CS174A Project Report

Authors: Yulin Lin, Chloe Andersen
Date: Tuesday, May 27th 2025

---

1. Task divisions:

Yulin:

- Create table commands
- Helped brainstorm triggers and Make Plan
- Worked on some of Registrar Interface
- Testing
- Project report

Chloe:

- Set up the codebase, Makefile, readme, database connection, and github
- Manually inserted all test data (version 2) into database
- Coded triggers/procedures in SQL and the command line interface in Java
    - Registrar Interface
    - Student Interface
    - Main Interface
- Testing
- Project report

2. Final Database + Integrity Constraints + Changes:

1. Final Database:

```SQL
CREATE TABLE Student ( PERM CHAR(7) PRIMARY KEY,
                       Name VARCHAR(30),
                       Address VARCHAR(50),
                       PIN CHAR(4),
                       MajorName VARCHAR(30) NOT NULL,
                       did VARCHAR(20) NOT NULL,
                       FOREIGN KEY MajorName REFERENCES Major(MajorName),
                       FOREIGN KEY did REFERENCES Department(did));
```

```SQL
CREATE TABLE Course (  CourseNumber VARCHAR(7) PRIMARY KEY CHECK
                       LENGTH(CourseNumber) BETWEEN 3 AND 7,
                       FirstName VARCHAR(15),
                       LastName VARCHAR(15),
                       MaxEnrollment INTEGER,
                       BuildingCode VARCHAR(10) CHECK LENGTH(BuildingCode)
                       BETWEEN 1 AND 5,
                       RoomNum VARCHAR(10) CHECK LENGTH(RoomNum) BETWEEN 3 AND 4,
                       CourseTitle VARCHAR(20));
```

```SQL
CREATE TABLE CourseOfferings (CourseNumber VARCHAR(7),
                     CourseOfferingslot VARCHAR(10),
                     YearQuarterOffered VARCHAR(5)
                     EnrollmentCode VARCHAR(10),
                     PRIMARY KEY (CourseNumber, YearQuarterOffered),
                     FOREIGN KEY (CourseNumber) REFERENCES Course(CourseNumber)
                            ON DELETE CASCADE);
```

```SQL
CREATE TABLE Major (MajorName VARCHAR(30) PRIMARY KEY,
                     NumElectives INTEGER,
                     did VARCHAR(20) NOT NULL UNIQUE,
                     FOREIGN KEY (did) REFERENCES Department(did));
```

```SQL
CREATE TABLE Department (did VARCHAR(20) PRIMARY KEY);
```

```SQL
CREATE TABLE Completed (PERM CHAR(7),
                        CourseNumber VARCHAR(7),
                        YearQuarterOffered VARCHAR(10) NOT NULL,
                        Grade VARCHAR(2) CHECK LENGTH(Grade) BETWEEN 1 AND 2,
                        PRIMARY KEY (PERM, CourseNumber, YearQuarterOffered),
                        FOREIGN KEY (PERM) REFERENCES Student(PERM)
                                ON DELETE CASCADE,
                        FOREIGN KEY (CourseNumber, YearQuarterOffered) REFERENCES
Course(CourseNumber, YearQuarterOffered) ON DELETE CASCADE
                        );
```

```SQL
CREATE TABLE EnrolledIn       (PERM CHAR(7),
                        CourseNumber VARCHAR(7),
                        EnrollmentCode VARCHAR(10) NOT NULL,
                        PRIMARY KEY (PERM, CourseNumber),
                        FOREIGN KEY (PERM) REFERENCES Student(PERM)
                                ON DELETE CASCADE,
                        FOREIGN KEY (CourseNumber) REFERENCES
                        Course(CourseNumber)
                        );
```

```SQL
CREATE TABLE ElectiveCourses  (MajorName VARCHAR(30),
                                CourseNumber VARCHAR(7),
                                PRIMARY KEY (MajorName, CourseNumber),
                                FOREIGN KEY (CourseNumber) REFERENCES
                                Course(CourseNumber));
                                FOREIGN KEY (MajorName) REFERENCES
                                Major(MajorName)
                                        ON DELETE CASCADE);
```

```SQL
CREATE TABLE MandatoryCourses (MajorName VARCHAR(30),
                                CourseNumber VARCHAR(7),
                                PRIMARY KEY (MajorName, CourseNumber),
                                FOREIGN KEY (CourseNumber) REFERENCES
```

```
                                    Course(CourseNumber)
                                    FOREIGN KEY (MajorName) REFERENCES
                                    Major(MajorName)
                                           ON DELETE CASCADE);
```

```
SQL
CREATE TABLE Prerequisite     (CourseNumber VARCHAR(7),
                               PrereqCourseNumber VARCHAR(7),
                               PRIMARY KEY (CourseNumber, PrereqCourseNumber
                               FOREIGN KEY (CourseNumber) REFERENCES
                               Course(CourseNumber),
                               FOREIGN KEY (PrereqCourseNumber) REFERENCES
                               Course(CourseNumber)
                               );
```

2. Integrity Constraints:
    a. **Primary key Student.PERM**
        - Participation: Total participation from Student to Major/Departments
        - Key Constraints: from Student to Major/Departments
        - Foreign Keys: Major.MajorName, Department.did

    b. **Primary key Course.CouseNumber**
        - Participation: none
        - Key Constraints: none
        - Foreign Keys: none
        - Weak Entity with CourseOfferings entity

    c. **Primary key CourseOfferings.CourseNumber**
        - Participation: none
        - Key Constraints: none
        - Foreign Keys: Course.CourseNumber

    d. **Primary key Majors.MajorName**
        - Participation: Total Participation from Major to Department, Total Participation from Major to ElectiveCourses, Total Participation from Major to MandatoryCourses
        - Key Constraints: from Major to Department
        - Foreign Keys: Department.did

    e. **Primary key Department.did**
        - Participation: none
        - Key Constraints: none
        - Foreign Keys: none

f. **Primary key Completed.PERM Foreign Students(PERM)**
   - Participation: Total Participation from Completed to CourseOfferings
   - Key Constraints: from Completed to CourseOfferings
   - Foreign Keys: Student.PERM, Course.CourseNumber, CourseOfferings.YearQuarterOffered

g. **Primary key EnrolledIn.PERM**
   - Participation: EnrolledIn.EnrollmentCode, Total Participation from EnrolledIn to CourseOfferings
   - Key Constraints: from EnrolledIn to CourseOfferings
   - Foreign Keys: Student.PERM, Course.CourseNumber

h. **Primary key ElectiveCourses.MajorName**
   - Participation: none
   - Key Constraints: none
   - Foreign Keys: Major.MajorName, Courses.CourseNumber

i. **Primary key MandatoryCourses.MajorName**
   - Participation: none
   - Key Constraints: none
   - Foreign Keys: Major.MajorName, Courses.CourseNumber

j. **Primary key Prerequisite.CourseNumber**
   - Participation: none
   - Key Constraints: none
   - Foreign Keys: Course.CourseNumber

3. Changes:
   a. In **CourseOfferings** table:
      i. The primary key on CourseNumber was changed to a composite primary key on both CourseNumber and YearQuarterOffered. This was to allow for one course to be offered multiple times in previous quarters and years.
      ii. The CHECK statement on YearQuarterOffered was removed, as it was not a valid command in Oracle SQL.
      iii. EnrollmentCode variable was added to the table in order to track enrollment codes for each course offering.
   b. In **Completed** table:
      i. The primary key on PERM was changed to a composite primary key on PERM, CourseNumber, YearQuarterOffered to allow for students to complete multiple courses in multiple years/quarters, but only be able to complete a specific course once.
   c. In **EnrolledIn** table:

    i. The primary key on PERM was changed to a composite primary key on PERM and CourseNumber. This allows the table to track a student enrolled in multiple courses.
    ii. The CHECK statement was removed, and instead enforced in the java application logic.
d. In **ElectiveCourses** table:
    i. The primary key on MajorName was changed to a composite primary key on MajorName and CourseNumber. This allows for one major to have multiple elective course offerings.
e. In **MandatoryCourses** table:
    i. The primary key on MajorName was changed to a composite primary key on MajorName and CourseNumber. This allows for one major to have multiple mandatory course offerings.
f. In **Prerequisites** table:
    i. A variable called PrereqCourseNumber was introduced to track the course(s) that are prerequisites for other courses. This is a foreign key referencing CourseNumber from the Course table
    ii. The primary key on CourseNumber was changed to a composite primary key on CourseNumber and PrereqCourseNumber

3. Violation of Integrity Constraints + Workarounds:

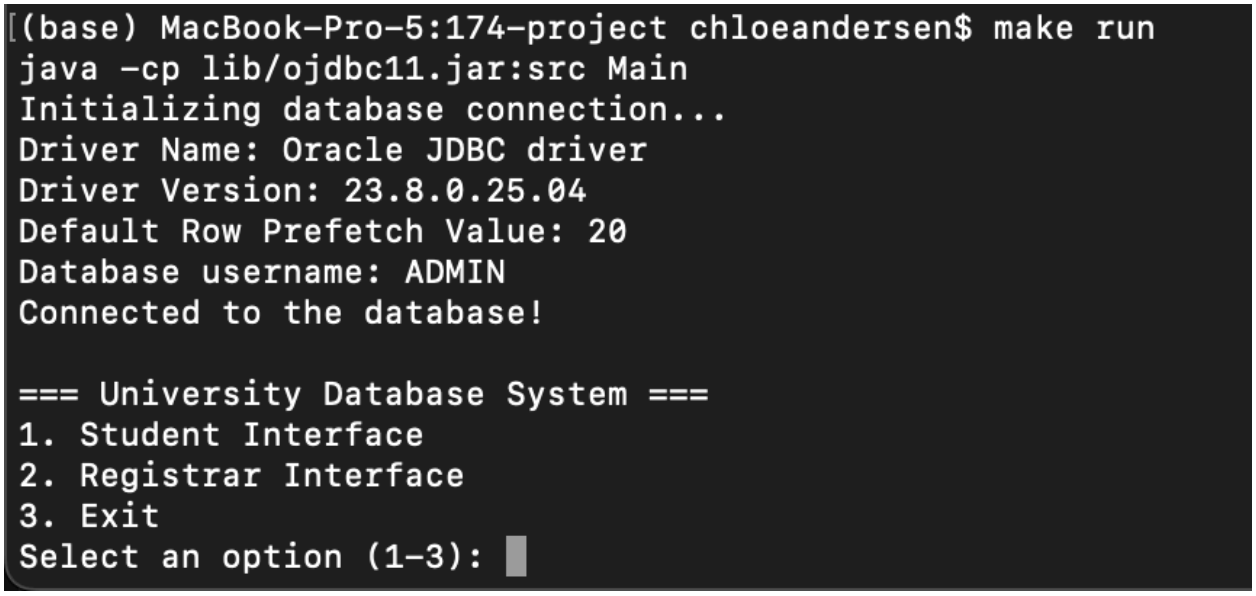
We ran into Integrity Constraints violations in the **CourseOfferings, Completed, EnrolledIn, ElectiveCourses, MandatoryCourses**, and **Prerequisites** tables when building out this project from our initial data design report.

For the **CourseOfferings, Completed, EnrolledIn, ElectiveCourses,** and **MandatoryCourses** tables, we had defined exactly one primary key. This turned out to be too restrictive. For example, in the **ElectiveCourses** table, we had the primary key be MajorName. This effectively meant that each major could only have one elective course. As we began implementation, we quickly realized that we would need to convert this singular primary key to a composite key, in order to allow multiple elective courses per major. A similar change was made to the **CourseOfferings, Completed, EnrolledIn,** and **MandatoryCourses** tables.

The other Integrity Constraint violation we ran into was within the **Prerequisites** table. Initially, the table only had one attribute (CourseNumber) which was also the primary key. Issues quickly arose, as we realized we would have to be able to store a correlation between two courses to display a prerequisite relationship, not just a singular course number. Additionally, we had to convert to a composite key, as one course number had to be able to be associated with multiple prerequisite course numbers (if one course has more than one prerequisite).

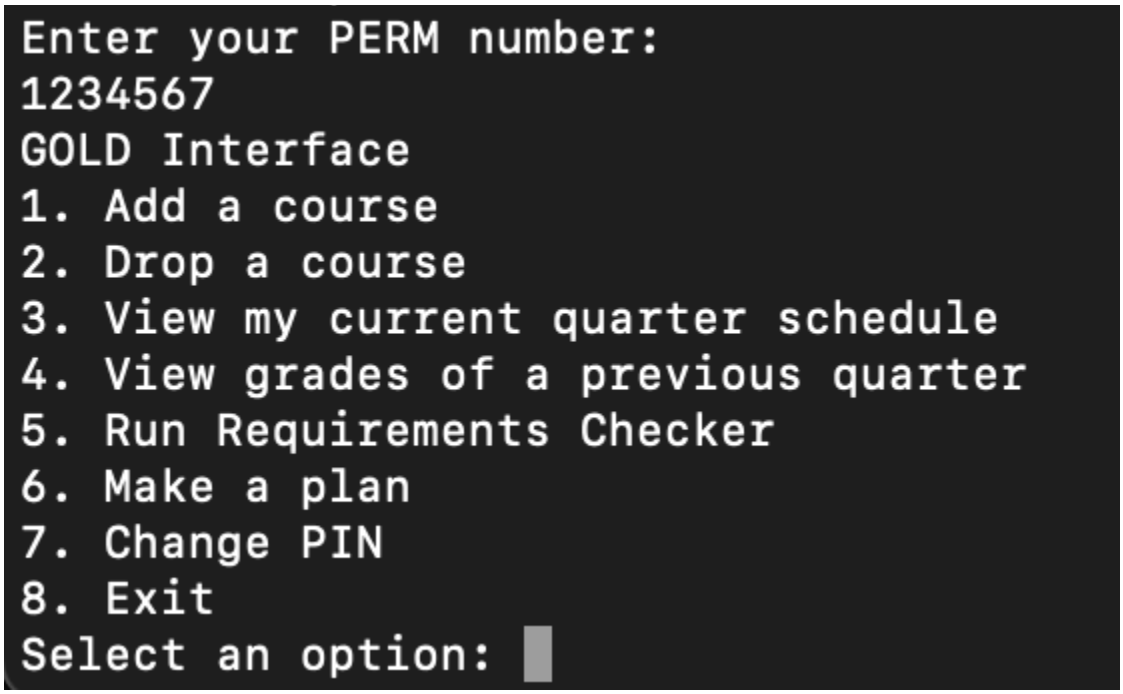
4. Interfaces + Descriptions:

1. MAIN Interface:



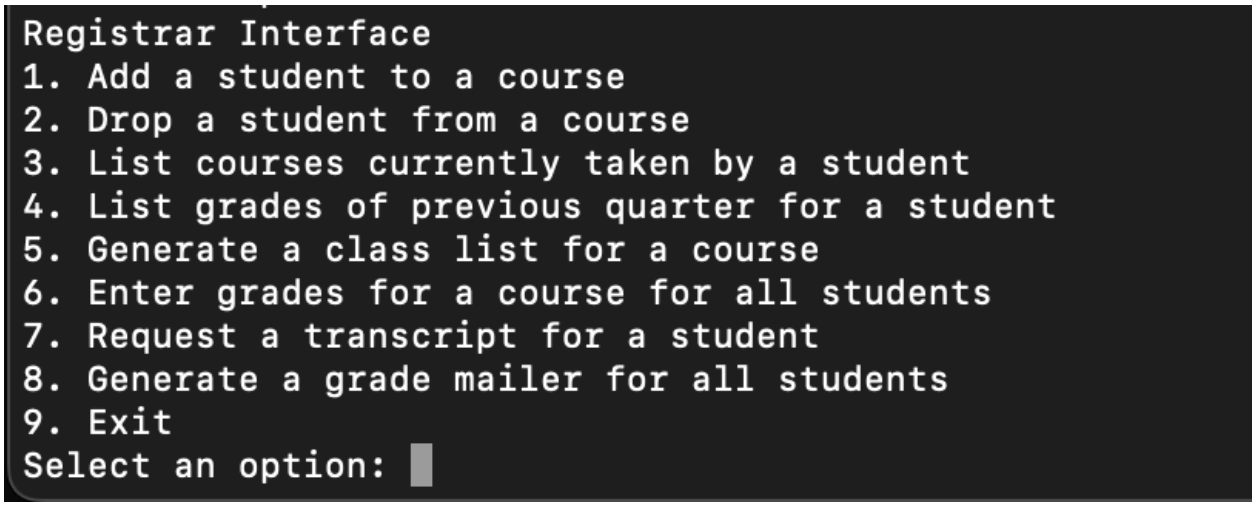
Description: after running ***make*** & ***make run***, this is the interface the user is met with. They are able to choose whether they want to enter the Student Interface, Registrar Interface, or exit the application.

2. STUDENT Interface:



Description: Upon entering **1** in the Main Interface, the user enters the Student/GOLD Interface. To be able to see any operations, the Student must enter their PERM number. Afterwards,they are able to perform the academic transactions enumerated above (numbers 1-8).

3. REGISTRAR Interface:



Description: Upon entering **2** in the Main Interface, the user enters the Staff/Registrar Interface. Within each transaction (numbers 1-9), the staff has to enter the PERM for the student that they want to manipulate data of. Then, they are able to perform the enumerated operations on the student's courses/academics.

5. All Java Classes + Methods:

Codebase is available at: https://github.com/chloeandersen-ucsb/174-project (currently private with Professor + TAs added as collaborators, please let us know if we should make it public)

Public Class Main:

The Main.java file is where the User first establishes a connection with the database, and chooses the interface that they would like to enter. This is handled primarily by the public class Main.

Java file with **public class Main** and all corresponding helper functions within: https://github.com/chloeandersen-ucsb/174-project/blob/main/src/Main.java

Public class DatabaseConnector:

The DatabaseConnector.java is a file referenced from section, and used within Main.java to establish a connection with the Oracle Database. All functionality occurs within the Public class DatabaseConnector in this file.

Java file with **public class DatabaseConnector** and all corresponding helper functions within: https://github.com/chloeandersen-ucsb/174-project/blob/main/src/DatabaseConnector.java

Public class StudentInterface:

The file StudentInterface.java has one public class called StudentInterface, that contains many helper files referenced within the public class itself, as well as the public class Main in Main.java. The functions help build the Student interface and reference SQL commands to perform operations that students are able to make on their data.

Java file with **public class StudentInterface** and all corresponding helper functions: https://github.com/chloeandersen-ucsb/174-project/blob/main/src/StudentInterface.java

1. **Add a course (referenced in displayMenu() function)**
    a. SQL Procedure:

```
Java
create or replace PROCEDURE AddCourse (
        p_perm CHAR,
        p_courseNumber VARCHAR,
        p_yearQuarter VARCHAR
)
AS
   v_count INTEGER;
   v_met_prereq BOOLEAN := TRUE;
   v_max INTEGER;
   v_enrolled INTEGER;
   v_enrollmentCode VARCHAR(10);
BEGIN
   -- see if student is already enrolled in 5 courses
   SELECT COUNT(*) INTO v_count
   FROM EnrolledIn
   WHERE PERM = p_perm;
   IF v_count >= 5 THEN
       RAISE_APPLICATION_ERROR(-20001, 'Student cannot enroll in more than 5 courses.');
   END IF;

       -- see if student is already enrolled in the course
   SELECT COUNT(*) INTO v_count
   FROM EnrolledIn
   WHERE PERM = p_perm AND CourseNumber = p_courseNumber;
   IF v_count > 0 THEN
       RAISE_APPLICATION_ERROR(-20002, 'Student is already enrolled in this course.');
   END IF;


   -- make sure all prerequisites are met for student to enroll in course
   FOR prereq IN (
       SELECT PrereqCourseNumber
       FROM Prerequisite
       WHERE CourseNumber = p_courseNumber
   )
```

```
   LOOP
       SELECT COUNT(*) INTO v_count
       FROM Completed
       WHERE PERM = p_perm AND CourseNumber = prereq.PrereqCourseNumber
             AND Grade IN ('A','A+','A-','B','B+','B-','C','C+','C-');
       IF v_count = 0 THEN
           v_met_prereq := FALSE; -- if a student has not completed ONE prerequisite, they
cannot enroll in the course
       END IF;
   END LOOP;
   IF NOT v_met_prereq THEN
       RAISE_APPLICATION_ERROR(-20003, 'Student is missing the prerequisites to enroll in
this course.');
   END IF;

   -- check max enrollment
   SELECT MaxEnrollment INTO v_max FROM Course WHERE CourseNumber = p_courseNumber;
   SELECT COUNT(*) INTO v_enrolled FROM EnrolledIn WHERE CourseNumber = p_courseNumber;
   IF v_enrolled >= v_max THEN
       RAISE_APPLICATION_ERROR(-20004, 'Student cannot enroll in this course because it is
full.');
   END IF;

   -- check course exists in CourseOfferings before enrolling student
   SELECT COUNT(*) INTO v_count
   FROM CourseOfferings
   WHERE CourseNumber = p_courseNumber
   AND YearQuarterOffered = p_yearQuarter;
   IF v_count = 0 THEN
       RAISE_APPLICATION_ERROR(-20005, 'Cannot enroll Student because course is not
offered in this quarter.');
   END IF;

   -- get enrollment code for course and quarter
   SELECT EnrollmentCode INTO v_enrollmentCode
   FROM CourseOfferings
   WHERE CourseNumber = p_courseNumber
   AND YearQuarterOffered = p_yearQuarter;

   -- if all looks good, enroll student
   INSERT INTO EnrolledIn (PERM, CourseNumber, EnrollmentCode)
   VALUES (p_perm, p_courseNumber, v_enrollmentCode);

   DBMS_OUTPUT.PUT_LINE('Student successfully enrolled in course.');
END AddCourse;
```

2. Drop a course

a. SQL Procedure:

```
Java
create or replace PROCEDURE DropCourse (
        p_perm CHAR,
        p_courseNumber VARCHAR
)
AS
   v_enrolledCount INTEGER;
   v_enrolledInCourse INTEGER;
BEGIN

   -- check total number of courses the student is enrolled in to make sure they are not
dropping the only course they're enrolled in
   SELECT COUNT(*) INTO v_enrolledCount
   FROM EnrolledIn
   WHERE PERM = p_perm;
   IF v_enrolledCount <= 1 THEN
       RAISE_APPLICATION_ERROR(-20001, 'Student cannot drop the only course they are
enrolled in.');
   END IF;


   -- check that the student is actually enrolled in the course they are trying to drop
   SELECT COUNT(*) INTO v_enrolledInCourse
   FROM EnrolledIn
   WHERE PERM = p_perm AND CourseNumber = p_courseNumber;
   IF v_enrolledInCourse = 0 THEN
       RAISE_APPLICATION_ERROR(-20002, 'Student is not enrolled in the specified course,
and therefore cannot drop it.');
   END IF;


   -- drop course
   DELETE FROM EnrolledIn
   WHERE PERM = p_perm AND CourseNumber = p_courseNumber;

   DBMS_OUTPUT.PUT_LINE('Student successfully dropped the course.');
END DropCourse;
```

3. View my current quarter schedule (referenced in displayMenu() function)

a. SQL:

```
SQL
SELECT e.CourseNumber, co.YearQuarterOffered, c.CourseTitle,
                          co.CourseOfferingslot, co.EnrollmentCode
FROM EnrolledIn e
JOIN CourseOfferings co ON e.CourseNumber = co.CourseNumber
                       AND co.YearQuarterOffered = ?
JOIN Course c ON e.CourseNumber = c.CourseNumber WHERE e.PERM = ?
```

4. **View grades of a previous quarter (referenced in displayMenu() function)**
    a. SQL:

```sql
SELECT c.CourseNumber, c.CourseTitle, co.Grade
FROM Completed co
JOIN Course c ON co.CourseNumber = c.CourseNumber
WHERE co.PERM = ?
AND co.YearQuarterOffered = ?
```

5. **Run requirements checker (referenced in displayMenu() function)**
    a. SQL Procedure

```sql
CREATE OR REPLACE PROCEDURE CheckRequirements (
   p_perm CHAR,
   p_currentQuarter VARCHAR
)
AS
   v_major VARCHAR(30);

   v_required_electives INT;

   -- mandatory variables
   v_met_mandatory_list VARCHAR(1000) := '';
   v_in_progress_mandatory_list VARCHAR(1000) := '';
   v_missing_mandatory_list VARCHAR(1000) := '';
   v_missing_mandatory_count INT := 0;

   -- electives variables
   v_met_electives_list VARCHAR(1000) := '';
   v_in_progress_electives_list VARCHAR(1000) := '';
   v_missing_electives INT := 0;
BEGIN
   -- major and elective requirement
   SELECT s.MajorName, m.NumElectives INTO v_major, v_required_electives
   FROM Student s JOIN Major m ON s.MajorName = m.MajorName
   WHERE s.PERM = p_perm;

   -- mandatory courses
   FOR mc IN (
       SELECT CourseNumber
       FROM MandatoryCourses
       WHERE MajorName = v_major
   )
   LOOP
       DECLARE
```

```
        v_completed_count INT;
        v_in_progress_count INT;
    BEGIN
        SELECT COUNT(*) INTO v_completed_count
        FROM Completed
        WHERE PERM = p_perm
          AND CourseNumber = mc.CourseNumber
          AND Grade IN ('A','A+','A-','B','B+','B-','C','C+','C-');

        SELECT COUNT(*) INTO v_in_progress_count
        FROM EnrolledIn e
        JOIN CourseOfferings co ON e.CourseNumber = co.CourseNumber
        WHERE e.PERM = p_perm
          AND e.CourseNumber = mc.CourseNumber
          AND co.YearQuarterOffered = p_currentQuarter;

        IF v_completed_count > 0 THEN
            v_met_mandatory_list := CASE
                WHEN v_met_mandatory_list IS NULL THEN mc.CourseNumber
                ELSE v_met_mandatory_list || ', ' || mc.CourseNumber
            END;
        ELSIF v_in_progress_count > 0 THEN
            v_in_progress_mandatory_list := CASE
                WHEN v_in_progress_mandatory_list IS NULL THEN mc.CourseNumber
                ELSE v_in_progress_mandatory_list || ', ' || mc.CourseNumber
            END;
        ELSE
            v_missing_mandatory_list := CASE
                WHEN v_missing_mandatory_list IS NULL THEN mc.CourseNumber
                ELSE v_missing_mandatory_list || ', ' || mc.CourseNumber
            END;
            v_missing_mandatory_count := v_missing_mandatory_count + 1;
        END IF;
    END;
END LOOP;

-- electives
DECLARE
    v_taken_electives INT := 0;
    v_in_progress_electives INT := 0;
BEGIN
    FOR ec IN (
        SELECT CourseNumber
        FROM ElectiveCourses
        WHERE MajorName = v_major
    )
    LOOP
        DECLARE
            v_completed INT;
            v_in_progress INT;
```

```
            BEGIN
                SELECT COUNT(*) INTO v_completed
                FROM Completed
                WHERE PERM = p_perm
                  AND CourseNumber = ec.CourseNumber
                  AND Grade IN ('A','A+','A-','B','B+','B-','C','C+','C-');

                SELECT COUNT(*) INTO v_in_progress
                FROM EnrolledIn e
                JOIN CourseOfferings co ON e.CourseNumber = co.CourseNumber
                WHERE e.PERM = p_perm
                  AND e.CourseNumber = ec.CourseNumber
                  AND co.YearQuarterOffered = p_currentQuarter;

                IF v_completed > 0 THEN
                    v_met_electives_list := CASE
                        WHEN v_met_electives_list IS NULL THEN ec.CourseNumber
                        ELSE v_met_electives_list || ', ' || ec.CourseNumber
                    END;
                    v_taken_electives := v_taken_electives + 1;
                ELSIF v_in_progress > 0 THEN
                    v_in_progress_electives_list := CASE
                        WHEN v_in_progress_electives_list IS NULL THEN ec.CourseNumber
                        ELSE v_in_progress_electives_list || ', ' || ec.CourseNumber
                    END;
                    v_in_progress_electives := v_in_progress_electives + 1;
                END IF;
            END;
        END LOOP;

        -- dynamically calculate missing electives
        v_missing_electives := GREATEST(v_required_electives - v_taken_electives -
v_in_progress_electives, 0);
    END;

    DBMS_OUTPUT.PUT_LINE('Required Met: ' || NVL(v_met_mandatory_list, 'None'));
    DBMS_OUTPUT.PUT_LINE('Required in Progress ' || p_currentQuarter || ': ' ||
NVL(v_in_progress_mandatory_list, 'None'));
    DBMS_OUTPUT.PUT_LINE('Remaining Required: ' || NVL(v_missing_mandatory_list, 'None'));
    DBMS_OUTPUT.PUT_LINE('----------------');
    DBMS_OUTPUT.PUT_LINE('Electives Met: ' || NVL(v_met_electives_list, 'None'));
    DBMS_OUTPUT.PUT_LINE('Electives In Progress ' || p_currentQuarter || ': ' ||
NVL(v_in_progress_electives_list, 'None'));
    DBMS_OUTPUT.PUT_LINE('Remaining Electives: ' || v_missing_electives);
END CheckRequirements;
```

6. **Make a plan (referenced in displayMenu() function)**
    a. SQL Procedure:

SQL

```
create or replace PROCEDURE MakePlan (
        p_perm CHAR,
        p_currentQuarter VARCHAR
)
AS
   v_major VARCHAR(30);
   v_num_electives INT;
   v_quarters DBMS_SQL.VARCHAR2_TABLE := DBMS_SQL.VARCHAR2_TABLE();
   v_course_counter INT;
BEGIN
   -- Get student's major and elective requirements
   SELECT s.MajorName, m.NumElectives
   INTO v_major, v_num_electives
   FROM Student s
   JOIN Major m ON s.MajorName = m.MajorName
   WHERE s.PERM = p_perm;

   -- Define quarters to plan (given sample data with 3 future quarters)
   v_quarters(1) := '25F';
   v_quarters(2) := '26W';
   v_quarters(3) := '26S';

   -- Loop over the future quarters
   FOR q IN 1 .. v_quarters.COUNT LOOP
       DECLARE
           v_quarter VARCHAR2(10) := v_quarters(q);
           v_count INT := 0;
       BEGIN
           -- don't need to plan for current quarter
           IF v_quarter = p_currentQuarter THEN
               DBMS_OUTPUT.PUT_LINE('Quarter ' || v_quarter || ': (skipped - current
quarter)');
               CONTINUE;
           END IF;
           DBMS_OUTPUT.PUT_LINE('Quarter ' || v_quarter || ':');

           FOR course_rec IN (
               SELECT CourseNumber
               FROM (
                   -- IF MISSING MANDATORY COURSE, ADD IT TO THE PLAN
                   -- similar implementation from requirements check
                   SELECT mc.CourseNumber, 'MANDATORY' AS Type
                   FROM MandatoryCourses mc
                   WHERE mc.MajorName = v_major
                     AND mc.CourseNumber NOT IN (
                         SELECT CourseNumber
                         FROM Completed
                         WHERE PERM = p_perm
                           AND Grade IN ('A', 'A+', 'A-', 'B', 'B+', 'B-', 'C', 'C+',
'C-')
```

```
                    )

                UNION ALL

                -- IF MISSING ELECTIVES, ADD IT TO THE PLAN
                -- similar implementation from requirements check
                SELECT ec.CourseNumber, 'ELECTIVE' AS Type
                FROM ElectiveCourses ec
                WHERE ec.MajorName = v_major
                  AND ec.CourseNumber NOT IN (
                      SELECT CourseNumber
                      FROM Completed
                      WHERE PERM = p_perm
                        AND Grade IN ('A', 'A+', 'A-', 'B', 'B+', 'B-', 'C', 'C+',
'C-')
                  )
            ) all_missing
            WHERE CourseNumber IN (
                SELECT co.CourseNumber
                FROM CourseOfferings co
                WHERE co.YearQuarterOffered = v_quarter
            )
            AND NOT EXISTS (
                SELECT 1
                FROM Prerequisite p
                WHERE p.CourseNumber = all_missing.CourseNumber
                  AND p.PrereqCourseNumber NOT IN (
                      SELECT CourseNumber
                      FROM Completed
                      WHERE PERM = p_perm
                        AND Grade IN ('A', 'A+', 'A-', 'B', 'B+', 'B-', 'C', 'C+',
'C-')
                  )
            )
            FETCH FIRST 5 ROWS ONLY
        )
        LOOP
            DBMS_OUTPUT.PUT_LINE('  - ' || course_rec.CourseNumber);
            v_count := v_count + 1;
        END LOOP;

        IF v_count = 0 THEN
            DBMS_OUTPUT.PUT_LINE('  (no eligible courses)');
        END IF;
    END;
END LOOP;

-- Need to indicate if any courses are unschedulable but still need to be taken
eventually
DBMS_OUTPUT.PUT_LINE(CHR(10) || 'Unscheduled Courses:');
```

```
   FOR course IN (
       SELECT CourseNumber
       FROM (
           SELECT mc.CourseNumber
           FROM MandatoryCourses mc
           WHERE mc.MajorName = v_major
             AND mc.CourseNumber NOT IN (
                 SELECT CourseNumber
                 FROM Completed
                 WHERE PERM = p_perm
                   AND Grade IN ('A', 'A+', 'A-', 'B', 'B+', 'B-', 'C', 'C+', 'C-')
             )
           UNION
           SELECT ec.CourseNumber
           FROM ElectiveCourses ec
           WHERE ec.MajorName = v_major
             AND ec.CourseNumber NOT IN (
                 SELECT CourseNumber
                 FROM Completed
                 WHERE PERM = p_perm
                   AND Grade IN ('A', 'A+', 'A-', 'B', 'B+', 'B-', 'C', 'C+', 'C-')
             )
       )
       WHERE CourseNumber NOT IN (
           SELECT DISTINCT CourseNumber
           FROM CourseOfferings
           WHERE YearQuarterOffered IN ('25F', '26W', '26S')
       )
   )
   LOOP
       DBMS_OUTPUT.PUT_LINE('  - ' || course.CourseNumber);
   END LOOP;
END MakePlan;
```

7. Change PIN (referenced in displayMenu() function)

a. SQL:

```
SQL
create or replace PROCEDURE SetPin (
   p_perm CHAR,
   p_old_pin VARCHAR2,
   p_new_pin VARCHAR2
)
AS
   v_stored_hash VARCHAR2(64);
   v_old_hash    VARCHAR2(64);
```

```
    v_new_hash    VARCHAR2(64);
BEGIN
    -- Get current PIN hash from the DB
    SELECT PIN INTO v_stored_hash
    FROM Student
    WHERE PERM = p_perm;

    -- Hash the old and new PINs using SQL context
    SELECT STANDARD_HASH(p_old_pin, 'SHA256') INTO v_old_hash FROM dual;
    SELECT STANDARD_HASH(p_new_pin, 'SHA256') INTO v_new_hash FROM dual;

    -- Compare hashes
    IF v_stored_hash != v_old_hash THEN
        RAISE_APPLICATION_ERROR(-20001, 'Old PIN is incorrect.');
    END IF;

    -- Update to new PIN hash
    UPDATE Student
    SET PIN = v_new_hash
    WHERE PERM = p_perm;

    DBMS_OUTPUT.PUT_LINE('PIN updated successfully.');
END;
```

Public class RegistrarInterface:

The file RegistrarInterface.java has one public class called RegistrarInterface, that contains many helper files referenced within the public class itself, as well as the public class Main in Main.java. The functions help build the Registrar interface, and call SQL commands to perform operations that staff are able to make on student's data.

Java file with **public class RegistrarInterface** and **all corresponding helper functions**: https://github.com/chloeandersen-ucsb/174-project/blob/main/src/RegistrarInterface.java

1. **Add a student to a course (referenced in displayMenu() function)**
    a. SQL Procedure (same one used in Student Interface)

```
SQL
create or replace PROCEDURE AddCourse (
        p_perm CHAR,
        p_courseNumber VARCHAR,
        p_yearQuarter VARCHAR
)
AS
   v_count INTEGER;
   v_met_prereq BOOLEAN := TRUE;
   v_max INTEGER;
   v_enrolled INTEGER;
   v_enrollmentCode VARCHAR(10);
BEGIN
   SELECT COUNT(*) INTO v_count
   FROM EnrolledIn
   WHERE PERM = p_perm;
   IF v_count >= 5 THEN
       RAISE_APPLICATION_ERROR(-20001, 'Student cannot enroll in more than 5 courses.');
   END IF;
   SELECT COUNT(*) INTO v_count
   FROM EnrolledIn
   WHERE PERM = p_perm AND CourseNumber = p_courseNumber;
   IF v_count > 0 THEN
       RAISE_APPLICATION_ERROR(-20002, 'Student is already enrolled in this course.');
   END IF;
   FOR prereq IN (
       SELECT PrereqCourseNumber
       FROM Prerequisite
       WHERE CourseNumber = p_courseNumber
   )
   LOOP
       SELECT COUNT(*) INTO v_count
       FROM Completed
       WHERE PERM = p_perm AND CourseNumber = prereq.PrereqCourseNumber
             AND Grade IN ('A','A+','A-','B','B+','B-','C','C+','C-');
       IF v_count = 0 THEN
```

```
            v_met_prereq := FALSE;
        END IF;
    END LOOP;
    IF NOT v_met_prereq THEN
        RAISE_APPLICATION_ERROR(-20003, 'Student is missing the prerequisites to enroll in
this course.');
    END IF;
    SELECT MaxEnrollment INTO v_max FROM Course WHERE CourseNumber = p_courseNumber;
    SELECT COUNT(*) INTO v_enrolled FROM EnrolledIn WHERE CourseNumber = p_courseNumber;
    IF v_enrolled >= v_max THEN
        RAISE_APPLICATION_ERROR(-20004, 'Student cannot enroll in this course because it is
full.');
    END IF;
    SELECT COUNT(*) INTO v_count
    FROM CourseOfferings
    WHERE CourseNumber = p_courseNumber
    AND YearQuarterOffered = p_yearQuarter;
    IF v_count = 0 THEN
        RAISE_APPLICATION_ERROR(-20005, 'Cannot enroll Student because course is not
offered in this quarter.');
    END IF;
    SELECT EnrollmentCode INTO v_enrollmentCode
    FROM CourseOfferings
    WHERE CourseNumber = p_courseNumber
    AND YearQuarterOffered = p_yearQuarter;
    INSERT INTO EnrolledIn (PERM, CourseNumber, EnrollmentCode)
    VALUES (p_perm, p_courseNumber, v_enrollmentCode);
    DBMS_OUTPUT.PUT_LINE('Student successfully enrolled in course.');
END AddCourse;
```

2. Drop a student from a course (referenced in displayMenu() function)

a. SQL Procedure (same one used in Student Interface)

```
SQL
create or replace PROCEDURE DropCourse (
        p_perm CHAR,
        p_courseNumber VARCHAR
)
AS
    v_enrolledCount INTEGER;
    v_enrolledInCourse INTEGER;
BEGIN
    SELECT COUNT(*) INTO v_enrolledCount
    FROM EnrolledIn
    WHERE PERM = p_perm;
    IF v_enrolledCount <= 1 THEN
```

```
        RAISE_APPLICATION_ERROR(-20001, 'Student cannot drop the only course they are
enrolled in.');
    END IF;
    SELECT COUNT(*) INTO v_enrolledInCourse
    FROM EnrolledIn
    WHERE PERM = p_perm AND CourseNumber = p_courseNumber;
    IF v_enrolledInCourse = 0 THEN
        RAISE_APPLICATION_ERROR(-20002, 'Student is not enrolled in the specified course,
and therefore cannot drop it.');
    END IF;
    DELETE FROM EnrolledIn
    WHERE PERM = p_perm AND CourseNumber = p_courseNumber;

    DBMS_OUTPUT.PUT_LINE('Student successfully dropped the course.');
END DropCourse;
```

3. **List courses currently taken by a student (referenced in displayMenu() function)**
    a. SQL:

```
SQL
SELECT CourseNumber FROM EnrolledIn WHERE PERM = ?
```

4. **List grades of previous quarter for a student (referenced in displayMenu() function)**
    a. SQL:

```
SQL
SELECT c.CourseNumber, c.CourseTitle, co.Grade
FROM Completed co
JOIN Course c ON co.CourseNumber = c.CourseNumber
WHERE co.PERM = ?
AND co.YearQuarterOffered = ?
```

5. **Generate class list for a course (referenced in displayMenu() function)**
    a. SQL:

```
SQL
SELECT e.PERM, s.NAME
FROM EnrolledIn e
JOIN Student s ON e.PERM = s.PERM
WHERE e.CourseNumber = ?
```

6. **Enter grades for a course for all students (referenced in displayMenu() function)**
    a. SQL:

```
SQL
SELECT CourseNumber, YearQuarterOffered
FROM CourseOfferings
WHERE EnrollmentCode = ?

-- and later...

INSERT INTO Completed (PERM, CourseNumber, YearQuarterOffered, Grade)
VALUES (?, ?, ?, ?)

DELETE FROM EnrolledIn
WHERE PERM = ? AND EnrollmentCode = ?
```

7. **Request a transcript for a student (referenced in displayMenu() function)**
    a. SQL:

```
SQL
SELECT CourseNumber, Grade, YearQuarterOffered
FROM Completed
WHERE PERM = ?
ORDER BY
        SUBSTR(YearQuarterOffered, 1, 2) DESC,
        CASE SUBSTR(YearQuarterOffered, 3, 1)
                    WHEN 'W' THEN 1
                    WHEN 'F' THEN 2
                    WHEN 'S' THEN 3
END
```

8. **Generate a grade mailer for all students (referenced in displayMenu() function)**
    a. SQL

```
SQL
SELECT s.NAME, s.PERM, c.CourseNumber, c.CourseTitle, co.Grade
FROM Student s
JOIN Completed co ON s.PERM = co.PERM
JOIN Course c ON co.CourseNumber = c.CourseNumber
WHERE co.CourseNumber = ? AND co.YearQuarterOffered = ?
ORDER BY s.NAME, c.CourseNumber
```