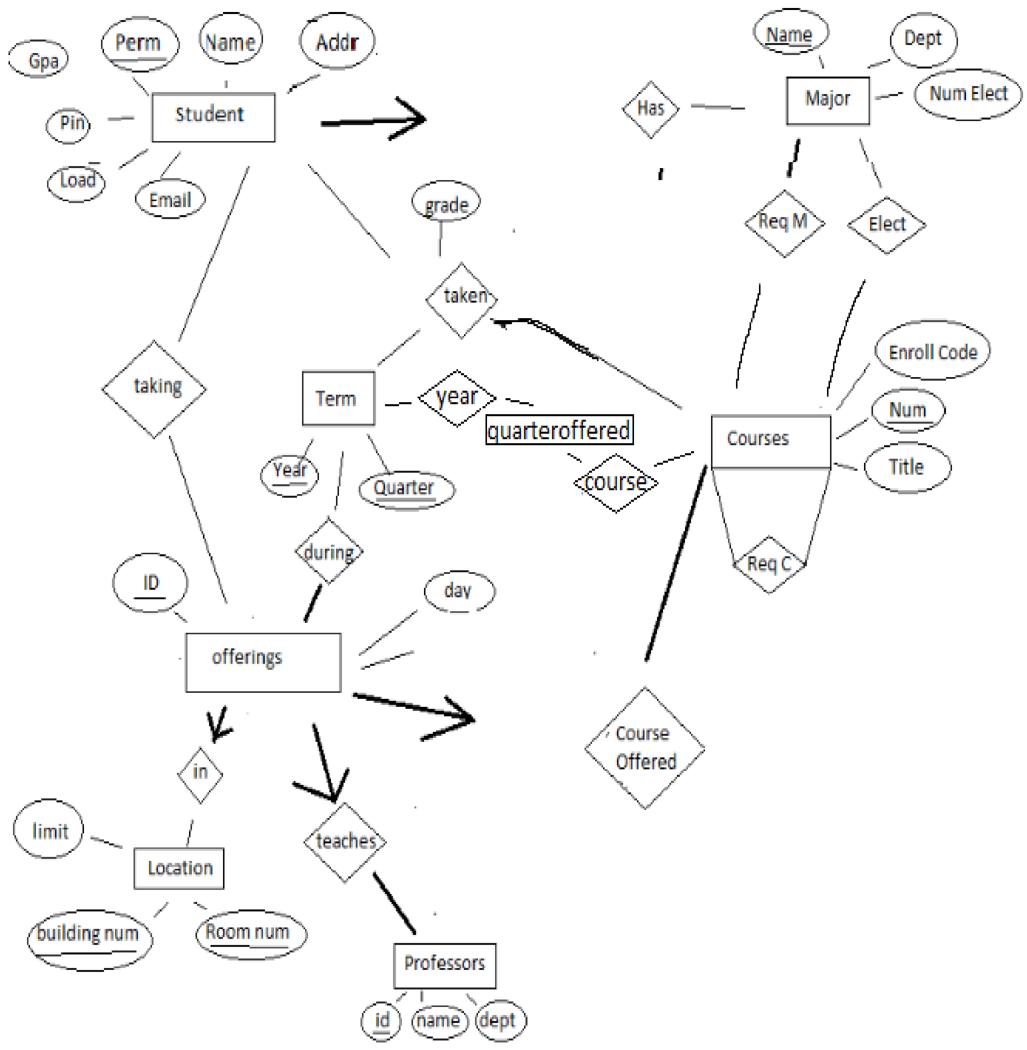
CS174A Project Report

Elijah Anderson, Thomas Harbeck

Task Divisions

- Elijah: ER and Relational Schemas, Available Transactions, SQL Tables, Student interface
- Thomas: SQL Tables, Integrity constraints, Integrity Constraints, SQL Tables, Registrar interface

ER Diagram Schema



Available Transactions

- **add a course,**
  - Students will add courses from Offerings
    Offerings: Reading quarter == current Quarter for course
      Student: Read load < 5, increase if adding is success
      Taking: Insert new row if success
      Courses: Read Enroll Code
      ReqC: Read All required courses Nums for given course.

Taken: Check Required Nums in comparison with Taken nums

• **drop a course**

- Student will drop courses from Taking
  Taking: Deletion of row if success
  Taking: Delete Row, Read Course num
  Course: Read Num
  Courses: Read Num
  ReqC: Check if course is needed for a current course (concurrent enrollment)
  Student: Reduce load by 1 if success

• **list the courses enrolled in the current quarter,**
Taking: List all courses in taking for given student

• **list the grades of courses enrolled in the previous quarter,**
Taken: List courses for given student searching on (Perm, quarter)

• **requirements check,**
Major: Check major for given student, read elective units required, will compare to total
ReqM: Check required course Nums for major
Elect: Get required electives Nums for Major
Taken: Check if ReqM for given major - taken of student(grade != F) = {}. Check taken against Elect to get total Units

• **make a plan,**

Taken: Read Course Num of grade != F
ReqM: Read Course Nums,
ReqC: Read Course Nums
Major: Read Num Elect
Elect: Read Course nums
Courses: Read Course Nums
Offerings: Read Quarters and Course num

Will compare ReqM and Elect with Taken to see needed elective units and Required courses. Get ReqC for the ReqM courses still needed to be taken. Then go through offerings to find best way to complete all of the requirements.

• **change the PIN.**
Student: Update Pin

A Registrar is allowed to perform the following transactions:

- add a student to a course
  Offering: Read Course Num
  Courses: Read Enroll Code
  Taking: Insert new Row

Student: Read Load < 5, increase if success
  - Assume Registrar can add even if student has not completed requirements

- drop a student from a course,
  Student: Read Load, decrease if success
  Taking: Delete Row, Read Course num
  Course: Read Num

- list the courses taken by a student,
  Taken: List all rows given a student perm

- list the grades of the previous quarter for a student,
  Taken: Read Grades from rows of search (Perm, quarter)

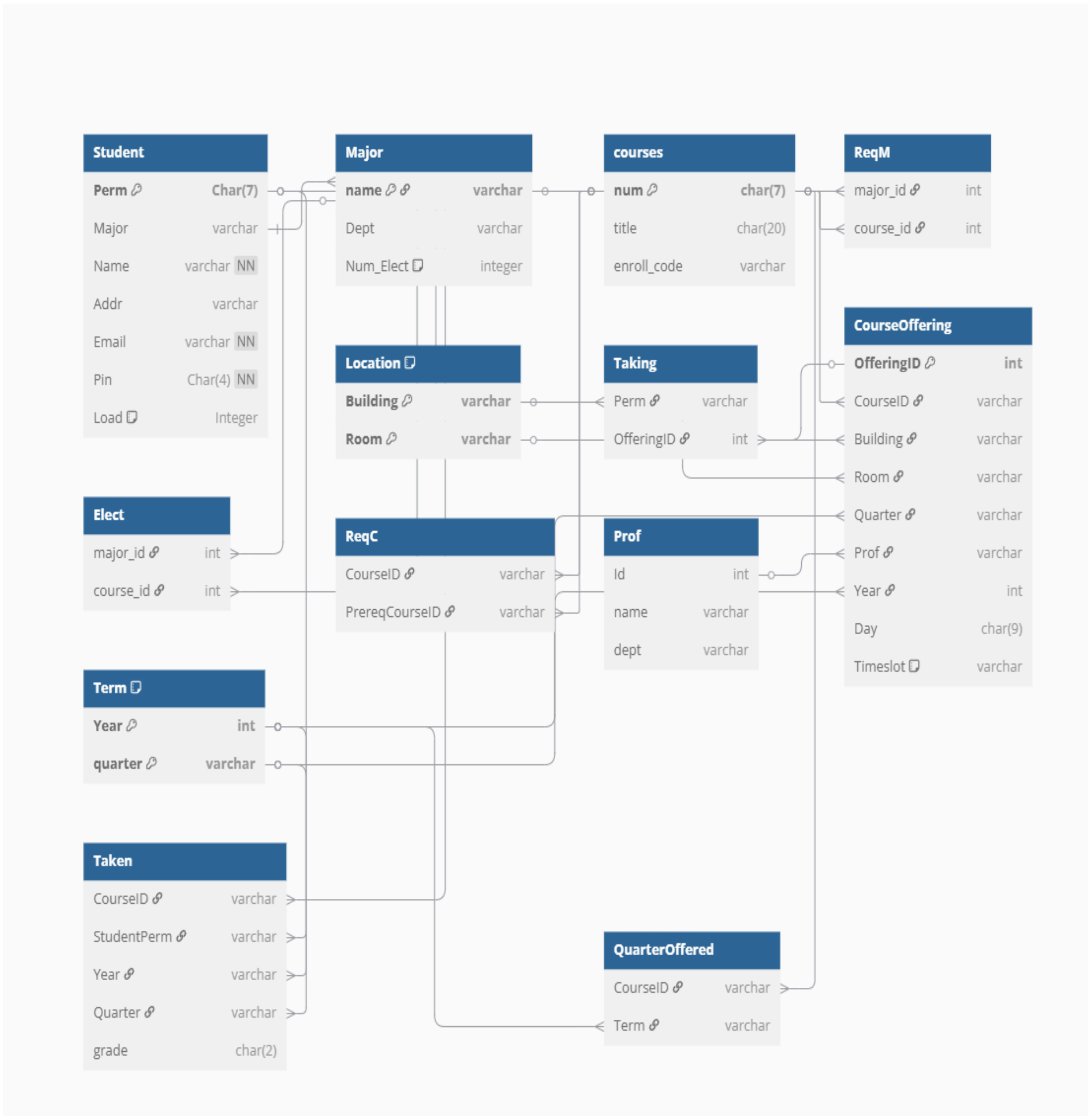
- generate a class list for a course,
  Taking: Given a course num, list all all students searching on Course Num.

- enter grades for a course, via a list of students and their grades,
  Taken: Insert a new row in Taken for course, student, grade, current quarter.
  Taken: Possibly update grade and quarter instead if replacing grades is allowed

- request a transcript to be printed out for a student, and
  Taken: Search through all taken classes by student, sorted by quarter. Real all

- generate a grade mailer for all students.
  Student: Read all Emails.

Relational Schema



Code for Relation DBdiagram.io

```
Table Student {
Perm Char(7) [primary key]
Major varchar
Name varchar [not null]
Addr varchar
Email varchar [unique, not null]
Pin Char(4)  [not null]
```

```
Load Integer [note: '0 < load <= 5']
}


Table Major {
name varchar [primary key]
Dept varchar
Num_Elect integer [note: 'Num_Elect >= 0']
}


Table courses {
num char(7) [primary key]
title char(20)
enroll_code varchar


}


Table ReqM {
major_id int [ref: > Major.name]
course_id int [ref: > courses.num]


}


Table Elect{
major_id int [ref: > Major.name]
course_id int [ref: > courses.num]


}




Table Location{
Building varchar [primary key]
```

```
Room varchar [primary key]
}


Table Taking {
Perm varchar [ref: > Student.Perm]
OfferingID int [ref: > CourseOffering.OfferingID]


}
Table CourseOffering {
OfferingID int [pk]
CourseID varchar [ref: > courses.num]
Building varchar [ref: > Location.Building]
Room varchar [ref: > Location.Room]
Quarter varchar [ref: > Term.quarter]
Prof varchar [ref: > Prof.Id]
Year int [ref: > Term.Year]
Day char(9)
Timeslot varchar [note: "In the form of [start time, endtime]"]
}


Table Term{
Year int [primary key]
quarter varchar [primary key]
}


Table ReqC {
CourseID varchar [ref: > courses.num]
PrereqCourseID varchar [ref: > courses.num]
}


Table Prof{
Id int
name varchar
dept varchar
}
```

```
Table QuarterOffered{
CourseID varchar [ref: > courses.num]
Term varchar [ref: > Term.Year]
}


Table Taken{
CourseID varchar [ref: > courses.num]
StudentPerm varchar [ref: > Student.Perm]
Year varchar [ref: > Term.Year]
Quarter varchar [ref: >Term.quarter]
grade char(2)
}


Ref: Student.Major < Major.name
```

Integrity Constraints

- Offerings
    - Key: Each ID has 1 location, 1 professor, 1 course in courses offered.
    - Participation: Must be in a location, have a professor, have a course offered, and be during some term.
    - Timeslot: Not Null
    - Day: Not Null
    - PRIMARY KEY (ID),
    - FOREIGN KEY (CoursesNum) REFERENCES Courses,
    - FOREIGN KEY (DuringYear, DuringQuarter) REFERENCES Term,
    - FOREIGN KEY (LocationBuilding, LocationRoom) REFERENCES Location,
    - FOREIGN KEY (ProfID) REFERENCES Professors

    - To deal with violations: All of the constraints are function parameters to ensure they are met, and foreign keys are used to link to other tables
- Courses
    - Primary Key: Num
    - Enroll Code: Unique, Not Null
    - Must be offered at least once
    - Title: Not Null
    - To deal with violations: Two functions (insertTakenCourse, insertTakingCourse) are used to differentiate between courses that have been taken and have a grade, and courses that are in the process of being taken. This way, it can be checked if a student has already taken a course, or if they do not meet the prerequisite requirements to take the course.
- Major
    - Must have at least one requirement
    - Primary Key: Name
    - Num Elect >= 0
    - To deal with violations: The number of requirements is a function parameter along with the insertion.
- Student
    - GPA >= 0
    - Pin >= 0000
    - Load >= 0 and <= 5
    - Email Not Null
    - Name Not Null
    - Primary Key Perm
    - Student must only 1 major and must have a major
    - To deal with violations: The function throws an error if a valid major is not provided.
- Professors
    - Must have a valid department
    - To deal with violations: The function requires a department definition
- Term
    - Primary Key [Year, Quarter]
    - To deal with violations: Throws an error if the term already exists
- Location
    - Primary Key, [Building num, Room num]
    - To deal with violations: Throws an error if the location already exists
- PIN
    - 5-digits
    - Only integers allowed
    - To deal with violations: VerifyPin function is used

SQL Tables

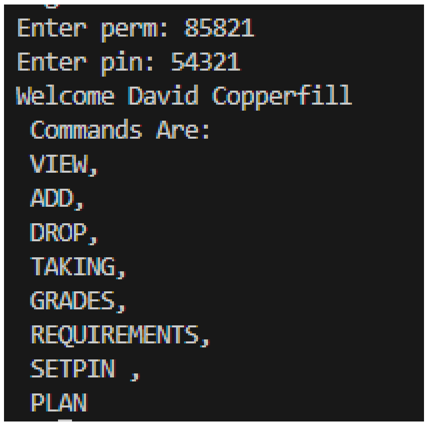
```
CREATE TABLE Majors (
    MajorName VARCHAR2(10) PRIMARY KEY,
    dept VARCHAR2(10) NOT NULL,
    num_elect INTEGER NOT NULL
);
CREATE TABLE Professors (
    ID INTEGER PRIMARY KEY,
    Name VARCHAR(20),
    Dept VARCHAR(10)
);
CREATE TABLE Locations (
    Building VARCHAR(7),
    Room VARCHAR(7),
    PRIMARY KEY (Building, Room)
);
CREATE TABLE Students (
    perm VARCHAR2(7) PRIMARY KEY,
    name VARCHAR2(40) NOT NULL,
    address VARCHAR2(40) NOT NULL,
    Dept VARCHAR(10) NOT NULL,
    pin VARCHAR2(64) NOT NULL,
    Major VARCHAR(10) NOT NULL,
    Load INTEGER DEFAULT 0,
    FOREIGN KEY (Major) REFERENCES Majors(MajorName)
);
CREATE TABLE Courses (
    CourseID VARCHAR(7) PRIMARY KEY,
    Title VARCHAR(20),
    enroll_code VARCHAR(7)
);
CREATE TABLE Term (
    TermYear INTEGER,
    Quarter VARCHAR(1),
    PRIMARY KEY (TermYear, Quarter)
);
CREATE TABLE Taken (
    TCourseID VARCHAR(7),
    StudentPerm VARCHAR2(7),
    TakenYear INTEGER NOT NULL,
    Quarter VARCHAR(1) NOT NULL,
    grade VARCHAR(2) NOT NULL,
    PRIMARY KEY (TCourseID, StudentPerm, TakenYear, Quarter),
    FOREIGN KEY (StudentPerm) REFERENCES Students(perm) ON DELETE CASCADE,
    FOREIGN KEY (TCourseID) REFERENCES Courses(CourseID),
    FOREIGN KEY (TakenYear, Quarter) REFERENCES Term(TermYear, Quarter)
);
CREATE TABLE QuarterOffered (
    CourseID VARCHAR2(7),
    Quarter VARCHAR2(1),  -- 'W', 'S', 'F'
    PRIMARY KEY (CourseID, Quarter),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);
CREATE TABLE Offerings (
    OfferingID INTEGER PRIMARY KEY,
    CourseID VARCHAR(7),
    Prof INTEGER,
    Building VARCHAR(7),
    Room VARCHAR(7),
    Quarter VARCHAR(1),
    Year INTEGER,
    Day VARCHAR(9),
    Capacity INTEGER,
    Timeslot VARCHAR(10),
    FOREIGN KEY (Year, Quarter) REFERENCES Term(TermYear, Quarter),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID) ON DELETE CASCADE,
    FOREIGN KEY (Building, Room) REFERENCES Locations(Building, Room) ON
DELETE CASCADE,
    FOREIGN KEY (Prof) REFERENCES Professors(ID),
    FOREIGN KEY (CourseID, Quarter) REFERENCES QuarterOffered ON DELETE
CASCADE
);
CREATE TABLE Taking (
    OfferingID INTEGER,
    StudentPerm VARCHAR2(7),
    CourseID VARCHAR(7),
    PRIMARY KEY (OfferingID, StudentPerm, CourseID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID) ON DELETE CASCADE,
    FOREIGN KEY (StudentPerm) REFERENCES Students(perm) ON DELETE CASCADE,
    FOREIGN KEY (OfferingID) REFERENCES Offerings(OfferingID) ON DELETE
CASCADE
);
CREATE TABLE Electives (
    MajorName VARCHAR2(10),
    CourseID VARCHAR(7),
    PRIMARY KEY (MajorName, CourseID),
    FOREIGN KEY (MajorName) REFERENCES Majors(MajorName) ON DELETE
CASCADE,
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID) ON DELETE CASCADE
);
CREATE TABLE ReqMajor (
    MajorName VARCHAR2(10),
    CourseID VARCHAR(7),
    PRIMARY KEY (MajorName, CourseID),
    FOREIGN KEY (MajorName) REFERENCES Majors(MajorName) ON DELETE
CASCADE,
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID) ON DELETE CASCADE
);
CREATE TABLE ReqCourses (
    CourseID1 VARCHAR2(7),
    CourseID2 VARCHAR(7),
    PRIMARY KEY (CourseID1, CourseID2),
    FOREIGN KEY (CourseID1) REFERENCES Courses(CourseID) ON DELETE
CASCADE,
    FOREIGN KEY (CourseID2) REFERENCES Courses(CourseID) ON DELETE CASCADE
);
CREATE OR REPLACE TRIGGER set_student_load_to_zero
BEFORE INSERT ON Students
FOR EACH ROW
BEGIN
    :NEW.Load := 0;
END;
/
CREATE OR REPLACE TRIGGER increment_load_on_insert
BEFORE INSERT ON Taking
FOR EACH ROW
DECLARE
    current_load INTEGER;
BEGIN
```

```
    SELECT Load INTO current_load FROM Students WHERE perm =
:NEW.StudentPerm;
    IF current_load >= 5 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Student cannot enroll in more
than 5 courses.');
    END IF;
    UPDATE Students
    SET Load = Load + 1
    WHERE perm = :NEW.StudentPerm;
END;
/
CREATE OR REPLACE TRIGGER decrement_load_on_delete
AFTER DELETE ON Taking
FOR EACH ROW
BEGIN
    UPDATE Students
    SET Load = Load - 1
    WHERE perm = :OLD.StudentPerm;
END;
/
CREATE OR REPLACE FUNCTION VerifyPin(
    p_perm IN VARCHAR2,
    p_pin  IN VARCHAR2
) RETURN NUMBER IS
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count
    FROM STUDENTS
    WHERE perm = p_perm AND pin = p_pin;
    RETURN CASE WHEN v_count = 1 THEN 1 ELSE 0 END;
END;
/
CREATE OR REPLACE FUNCTION SetPin(
    p_perm       IN VARCHAR2,
    p_oldpin     IN VARCHAR2,
    p_newpin     IN VARCHAR2
) RETURN NUMBER IS
BEGIN
    -- check if old PIN matches
    IF VerifyPin(p_perm, p_oldpin) = 1 THEN
```

```
        UPDATE STUDENTS
        SET pin = p_newpin
        WHERE perm = p_perm;
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END;
```

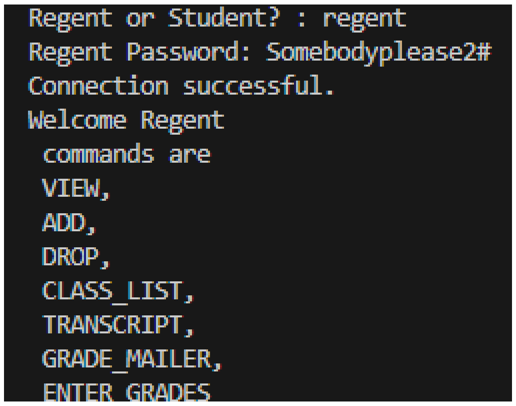
Interfaces

- Gold Interface



```
Enter perm: 85821
Enter pin: 54321
Welcome David Copperfill
 Commands Are:
 VIEW,
 ADD,
 DROP,
 TAKING,
 GRADES,
 REQUIREMENTS,
 SETPIN ,
 PLAN
```

-
- Registrar Interface



```
Regent or Student? : regent
Regent Password: Somebodyplease2#
Connection successful.
Welcome Regent
 commands are
 VIEW,
 ADD,
 DROP,
 CLASS_LIST,
 TRANSCRIPT,
 GRADE_MAILER,
 ENTER_GRADES
```

JAVA Classes

- displayGraduationPlan
  - Used by the Gold Interface to make a plan for student graduation
- displayTakingCourse

    - Used by the Gold and Registrar Interfaces to generate a list of courses a student is enrolled in the current quarter
- GetName
    - Used in the Gold Interface to display name of user after logging in
- showTakenGrades
    - Used in the Gold Interface to list the grades of courses a student was enrolled in previous quarters
- showAvailableOfferings
    - Used by the Gold and Registrar Interfaces to view courses available to add, based on the current year and quarter
- AddCourse
    - Used by the Gold and Registrar Interfaces to add a student to a course based on their perm number and offered course's ID
    - Uses insertTakingCourse to check if the offered course ID matches the database for course IDs
- showStudentInfo
    - Used in testing to print the name of a student based on their perm number
    - Uses connectAsAppUser
- VerifyPin
    - Used by the Gold Interface to log a student into their account
    - Uses hashPin
    - Uses connectAsAppUser
- SetPin
    - Used in Gold Interface testing to set a student PIN
    - Uses hashPin
    - Uses connectAsAppUser
- checkStudentRequirements
    - Used in Gold Interface for a student's requirements check
- connectAsAppUser
    - Used in the back end of both interfaces to connect to the database as a user
- connectAsUser
    - Used in the back end of both interfaces to connect to the database as a user
- insertNewCourseRequirement
    - Used in testing to insert a new course requirement
    - Uses connectAsUser
- insertNewStudent
    - Used in testing to insert a new student
    - Uses hashPin
    - Uses connectAsUser
- insertNewProfessor
    - Used in testing to insert a new professor
    - Uses connectAsUser
- insertOffering
    - Used in testing to insert a new course offering

    - Uses connectAsUser
- insertNewMajor
    - Used in testing to insert a new department
    - Uses connectAsUser
- dropTakingCourse
    - Used by the Gold and Registrar Interfaces to drop a student from a course
- insertTakenCourse
    - Used in testing to insert the courses a student has taken
    - Uses connectAsUser
- insertTakingCourse
    - Checks if a student is enrolled in a course, or have already completed a course, otherwise enrolls them in the course
- insertNewTerm
    - Used in testing to insert a new term (year, quarter)
    - Uses connectAsUser
- insertNewLocation
    - Used in testing to insert a new location for a class
    - Uses connectAsUser
- insertReqMajor
    - Used in testing to insert a new major requirement
    - Uses connectAsUser
- insertElectiveCourse
    - Used in testing to insert an elective
    - Uses connectAsUser
- insertNewCourse
    - Used in testing to generate a new course
    - Uses connectAsUser
- hashPin
    - Used to create a hashed pin
- listStudentsInCourse
    - Used by the Registrar Interface to generate a class list for a course
- generateTranscript
    - Used by the Registrar Interface to request a transcript to be printed out for a student
- enterGrades
    - Used by the Registrar Interface to key in the name of a file which contains a list of students and their grades to enter grades for a course
    - Uses checkIfExists if a SQL exception is encountered to ensure the class exists
- checkIfExists
    - Used in the back end of enter grades
- requestGradeMailer
    - Used by the Registrar Interface to generate a grade mailer for all students