THE UNIVERSITY OF SYDNEY

Assignment 3

ISYS2120: Data and Information Management

Group R10_82

Contents

Listings

1 Executive summary

Given the base web application and database provided in Week 8 containing university data, this report documents the processes used by our team to implement functionalities for various tables in the schema. Each team member implemented 4 basic functionalities which followed the same structure: function 1 is a simple `SELECT-FROM` query, function 2 is a `SELECT-FROM-WHERE` query, page 3 is a `GROUP BY` query, and function 4 is an `INSERT INTO` query. Our team also implemented 4 schema extensions to the database by creating new tables relevant to the original domain. These extensions range from adding exam data to the database to adding university library data to the schema. The original and final database schema diagrams can be found in Appendix A.

2 Individual contributions

This section contains the code that each member has written in `database.py` and implementation information about any extensions an individual has made.

Quick links

2.1 Antriksh Dhand: 510415022

I chose to work on implementing information about university classrooms into the web app. Below are the related functions I have implemented in the `database.py` file.

2.1.1 List all classrooms

```python
def get_classroom_list():
    # connect to database and set up cursor
    conn = database_connect()
    if (conn is None):
        return None
    cur = conn.cursor()

    # attempt to query database
    val = None
    try:
        sql = """
            SELECT  classroomId AS "Classroom ID",
                    seats AS "Number of seats",
                    type AS "Type of room"
              FROM unidb.classroom
              """
        cur.execute(sql)
        val = cur.fetchall()
    except:
        # if error occurs, print error message
        # val will remain null
        print("Error fetching from database")

    cur.close()     # Close the cursor
    conn.close()    # Close the connection to the db

    return val
```

Listing 1: Function definition to query list of classrooms from database

Table 1: Sample output from running Listing 1

Classroom ID	**Number of seats**	**Type of room**
CAR373	160	tiered
CAR375	160	tiered
...		
EALT	200	sloping
EA403	40	flat

2.1.2 Search for classrooms based on seating capacity

```python
def get_classroom_search(seats: int) -> list:
    # connect to database and set up cursor
    conn = database_connect()
    if (conn is None):
        return None
    cur = conn.cursor()

    # attempt to query database
    val = None
    try:
        sql = """
            SELECT  classroomId AS "Classroom ID",
                    seats AS "Number of seats",
                    type AS "Type of room"
              FROM unidb.classroom
             WHERE seats > %s
              """
        cur.execute(sql, (seats,))
        val = cur.fetchall()
    except:
        # if error occurs while querying
        print("Error fetching from database")

    cur.close()     # Close the cursor
    conn.close()    # Close the connection to the db

    return val
```

Listing 2: Function definition to search for classrooms with more than a certain number of seats

Table 2: Sample output from running Listing 2 with input 250

Classroom ID	Number of seats	Type of room
BoschLT1	270	tiered
BoschLT2	267	tiered
...		
CheLT3	300	tiered
EAA	500	sloping

2.1.3 Report on number of classrooms

```python
def get_classroom_report() -> list:
    # connect to database and set up cursor
    conn = database_connect()
    if (conn is None):
        return None
    cur = conn.cursor()

    # attempt to query database
    val = None
    try:
        sql = """
                SELECT  type AS "Type of classroom",
                        COUNT(classroomId) AS "Number of classrooms"
                  FROM unidb.classroom
              GROUP BY type
              """
        cur.execute(sql)
        val = cur.fetchall()
    except:
        # if error occurs while querying
        print("Error fetching from database")

    cur.close()     # Close the cursor
    conn.close()    # Close the connection to the db

    return val
```

Listing 3: Function definition to produce a grouped-aggregate report on the number of classrooms by type

Table 3: Sample output from running Listing 3

Type of classroom	Number of classrooms
flat	4
tiered	22
sloping	3

2.1.4 Add a new classroom to database

```python
def add_new_classroom(classroom_id: int, seats: int, class_type: str, lat: float,
    long: float) -> int:
    # connect to database and set up cursor
    conn = database_connect()
    if (conn is None):
        return None
    cur = conn.cursor()

    # attempt to append to database
    try:
        sql = """
                INSERT INTO unidb.classroom (classroomid, seats, type, lat, long)
                VALUES (%s, %s, %s, %s, %s);
              """
        cur.execute(sql, (classroom_id, seats, class_type, lat, long))
    except Exception as e:
        # if error occurs while querying return -1
        print(e)
        print("Error adding to database")
        return -1

    cur.close()     # Close the cursor
    conn.commit()   # Commit changes to the database
    conn.close()    # Close the connection to the db

    return 0 # return 0 on success
```

Listing 4: Function definition to add a new classroom entry into the database

2.1.5 Extension 1: Adding exam data

My first extension involves extending the schema to include data about examinations for unit of study offerings. Students can then access their own personalised exam timetable through the web app in a similar fashion to how students access this information in the real-world.

The inspiration for this extension was actually the recent release of Semester 2 examination timetables on `https://exams.sydney.edu.au/`. I wanted to accurately extend the database to include the same information that we use in the real-world regarding our examinations, including the date, time, writing time, reading time, venue, session type (whether it is a main exam or a replacement exam) and exam type (open-book, Live+ etc.). The University's website does not include information regarding the time-zone of the exam, however I endeavoured to include this to make the web app easily accessible to international students as well.

The relevant code chunks for the implementation of this extension into the web app are briefly described below.

1. **Defining the relevant tables**
   Three new tables were constructed to include examination data in the schema: `Exam`, `ExamSession` and `ExamType`. Please see Figure 12 for the database's relational schema diagram with these tables added, and please refer to Section 3.1 for the relevant data definition language (DDL) statements written to create these tables.

2. **Inserting sample data into the new tables**
   Please see Appendix for the `INSERT INTO` statements used to populate these tables with dummy data.

3. **Querying relevant data**
   Listing 5 is the SQL query function implemented in `database.py` to query the database for a specific student's examination data. See Table 4 for a sample table output and Figure 1 to see the results on the webpage.

```sql
    SELECT uosCode, uosName, uosOffering.semester, year, sessionType,
    examTime, examTypeName, readingTime, writingTime, venue
      FROM unidb.Transcript
INNER JOIN unidb.UosOffering USING (uosCode, semester, year)
INNER JOIN unidb.Exam USING (uosCode, semester, year)
INNER JOIN unidb.UnitOfStudy USING (uosCode)
INNER JOIN unidb.ExamSession USING (sessionId, year)
INNER JOIN unidb.ExamType USING (examTypeId)
     WHERE studId = %s
       AND sessionType = 'MAIN';
```

Listing 5: SQL query to retrieve student's exam data

4. **Pipelining the data**
   Listing 6 is the function written in `routes.py` to pipeline the data from the backend

Table 4: Sample output from running Listing 5

UOS Code	**UOS Name**	**Semester**	**Year**	**Session Type**	**Exam Time**	**Exam Type**	**Reading Time**	**Writing Time**	**Venue**
INFO3005	Organisational Database Systems	S1	2005	MAIN	2005-05-01 11:00:00+10:00	In-person	0:10:00	2:00:00	CAR159
COMP5338	Advanced Data Models	S1	2006	MAIN	2006-06-13 12:00:00+10:00	Short-release	0:10:00	2:30:00	Online
ISYS2120	Database Systems I	S1	2010	MAIN	2010-07-01 17:00:00+10:00	Open-book	0:05:00	2:40:00	Online

(database) to the frontend (web app). Note that lines 237 and 238 are unrelevant to this extension.

```
231 @app.route('/exam_timetable', methods=['POST', 'GET'])
232 def exam_timetable():
233     page['title'] = 'Personal exam timetable'
234 
235     sid = session['sid']
236     exam_timetable = database.get_exam_timetable(sid)
237     classrooms = database.get_classroom_locations()
238     map = produce_map(exam_timetable, classrooms)
239 
240     # if result is null show error message
241     if exam_timetable is None or exam_timetable == ():
242         exam_timetable = {}
243         flash('Error, this student has no exams timetabled!')
244 
245     return render_template(
246         'exam_timetable.html',
247         page=page,
248         session=session,
249         exam_timetable=exam_timetable,
250         classrooms=classrooms,
251         map=map
252     )
```

Listing 6: Flask backend to call database function and push results to frontend

5. **Displaying on the web app**

Listing 7 is the extract from `exam_timetable.html` relevant to presenting the timetable data in a table format. Note in line 29 we use the `datetime` module's `astimezone` function to present the exam time in the user's local time zone. This is extremely handy for international or remote students. Furthermore, lines 33-37 implement the idea that if the venue is `null` then the exam is online.

```
6  <table class="pure-table">
7      <thead>...</thead>
21     <tbody>
22         {% for row in exam_timetable %}
23             <tr>
24                 <td> {{ row[0] }}</td>
25                 <td> {{ row[1] }} </td>
26                 <td> {{ row[2] }} </td>
27                 <td> {{ row[3] }} </td>
28                 <td> {{ row[4] }} </td>
29                 <td> {{ row[5].astimezone() }} </td>
30                 <td> {{ row[6] }} </td>
31                 <td> {{ row[7] }} </td>
32                 <td> {{ row[8] }} </td>
```

```
                {% if row[9] is none %}
                <td>Online</td>
                {% else %}
                <td> {{ row[9] }} </td>
                {% endif %}
            </tr>
        {% endfor %}
    </tbody>
</table>
```

Listing 7: Table definition in `exam_timetable.html`

2.1.6 Extension 2: Adding classroom locations

My second extension involves adding location fields to the pre-existing `Classroom` table. There are various uses for this, but as I was originally extending the schema with exam data I chose to implement a “Where are my exams?” feature where students can see the location of their in-person examinations on an interactive map.

Once again, the code chunks from my submission which are relevant to this extension are described below.

1. **Adding new columns**

   Two new columns were added to the `Classroom` table: `lat` and `long`. Please refer to Section 3.1 for the relevant DDL statements written to create these columns.

2. **Inserting sample data into the new tables**

   Please see Appendix B.1 for the `UPDATE-SET-WHERE` statements used to populate the clsas-rooms with location data.

3. **Querying relevant data**

   A simple query was written to extract classroom location data from the database (see Listing 8). This was embedded in the function `get_classroom_locations()`.

```sql
SELECT classroomId, lat, long
  FROM unidb.Classroom;
```

Listing 8: SQL query to retrieve classroom location data

4. **Creating the map visualisation**

   The map was created using Python’s `folium` library which was installed onto the server manually. Listing 9 contains the code used to generate the map placed in `helpers.py`; it essentially uses a for-loop to iterate through each entry in the student’s exam timetable before checking whether the exam is online or not. If the exam is not online (and therefore has a location), it generates a `folium.Marker()` object for it and places it onto the map. The function then returns the HTML representation of the map to allow it to be placed on the webpage using Flask (see Figure 1).

```python
import folium

def produce_map(exam_timetable: dict, classrooms: dict) -> str:
    m = folium.Map(
        location=[-33.88841430895747, 151.18715475049206], # center of USYD
        zoom_start=16,
        tiles="OpenStreetMap"
    )

    for exam in exam_timetable:
        venue = exam[9]

        # do nothing if the exam is online
        if venue is None:
            break

        # otherwise get the lat/long of the exam venue
        lat = None
        long = None
        for classroom in classrooms:
            if classroom[0] == venue:
                lat = classroom[1]
                long = classroom[2]
        if (lat is None) or (long is None):
            return -1

        folium.Marker(
            location=[lat, long],
            popup=f"Unit: {exam[0]}<br>Location: {exam[9]}"
        ).add_to(m)

    return m._repr_html_()
```

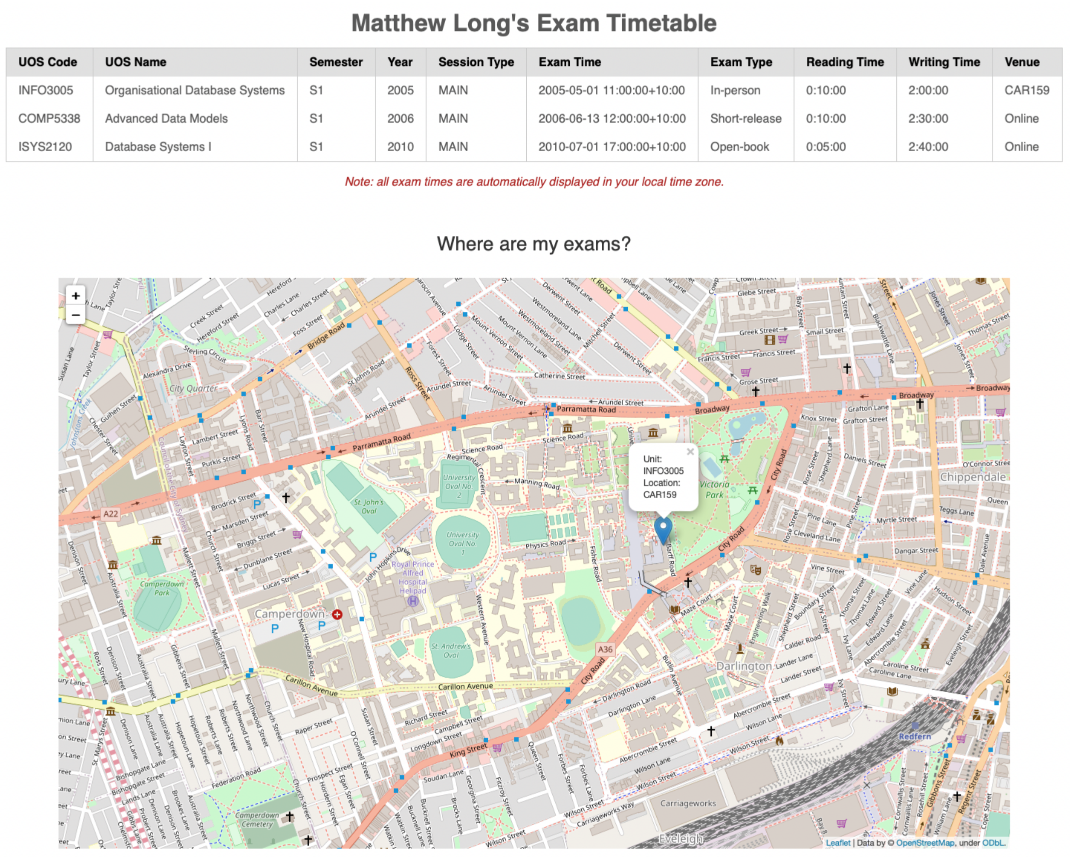
Listing 9: Folium code to generate the map of a student's in-person exams, defined in `helpers.py`.

5. **Pipelining the data**
   Please refer to lines 237 and 238 in Listing 6 for the function calls relevant to the production of the map.

6. **Frontend**
   Implementing the code into the frontend was simple due to `folium` exporting the map as HTML. All we need is to print the map's HTML inside `exam_timetable.html` which can be done using the `safe` keyword (see Listing 10).

```html
<div id="container" style="max-width: 50%; margin: auto;">
    <h2 style="text-align:center;">Where are my exams?</h2>
    <br>
    {{ map | safe }}
</div>
```

Listing 10: Using `jinja2`'s `safe` flag to place the map in the frontend



Matthew Long's Exam Timetable

UOS Code	UOS Name	Semester	Year	Session Type	Exam Time	Exam Type	Reading Time	Writing Time	Venue
INFO3005	Organisational Database Systems	S1	2005	MAIN	2005-05-01 11:00:00+10:00	In-person	0:10:00	2:00:00	CAR159
COMP5338	Advanced Data Models	S1	2006	MAIN	2006-06-13 12:00:00+10:00	Short-release	0:10:00	2:30:00	Online
ISYS2120	Database Systems I	S1	2010	MAIN	2010-07-01 17:00:00+10:00	Open-book	0:05:00	2:40:00	Online

Note: all exam times are automatically displayed in your local time zone.

Where are my exams?

Figure 1: Output of extensions 1 and 2 in the Flask web app

2.2 Udit Samant: 500700976

For this Assignment, I have implemented the information on university lectures and their locations into the web app.

2.2.1 List all the locations of lectures

```python
def list_locations():
    # Get the database connection and set up the cursor
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        # Execute Query
        cur.execute("""
        SELECT uosCode, uosName, semester, year, classtime, classroomId
        FROM UniDB.Lecture JOIN UniDB.Classroom USING (classroomId)
        JOIN UniDB.UnitOfStudy USING (uosCode)
        ORDER BY uosCode, year, semester, uosName
        """)
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                         # Close the cursor
    conn.close()                        # Close the connection to the db
    return val
```

Listing 11: Function definition to query lecture locations from the database

Table 5: Sample output from Listing 11

UOS Code	**UOS Name**	**Semester**	**Year**	**Class Time**	**Classroom ID**
COMP5046	Statistical Natural Language Processing	S1	2010	Tue14	SITLT
COMP5138	Database Management Systems	S2	2006	Mon18	SITLT
...					
ISYS2120	Database Systems I	S1	2010	Tue13	BoschLT2
ISYS2120	Database Systems I	S2	2010	Mon09	QuadLT

2.2.2 Search for units that have a class at a particular time

```python
def search_units(classtime):
    # Get the database connection and set up the cursor

    sql = """
    SELECT uosCode, uosName, semester, year, classtime, classroomId
    FROM UniDB.Lecture JOIN UniDB.Classroom USING (classroomId)
    JOIN UniDB.UnitOfStudy USING (uosCode)
    WHERE classtime = %s
    ORDER BY uosCode, year, semester, uosName
    """

    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        # NOTE: column ordering is IMPORTANT
        cur.execute(sql, (classtime, ))
        val = cur.fetchall()

    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 12: Function definition to query a particular day time for classes from the database

Table 6: Sample output from Listing 12 when searching for 'Mon12' tutorials

UOS Code	UOS Name	Semester	Year	Class Time	Classroom ID
INFO1003	Introduction to IT	S1	2006	Mon12	CheLT4
INFO1003	Introduction to IT	S2	2006	Mon12	SITLT

2.2.3 Report for how many classes occur in each room

```
def location_counts():
        # Get the database connection and set up the cursor
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        # Execute Query
        cur.execute("""
        SELECT classroomId, COUNT(*) AS "count"
        FROM UniDB.Lecture JOIN UniDB.Classroom USING (classroomId)
        JOIN UniDB.UnitOfStudy USING (uosCode)

        GROUP BY classroomId
        ORDER BY COUNT(*) DESC
        """)

        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 13: Function definition to summarise number of classes per locations from the database

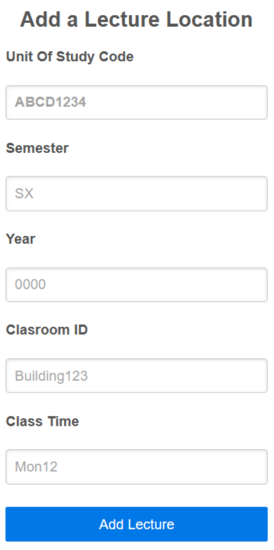
Table 7: Sample output from running Listing 13

Classroom ID	**Count of Classes**
SITLT	3
CAR159	2
...	
BoschLT2	1
QuadLT	1

2.2.4 Add a new lecture location

```python
def add_units(uosCode, semester, year, class_id, class_time):
        # Get the database connection and set up the cursor
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = False
    try:
        cur.execute("""
        INSERT INTO UniDB.lecture(uoscode, semester, year, classtime, classroomid)
        VALUES (%s , %s , %s, %s , %s)
        """, (uosCode, semester, year, class_time, class_id))

        val = True
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    conn.commit()
    cur.close()                         # Close the cursor
    conn.close()                        # Close the connection to the db
    return val
```

Listing 14: Function definition to add a new lecture location to the database


Add a Lecture Location

Unit Of Study Code	ABCD1234
Semester	SX
Year	0000
Clasroom ID	Building123
Class Time	Mon12

Figure 2: GUI to add a lecture location using the function in Listing 14

2.3 Sulav Malla: 500495980

I have chosen to work on implementing the information about academic staff, querying the relation, and displaying this query formatted neatly on the web app. These are the functions I have implemented in the database.py file to get the appropriate result to display on each page.

2.3.1 List all academic staff

```
def list_academic_staff():
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        # NOTE: column ordering is IMPORTANT
        cur.execute("""
        SELECT id, name, deptid, address
        FROM UniDB.AcademicStaff
        ORDER BY id, name
        """)
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                         # Close the cursor
    conn.close()                        # Close the connection to the db
    return val
```

Listing 15: Function definition to query list of academic staff from database

Table 8: Sample output from running Listing 15

Id	Name	Department	Address
0987654	Simon Poon	SIT	Sydney
1122334	1122334	SIT	Glebe
...			
6339103	Uwe Roehm	SIT	Cremorne
7891234	Sanjay Chawla	SIT	Neutral Bay

2.3.2 Search for all staff in particular department

```python
def staff_search(deptid):
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        # NOTE: column ordering is IMPORTANT
        cur.execute("""
            SELECT id, name, address
            FROM UniDB.AcademicStaff
            where deptid = %s
            ORDER BY id, name
            """, (deptid,))
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 16: Function definition to query list of academic staff given the department id

Table 9: Sample output of Listing 16, searching for all staff in the SIT department

Id	Name	Address
0987654	Simon Poon	Sydney
1122334	1122334	Glebe
...		
6339103	Uwe Roehm	Cremorne
7891234	Sanjay Chawla	Neutral Bay

2.3.3 Report on the number of staff in each department

```python
def grp_staff():
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        # NOTE: column ordering is IMPORTANT
        cur.execute("""SELECT deptid, count(id) as "Number of Staff"
                        FROM UniDB.AcademicStaff
                        GROUP BY deptid
                        ORDER BY count(id)""")
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                         # Close the cursor
    conn.close()                        # Close the connection to the db
    return val
```

Listing 17: Function definition to query the number of staff members in each department

Table 10: Sample output of Listing 17

Department	**Number of Staff**
SIT	7

2.3.4 Add a new academic staff member

```
def add_staff(Id, name, deptid, password, address, salary):
    conn = database_connect()

    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        cur.execute("""INSERT INTO UniDB.AcademicStaff(id, name, deptId, password,
 address, salary) VALUES (%s, %s, %s, %s, %s, %s) """, (Id, name, deptid,
password, address, salary))
    except:
        conn.rollback()
        # If there were any errors, we print something nice and return a NULL
value
        print("Error fetching from database")
        traceback.print_exc()

    cur.close()
    conn.commit()                          # Close the cursor
    conn.close()                          # Close the connection to the db
    return val
```

Listing 18: Function definition of adding a new staff member into the database

2.3.5 Extension: Adding new relation ThesisStudent to the database

For my extension I chose to add a new relation to the database named ThesisStudent which stores `studId` (references the `studId` from the Student table) and `academicId` which references the AcademicStaff relation. The idea here is to see, for the academic staff who mentor students, who they are mentoring. Hence the primary key is the combination of both the student id and academic id.

After, creating tables I inserted some dummy values (see Appendix B.4). Then in `routes.py` I queried the table and made a HTML page to display all the students and their mentors, and also a table showing how many students academic staff mentor (see Listings 19 and 20).

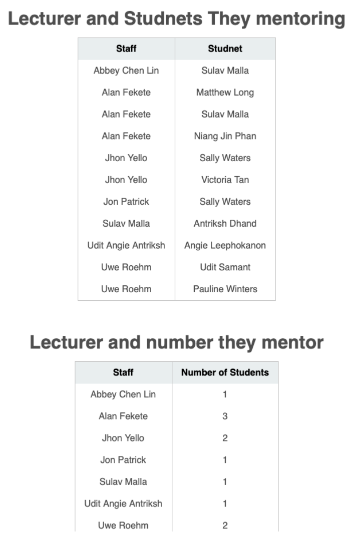
```
################################################################################
# Returns the lecturers who have thesis students and the students they mentor
################################################################################
def thesis_student():
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        cur.execute("""SELECT D.name, S.name
                        FROM (UniDB.ThesisStudent Ts join UniDB.Student S on (S.
    studid = Ts.studentid)) join UniDB.AcademicStaff D on (D.id = Ts.academicid)
                        where D.name is not NULL
                        ORDER BY D.name""")
        val = cur.fetchall()
    except:
        print("Error fetching from database")

    cur.close()                    # Close the cursor
    conn.close()                   # Close the connection to the db
    return val

################################################################################
# Returns the thesis supervisors and the number of students they mentor
################################################################################

def num_staff_student():
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        # SQL statement to returns query.
        cur.execute("""SELECT D.name, count(S.name)
                        FROM (UniDB.ThesisStudent TS join UniDB.Student S on (S.
    studid = Ts.studentid)) join UniDB.AcademicStaff D on (D.id = Ts.academicid)
                        GROUP BY D.name
                        HAVING count(S.name) > 0
                        ORDER BY D.name """)
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 19: Extension functions in `database.py` to query data in the `ThesisStudent` table

```
################################################################################
# Adding thesis students and the corresponding staff to the website
################################################################################
@app.route("/Thesis-Student")
def grp_thesisstudnet():
    entries = database.thesis_student()
    counting = database.num_staff_student()

    if( entries is None or entries == ()):
        entries = []
        flash("Error, there are no Valid Group")

    page['title'] = "Lecturer and Students They mentoring"
    return render_template('thesis_student.html', page = page, session = session,
    Staff = entries, countStudents = counting)
```

Listing 20: Flask function in `routes.py` to pass data to the HTML page

The HTML page which is rendered by Listing 20 is shown in Figure 3.



Lecturer and Studnets They mentoring

Staff	Studnet
Abbey Chen Lin	Sulav Malla
Alan Fekete	Matthew Long
Alan Fekete	Sulav Malla
Alan Fekete	Niang Jin Phan
Jhon Yello	Sally Waters
Jhon Yello	Victoria Tan
Jon Patrick	Sally Waters
Sulav Malla	Antriksh Dhand
Udit Angie Antriksh	Angie Leephokanon
Uwe Roehm	Udit Samant
Uwe Roehm	Pauline Winters

Lecturer and number they mentor

Staff	Number of Students
Abbey Chen Lin	1
Alan Fekete	3
Jhon Yello	2
Jon Patrick	1
Sulav Malla	1
Udit Angie Antriksh	1
Uwe Roehm	2

Figure 3: HTML page to view information about thesis students

2.4 Cody Hu: 500513701

2.4.1 List all the prerequisite pairs from Requires and UnitOfStudy Table

```python
def get_all_prerequisites():
    # Get the all the prerequisites
    conn = database_connect()
    if(conn is None):
        return None

    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""
        SELECT R.uosCode, Ust.uosName, Und.uosName, R.enforcedSince
        FROM UniDB.Requires R JOIN UniDB.UnitOfStudy Ust USING (uosCode)
        JOIN UniDB.UnitOfStudy Und ON (R.prereqUosCode = Und.uosCode);
                    """)
        val = cur.fetchall()
    except Exception as e:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")
        print(e)

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 21: Function to query the list of prerequisite pairs from the database with their enforcement date

Table 11: Sample output from running Listing 21

UOS Code	Unit	Prerequisite Code	Prerequisite Name	enforceDate
ISYS2120	Database Systems I	INFO1003	Introduction to IT	2002-01-01
DATA3404	Database Systems II	ISYS2120	Database Systems I	2004-11-01
...				
INFO2005	Database Management Introductory	INFO1003	Introduction to IT	2002-01-01
INFO3005	Organisational Database Systems	INFO2005	Database Management Introductory	2002-01-01

2.4.2 Search for all the units which are prerequisites of a given unit

```python
def search_prerequisites(unit : str):
    # Get the all the prerequisites
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        # NOTE: column ordering is IMPORTANT
        sql = """
            SELECT uosCode, prereqUosCode, enforcedSince
            FROM UniDB.Requires
            WHERE uosCode = %s;
            """

        cur.execute(sql, (unit,))
        val = cur.fetchall()
    except Exception as e:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")
        print(e)

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 22: Function to get the list of prerequisite courses based on a given unit code

Table 12: Sample output from running Listing 22 with input `COMP5338`

UOS Code	Pre Code	enforceDate
COMP5338	COMP5138	2004-01-01
COMP5338	ISYS2120	2004-01-01

2.4.3 Report on prerequisites for each unit of study

```python
def get_report():
    # Get the database connection and set up the cursor
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        # NOTE: column ordering is IMPORTANT
        cur.execute("""
                    SELECT uosCode, count(uosCode) as "Number of Requirements"
                    FROM UniDB.Requires
                    GROUP BY uosCode
                    ORDER BY count(uosCode) DESC, uosCode;
                    """)
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 23: Function to generate a report showing number of prerequisites for each unit of study

Table 13: Sample output from running Listing 23

UOS Code	**Number of Prerequisite Course**
COMP5338	2
INFO3005	1
COMP5046	1
DATA3404	1
INFO2005	1
ISYS2120	1

2.4.4 Add a new prerequisite pair to database

```python
def insert_prerequisites(course:str, prerequisites:str, enforce_date:str):
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()

    # add new data to database
    try:
        sql = """
                INSERT INTO unidb.Requires (uoscode, prerequoscode, enforcedsince)
                VALUES (%s, %s, %s);
                """

        cur.execute(sql, (course, prerequisites, enforce_date))
    except Exception as e:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error when insert into databse")
        print(e)
        return 1

    cur.close()                         # Close the cursor
    conn.commit()                       # commit the changes
    conn.close()                        # Close the connection to the db
    return 0
```

Listing 24: Function that allows the user to add a new prerequisite unit pair to the database

When the user successfully adds to the database, the HTML page will prompt a message to inform the user that his/her action was successful. If the data wasn't added to the data set, the HTML will also inform the user with an message: "Error, failed to add a new prerequisite unit pair to the data set."

2.4.5 Extension: Adding allocated unit of study tables

My extension is adding two new tables to the schema, one to include data about the allocated unit of study table for each subject and the other one to contain information about different table types. There are five different types of study table initially, Table A: available majors/minors, Table S: shared pool, Table D: Dalyell stream units, Table O: Open Learning Environment and Table R which is for higher degree by research. Users can search for a unit's allocated unit of study table, and also ask the database to generate a report to see how many units are allocated to each table types. Users can reallocate a unit to a new table or remove from a existing table. Additionally, users can add a new type of study table into our database by providing the new table code (a single character) and a name.

The inspiration for this extension is to help student when selecting subject during enrolment. Units of Study Handbook provide information on how to select subjects for all different majors. However, the handbook provide the course name and the course code but they do not show which table this course belongs to. This may cause some inconvenience since student are selecting subject from Selection Tables in the enrolment website. Hence I want to include this information into our database. Below are the four functions provided for this extension.

Functionality 1: Unit Search

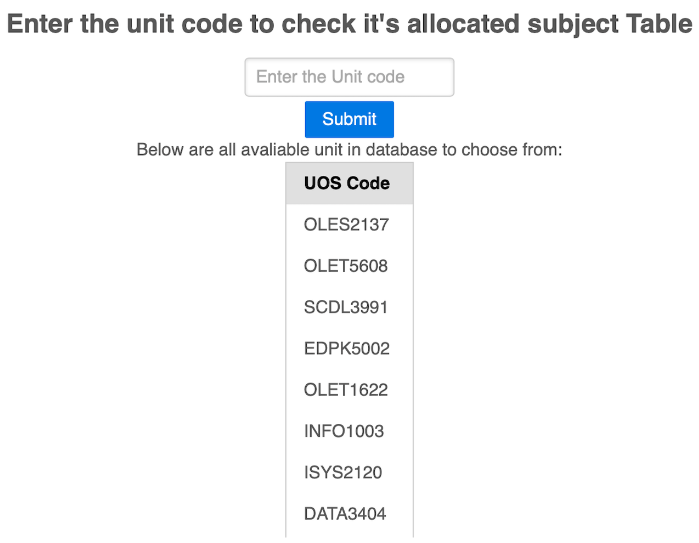
```python
def search_table(unit : str):
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        # NOTE: column ordering is IMPORTANT
        sql = """
                SELECT S.uosCode, S.tableBelong, U.tableName
                FROM UniDB.subjectTables S JOIN UniDB.unitTables U ON (S.
    tableBelong = U.tableCode)
                WHERE uosCode = %s
                ORDER BY tableBelong;
              """
        cur.execute(sql, (unit,))
        val = cur.fetchall()
    except Exception as e:
        print("Error fetching from database")
        print(e)

    cur.close()                         # Close the cursor
    conn.close()                        # Close the connection to the db
    return val
```

Listing 25: Extract the list of study table based on a given unit code

Table 14: Sample output from running Listing 25 with input MATH1002

UOS Code	Table Code	Table Name
MATH1002	A	Degree Core Units of Study
MATH1002	S	Shared Pool

Additionally, all available subjects are provided in a table when user tries to use the function above in the web page. A screen shot of the web page is provided below, please refer to Figure 4.



Enter the unit code to check it's allocated subject Table

Below are all avaliable unit in database to choose from:

UOS Code
OLES2137
OLET5608
SCDL3991
EDPK5002
OLET1622
INFO1003
ISYS2120
DATA3404

Figure 4: GUI to search for a unit's allocated tables

Functionality 2: Report on study table

```python
def extension_report():
    # Get the database connection and set up the cursor
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        # NOTE: column ordering is IMPORTANT
        cur.execute("""
                    SELECT tableBelong, count(uosCode)
                    FROM uniDB.subjectTables
                    Group By tableBelong
                    ORDER BY count(uosCode) DESC, tableBelong;
                    """)
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 26: Generate a report showing number of units allocated to each table type

Table 15: Sample output from running Listing 26. Note that the result is sorted by the count of allocated units.

Table Type	Number of Courses Allocated
S	9
A	5
O	3
D	1
R	1

Functionality 3: Reallocate Units to Existing Table

```python
def allocate_table(course:str, tableBelong:str):
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()

    try:
        sql = """
                INSERT INTO unidb.subjectTables (uosCode, tableBelong)
                VALUES (%s, %s);
            """
        cur.execute(sql, (course, tableBelong))
    except Exception as e:
        print("Error when insert into databse")
        print(e)
        return 1

    cur.close()                     # Close the cursor
    conn.commit()                   # commit the changes
    conn.close()                    # Close the connection to the db
    return 0

def remove_from_table(course:str, tableBelong:str):
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()

    try:
        sql = """
        DELETE FROM unidb.subjectTables
        WHERE uosCode = %s AND tableBelong = %s;
            """
        cur.execute(sql, (course, tableBelong))
    except Exception as e:
        print("Error when deleting row from databse")
        print(e)
        return 1

    cur.close()                     # Close the cursor
    conn.commit()                   # commit the changes
    conn.close()                    # Close the connection to the db
    return 0
```

Listing 27: Allocating a given unit to a new study table or removing a given unit from a study table

Figure 5 shows the GUI for reallocating units. Users can simply type a course code and choose the table type that they want to allocate to or remove from. The drop-down list shown in Figure 6 is dynamic; when a new type of study table has been added to the database, the list will include that new table type. Listing 28 (the HTML code from `allocate.html`) also shows that the table option is formed dynamically.

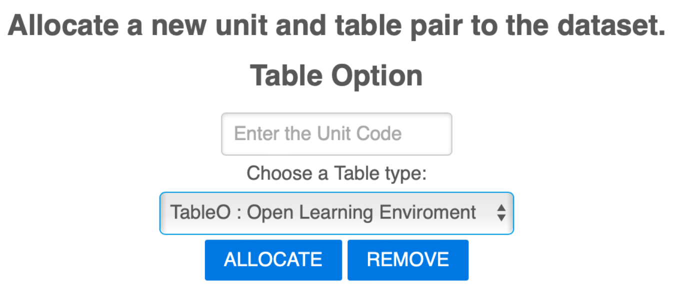

Allocate a new unit and table pair to the dataset.
Table Option
Enter the Unit Code
Choose a Table type:
TableO : Open Learning Enviroment
ALLOCATE
REMOVE

Figure 5: GUI for Reallocating Units

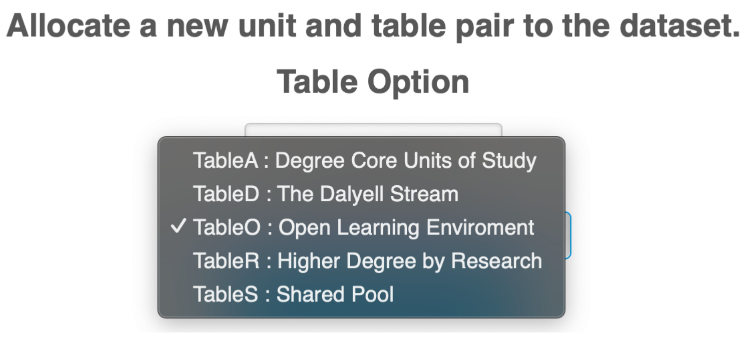
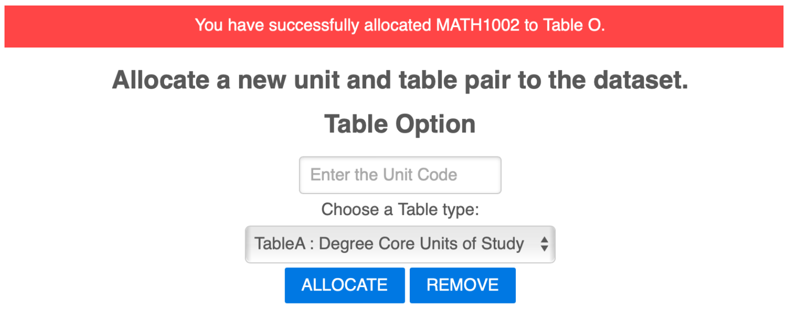

Allocate a new unit and table pair to the dataset.
Table Option
TableA : Degree Core Units of Study
TableD : The Dalyell Stream
✓ TableO : Open Learning Enviroment
TableR : Higher Degree by Research
TableS : Shared Pool

Figure 6: Dynamic Dropdown List

```
<div>
    <label for="table-type">Choose a Table type:</label>
    <select name="table-type" id="table-type">
        {% for key in dynamic %}
            <option value = {{key[0]}} >Table{{key[0]}} : {{key[1]}} </option>
        {% endfor %}
    </select>
</div>
```

Listing 28: Dynamic dropdown list code

When the user successfully allocated a unit to a study table or removed a unit from a table that it has been allocated to, an informative message will shown up in the webpage (see Figure 7). Additionally, if the user's action was not achieved, a failed message will pop up in the same manner.


You have successfully allocated MATH1002 to Table O.

Allocate a new unit and table pair to the dataset.

Table Option

Figure 7: Success message on successful allocation of unit to study table

To achieve condition checking for removing units from table, an additional function (see Listing 29) will run after the `remove_from_table` method has been called. This function is written to try to `SELECT` the data which is suppose to be deleted and return 1 if the data is found.

```python
def check_check(course:str, tableBelong:str):
    # if we find the target then we return 1
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()

    try:
        sql = """
            SELECT *
            FROM unidb.subjectTables
            WHERE uosCode = %s AND tableBelong = %s;
              """
        cur.execute(sql, (course, tableBelong))
        val = cur.fetchone()
    except Exception as e:
        print("Error when deleting row from databse")
        return 1
    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    if val is None:
        return 0    # we did not find the data, so deletion is successful
    return 1
```

Listing 29: Function that helps to check if the deletion is successfull

Functionality 4: Insert New Unit of Study Table

```python
def insert_unit_table(tableCode : str, tableName : str):
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()

    # add new data to database
    try:
        sql = """
                INSERT INTO unidb.unitTables (tableCode, tableName)
                VALUES (%s, %s);
              """

        cur.execute(sql, (tableCode, tableName))
    except Exception as e:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error when insert into database")
        print(e)
        return 1

    cur.close()                     # Close the cursor
    conn.commit()                   # commit the changes
    conn.close()                    # Close the connection to the db
    return 0
```

Listing 30: Function that allows the user to add a new study table to the database

User can add a new type of study table at anytime, and the dynamic table list in Function 3 will get modified as well. Informative messages will be shown in webpage according to the user's action has been achieved successfully or not.

2.5 Yan Rong: 500514742

2.5.1 List all units with their textbook

```python
def get_textbook():
    # This function is querying a table that contains all the units with their textbook
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""
            SELECT uosCode, semester, year, uosName, textbook
            FROM UniDB.UoSOffering JOIN UniDB.UnitOfStudy USING (uosCode)
            ORDER BY uosCode, year, semester
                    """)
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL value
        print("Error fetching from database")

    cur.close()                        # Close the cursor
    conn.close()                       # Close the connection to the db
    return val
```

Listing 31: Function to query a table that contains all the units (by UoSCode, semester, year, and unit name) with their textbook

Table 16: Sample output from running Listing 31

UOS Code	**Semester**	**Year**	**Unit name**	**Textbook**
COMP5046	S1	2010	Statistical Natural Language Processing	None
COMP5138	S2	2006	Database Management Systems	Ramakrishnan/Gehrke
COMP5138	S1	2010	Database Management Systems	Ramakrishnan/Gehrke
...				
INFO3005	S1	2005	Organisational Database Systems	Hoffer
ISYS2120	S1	2006	Database Systems I	Kifer/Bernstein/Lewis
ISYS2120	S1	2009	Database Systems I	Kifer/Bernstein/Lewis
ISYS2120	S1	2010	Database Systems I	Kifer/Bernstein/Lewis

2.5.2 Show how many UnitOfferings use each textbook

```python
def get_textbook_report():
    # This function is querying a table that summarise a textbook is used by how many UoS
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""
            SELECT textbook, COUNT(uosCode)
            FROM UniDB.UoSOffering
            WHERE textbook IS NOT NULL
            GROUP BY textbook
            ORDER BY textbook
                    """)
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 32: Function to query a table that summarise a textbook is used by how many UoS

Table 17: Sample output from running Listing 32

Textbook	**Number of UoS**
Hoffer	2
Kifer/Bernstein/Lewis	3
. . .	
Ramakrishnan/Gehrke	3
Snyder	2

2.5.3 Search for all UnitOfferings which use a particular textbook

```python
def get_textbook_search(textbook):
    # This function is querying a table that contains all the units (by UoSCode, semester, year, and unit name) with a particular textbook
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""
            SELECT uosCode, semester, year, uosName, textbook
            FROM UniDB.UoSOffering JOIN UniDB.UnitOfStudy USING (uosCode)
            WHERE textbook=%s
            ORDER BY uosCode, year, semester
                    """, (textbook,))
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 33: Function to query a table that contains all the units with a particular textbook

Table 18: Sample output from running Listing 33 with input `Kifer/Bernstein/Lewis`

UOS Code	**Semester**	**Year**	**Unit name**	**Textbook**
ISYS2120	S1	2006	Database Systems I	Kifer/Bernstein/Lewis
ISYS2120	S1	2009	Database Systems I	Kifer/Bernstein/Lewis
ISYS2120	S1	2010	Database Systems I	Kifer/Bernstein/Lewis

2.5.4 Change the textbook for a given UnitOffering

```
def get_textbook_update(uoscode, year, semester, textbook):
    # This function changes the textbook used for a particular unit
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    val = None
    try:
        cur.execute("""
            UPDATE UniDB.UoSOffering
            SET textbook=%s
            WHERE uosCode=%s AND semester=%s AND year =%s;
                    """, (textbook, uoscode, semester,year,))
        conn.commit()
        val = None
    except Exception as e:
        print("Query Failed with error {}".format(e))
        conn.rollback()

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val

def get_textbook_update_check(uoscode, year, semester, textbook):
    # This function verifies change result
    conn = database_connect()
    if(conn is None):
        return None
    cur = conn.cursor()
    val = None
    try:
        cur.execute("""
            SELECT 1
            FROM UniDB.UoSOffering
            WHERE textbook=%s AND uosCode=%s AND semester=%s AND year =%s
                    """, (textbook, uoscode, semester,year,))
        val = cur.fetchall()
    except:
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 34: These two functions change the textbook used for a particular unit and then verifies the result by query

The function `get_textbook_update` is used to update the textbook used for a particular unit of study. Since the unit of study is identified by its UoScode, semester and year (primary key), these values are all required when changing the textbook for a particular unit of study. The function `get_textbook_update_check` is used to check whether user have successfully updated information or not. If the unit of study does not exist in the table function or update failed `get_textbook_update_check` will return empty tuple. Then the web app can notify the user that an error has occurred.

2.5.5 Extension: Adding university libraries

My extension is adding two tables into unidb schema (library table and book table) to manage books in libraries across entire university. The book table contains the numeric and descriptive information about books in university libraries. The information includes title, author, publisher, number of available copies, number of total copies and library ID (where to find this book). The library table contains descriptive information about uni libraries. Those are library ID, name and location. By joining book and library table, user can easily retrieve all information they need for a particular book.

I was inspired by the task I did. University provides numerous numbers of unit of study and most of them uses text book, and students might get textbook from university libraries; therefore, there is demand on manage books in libraries (thus provide more conveniences to our users).

Functionality 1: Browsing books in each libraries or entire university

```python
def get_distinct_library():
    # This function returns a list of distinct libraries in uni
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""SELECT DISTINCT(libraryName)
                       FROM UniDB.Libraries
                       ORDER BY libraryName;""")
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                        # Close the cursor
    conn.close()                       # Close the connection to the db
    return val
```

Listing 35: The query that returns a list of distinct library name

The code in Listing 35 is used to get a distinct list of libraries in the uni, and this list will be used to generate a dynamic drop-down of choices that will help our user to view all books in a specific library (see Listing 36).

Table 19: Sample output from running Listing 35

libraryName
Central library
Conservatorium Library
Engineering faculty library
Law building library
Medical faculty library

```
{% include 'extension_top.html' %}
<div id="content">
    <h1 class="page-title">All books with their library</h1>

    <form class="pure-form pure-form-stacked login" method="POST" action="{{
    url_for('library') }}">

        <h2>Browsing books by their library</h2>
        <label for="library">Choose library:</label>
        <select name="libraries" id="library">
            {% for lib in libs %}
                <option value={{lib[0]}}>{{lib[0]}}</option>
            {% endfor %}
        </select>
        <input class="pure-button pure-button-primary" type="submit" value="Submit
    ">

    </form>
    <br><br>

    <h3 style="text-align: center;">{{ page.subtitle }}</h3>

    <table class="pure-table">
        <thead>
            <tr><th>Title</th><th>Number of available Copies</th><th>Library</th><
    th>Location</th></tr>
        </thead>
        <tbody>
            {% for unit in units %}
                <tr>
                    <td> {{unit[0]}}</td>
                    <td> {{unit[1]}} </td>
                    <td> {{unit[2]}} </td>
                    <td> {{unit[3]}} </td>
                </tr>
            {% endfor %}
        </tbody>
    </table>
```

```
36 </div>
37 </body>
38 </html>
```

Listing 36: Dynamic dropdown list code

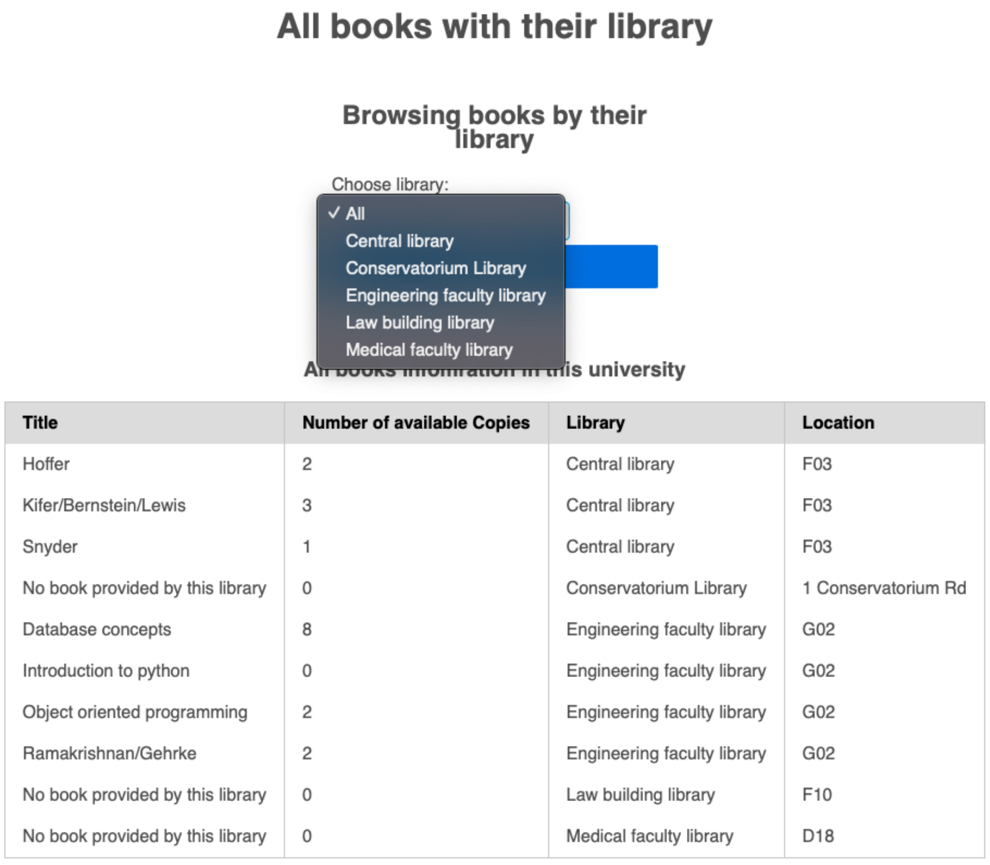
Lines 9 to 13 in Listing 36 acts as a loop to display all libraries in drop-down of choices.

The two function below are used to query table for books with their detailed information. Function `get_library` in Listing 37 return all books' information in uni. Function `get_library_with_selection` in Listing 37 return all books' information in a specific library.

```
def get_library():
    # This function returns all books information in uni
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        cur.execute("""SELECT COALESCE(title, 'No book provided by this library'), COALESCE(availableCopies, 0), libraryName, address
                        FROM UniDB.Books RIGHT JOIN UniDB.Libraries USING (libraryId)
                        ORDER BY libraryName, title;""")
        val = cur.fetchall()
    except:
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val

def get_library_with_selection(lib):
    # This function returns all books in a specifc library
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        cur.execute("""SELECT COALESCE(title, 'No book provided by this library'), COALESCE(availableCopies, 0), libraryName, address
                        FROM UniDB.Books RIGHT JOIN UniDB.Libraries USING (libraryId)
                        WHERE libraryName = %s
                        ORDER BY title;""",(lib,))
        val = cur.fetchall()
    except:
        print("Error fetching from database")

    cur.close()                         # Close the cursor
    conn.close()                        # Close the connection to the db
    return val
```

Listing 37: Those two functions are used to return a table of all books with their libraries or filter books by specific library

The website appearance is seen in Figure 8.



All books with their library

Browsing books by their library

Choose library:

All books information in this university

Title	Number of available Copies	Library	Location
Hoffer	2	Central library	F03
Kifer/Bernstein/Lewis	3	Central library	F03
Snyder	1	Central library	F03
No book provided by this library	0	Conservatorium Library	1 Conservatorium Rd
Database concepts	8	Engineering faculty library	G02
Introduction to python	0	Engineering faculty library	G02
Object oriented programming	2	Engineering faculty library	G02
Ramakrishnan/Gehrke	2	Engineering faculty library	G02
No book provided by this library	0	Law building library	F10
No book provided by this library	0	Medical faculty library	D18

Figure 8: Library page offer drop-down of choices for users to browsing books by library

Functionality 2: Summary report about how many available books each library has

```python
def get_library_report():
    # This function summarise the number of available copies each library has
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""SELECT libraryName, address, COALESCE(SUM(availableCopies),
 0)
                       FROM UniDB.Books RIGHT JOIN UniDB.Libraries USING (
libraryId)
                       GROUP BY libraryName, address
                       ORDER BY libraryName, address;""")
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
value
        print("Error fetching from database")

    cur.close()                    # Close the cursor
    conn.close()                   # Close the connection to the db
    return val
```

Listing 38: This function summarise the number of available copies each library has

Table 20: Sample output from running Listing 38

Library	**Location**	**Number of available books**
Central library	F03	6
Conservatorium Library	1 Conservatorium Rd	0
Engineering faculty library	G02	12
Law building library	F10	0
Medical faculty library	D18	0

Functionality 3: Search for a book

This function allows you to search the information (number of available copies, location) of a book by title in entire university or a specific library. This functionality also requires a list of distinct library as functionality 1. The list of distinct libraries can be retrieved by code 35.

```
 1 {% include 'extension_top.html' %}
 2 <div id="content">
 3     <h1 class="page-title">{{ page.title }}</h1>
 4     <form class="pure-form pure-form-stacked login" method="POST" action="{{
       url_for('library_search') }}">
 5         <h2>Search for books and their location</h2>
 6         <label for="library">Choose library:</label>
 7         <select name="libraries" id="library">
 8             {% for lib in libs %}
 9                 <option value={{lib[0]}}>{{lib[0]}}</option>
10             {% endfor %}
11         </select>
12         <br>
13         <label for="bookTitle">Please enter book title:</label>
14         <input type="text" name="bookTitle" placeholder="Book title" required>
15         <input class="pure-button pure-button-primary" type="submit">
16     </form>
17     <br>
18     <br>
19     <h3 style="text-align: center;">{{ page.subtitle }}</h3>
20     {% if submitted %}
21     <table class="pure-table">
22         <thead>
23             <tr><th>Book title</th><th>Number of available Copies</th><th>Library<
       /th><th>Location</th></tr>
24         </thead>
25         <tbody>
26             {% for unit in units %}
27                 <tr>
28                     <td> {{unit[0]}}</td>
29                     <td> {{unit[1]}} </td>
30                     <td> {{unit[2]}} </td>
31                     <td> {{unit[3]}} </td>
32                 </tr>
33             {% endfor %}
34         </tbody>
35     </table>
36     {% endif %}
37 </div>
38 <br>
39 </body>
40 </html>
```

Listing 39: Dynamic dropdown list code

Line 7 to 11 in Listing 39 acts as a loop to display all libraries in drop-down of choices.

```
def get_library_search(book):
    # This function search book in all library
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""SELECT title, availableCopies, libraryName, address
                       FROM UniDB.Books RIGHT JOIN UniDB.Libraries USING (libraryId)
                       WHERE title = %s;
                    """, (book,))
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val

def get_library_search_with_selection(library, book):
    # This function search book in a particular library
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""SELECT title, availableCopies, libraryName, address
                       FROM UniDB.Books RIGHT JOIN UniDB.Libraries USING (libraryId)
                       WHERE title = %s AND libraryName = %s;
                    """, (book, library, ))
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 40: Those two functions are used to return a table of books, which is filtered by book title and library

The website appearance is shown in Figure 9.

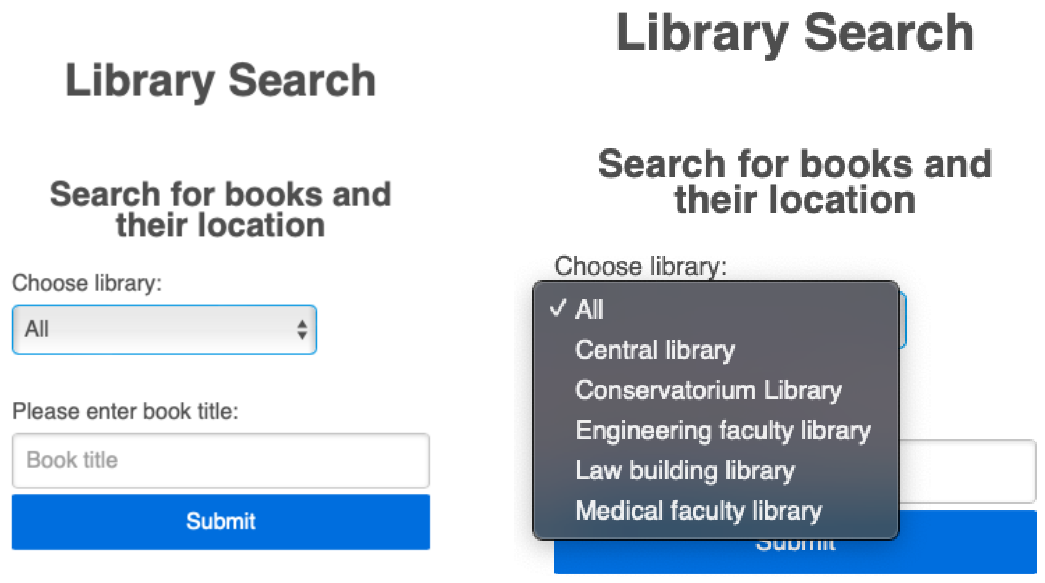


Figure 9: Search page offer drop-down of choices for users to search book in a particular library

Functionality 4: Return a book to library

```
def get_library_update(book):
    # This function perform an action of return book
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""UPDATE UniDB.Books
                       SET availableCopies = availableCopies + 1
                       WHERE title=%s;
                    """, (book,))
        conn.commit()

        val = None
    except Exception as e:
        print("Query Failed with error {}".format(e))
        conn.rollback()

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val

def get_library_search_existing_book(book):
    # This function checks wether the book is available for return
    conn = database_connect()
    if(conn is None):
        return None
    # Sets up cursor
    cur = conn.cursor()
    val = None
    try:
        # Try getting all the information returned from the query
        cur.execute("""SELECT 1
                        FROM UniDB.Books RIGHT JOIN UniDB.Libraries USING (
    libraryId)
                        WHERE title = %s;
                    """, (book, ))
        val = cur.fetchall()
    except:
        # If there were any errors, we print something nice and return a NULL
    value
        print("Error fetching from database")

    cur.close()                     # Close the cursor
    conn.close()                    # Close the connection to the db
    return val
```

Listing 41: Those two functions are used to return a book (increment quantity of a book by one), and check whether this operation is valid or not

The website appearance is shown in Figure 10.

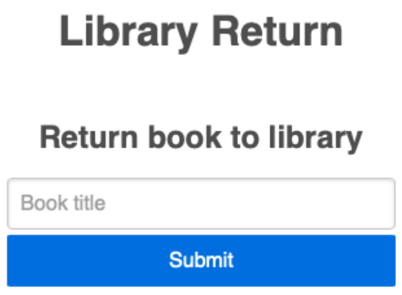


Figure 10: Page to return a book

3 Extensions

The team has implemented a total of 5 extensions. These are:

1. Include examination data in schema – Antriksh
2. Add location attributes to classroom data – Antriksh
3. Add unit of study tables in schema – Qixuan (Cody)
4. Add university libraries and books in schema – Yan Rong
5. Add thesis student data into schema – Sulav

Below we have the DDL statements each student used to extend the database's schema.

Quick links

3.1 Antriksh's extensions

See listings 42 and 43 below for the DDL statements used to extend the database schema for extensions 1 and 2.

```sql
SET SCHEMA 'unidb';
DROP TABLE IF EXISTS Exam;
DROP TABLE IF EXISTS ExamType;
DROP TABLE IF EXISTS ExamSession;

CREATE TABLE ExamType (
    examTypeId          CHAR(1),          -- 'A', 'B', 'C', 'D', or 'E'
    examTypeName        VARCHAR(40),      -- 'Live+', 'Record+', 'Open-book', 'Short-release', or 'Extended-release'
    PRIMARY KEY (examTypeId)
);

CREATE TABLE ExamSession (
    sessionId       SERIAL,         -- auto-create the sessionId key
    semester        CHAR(2),        -- 'S1' or 'S2' (following the original schema design)
    year            INTEGER,
    sessionType     CHAR(4),        -- 'MAIN', 'REP1', or 'REP2' (assuming 2 replacement exam periods per semester)
    PRIMARY KEY (sessionId),
    UNIQUE (sessionId, year)
);

CREATE TABLE Exam (
    uosCode         CHAR(8),
    semester        CHAR(2),
    year            INTEGER,
    sessionId       INTEGER,
    examTime        TIMESTAMPTZ,        -- include time zone
    examTypeId      CHAR(1),
    readingTime     INTERVAL,           -- e.g. '90 minutes', '1 hour and 20 minutes'
    writingTime     INTERVAL NOT NULL,
    venue           VARCHAR(8),         -- null indicates online exam
    PRIMARY KEY (uoSCode, semester, year, examTime),
    FOREIGN KEY (uosCode, semester, year) REFERENCES UoSOffering(uosCode, semester, year),
    FOREIGN KEY (examTypeId) REFERENCES ExamType(examTypeId),
    FOREIGN KEY (venue) REFERENCES Classroom(classroomId),
    FOREIGN KEY (sessionId, year) REFERENCES ExamSession(sessionId, year)
);
```

Listing 42: Antriksh's `CREATE TABLE` statements for extension 1

```sql
ALTER TABLE unidb.Classroom
ADD lat REAL,
ADD long REAL;
```

Listing 43: Antriksh's ALTER TABLE statement for extension 2

You can find the INSERT and UPDATE-SET-WHERE statements for these tables and columns in Appendix B.1.

3.2 Cody's extensions

See listing 44 below for the DDL statements used to extend the database schema for extension 3.

```sql
set schema 'unidb';
DROP TABLE IF EXISTS SubjectTables;
DROP TABLE IF EXISTS UnitTables;

/* create the schema */
CREATE TABLE UnitTables (
  tableCode    CHAR(1) NOT NULL,
  tableName    VARCHAR(50) NOT NULL,
  PRIMARY KEY (tableCode)
);

CREATE TABLE SubjectTables (
  uoSCode        CHAR(8),
  tableBelong    VARCHAR(20) NOT NULL,
  PRIMARY KEY (uoSCode, tableBelong),
  FOREIGN KEY (uoSCode) REFERENCES UnitOfStudy(uoSCode),
  FOREIGN KEY (tableBelong) REFERENCES UnitTables(tableCode)
);
```

Listing 44: Cody's `CREATE TABLE` statements for extension 3

The `INSERT` statements for `UnitTables` and `SubjectTables` tables can be found in Appendix B.2.

3.3 Yan's extensions

See listing 45 below for the DDL statements used to extend the database schema for extension 4.

```sql
set schema 'unidb';
DROP TABLE IF EXISTS Books;
DROP TABLE IF EXISTS Libraries;

CREATE TABLE Libraries (
  libraryId INTEGER PRIMARY KEY,
  libraryName VARCHAR(50) NOT NULL,
  address VARCHAR(200) NOT NULL
);

CREATE TABLE Books (
  title VARCHAR(50) PRIMARY KEY,
  author VARCHAR(50) NOT NULL,
  publisher VARCHAR(50) NOT NULL,
  availableCopies INTEGER NOT NULL,
  totalCopies INTEGER NOT NULL,
  libraryId INTEGER NOT NULL,
  FOREIGN KEY (libraryId) REFERENCES Libraries(libraryId)
);
```

Listing 45: Yan's `CREATE TABLE` statements for extension 4

The `INSERT` statements for extension 4 can be found in Appendix B.3.

3.4 Sulav's extensions

See listing 46 below for the DDL statement used to extend the database schema for extension 5.

```sql
set schema 'unidb';
DROP TABLE IF EXITS unidb.ThesisStudent;

CREATE TABLE ThesisStudent (
    studId INTEGER references Student(studId) primary key,
    academicId CHAR(9) references academicstaff(id)
)
```

Listing 46: Sulav's `CREATE TABLE` statement for extension 5

4 Testing

A thorough testing process was undertaken by all group members to ensure each of our queries and web apps were functioning correctly. Each member gave their codebase and SQL queries to another member who then tested these two parts individually. The testing pairs were as follows:

- Antriksh's code tested by Cody
- Cody's code tested by Antriksh
- Udit's code tested by Sulav
- Sulav's code tested by Yan
- Yan's code tested by Udit

The general testing procedure to test the team member's SQL queries was to copy the SQL query from the user's codebase and run it directly in the pgAdmin query tool. This required us to have access to each other's databases, for testing purposes only. The output from pgAdmin was then compared with the sample output in the report.

The website development testing was conducted by running the other team member's codebase locally. Various inputs were then tested in the GUI including expected input, edge cases, and nonsensical inputs.

The complete process, including testing methodology and results for each testing pair, is shown in the YouTube video at `https://youtu.be/tc5CoZR2p6w/`.

4.1 Testing Cody's codebase (by Antriksh)

As shown in the video, all SQL queries written by Cody had generated desired output and the website development is great. All functionalities were able to generate valid result and display it in clean format. However, there were some improvements can be made for the website development:

1. Instead of giving same type of error messages for all mistakes, it is recommend to have a more informative error message. For example, if the user gives an invalid input, then the error message will indicate user that their input is invalid. If the data that user tries to add is already in the database, then the error message could be telling that. In this case, it would help the user to understand why their action was not achieved.
2. Capital and lower-case letter handling could be considered when inserting data into Table or removing data from Table. For example, the result for inputting 'MATH1002' should be the same as 'Math1002' or 'math1002'.
3. User input checking is not enough for adding a new type of unit of study table into the database. During testing, I tried to set Table code to be more than one character, the data will not be inserted. However, when table code was set to be 0, the data did get inserted and number is not a valid table code.
4. When user tries to select a enforcement date, it would be good to restrict user to select a date that is later than the current date.

4.2 Testing Antriksh's codebase (by Cody)

The video shows that Antriksh's queries all run successfully and produce the expected output as seen in the report. Some general pointers of feedback gathered through the testing process are provided below:

- The queries do not take excessively long to run.
- The web application is generally well built and intuitive, including the implementation of drop-down boxes on the menu bar to separate different parts of the assignment (pre-built functions, basic functions, and extensions).
- The GUI helps limit the amount of random inputs a user could give. For example, `ClassroomId` is automatically limited to a maximum of 8 characters through the HTML `<input>` tag. This reduces some backend checking which would have had to be done otherwise.
- The use of mapping in Antriksh's extension was nicely implemented. The map automatically updates depending on how many rows of output are present in the query.
- Error messages are generally informative and guide the user as to what they are inputting incorrectly. For example, the error message on the 'Classroom search' page says "There is an error in the input. Please input a positive integer." This automatically covers multiple cases such as the user inputting a string or a negative number.
- One improvement I would suggest to Antriksh is to make the input of latitude and longitude more robust.
  - The limits of latitude and longitude are not implemented in the program. Latitude has a range of $[-90, 90]$ and longitude has a range of $[-180, 180]$. However, a user can input any floating point number in these boxes and no error message will appear. Ideally, the limits of the latitude and longitude should be the boundaries of the University of Sydney campus areas (essentially Greater Sydney, accounting for the Camden and Lidcombe campuses).
  - A beneficial feature for Antriksh to include in the future development of this project is a point and click latitude and longitude finder. Currently the process is reliant on the user visiting Google Maps or a similar mapping service to extract the coordinates of the classroom. It would be useful if a popup map was provided on the website where a user could 'pin' the location they require and the coordinates automatically get filled in.

4.3 Testing Udit's codebase (by Sulav)

The video linked shows all SQL Queries and Website functionalities that Udit wrote. All the functionalities worked as expected. Listed below are some comments that have been documented through testing. These include some key areas where the functionalities and coding implementations could have been improved on, and also some areas where the program functionality was appreciated

- The application GUI was quite intuitive to use. Tab titles were very clear and the page descriptions complemented them quite well. This meant that navigating the pages was quite good.
- The list locations page could sort the output in a more meaningful manner. Currently, the data is ordered in a way that is not useful. It is difficult trying to find a unit in a particular year, because the units are not sorted about available year.

- Otherwise counts and list location functionalities have been implemented properly. Not much else to comment on them
- The search page could be more specific with the error messages. Extending this issue further, the back end program should have better error handling. Entering an incorrect class time like *"adfadsaf"* should give some form error but does not.
- Likewise, the add page could also be more specific with error messages. Error messages in the add page provide no value to the user. They do not mention at all why the unit could not be added into the lecture tables. Was there an invalid entry somewhere or was the current combination of entries already in the database? This scenarios should be properly handle in the back-end and properly indicated back to the user.
- Lastly, the add page allows the user the to add any combination of entries into the database. This means that user can add in a UOS offering from 2006. Should this be allowed? This has not been addressed in the code at all.
- The SQL statements were perfect, and worked as expected.

4.4 Testing Sulav's codebase (by Yan)

The video uploaded to YouTube shows how Sulav has implemented all the functionalities that were required. The functionalities and implementations were all as expected. There are a few improvements that are required to be made which are listed below.

- The presence of spelling mistakes on the webpages reduces the professional look of the application.
- There needs to be more error handling embedded into the forms. For example, in the 'Insert new academic staff member' form, there is little to no error handling outside of the in-built database integrity functions (primary key constraint etc.).
- Further improvement to the extension function would be good. Perhaps a student could have a primary thesis supervisor and also a secondary, to emulate the real world better.

4.5 Testing Yan's codebase (by Udit)

According to the testing video, Yan's SQL code has generated desired output and the websites met all requirements. Invalid input are handled correctly, and the error message on the web clearly indicates users to fix the error. However, there are improvement could be made:

- The library return function should perform more check. The system should check did the user borrowed the book, should check if increment the current number of available copy, will the umber of available copy exceed the total number of copy.
- When search for a book, capitalisation should be ignored. That could be done either by python or SQL. e.g Converting user input and value from database to upper case and then compare.
- Could offer search function in the drop-down of selection. When the drop-down of list offers numerous choices, searching might be more efficient.
- In the update textbook page, there could be a drop-down of choices for the unitOffering, typing three attributes at the same time is inconvenient and easy to make mistakes.

5 Security

5.1 Security Goals

5.1.1 Availability

Availability is ensuring that the data and services required by users is available for use, especially during times where they may be required. In this context, the availability security goal would be to aim to keep the data regarding lectures, units of study, transcripts, etc available when students or staff require it. For example, it would be a violation of the Availability Security goal if students are not able to access their transcripts from Sydney Students. It would be especially concerning if students were unable to access their transcripts when grad program applications begin.

5.1.2 Integrity

Integrity is ensuring that data that the application stores and displays remains true, accurate and reflects the state of the real world. In this context, the integrity security goal would be to aim to keep information regarding lectures, units of study, transcripts, etc remain correct for display. For example, it would be a violation of the Integrity Security Goal if Students see incorrect lecture location, UOS Offerings, Scheduled Exams, etc in the web application.

5.1.3 Confidentiality

Confidentiality is ensuring that the data the application stores and displays is only accessible to users that should have access to it. In this context, the confidentiality security goal would aim to keep information only accessible to to users who have access to it. For example, it would be a violation of the confidentiality secuirty goal if students apart from a particular user are able to view the password of a the user.

5.2 Security Mechanisms

Security Mechanisms refer to how the system achieve these security goals, and it how it will enforce the security policies. In this context, how will it ensure that the academic data is only accessible by academic staff and that they can only add new staff and not remove, or update existing staff.

In our cases one way we have implemented security mechanism is we have had 1 data owner who is responsible for who can access the database what they can access in the database. The only person who could create new relation in the database is the owner and each team member would tell the data owner the new relations they needed and all access requirement they needed to complete the task, this way each person could only access information they needed.

The data owner ran the `CREATE TABLE` statements listed in Listings 42, 46, 45, and 44.

In order to give users the correct permissions, the data owner also ran the `GRANT` statements in Listing 47.

```sql
GRANT USAGE ON SCHEMA unidb to y22s2i2120_qihu6986, y22s2i2120_adha5655,
    y22s2i2120_usam6049, y22s2i2120_yron6616;

-- Antriksh Grant
```

```
GRANT SELECT ON unidb.Student, unidb.Transcript, unidb.UoSOffering, unidb.
    UnitOfStudy, unidb.Classroom TO y22s2i2120_adha5655;
GRANT INSERT ON unidb.Classroom TO y22s2i2120_adha5655;
GRANT select on Exam to y22s2i2120_adha5655;
GRANT select on ExamType to  y22s2i2120_adha5655;
GRANT select on ExamSession to  y22s2i2120_adha5655;

-- Udit Grant
GRANT SELECT, INSERT ON UniDB.Lecture TO y22s2i2120_usam6049
GRANT SELECT ON UniDB.UoSOffering, UniDB.Classroom, UniDb.student TO
    y22s2i2120_usam6049

-- Cody Grant
GRANT SELECT ON unidb.Student, unidb.UoSOffering, unidb.UnitOfStudy, unidb.
    Requires TO y22s2i2120_qihu6986;
GRANT INSERT ON unidb.Requires TO y22s2i2120_qihu6986;
GRANT SELECT, INSERT ON unidb.SubjectTables, unidb.UnitTables TO
    y22s2i2120_qihu6986;
GRANT DELETE ON unidb.SubjectTables TO y22s2i2120_qihu6986;

-- Yan Grant
GRANT SELECT ON unidb.Student, unidb.UoSOffering, unidb.UnitOfStudy TO
    y22s2i2120_yron6616;
GRANT UPDATE ON unidb.UoSOffering, unidb.UnitOfStudy TO y22s2i2120_yron6616; (
GRANT SELECT ON unidb.Libraries, unidb.Books TO y22s2i2120_yron6616;
GRANT UPDATE ON unidb.Books TO y22s2i2120_yron6616;
```

Listing 47: The `GRANT` statements ran by the data owner

5.3 Security Threats

Firstly, what is a Security Threat? It is all the factors to which a system is exposed to. In this case the system is the web application(s) that the team has developed. While discussing Security overall it important to consider the possible areas where the system is particularly vulnerable. This section considers those security threats, and provides potential solutions to them.

- SQL Injection Attacks
  SQL Injection Attacks are something to look out for. Many of the functionalities in the web application take a text input and then take the input inserting into an SQL Query. This is quite dangerous as if this is not accounted for (which it isn't) could potentially allow users to enter SQL into these input boxes. This would mean that the integrity, confidentiality could potentially be violated. Users could use SQL Injection Attacks to insert incorrect or untrue data into the database. The user could also retrieve information that they should not have access to.
- DoS Attacks DoS Attacks or Denial of Service attacks are a genuine concern for the web application. What is a DoS attack? A DoS attack is an attack on the availability of a particular service. It is when a user or a group users take control of over fifty-percent of the available connections. This is quite easy to do for this application. There can be at most five connections to the database

at any point in time. This would mean that if a particular individual desired, it would be possible to take up all five connections at once, rendering all unable to access the database. Even as the programmers of the application it would be impossible to disconnect the user from the server/databse.

- Incorrect Information Inserted Currently it very easy to insert incorrect information into the database. A good example would be the staff table functionalities. Currently, anyone with the link with access to the USYD network can add information regarding staff. This would meant that anyone with this access can add incorrect to the staff table. Also currently, all the website pages have access to the student table (username, and password) this is not entirely safe. The lack of security mechanisms in this aspect violate the Confidentiality and Integrity Security goals.

A Relational schema diagrams

Below are two relational schema diagrams, one describing the original database's schema and one including all completed extensions.

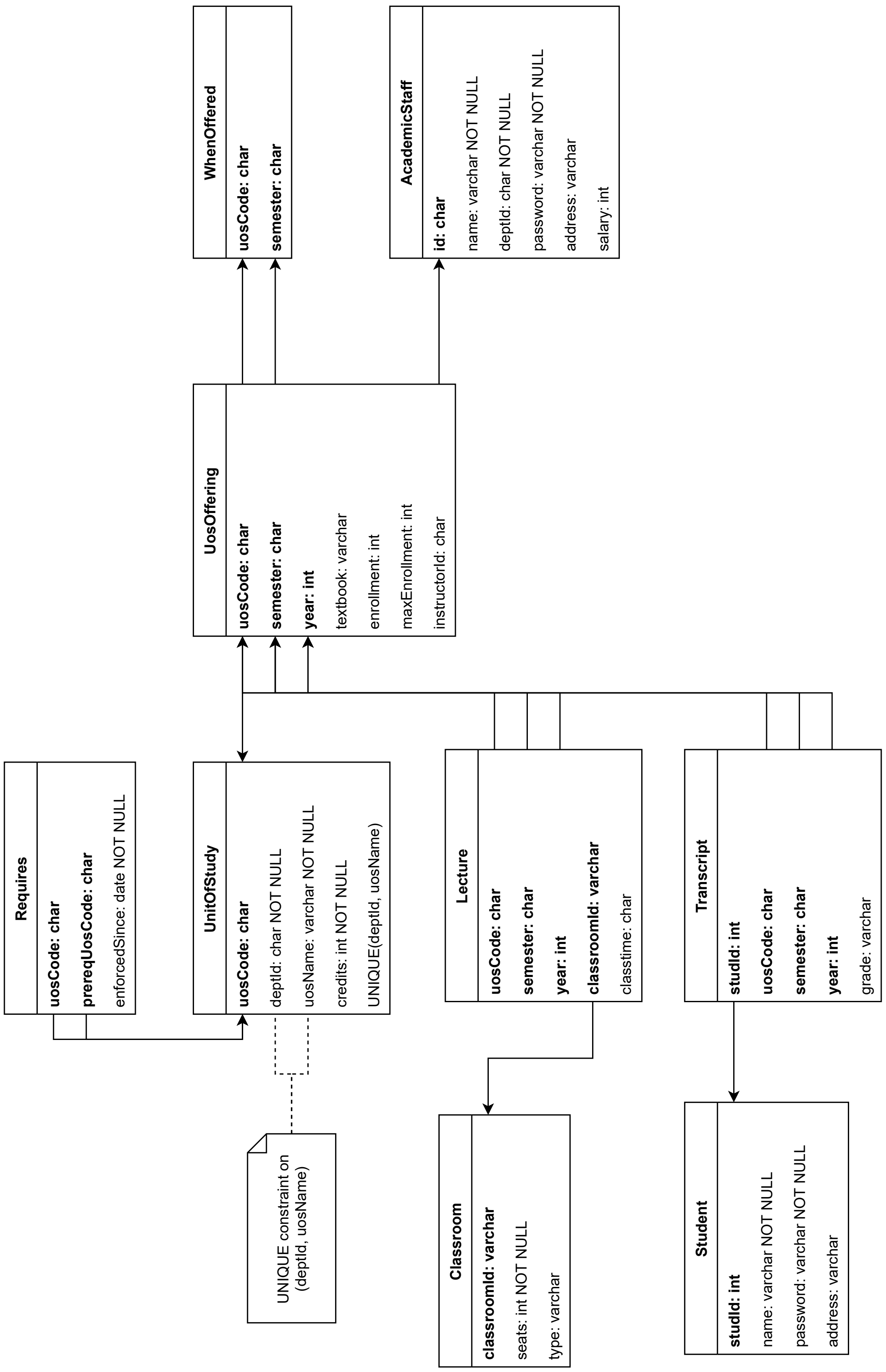


Figure 11: Relational schema diagram of the original database

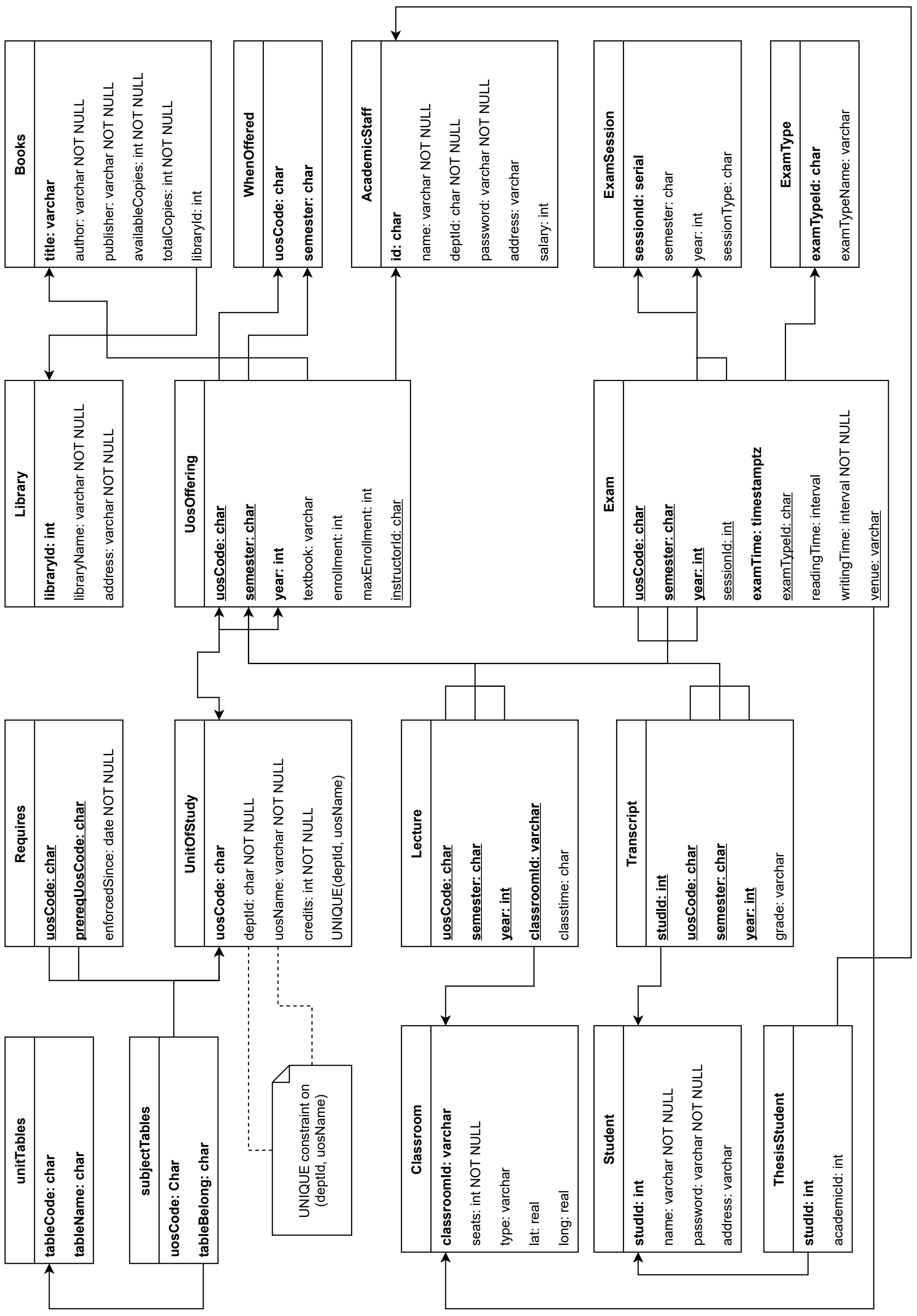


Figure 12: Relational schema diagram of the database with implemented extensions

B DDL statements for extensions

Quick links

B.1 Antriksh's extensions

```sql
-- these values are mostly constant
INSERT INTO ExamType VALUES ('A', 'Live+');
INSERT INTO ExamType VALUES ('B', 'Record+');
INSERT INTO ExamType VALUES ('C', 'Open-book');
INSERT INTO ExamType VALUES ('D', 'Short-release');
INSERT INTO ExamType VALUES ('E', 'Extended-release');
INSERT INTO ExamType VALUES ('F', 'In-person');

-- insert exam sessions from 2000 onwards (generated using a python script)
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2000, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2000, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2000, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2000, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2000, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2000, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2001, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2001, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2001, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2001, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2001, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2001, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2002, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2002, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2002, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2002, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2002, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2002, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2003, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2003, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2003, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2003, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2003, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2003, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2004, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2004, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2004, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2004, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2004, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2004, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2005, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2005, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2005, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2005, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2005, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2005, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2006, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2006, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2006, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2006, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2006, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2006, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2007, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2007, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2007, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2007, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2007, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2007, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2008, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2008, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2008, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2008, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2008, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2008, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2009, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2009, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2009, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2009, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2009, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2009, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2010, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2010, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2010, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2010, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2010, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2010, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2011, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2011, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2011, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2011, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2011, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2011, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2012, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2012, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2012, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2012, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2012, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2012, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2013, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2013, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2013, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2013, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2013, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2013, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2014, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2014, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2014, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2014, 'MAIN');
```

```sql
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2014, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2014, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2015, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2015, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2015, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2015, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2015, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2015, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2016, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2016, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2016, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2016, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2016, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2016, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2017, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2017, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2017, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2017, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2017, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2017, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2018, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2018, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2018, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2018, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2018, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2018, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2019, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2019, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2019, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2019, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2019, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2019, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2020, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2020, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2020, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2020, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2020, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2020, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2021, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2021, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2021, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2021, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2021, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S2', 2021, 'REP2');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2022, 'MAIN');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2022, 'REP1');
INSERT INTO ExamSession VALUES (DEFAULT, 'S1', 2022, 'REP2');

-- insert dummy exams based on units already in the db
```

```sql
SET TIMEZONE TO 'Australia/Sydney';
SET datestyle = dmy;

INSERT INTO Exam VALUES
    (
        'INFO1003',
        'S1',
        2006,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S1' AND year = 2006
    AND sessionType = 'MAIN'),
        '20/06/2006 09:00:00',
        'F',
        '10 minutes',
        '2 hours',
        'BoschLT1'
    );

INSERT INTO Exam VALUES
    (
        'INFO1003',
        'S2',
        2006,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S2' AND year = 2006
    AND sessionType = 'MAIN'),
        '29/11/2006 16:00:00',
        'F',
        '5 minutes',
        '2 hours 10 minutes',
        'SITLT'
    );

INSERT INTO Exam VALUES
    (
        'ISYS2120',
        'S1',
        2006,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S1' AND year = 2006
    AND sessionType = 'MAIN'),
        '17/06/2006 13:00:00',
        'F',
        '10 minutes',
        '2 hours 30 minutes',
        'EA404'
    );


INSERT INTO Exam VALUES
    (
        'ISYS2120',
        'S1',
        2009,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S1' AND year = 2009
    AND sessionType = 'MAIN'),
        '22/06/2009 11:00:00',
        'F',
        '10 minutes',
        '2 hours 30 minutes',
        'CheLT4'
    );

INSERT INTO Exam VALUES
    (
        'ISYS2120',
        'S1',
        2010,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S1' AND year = 2010
    AND sessionType = 'MAIN'),
        '01/07/2010 17:00:00',
        'C',
        '5 minutes',
        '2 hours 40 minutes',
        null
    );

INSERT INTO Exam VALUES
    (
        'DATA3404',
        'S2',
        2008,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S2' AND year = 2008
    AND sessionType = 'MAIN'),
        '01/12/2008 14:00:00',
        'F',
        '20 minutes',
        '2 hours 20 minutes',
        'CAR159'
    );

INSERT INTO Exam VALUES
    (
        'DATA3404',
        'S2',
        2008,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S2' AND year = 2008
    AND sessionType = 'REP1'),
        '16/12/2008 10:00:00',
        'F',
        '20 minutes',
        '2 hours 20 minutes',
        'FarrelLT'
    );

INSERT INTO Exam VALUES
    (
        'DATA3404',
        'S2',
        2008,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S2' AND year = 2008
    AND sessionType = 'REP2'),
        '25/12/2008 14:00:00',
        'F',
        '20 minutes',
        '2 hours 20 minutes',
        'QuadLT'
    );

INSERT INTO Exam VALUES
    (
        'COMP5138',
        'S2',
        2006,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S2' AND year = 2006
    AND sessionType = 'MAIN'),
        '29/11/2006 09:00:00',
        'F',
        '10 minutes',
        '3 hours',
        'CAR175'
    );

INSERT INTO Exam VALUES
    (
        'COMP5138',
        'S1',
        2010,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S1' AND year = 2010
    AND sessionType = 'MAIN'),
        '01/07/2010 09:00:00',
        'F',
        '10 minutes',
        '3 hours',
        'MechLT'
    );

INSERT INTO Exam VALUES
    (
        'COMP5046',
        'S1',
        2010,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S1' AND year = 2010
    AND sessionType = 'MAIN'),
        '26/06/2010 09:00:00',
        'F',
        '10 minutes',
        '3 hours',
        'BoschLT4'
    );

INSERT INTO Exam VALUES
    (
        'COMP5338',
        'S1',
        2006,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S1' AND year = 2006
    AND sessionType = 'MAIN'),
        '13/06/2006 12:00:00',
        'D',
        '10 minutes',
        '2 hours 30 minutes',
        null
    );

INSERT INTO Exam VALUES
    (
        'COMP5338',
        'S2',
        2006,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S2' AND year = 2006
    AND sessionType = 'MAIN'),
        '13/11/2006 15:00:00',
        'D',
        '10 minutes',
        '2 hours 30 minutes',
        null
    );

INSERT INTO Exam VALUES
    (
        'INFO2005',
        'S2',
        2004,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S2' AND year = 2004
    AND sessionType = 'MAIN'),
        '23/11/2004 09:00:00',
        'F',
        '5 minutes',
```

```sql
        '2 hours 25 minutes',
        'QuadLT'
    );

INSERT INTO Exam VALUES
    (
        'INFO3005',
        'S1',
        2005,
        (SELECT sessionId FROM ExamSession WHERE semester = 'S1' AND year = 2005
    AND sessionType = 'MAIN'),
        '01/05/2005 11:00:00',
        'F',
        '10 minutes',
        '2 hours',
        'CAR159'
    );

/* SETTING LOCATION DATA FOR LECTURE THEATRES */

-- Bosch Building lecture theatres
UPDATE unidb.Classroom
SET lat = -33.88927615009087, long = 151.18525296608223
WHERE classroomId IN ('BoschLT1', 'BoschLT2', 'BoschLT3', 'BoschLT4');

-- Chemistry Building lecture theatres
UPDATE unidb.Classroom
SET lat = -33.887670477912124, long = 151.18938789726445
WHERE classroomId IN ('CheLT1', 'CheLT2', 'CheLT3', 'CheLT4');

-- Carslaw lecture theatres
UPDATE unidb.Classroom
SET lat = -33.887993084435905, long = 151.1908708923028
WHERE classroomId IN ('CAR157', 'CAR159', 'CAR273', 'CAR275');

UPDATE unidb.Classroom
SET lat = -33.88802278239188, long = 151.19053418540216
WHERE classroomId IN ('CAR173', 'CAR175', 'CAR373', 'CAR375');

-- Eastern Avenue buildings
UPDATE unidb.Classroom
SET lat = -33.88814690843262, long = 151.19033352226836
WHERE classroomId IN ('EAA', 'EALT', 'EA403', 'EA404', 'EA405', 'EA406');

-- PNR Farrel lecture theatre
UPDATE unidb.Classroom
SET lat = -33.89019849155387, long = 151.19294413098282
WHERE classroomId IN ('FarrelLT');

```

```sql
-- Mechanical lecture theatre
UPDATE unidb.Classroom
SET lat = -33.88904661716015, long = 151.19394710338895
WHERE classroomId IN ('MechLT');

-- Quadrangle lecture theatre
UPDATE unidb.Classroom
SET lat = -33.886201247928895, long = 151.18915263596094
WHERE classroomId IN ('QuadLT');

-- SIT lecture theatre
UPDATE unidb.Classroom
SET lat = -33.888187543793535, long = 151.19424876755252
WHERE classroomId IN ('SITLT');

COMMIT;
```

Listing 48: `INSERT INTO` statements for Antriksh's exam data and classroom location extensions

B.2 Cody's extensions

```sql
set schema 'unidb';

/* get some study units from other table into UnitOfStudy */
INSERT INTO UnitOfStudy VALUES ('OLES2137', 'SLC', 'Experience China', 6);
INSERT INTO UnitOfStudy VALUES ('OLET5608', 'MAT', 'Linear Modelling', 6);
INSERT INTO UnitOfStudy VALUES ('SCDL3991', 'SIT', 'Science Dalyell Individual
    Research', 6);
INSERT INTO UnitOfStudy VALUES ('EDPK5002', 'MAT', 'Quantitative Methods', 6);
INSERT INTO UnitOfStudy VALUES ('OLET1622', 'MAT', 'Numbers and Numerics', 2);

/* insert into UnitTables */
INSERT INTO UnitTables VALUES ('O', 'Open Learning Enviroment');
INSERT INTO UnitTables VALUES ('A', 'Degree Core Units of Study');
INSERT INTO UnitTables VALUES ('D', 'The Dalyell Stream');
INSERT INTO UnitTables VALUES ('R', 'Higher Degree by Research');
INSERT INTO UnitTables VALUES ('S', 'Shared Pool');

/* insert into SubjectTables */
INSERT INTO SubjectTables VALUES ('OLES2137', 'O');
INSERT INTO SubjectTables VALUES ('OLET5608', 'O');
INSERT INTO SubjectTables VALUES ('SCDL3991', 'D');
INSERT INTO SubjectTables VALUES ('EDPK5002', 'R');
INSERT INTO SubjectTables VALUES ('OLET1622', 'O');

INSERT INTO SubjectTables VALUES ('INFO1003', 'S');
INSERT INTO SubjectTables VALUES ('ISYS2120', 'S');
INSERT INTO SubjectTables VALUES ('DATA3404', 'S');
INSERT INTO SubjectTables VALUES ('COMP5046', 'S');
INSERT INTO SubjectTables VALUES ('COMP5138', 'S');
INSERT INTO SubjectTables VALUES ('COMP5338', 'S');
INSERT INTO SubjectTables VALUES ('INFO2005', 'S');
INSERT INTO SubjectTables VALUES ('INFO3005', 'S');
INSERT INTO SubjectTables VALUES ('MATH1002', 'S');

INSERT INTO SubjectTables VALUES ('ISYS2120', 'A');
INSERT INTO SubjectTables VALUES ('DATA3404', 'A');
INSERT INTO SubjectTables VALUES ('INFO2005', 'A');
INSERT INTO SubjectTables VALUES ('INFO3005', 'A');
INSERT INTO SubjectTables VALUES ('MATH1002', 'A');

COMMIT;
```

Listing 49: `INSERT INTO` statements for Cody's extensions tables

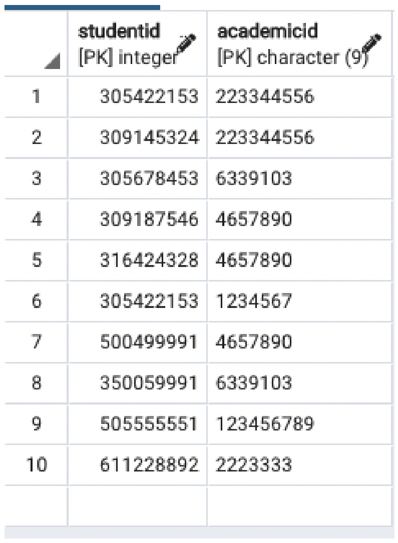
B.3 Yan's extensions

```sql
set schema 'unidb';
/*Insert into libraries table*/
INSERT INTO libraries VALUES (1, 'Central library', 'F03');
INSERT INTO libraries VALUES (2, 'Engineering faculty library', 'G02');
INSERT INTO libraries VALUES (3, 'Medical faculty library', 'D18');
INSERT INTO libraries VALUES (4, 'Law building library', 'F10');
INSERT INTO libraries VALUES (5, 'Conservatorium Library', '1 Conservatorium Rd');

/*Insert into Books table*/
INSERT INTO Books VALUES ('Snyder', 'Snyder', 'p1', 1, 3, 1);
INSERT INTO Books VALUES ('Kifer/Bernstein/Lewis', 'Kifer/Bernstein/Lewis', 'p2',
    3, 3, 1);
INSERT INTO Books VALUES ('Ramakrishnan/Gehrke', 'Ramakrishnan/Gehrke', 'p2', 2,
    5, 2);
INSERT INTO Books VALUES ('Hoffer', 'Hoffer', 'p5', 0, 2, 1);
INSERT INTO Books VALUES ('Introduction to python', 'Peter', 'p3', 0, 1, 2);
INSERT INTO Books VALUES ('Database concepts', 'Andrew', 'p3', 5, 6, 2);
INSERT INTO Books VALUES ('Object oriented programming', 'Andy', 'p3', 2, 3, 2);

COMMIT;
```

Listing 50: `INSERT INTO` statements for Yan's extensions tables

B.4 Sulav's extensions

DDL statements for Sulav's extensions could not be provided, however Figure 13 contains proof that the table is in fact filled with some data.



	studentid [PK] integer	academicid [PK] character (9)
1	305422153	223344556
2	309145324	223344556
3	305678453	6339103
4	309187546	4657890
5	316424328	4657890
6	305422153	1234567
7	500499991	4657890
8	350059991	6339103
9	505555551	123456789
10	611228892	2223333

Figure 13: Proof of insertions into the `ThesisStudent` table