Class: COP 4710: Database Design

Professor: William Hendrix

Team Members: Christian Flores, Jacob Kearschner, Weijie Kang, Gerardo Wibmer

Date: 04/26/2023

Final Project Report

Our project is a class scheduler database system designed to simplify the process of scheduling classes and achieving relative aesthetic parity with the old USF schedule search functionality. The system generates schedules based on the classes requested by the user and includes a cart functionality, allowing users to view and manage the classes they have added for the moment. The database is populated with data obtained from OASIS class schedule search functionality, ensuring the accuracy and up-to-date nature of the information stored. The database consists of several interrelated tables that store information about courses, courses attributes, sections, subjects, instructors, meetings, locations, and buildings, linked by primary and foreign keys to maintain data relationships. The end result is a comprehensive and user-friendly solution for scheduling classes.

In the development of our API, we carefully evaluated various full-stack frameworks and programming languages, including Kore, Spring, C, React, and Java. After some research, we selected Spring Boot for the backend, as it offers a wealth of online resources and aligns with our team's knowledge in Java. For the frontend, we opted for the React library as it offers a manageable learning process and efficient work approach, aligning with our time limitation. Furthermore, we opted for PostgreSQL as our database solution since we are familiar with it from our coursework. Below you can see the interaction between the API components.

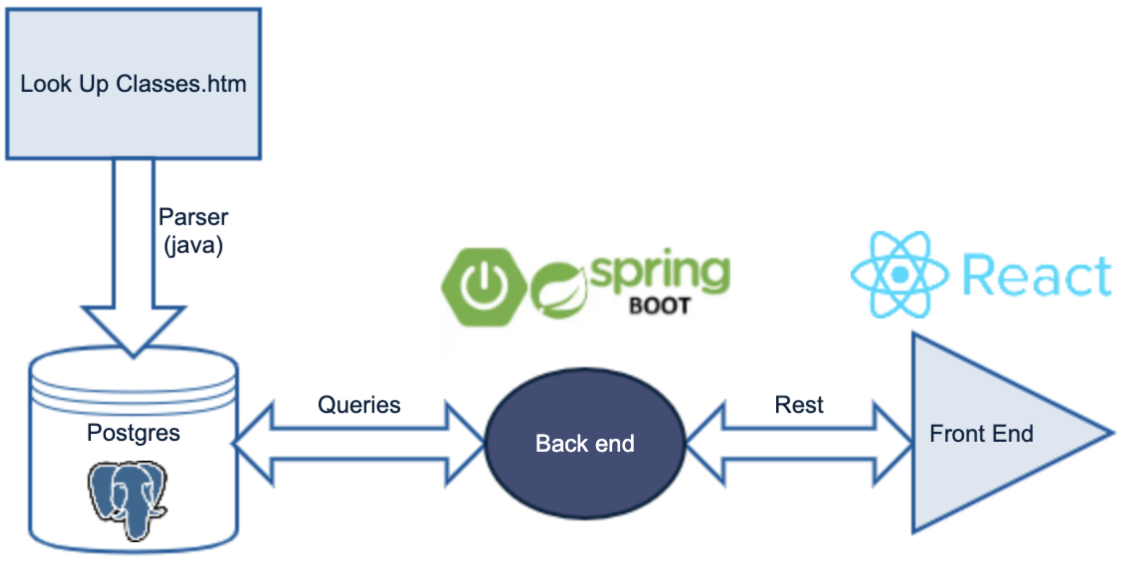
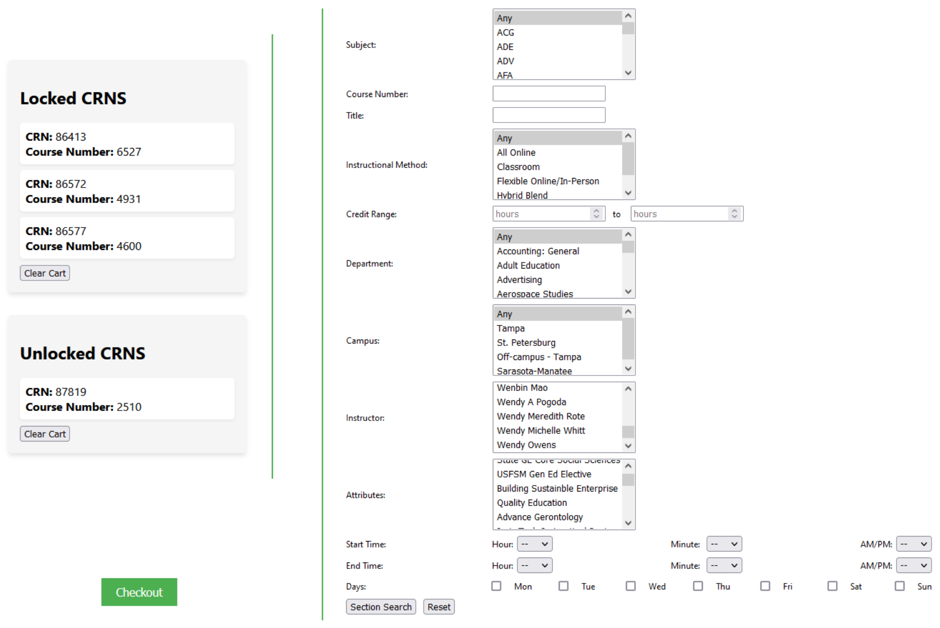


Image 1: API overview

Front End:

As previously mentioned, we utilize React for our API's frontend, the system features three distinct user interaction windows and a consistently accessible cart for viewing added classes. Initially, users encounter the search engine window, where they can input their preferences and search for classes of interest. Following this, the second window presents a comprehensive overview of all available classes that meet the user's criteria. Lastly, the third window showcases the user's schedule, generated based on the classes added to the cart. This allows users to easily manage their class schedule. Together, these three windows create a smooth and user-friendly experience for class scheduling.

In designing the first window, we drew inspiration from the USF OASIS class schedule search layout. This search engine allows users to select various criteria such as subject, course number, title, instructional method, credit range, department, campus, instructor, attributes, start time, and item. To create a user-friendly experience, we included input boxes, scroll boxes, and drop-down menus in the design, enabling users to easily navigate through the available options and customize their search preferences.


Locked CRNS

CRN: 86413
Course Number: 6527

CRN: 86572
Course Number: 4931

CRN: 86577
Course Number: 4600

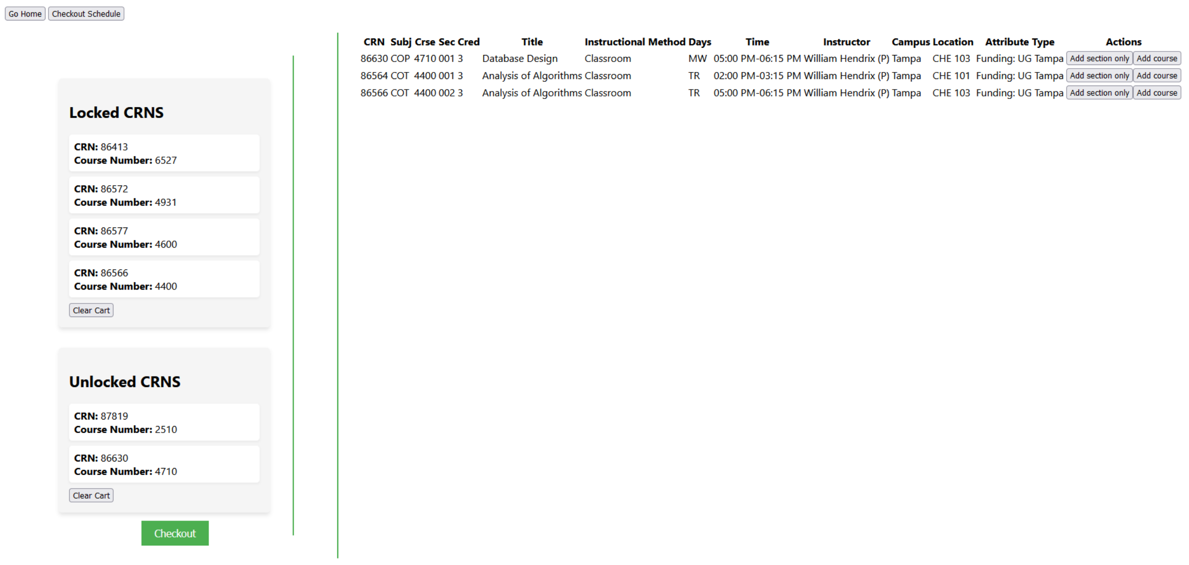
Unlocked CRNS

CRN: 87819
Course Number: 2510

Image 2: Look up Section

The search engine design integrates the automatic fetching of instructors and subjects from an API, ensuring users have access to the latest information (we'll discuss this feature in more detail later). Additionally, our search engine quickly responds to user inputs, updating the search form as they type.

In the second window of our project, we present all the classes that meet the criteria entered by the user in the search engine. This window displays important details about each of the classes, such as the instructor, the number of available seats, course information, location, CRN, and course number. To make a better user experience, we've added two buttons for each class. One button lets users add the requested class to a "shopping cart" that stores all of the desired classes in one place for the user to see. This is done by storing the CRN of the requested class inside a cookie and then displaying all of the created cookies in the cart. A second button is present for each class that adds every section of a course to the schedule parameters. This is done with the same cookie method as before but loops through each CRN with the same title. There is also a single button that can be used to clear the schedule cart. Finally there is a generate schedule button that will take the shopping cart created by the user and generate potential schedules.



CRN: 86413
Course Number: 6527

CRN: 86572
Course Number: 4931

CRN: 86577
Course Number: 4600

CRN: 86566
Course Number: 4400

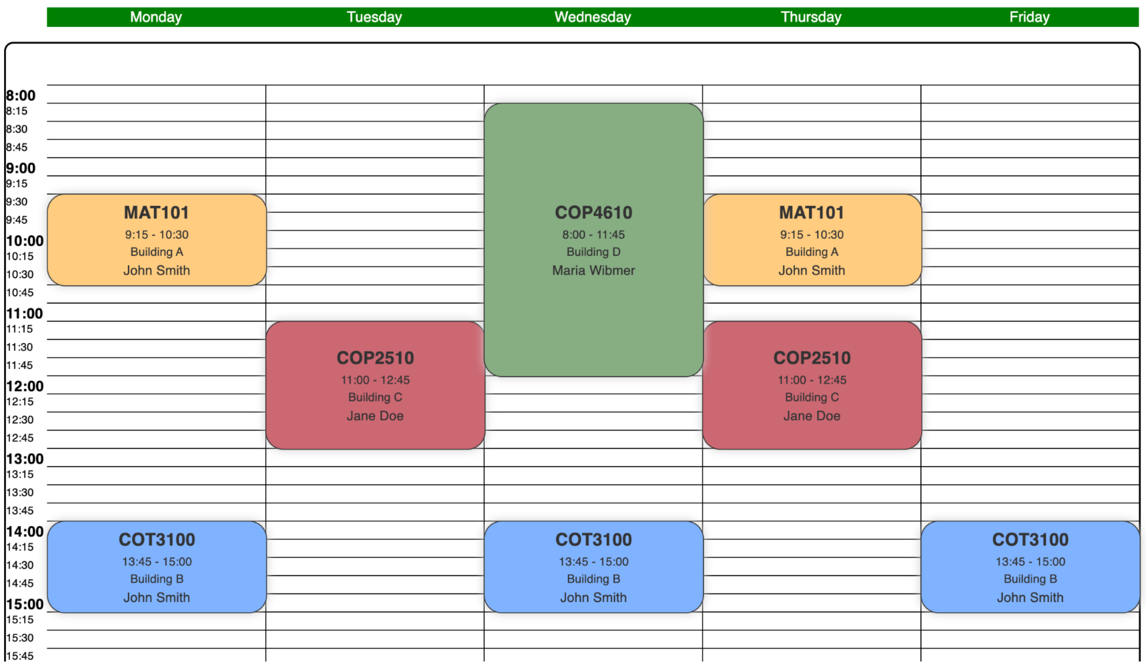
CRN: 87819
Course Number: 2510

CRN: 86630
Course Number: 4710

CRN	Subj	Crse	Sec	Cred	Title	Instructional Method	Days	Time	Instructor	Campus	Location	Attribute Type	Actions
86630	COP	4710	001	3	Database Design	Classroom	MW	05:00 PM-06:15 PM	William Hendrix (P)	Tampa	CHE 103	Funding: UG Tampa	Add section only Add course
86564	COT	4400	001	3	Analysis of Algorithms	Classroom	TR	02:00 PM-03:15 PM	William Hendrix (P)	Tampa	CHE 101	Funding: UG Tampa	Add section only Add course
86566	COT	4400	002	3	Analysis of Algorithms	Classroom	TR	05:00 PM-06:15 PM	William Hendrix (P)	Tampa	CHE 103	Funding: UG Tampa	Add section only Add course

Image 3: Look up Sections results

In the last part of our app, we've put together a nice-looking schedule using the classes that users have picked. We've made sure that it's easy to read by showing the class times, where they take place, the teacher's name, and the class title. We've also used different colors for each class to help users tell them apart. The schedule has days of the week as columns, which helps users see how their week will look. To see how this design turned out, check out Image 4 below.



Monday
Tuesday
Wednesday
Thursday
Friday
8:00
8:15
8:30
8:45
9:00
9:15
9:30
9:45
10:00
10:15
10:30
10:45
11:00
11:15
11:30
11:45
12:00
12:15
12:30
12:45
13:00
13:15
13:30
13:45
14:00
14:15
14:30
14:45
15:00
15:15
15:30
15:45
MAT101 9:15 - 10:30 Building A John Smith
COP2510 11:00 - 12:45 Building C Jane Doe
COP4610 8:00 - 11:45 Building D Maria Wibmer
MAT101 9:15 - 10:30 Building A John Smith
COP2510 11:00 - 12:45 Building C Jane Doe
COT3100 13:45 - 15:00 Building B John Smith
COT3100 13:45 - 15:00 Building B John Smith
COT3100 13:45 - 15:00 Building B John Smith

Image 4: Schedule generated

Data Collection and Parsing:

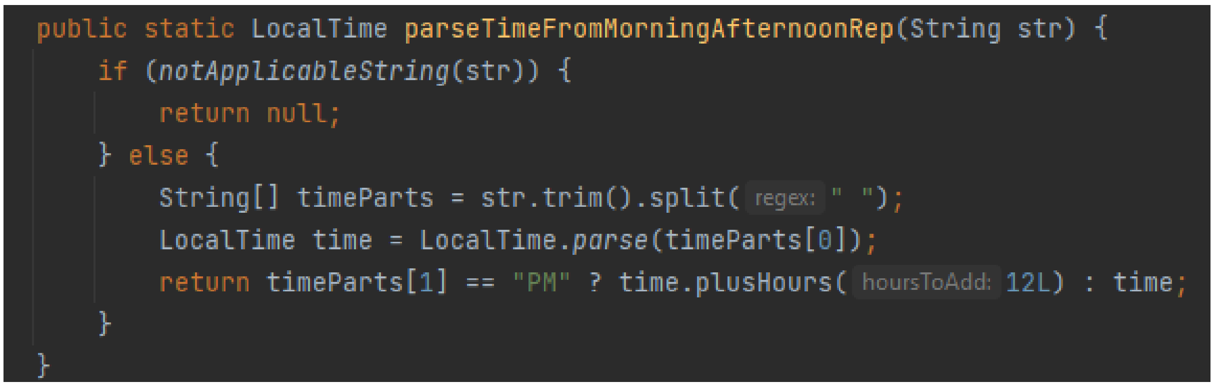
For this project, we gather data from the USF OASIS Look up classes section to populate our databases with information about courses, instructors, class schedules, locations, and other related details. To achieve this, we made a Java program called App and OasisParser.java, which together facilitated the parsing of the HTML file containing all the required data. The App class plays a crucial role in parsing the HTML content and extracting relevant data to populate various classes that represent different aspects of the class schedule. The code uses the jsoup library to parse the HTML content and navigate through the DOM elements.

Some important Parsing functions :

1. **extractString:** Retrieves the text content of an HTML node.
2. **notApplicableString:** Determines if a string contains "N/A" or "TBA".
3. **getDepartment**: Obtains the department name from the row node.
4. **parseTimeFromMorningAfternoonRep:** Converts a time string into a LocalTime object.
5. **isAdditionalMeetingRow:** Identifies if the current row is an additional meeting row.
6. **populateMeetings:** Fills a list of Meeting objects based on the meeting row nodes.
7. **extractLowerCredits:** Gathers the lower credit value from the credit string.
8. **parseDateWithArbitraryYear:** Transforms a date string with an arbitrary year into a LocalDate object.
9. **parseInstructors:** Analyzes the instructors and returns an InstructorsParseResult object.
10. **parseAttributes:** Examines the attributes and returns an AttributesParseResult object.
11. **getSection:** Obtains a Section object from the row node and the department.
12. **createHTMLDocFromFile:** Generates an HTML document from a given file path.
13. **getCleanTableBody:** Retrieves the clean table body node from the HTML document.
14. **parseSectionsFromDocument:** Analyzes the sections from the HTML document and returns a list of Section objects.

The main method in the App class reads the HTML file, "Look Up Classes.htm", processes the sections, and outputs the number of parsed sections. In addition to the App class, the OasisParser.java program also plays an essential role in the data extraction process. This java program involves some steps:

1. Import the necessary Java libraries to handle tasks such as date and time manipulation, and HTML parsing.
2. Employ a custom tool named EmptyTextFilter to process the parts of the HTML file by filtering out unnecessary text and focusing on the relevant data.
3. Use specific functions like parseTimeFromMorningAfternoonRep and parseDateWithArbitraryYear to convert time and date strings into more manageable formats. These functions operate by splitting the input string into components, creating LocalTime or LocalDate objects, and making adjustments as needed based on the provided data.



```java
public static LocalTime parseTimeFromMorningAfternoonRep(String str) {
    if (notApplicableString(str)) {
        return null;
    } else {
        String[] timeParts = str.trim().split( regex: " ");
        LocalTime time = LocalTime.parse(timeParts[0]);
        return timeParts[1] == "PM" ? time.plusHours( hoursToAdd: 12L) : time;
    }
}
```

Image 5: Parsing Date Function

In summary, the App class and OasisParser.java program effectively process the HTML file and extract the necessary data to populate various classes representing different aspects of the class schedule. Once the data is extracted, it is used to populate the PostgreSQL database, making the information available for frontend components to access and display.

Backend:

Section Lookup Page Queries

These Queries were used to properly fill the section lookup page with properly generated drop-down menus that display all unique subject, department, attributes, instructors, and campuses stored within the SQL database.

SELECT DISTINCT c.subject FROM Course c ORDER BY c.subject

SELECT DISTINCT c.department FROM Course c ORDER BY c.department

SELECT DISTINCT c.attributes FROM Course c JOIN c.attributes

SELECT ins.name FROM Instructor ins ORDER BY ins.name

SELECT DISTINCT l.campus FROM Location l

SELECT DISTINCT s.instructMethod FROM Section s WHERE (NOT s.instructMethod = ' ') ORDER BY s.instructMethod

Other options such as time, day, and credit range are present as well but are determined by user input rather than queries from a database.

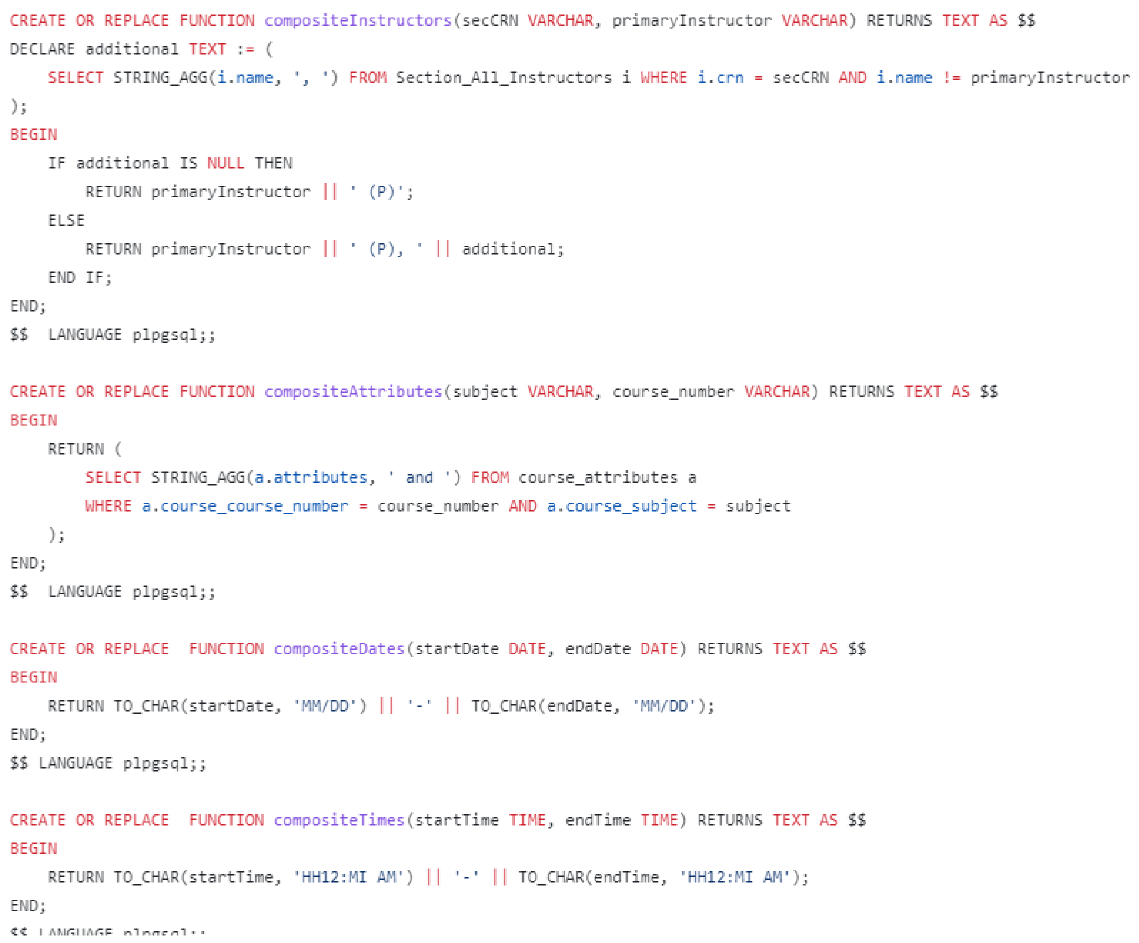
Section Lookup Process Queries

This section will show the queries used in the process of creating lo sections of the page that displays the user chosen sections. These include the select query that calls the lookup sections query, the look up section query that finds sections based on user input, and queries that composite the days, instructors, times, and attributes.

SELECT (lookupSections(?1,?2,?3,?4,?5,?6,?7,?8,?9,?10,?11,?12,?13)).*

```sql
CREATE OR REPLACE FUNCTION lookupSections(
    TEXT[],
    Course.course_number%TYPE,
    Course.title%TYPE,
    TEXT[],
    Course.credits%TYPE,
    Course.credits%TYPE,
    TEXT[],
    TEXT[],
    TEXT[],
    TEXT[],
    Meeting.start_time%TYPE,
    Meeting.end_time%TYPE,
    Meeting.days%TYPE
) RETURNS TABLE (crn VARCHAR(255),
                 subject VARCHAR(255),
                 courseNumber VARCHAR(255),
                 sectionNumber VARCHAR(255),
                 credits INTEGER,
                 title VARCHAR(255),
                 instructionalMethod VARCHAR(255),
                 permitRequired BOOL,
                 termDates TEXT,
                 days VARCHAR(255),
                 times TEXT,
                 instructor TEXT,
                 campus VARCHAR(255),
                 loc TEXT,
                 attribs TEXT,
                 department VARCHAR(255)) AS $$
```

```sql
BEGIN
    RETURN QUERY
        SELECT filtSec.crn,
               filtCourse.subject,
               filtCourse.course_number,
               filtSec.section_number,
               filtCourse.credits,
               filtCourse.title,
               filtSec.instructional_method,
               filtCourse.permit_required,
               compositeDates(filtSec.start_date, filtSec.end_date),
               m.days,
               compositeTimes(m.start_time, m.end_time),
               compositeInstructors(filtSec.crn, filtSec.name),
               l.campus,
               l.building || ' ' || l.room_number,
               compositeAttributes(filtCourse.subject, filtCourse.course_number),
               filtCourse.department
        FROM (
            SELECT *
            FROM Section s
            WHERE (array_length($9,1) IS NULL OR s.crn IN (SELECT i.crn FROM Section_All_Instructors i WHERE i.name = ANY($9)))
                AND ($11 IS NULL OR STRING_TO_ARRAY($13, NULL) && STRING_TO_ARRAY(coalesceDays(s.crn, $11, $12), NULL))
            ) AS filtSec
            JOIN
            (SELECT *
              FROM Course c
              WHERE (c.subject, c.course_number) IN
                    (SELECT a.course_subject, a.course_course_number
                     FROM Course_Attributes a
                     WHERE (array_length($10,1) IS NULL OR a.attributes = ANY($10)))) AS filtCourse
             ON filtSec.target_course_number = filtCourse.course_number AND filtSec.target_subject = filtCourse.subject
                NATURAL JOIN section_meetings
                NATURAL JOIN Meeting m
                NATURAL JOIN Location l
        WHERE (array_length($1,1) IS NULL OR filtCourse.subject = ANY($1))
          AND ($2 IS NULL OR filtCourse.course_number = $2)
          AND ($3 IS NULL OR filtCourse.title = $3)
          AND (array_length($4,1) IS NULL OR filtSec.instructional_method = ANY($4))
          AND ($5 IS NULL OR $6 IS NULL OR int4range($5, $6, '[]') @> filtCourse.credits)
          AND (array_length($7,1) IS NULL OR filtCourse.department = ANY($7))
          AND (array_length($8,1) IS NULL OR l.campus = ANY($8));
```

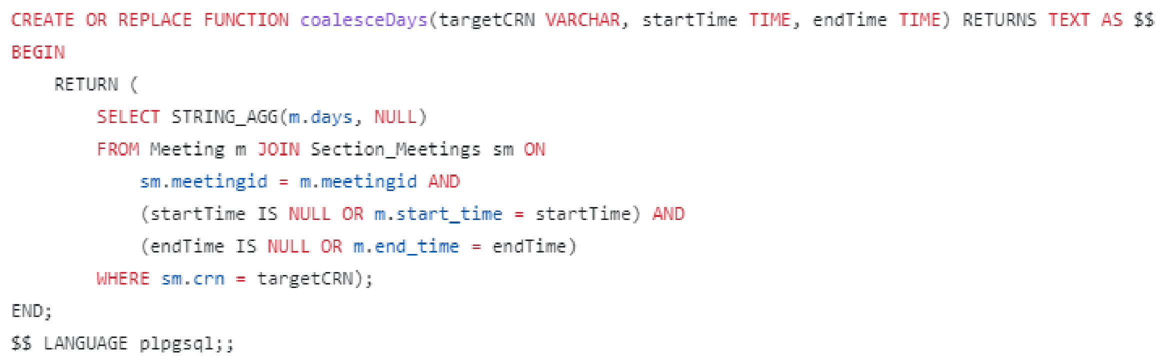


```
CREATE OR REPLACE FUNCTION compositeInstructors(secCRN VARCHAR, primaryInstructor VARCHAR) RETURNS TEXT AS $$
DECLARE additional TEXT := (
    SELECT STRING_AGG(i.name, ', ') FROM Section_All_Instructors i WHERE i.crn = secCRN AND i.name != primaryInstructor
);
BEGIN
    IF additional IS NULL THEN
        RETURN primaryInstructor || ' (P)';
    ELSE
        RETURN primaryInstructor || ' (P), ' || additional;
    END IF;
END;
$$  LANGUAGE plpgsql;;

CREATE OR REPLACE FUNCTION compositeAttributes(subject VARCHAR, course_number VARCHAR) RETURNS TEXT AS $$
BEGIN
    RETURN (
        SELECT STRING_AGG(a.attributes, ' and ') FROM course_attributes a
        WHERE a.course_course_number = course_number AND a.course_subject = subject
    );
END;
$$  LANGUAGE plpgsql;;

CREATE OR REPLACE  FUNCTION compositeDates(startDate DATE, endDate DATE) RETURNS TEXT AS $$
BEGIN
    RETURN TO_CHAR(startDate, 'MM/DD') || '-' || TO_CHAR(endDate, 'MM/DD');
END;
$$ LANGUAGE plpgsql;;

CREATE OR REPLACE  FUNCTION compositeTimes(startTime TIME, endTime TIME) RETURNS TEXT AS $$
BEGIN
    RETURN TO_CHAR(startTime, 'HH12:MI AM') || '-' || TO_CHAR(endTime, 'HH12:MI AM');
END;
$$ LANGUAGE plpgsql;;
```

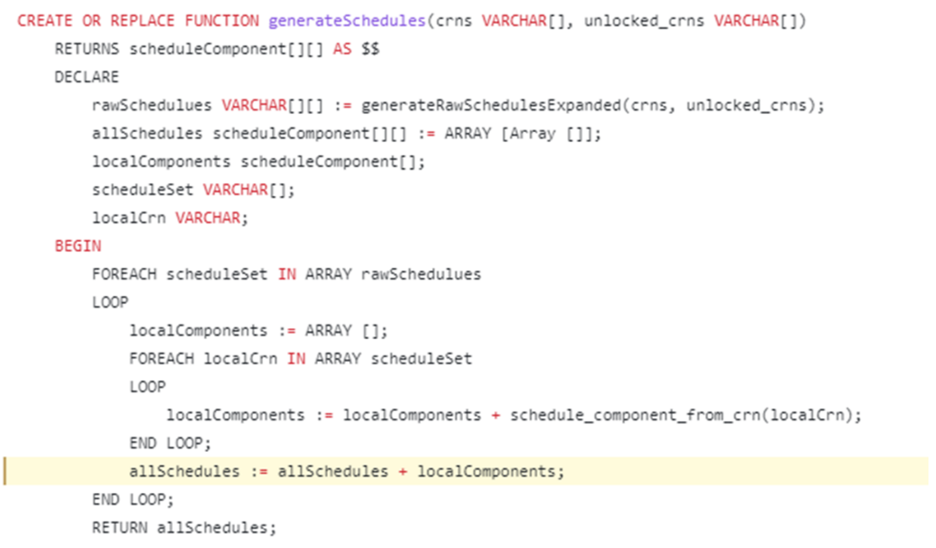


```
CREATE OR REPLACE FUNCTION coalesceDays(targetCRN VARCHAR, startTime TIME, endTime TIME) RETURNS TEXT AS $$
BEGIN
    RETURN (
        SELECT STRING_AGG(m.days, NULL)
        FROM Meeting m JOIN Section_Meetings sm ON
            sm.meetingid = m.meetingid AND
            (startTime IS NULL OR m.start_time = startTime) AND
            (endTime IS NULL OR m.end_time = endTime)
        WHERE sm.crn = targetCRN);
END;
$$ LANGUAGE plpgsql;;
```

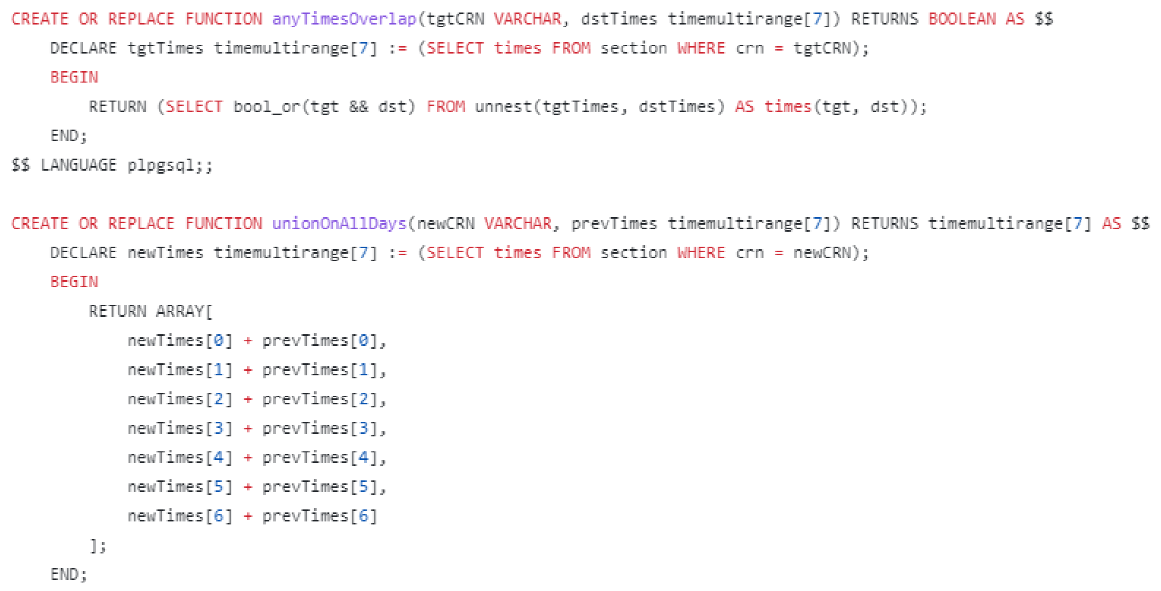
Schedule Generation queries

There are more than listed queries than shown in this report for generating the schedules for this site but these images will display the notable ones. These include the Select Query to display the overall schedule generated from the generate query, the generate query which generates the schedule, and the queries surrounding generating both the timeslots of the courses as well as the days the course is available.

SELECT (generateSchedules(?1,?2)).*

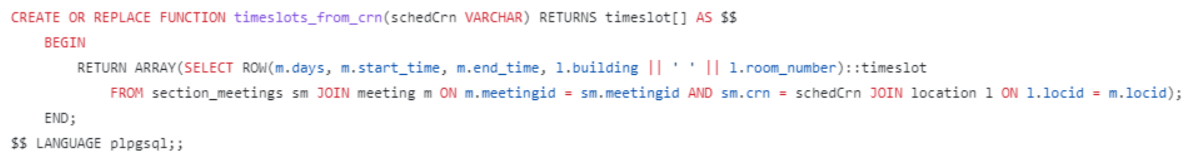


```
CREATE OR REPLACE FUNCTION generateSchedules(crns VARCHAR[], unlocked_crns VARCHAR[])
    RETURNS scheduleComponent[][] AS $$
    DECLARE
        rawSchedulues VARCHAR[][] := generateRawSchedulesExpanded(crns, unlocked_crns);
        allSchedules scheduleComponent[][] := ARRAY [Array []];
        localComponents scheduleComponent[];
        scheduleSet VARCHAR[];
        localCrn VARCHAR;
    BEGIN
        FOREACH scheduleSet IN ARRAY rawSchedulues
        LOOP
            localComponents := ARRAY [];
            FOREACH localCrn IN ARRAY scheduleSet
            LOOP
                localComponents := localComponents + schedule_component_from_crn(localCrn);
            END LOOP;
            allSchedules := allSchedules + localComponents;
        END LOOP;
        RETURN allSchedules;
```



```
CREATE OR REPLACE FUNCTION anyTimesOverlap(tgtCRN VARCHAR, dstTimes timemultirange[7]) RETURNS BOOLEAN AS $$
    DECLARE tgtTimes timemultirange[7] := (SELECT times FROM section WHERE crn = tgtCRN);
    BEGIN
        RETURN (SELECT bool_or(tgt && dst) FROM unnest(tgtTimes, dstTimes) AS times(tgt, dst));
    END;
$$ LANGUAGE plpgsql;;

CREATE OR REPLACE FUNCTION unionOnAllDays(newCRN VARCHAR, prevTimes timemultirange[7]) RETURNS timemultirange[7] AS $$
    DECLARE newTimes timemultirange[7] := (SELECT times FROM section WHERE crn = newCRN);
    BEGIN
        RETURN ARRAY[
            newTimes[0] + prevTimes[0],
            newTimes[1] + prevTimes[1],
            newTimes[2] + prevTimes[2],
            newTimes[3] + prevTimes[3],
            newTimes[4] + prevTimes[4],
            newTimes[5] + prevTimes[5],
            newTimes[6] + prevTimes[6]
        ];
    END;
```



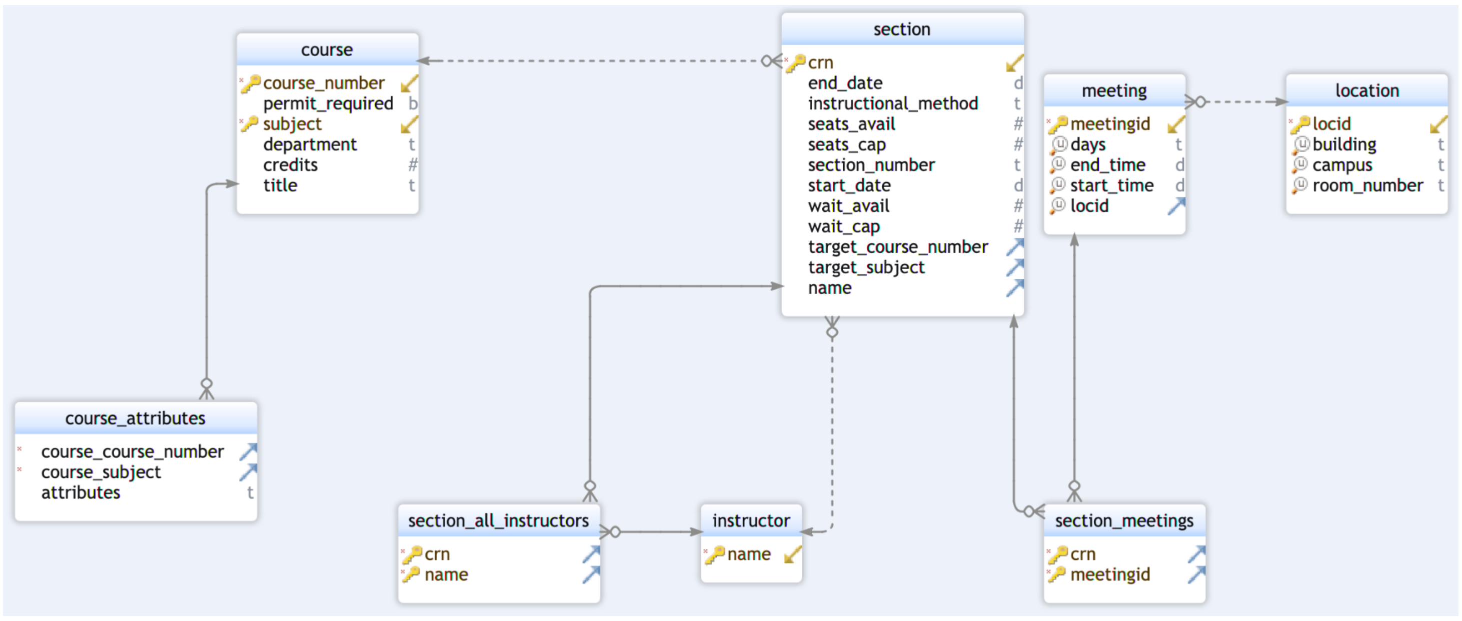
```
CREATE OR REPLACE FUNCTION timeslots_from_crn(schedCrn VARCHAR) RETURNS timeslot[] AS $$
    BEGIN
        RETURN ARRAY(SELECT ROW(m.days, m.start_time, m.end_time, l.building || ' ' || l.room_number)::timeslot
            FROM section_meetings sm JOIN meeting m ON m.meetingid = sm.meetingid AND sm.crn = schedCrn JOIN location l ON l.locid = m.locid);
    END;
$$ LANGUAGE plpgsql;;
```

Data Base:

From the first design to the last version, the design underwent substantial changes. The database's general structure and functionality were improved as a result of this upgrade. The way the courses were modeled was one of the major adjustments. The course id, course number, department, credits, title, and other features were all represented as a single object in the original design of courses. The courses were divided into two entities in the new design: the course entity, which contains the course's essential details, and the course_attributes entity, which contains the course's supplemental details. Due to the course entity only holding the essential information and storing any additional information, this move helps to safeguard the integrity of the data. Another change was the way instructors were modeled. In the original design, instructors were associated with sections through the additional_instructors entity. In the refined design, instructors were modeled as a separate entity, with the relationship between instructors and sections being established through the section and section_all_instructors entities. The location entity was also revised in the refined design. In the original design, the location entity was associated with the meeting entity through the meetingid foreign key. In the updated design, the relationship between the location and meeting entities was established through the locid foreign key. The section entity was also altered in the new design, with the relationships between sections, courses, instructors, and meetings being established through foreign keys.

In conclusion, by separating entities, creating relationships through foreign keys, and minimizing data redundancy, the improvements made in the update design enhanced the database's overall structure and effectiveness. These changes helped to ensure that the database is well-designed and optimized for data retrieval and manipulation.

EER Diagram:



Relational Schema:

1. course(course_number, subject, credits, department, permit_required, title)
   - Primary Key: (course_number, subject)

2. course_attributes(course_course_number, course_subject, attributes)
   - Primary Key: (course_course_number, course_subject)
   - Foreign Key: (course_course_number, course_subject) REFERENCES course(course_number, subject)

3. instructor(name)
   - Primary Key: name

4. location(locid, building, campus, room_number)
   - Primary Key: locid

- Unique Key: (building, campus, room_number)

5. meeting(meetingid, days, end_time, start_time, locid)
   - Primary Key: meetingid
   - Unique Key: (days, end_time, locid, start_time)
   - Foreign Key: locid REFERENCES location(locid)

6. section(crn, end_date, instructional_method, seats_avail, seats_cap, section_number, start_date, wait_avail, wait_cap, target_course_number, target_subject, name)
   - Primary Key: crn
   - Foreign Key: (target_course_number, target_subject) REFERENCES course(course_number, subject)
   - Foreign Key: name REFERENCES instructor(name)

7. section_all_instructors(crn, name)
   - Primary Key: (crn, name)
   - Foreign Key: name REFERENCES instructor(name)
   - Foreign Key: crn REFERENCES section(crn)

8. section_meetings(crn, meetingid)
   - Primary Key: (crn, meetingid)
   - Foreign Key: meetingid REFERENCES meeting(meetingid)
   - Foreign Key: crn REFERENCES section(crn)

Conclusion:

In conclusion, the class scheduler database system project demonstrates the successful development of an efficient and comprehensive solution for class scheduling, tailored to the needs of students. Utilizing advanced technologies such as React for the frontend, Spring Boot for the backend, and PostgreSQL for the database allowed the team to create a seamless experience while maintaining a robust data infrastructure.The various aspects of the project, including frontend design, data collection and parsing, and database design, were carefully carried out to ensure the accuracy and consistency of the information presented. By gathering data from the USF OASIS Look up classes section and populating the database, the system provides up-to-date and reliable information for students.