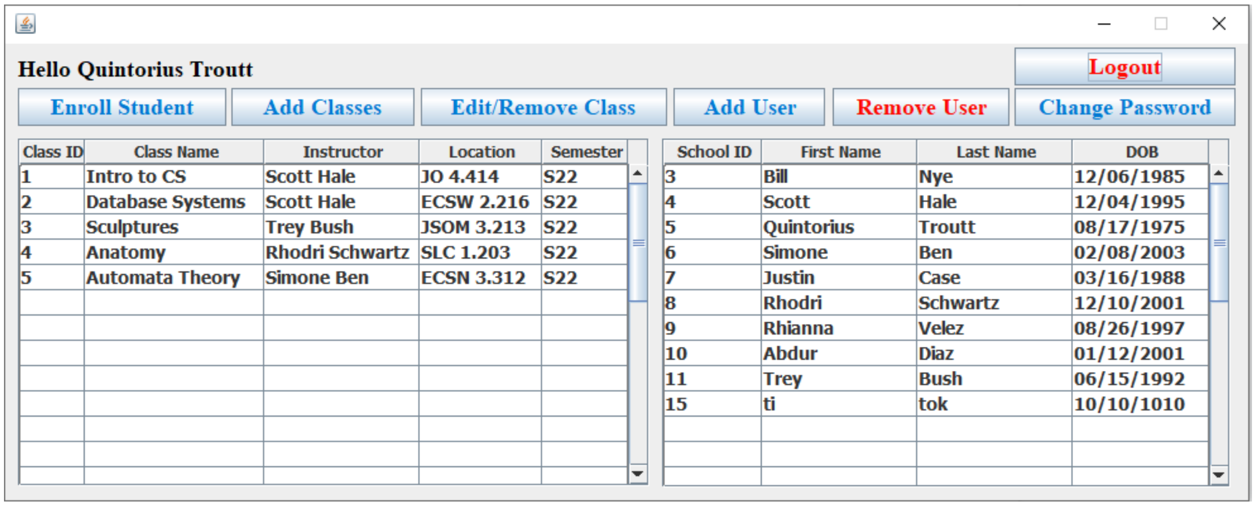
New Grade: 84

The Change password works similarly to the change password in student as well as admin



Admin View

The administrator has the most power between the three user types, with the power to enroll students into classes, add, edit and remove classes, add and remove users from the database, and change their own password.

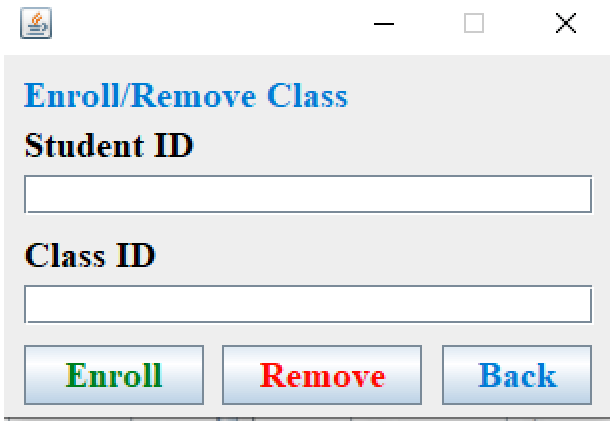


Hello Quintorius Troutt

Class ID	Class Name	Instructor	Location	Semester
1	Intro to CS	Scott Hale	JO 4.414	S22
2	Database Systems	Scott Hale	ECSW 2.216	S22
3	Sculptures	Trey Bush	JSOM 3.213	S22
4	Anatomy	Rhodri Schwartz	SLC 1.203	S22
5	Automata Theory	Simone Ben	ECSN 3.312	S22

School ID	First Name	Last Name	DOB
3	Bill	Nye	12/06/1985
4	Scott	Hale	12/04/1995
5	Quintorius	Troutt	08/17/1975
6	Simone	Ben	02/08/2003
7	Justin	Case	03/16/1988
8	Rhodri	Schwartz	12/10/2001
9	Rhianna	Velez	08/26/1997
10	Abdur	Diaz	01/12/2001
11	Trey	Bush	06/15/1992
15	ti	tok	10/10/1010

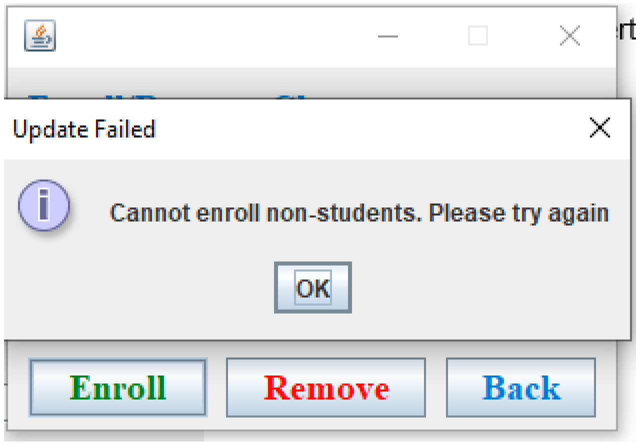
The enroll student option allows the Admin to add and remove students from certain classes via the student and class ID. The app will prompt the user on a failed update.



Enroll/Remove Class

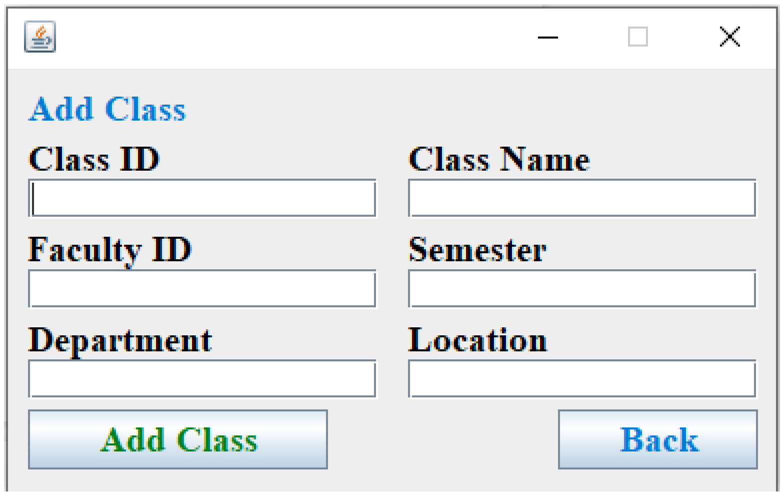
Student ID

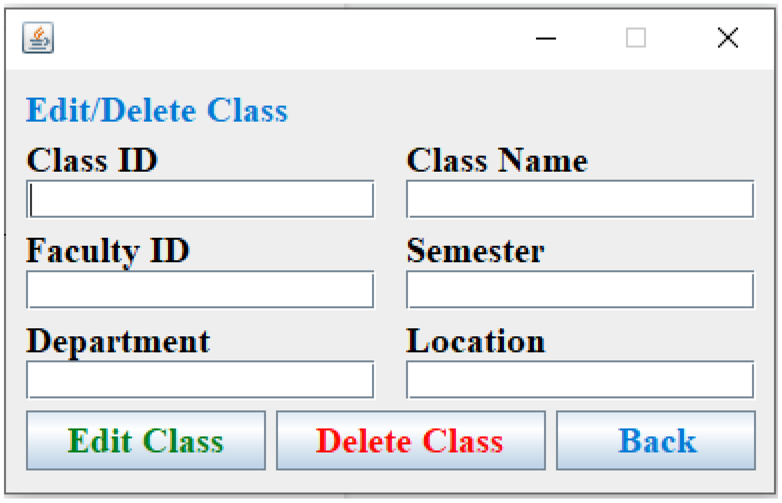
Class ID


Update Failed

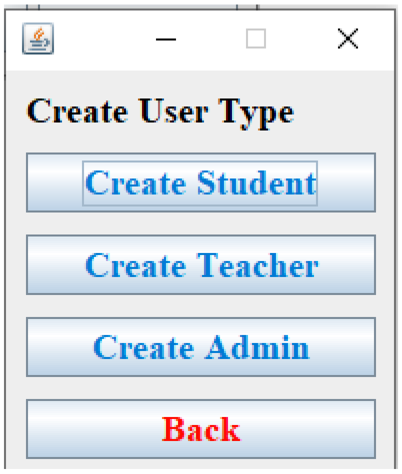
Cannot enroll non-students. Please try again

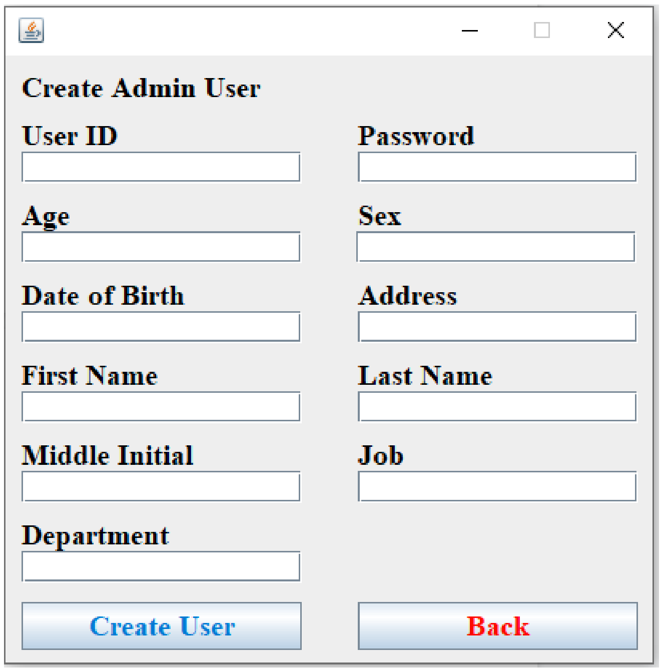
The add classes function allows the admin to create classes, provided they fill in the required information, and in the same vein, edit and delete classes based on information.

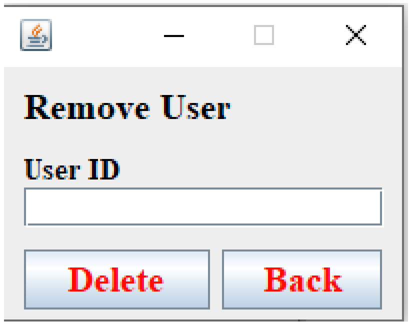




The admin can add and remove users as well, with different forms based on the type of entity they would like to add, and can remove users based on their user ID.







Conclusions and Future Work

Our project provides a rudimentary database for managing classes within a school environment. Creating our project reinforced important concepts of database management through SQL, and taught us how to apply the knowledge we learned in class within a group setting. We are currently missing some functionality, such as the user specific functions, and perhaps an enrolled table to relate students and classes. For further development, our system could be expanded in many different ways. The capability to drop or waitlist classes could be added, more functions such as tuition or on-campus housing can be added, and additional information such as minor or graduate status can also be added. Adding TA and professor information to the class attributes would be handy for students to get into contact with a professor. Adding realistic enrollment limit constraints would also help refine the system.

References

- Fundamentals of Database Systems (7th Edition) byRamez Elmasri, Shamkant B. Navathe, ISBN-13: 978-0133970777
- https://www.w3schools.com/
- https://www.postgresql.org/

Appendix

```
/doc
        /FinalProjectReport.pdf - this file!
/project
        /application
                /addAdmin.java
                /addClasses.java
                /addStudent.java
                /addTeacher.java
                /addUser.java
                /changepwd.java
                /editClasses.java
                /enrollStudents.java
                /login.java
                /removeUser.java
                /staffUser.java
                /studentUser.java
                /teacherUser.java
                /postgresql-42.5.1.jar
        /sql
                /create.sql
                /populate.sql
/readme
```