ISYS2120 Assignment 3 Report

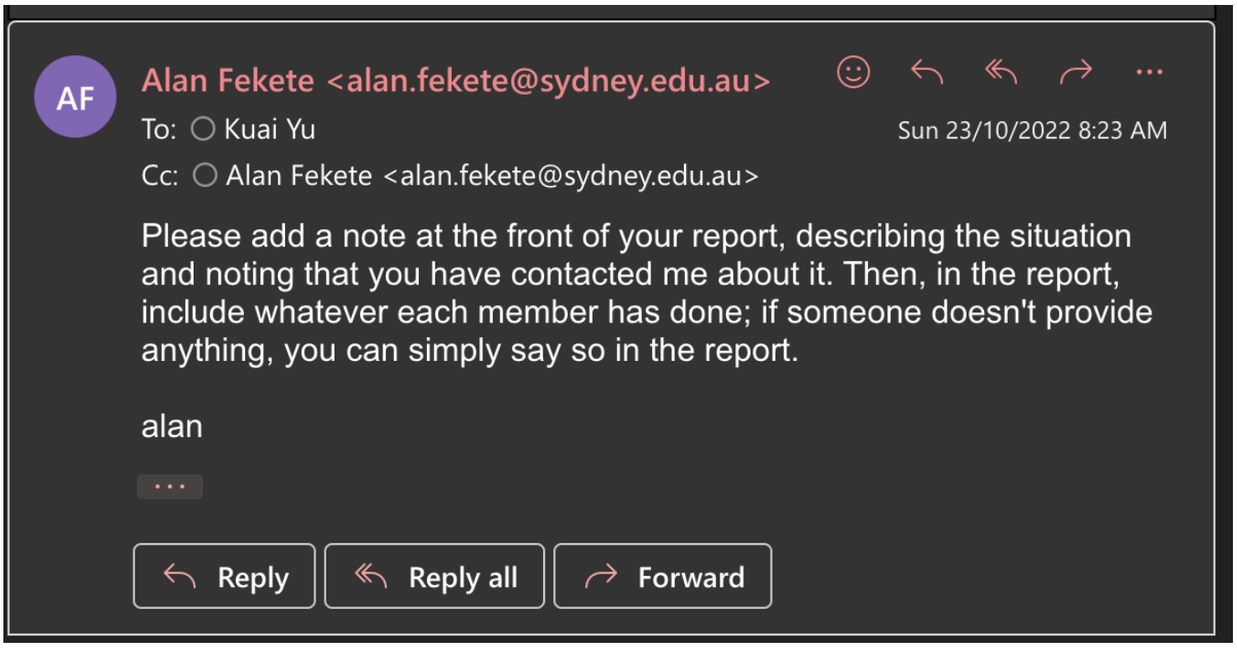
Kuai Yu 500034918

Kefeng Chen 510549198

Kevin Zheng 510233600

Chenjun Cao <- This person did nothing

This person has been contacted by other group members multiple times, but to date, no response yet. Note this person did attend the most recent tutorial on 21/11/2022. Prof. Alan Flake has been informed and is aware of this situation.



Part 1 Code:

Completed by: Kuai Yu 500034918

Classrooms page set:

List all classrooms:

```
def list_classrooms():
    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 classroomId, seats, type
            from unidb.classroom
            """
        )
        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
```

Display the number of classrooms by type:

```
def number_of_classroom_per_type():
    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 type, count(*)
            from unidb.classroom
            group by type
            """
        )
        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
```

Search classrooms by the number of seats:

```
def search_classrooms_by_number_of_seat(number_of_seat):
    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 classroomId, seats, type
            from unidb.classroom
            where seats >= %s
            """,
            (number_of_seat,),
        )
        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
```

Add classrooms:

```
def add_classroom(classroomId, seats, class_type):
    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 fr
        cur.execute(
            """
            insert into unidb.classroom (classroomId, seats, type)
            values (cast(%s as text), %s, cast(%s as text))
            """,
            (classroomId, seats, class_type),
        )
        conn.commit()
        val = cur.fetchall()
    except Exception as e:
        pass
        # If there were any errors, we print something nice and return a NULL value
```

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

Extensions:

Extension 1: Get booked classrooms:

```
def get_booked_classrooms():
    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 c.classroomId, c.start_time, c.end_time
            from unidb.kuyu5570_classroombooking c
            """
        )
        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
```

Extension 2: Get all classrooms that are available in the given period:

```
def get_available_classrooms_between_two_timestamps(start_time, end_time):
    conn = database_connect()
    if conn is None:
        return None
    # Sets up the rows as a dictionary
    cur = conn.cursor()
    val = None

    # convert utc format format to iso
    start_time = datetime.strptime(start_time, "%Y-%m-%dT%H:%M")
    end_time = datetime.strptime(end_time, "%Y-%m-%dT%H:%M")

    try:
        # Try getting all the information returned from the query
        # NOTE: column ordering is IMPORTANT
        cur.execute(
            """
            select c.classroomId, c.seats, c.type
            from unidb.classroom c left join unidb.kuyu5570_classroombooking cb using
(classroomId)
            where
```

```
            (not %s between cb.start_time and cb.end_time)
            and
            (not %s between cb.start_time and cb.end_time)
            and
            (not cb.start_time between %s and %s)
            and
            (not cb.end_time between %s and %s)
            or
            (cb.start_time is null and cb.end_time is null);
            """,
            (start_time, end_time, start_time, end_time, start_time, end_time),
        )
        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
```

Completed by: Kefeng Chen 510549198

Department page set:

List all staffs

```
def list_staff():
    # 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:
        cur.execute("""SELECT id, name, deptid, address FROM
unidb.academicstaff""")
        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
```

Search department

```
def list_staff_department(depart_name : str):
    # 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:
        cur.execute("""SELECT id, name, deptid, address FROM
unidb.academicstaff WHERE deptid = %s""", (depart_name, )) # TODO: Make
injection safe
        val = cur.fetchall()
    except:
        print("Error fetching from database")

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

Department count

```
def list_department_count():
     # 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:
        cur.execute("""SELECT deptid, COUNT(*) FROM unidb.academicstaff GROUP
BY 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
```

Add staff

```
def add_staff(id, name, deptid, password, address, salary):
    # 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:
        cur.execute("""INSERT INTO unidb.academicstaff VALUES(%s, %s, %s, %s,
%s, %s)""", (id, name, deptid, password, address, salary))
        conn.commit()
        val = cur.fetchone()
    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
```

Completed by: Kevin Zheng 510233600

Prerequisites UoS page set:

List all prerequisite pair:

```
def get_prereq():
    conn = database_connect()
    if (conn is None):
        return None

    cur = conn.cursor()
    val = None
    try:
        cur.execute("""SELECT r.uoSCode UoSCode, u.uoSName UosName,
                        r.prereqUoSCode PrereqUoSCode,
                        u2.uoSName PrereqUoSName, r.enforcedSince
                        FROM UniDB.Requires r
                        INNER JOIN UniDB.UnitOfStudy u
                        ON r.uoSCode = u.uoSCode
                        INNER JOIN UniDB.UnitOfStudy u2
                        ON r.prereqUoSCode = u2.uoSCode
                        """)
        val = cur.fetchall()
    except:
        print("Error fetching from database")

    cur.close()
    conn.close()
    return val
```

Search prerequisites of a given unit:

```
def search_prereq(uosCode):
    conn = database_connect()
    if (conn is None):
        return None

    cur = conn.cursor()
    val = None
    try:
        sql = """SELECT uoSCode, string_agg(prereqUoSCode, ', ') PrereqUoSCode
                    FROM UniDB.Requires
                    WHERE uoSCode = %s
                    GROUP BY uoSCode
                    """
        cur.execute(sql, (uosCode,))
        val = cur.fetchall()
```

```
    except:
        print("Error fetching from database")

    cur.close()
    conn.close()
    return val
```

A report of the number of prerequisites for each unit:

```
def get_pre_amount():
    conn = database_connect()
    if (conn is None):
        return None

    cur = conn.cursor()
    val = None
    try:
        cur.execute("""SELECT uoSCode, COUNT(*) "Number of Prerequires"
                       FROM UniDB.Requires
                       GROUP BY uoSCode
                       """)
        val = cur.fetchall()
    except:
        print("Error fetching from database")

    cur.close()
    conn.close()
    return val
```

Adding a new prerequisites:

```
def add(uoSCode, prereqUoSCode):
    conn = database_connect()
    if (conn is None):
        return None

    cur = conn.cursor()
    val = None
    try:
        sql = """INSERT INTO UniDB.Requires VALUES (%s, %s, now())"""
        cur.execute(sql, (uoSCode, prereqUoSCode))
        conn.commit()
        cur.execute("""SELECT * FROM UniDB.Requires WHERE uoSCode = %s AND prereqUoSCode =
%s""",
                    (uoSCode, prereqUoSCode))
        val = cur.fetchall()
    except:
        print("Error fetching from database")

    cur.close()
    conn.close()
    return val
```

Extension:

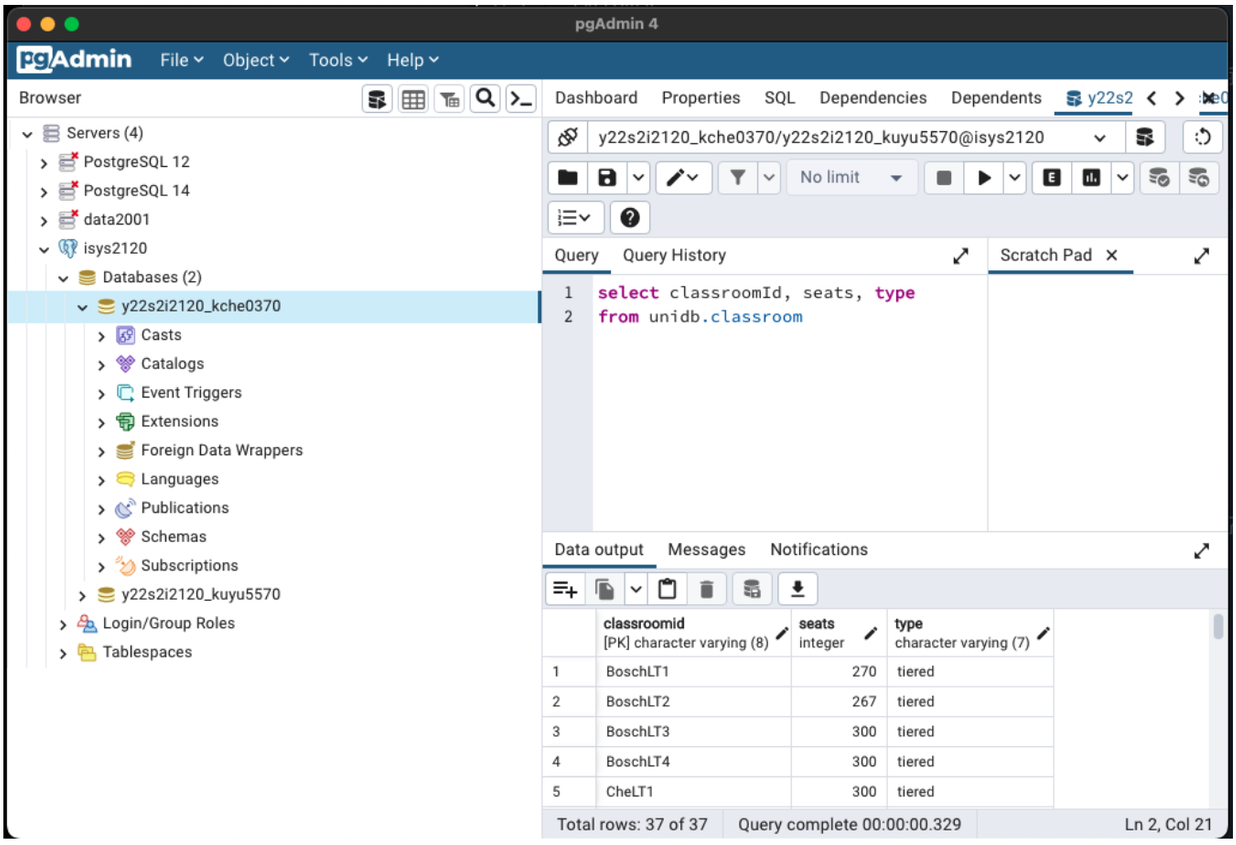
```
def get_credits(uosCode):
    conn = database_connect()
    if (conn is None):
        return None

    cur = conn.cursor()
    val = None
    try:
        sql = """SELECT r.uoSCode UoSCode,
                string_agg(r.prereqUoSCode, ', ') PrereqUoSCode,
                sum(u.credits) "Total Credits"
                FROM UniDB.Requires r
                INNER JOIN UniDB.UnitOfStudy u
                on r.prereqUoSCode = u.uoSCode
                WHERE r.uoSCode = %s
                GROUP BY r.uoSCode"""
        cur.execute(sql, (uosCode,))
        val = cur.fetchall()
    except:
        print("Error fetching from database")

    cur.close()
    conn.close()
    return val
```

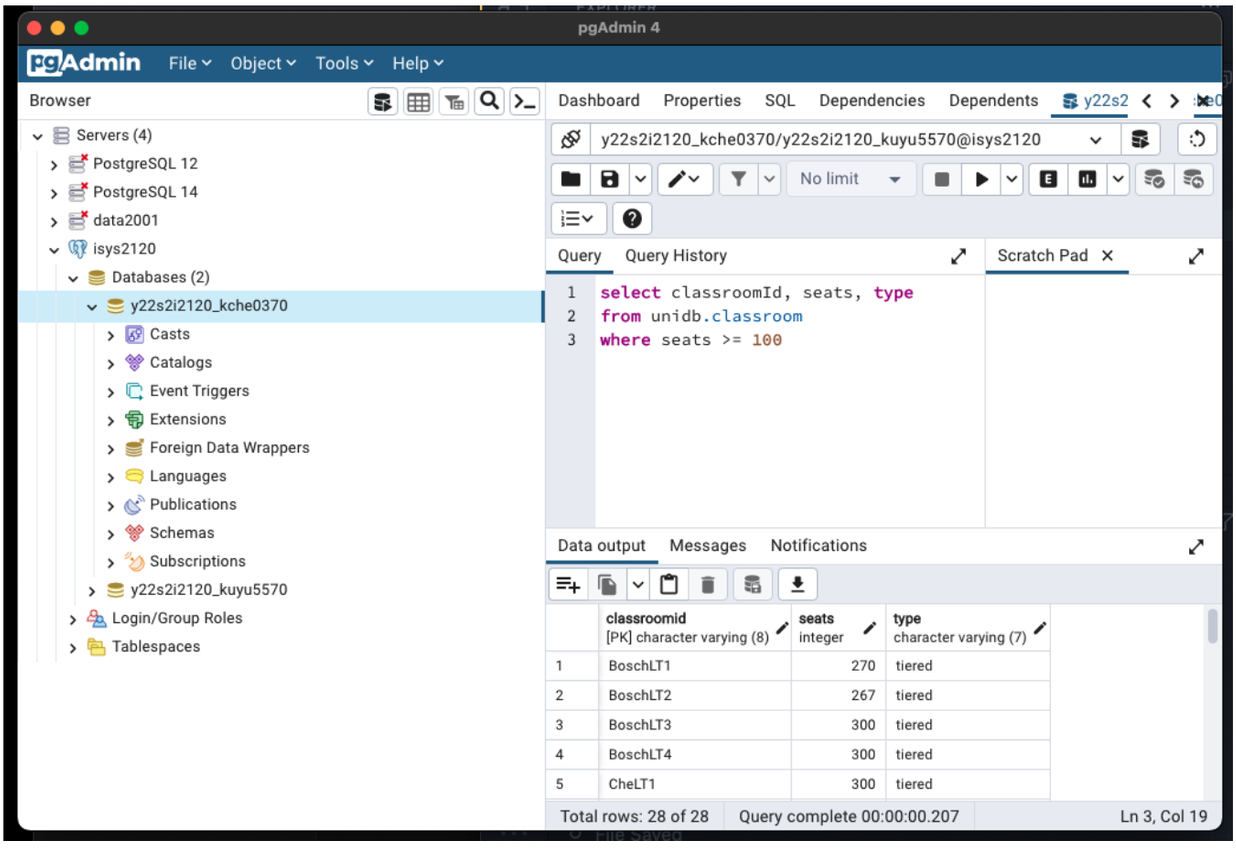
Part 2 Testing:

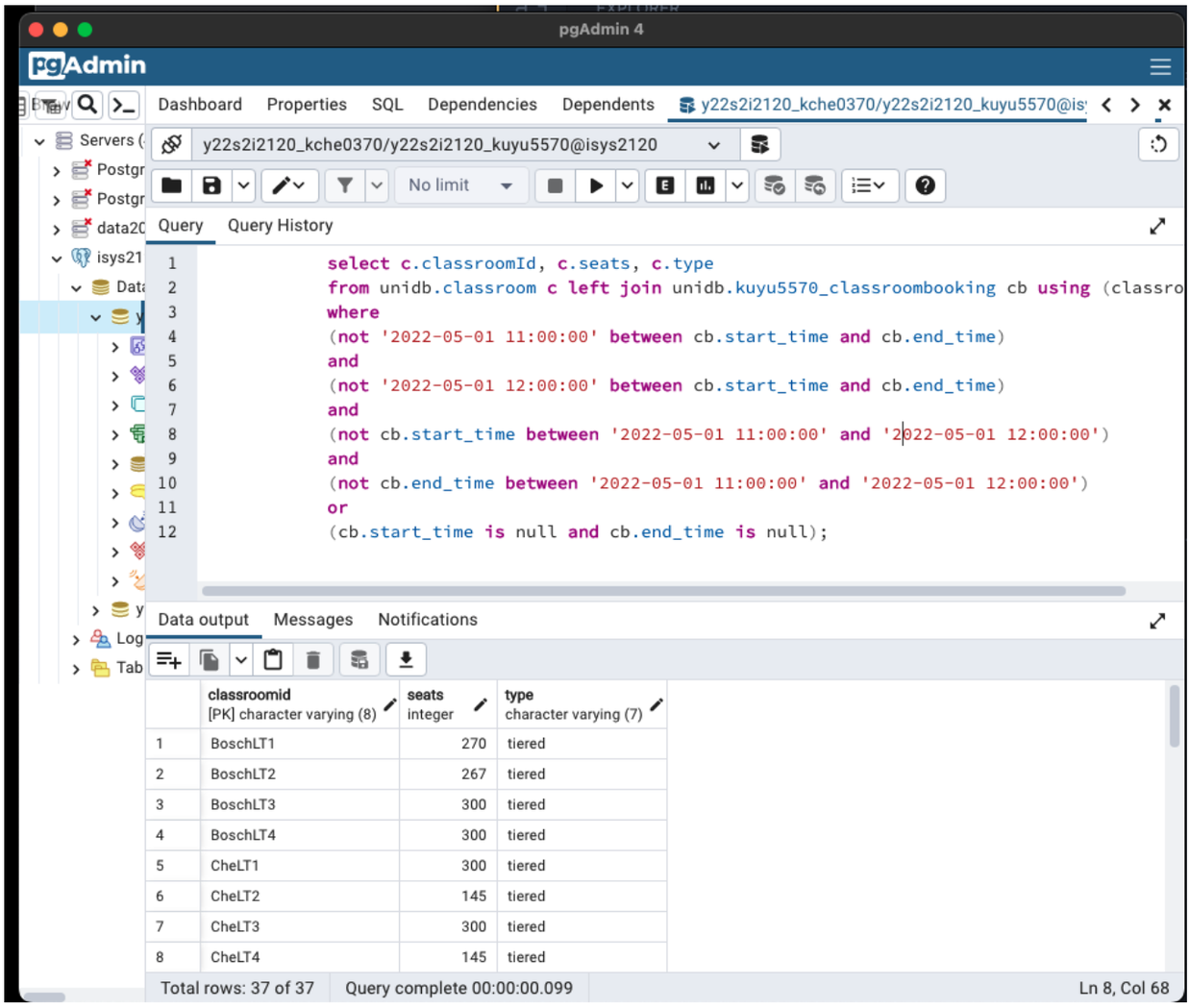
The testing for one member is conducted by another member of the group. To test the other members website, each member will receive a zipped codebase from another member. For each test, there are three parts

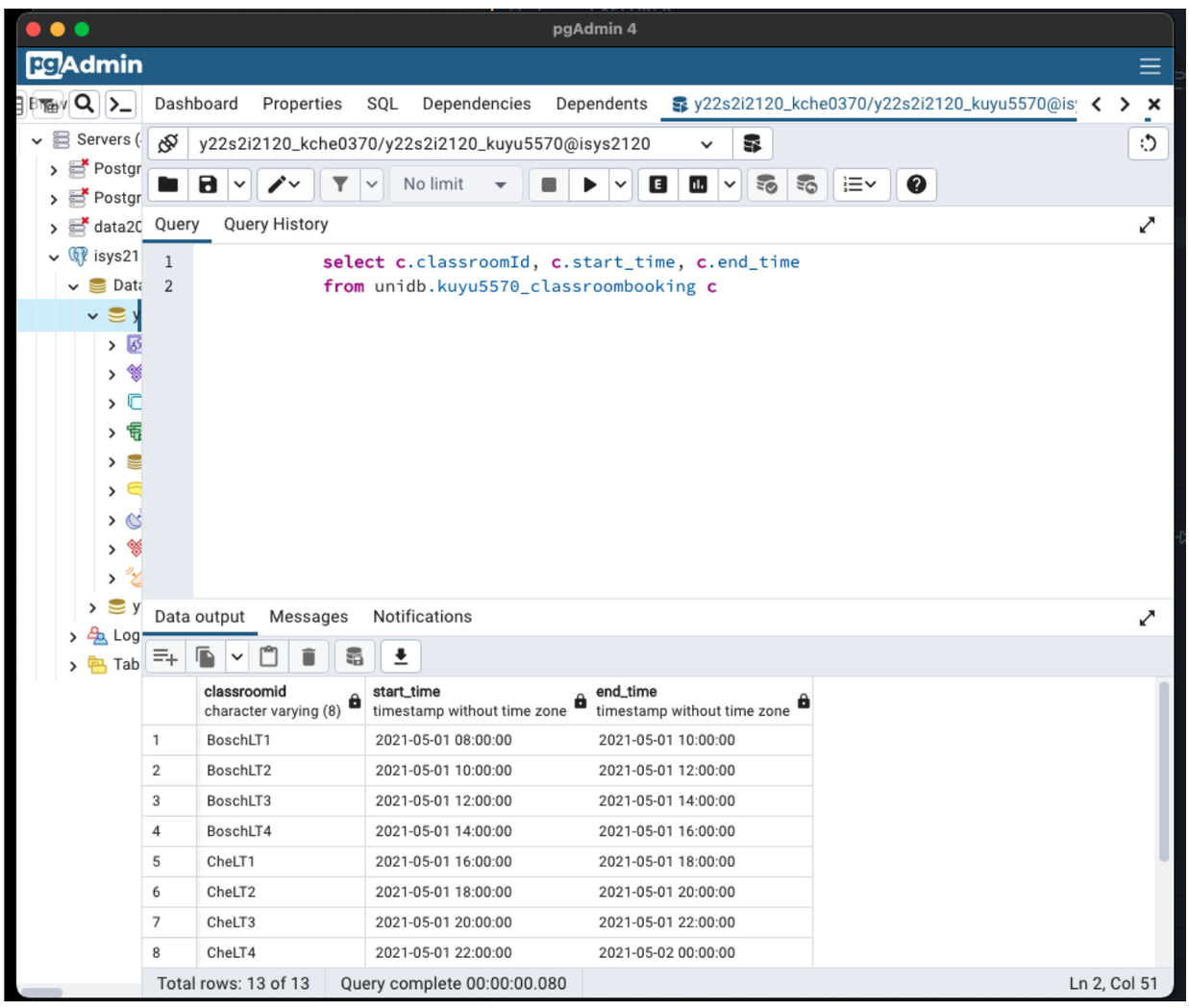
Classroom Tests

PgAdmin:



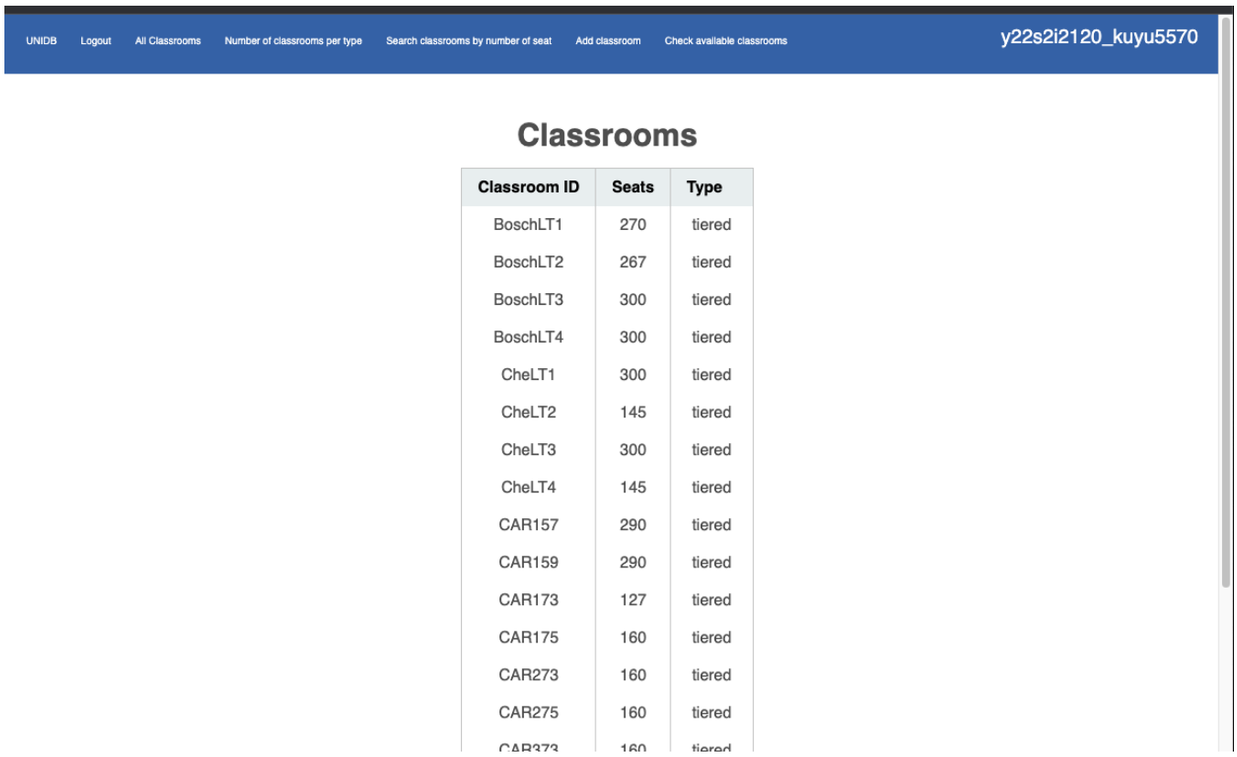


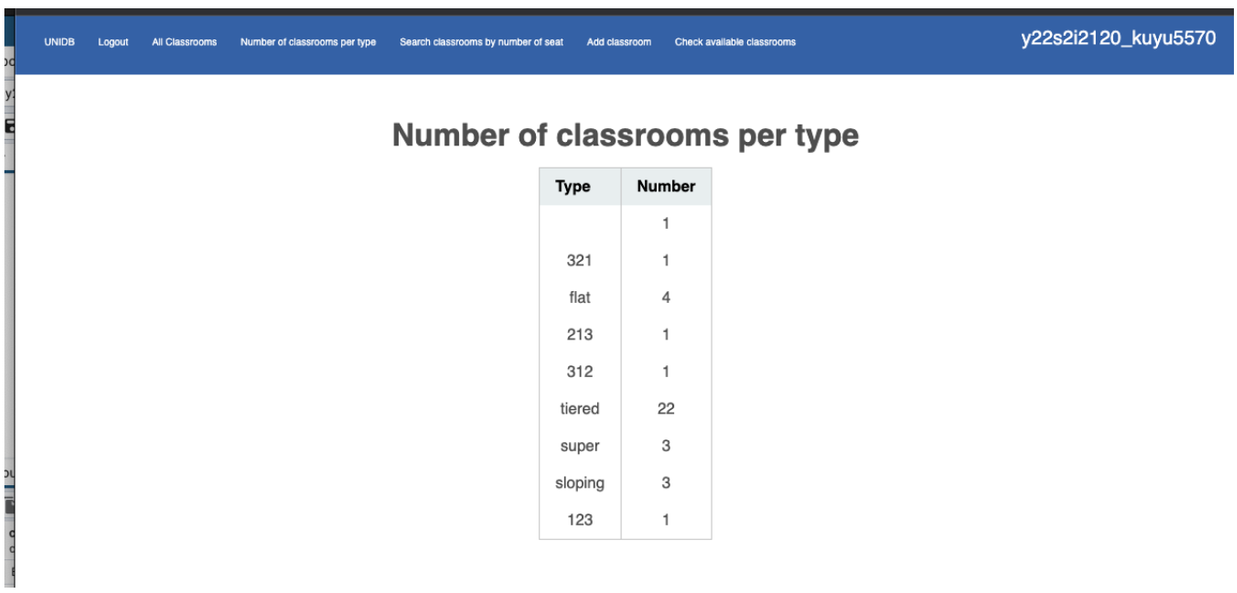

pgAdmin 4
pgAdmin
Dashboard
Properties
SQL
Dependencies
Dependents
y22s2i2120_kche0370/y22s2i2120_kuyu5570@isys2120
No limit
Query
Query History
select c.classroomId, c.seats, c.type
from unidb.classroom c left join unidb.kuyu5570_classroombooking cb using (classro
where
(not '2022-05-01 11:00:00' between cb.start_time and cb.end_time)
and
(not '2022-05-01 12:00:00' between cb.start_time and cb.end_time)
and
(not cb.start_time between '2022-05-01 11:00:00' and '2022-05-01 12:00:00')
and
(not cb.end_time between '2022-05-01 11:00:00' and '2022-05-01 12:00:00')
or
(cb.start_time is null and cb.end_time is null);
Data output
Messages
Notifications
classroomid [PK] character varying (8)
seats integer
type character varying (7)
1 BoschLT1 270 tiered
2 BoschLT2 267 tiered
3 BoschLT3 300 tiered
4 BoschLT4 300 tiered
5 CheLT1 300 tiered
6 CheLT2 145 tiered
7 CheLT3 300 tiered
8 CheLT4 145 tiered
Total rows: 37 of 37
Query complete 00:00:00.099
Ln 8, Col 68

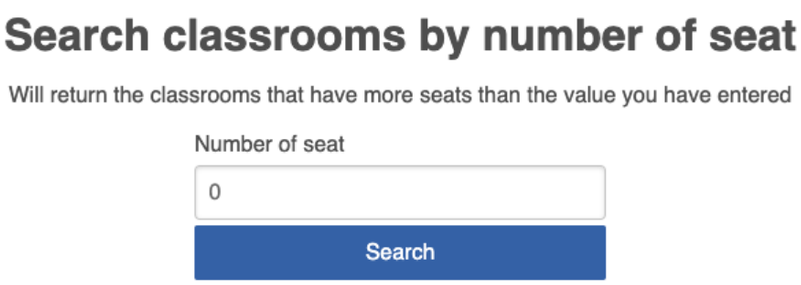

pgAdmin 4
pgAdmin
Dashboard
Properties
SQL
Dependencies
Dependents
y22s2i2120_kche0370/y22s2i2120_kuyu5570@isys2120
No limit
Query
Query History
select c.classroomId, c.start_time, c.end_time
from unidb.kuyu5570_classroombooking c
Data output
Messages
Notifications
classroomid character varying (8)
start_time timestamp without time zone
end_time timestamp without time zone
1 BoschLT1 2021-05-01 08:00:00 2021-05-01 10:00:00
2 BoschLT2 2021-05-01 10:00:00 2021-05-01 12:00:00
3 BoschLT3 2021-05-01 12:00:00 2021-05-01 14:00:00
4 BoschLT4 2021-05-01 14:00:00 2021-05-01 16:00:00
5 CheLT1 2021-05-01 16:00:00 2021-05-01 18:00:00
6 CheLT2 2021-05-01 18:00:00 2021-05-01 20:00:00
7 CheLT3 2021-05-01 20:00:00 2021-05-01 22:00:00
8 CheLT4 2021-05-01 22:00:00 2021-05-02 00:00:00
Total rows: 13 of 13
Query complete 00:00:00.080
Ln 2, Col 51

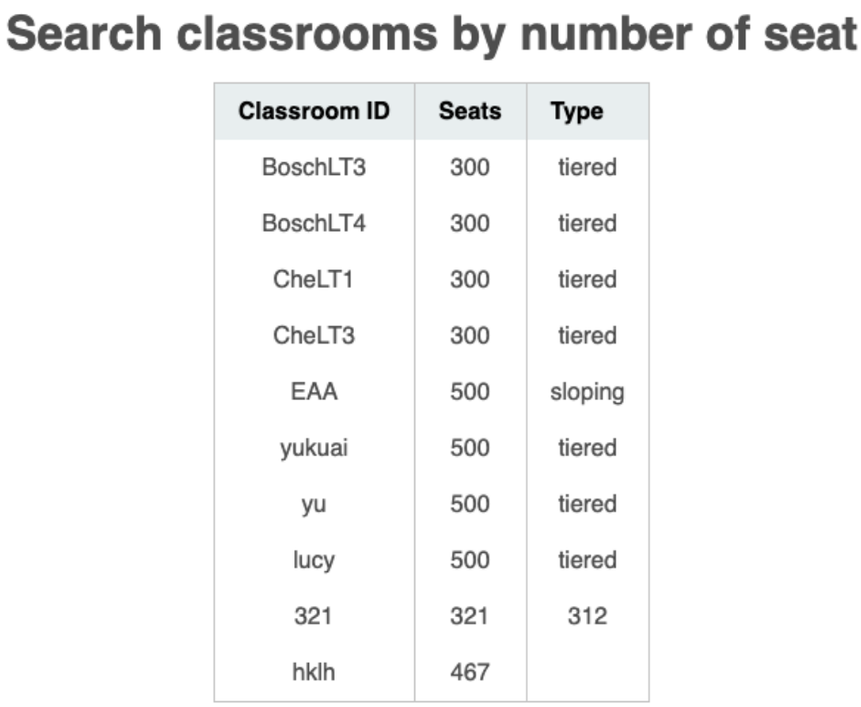
By extracting the SQL code from database.py and filling up “%s” with some values, the result that has been returned from that SQL query will be assessed manually For example, if a query is intended to find classrooms over 200 people, then. no classroom with less than 200 seats should appear in the data frame.

Website:









Classroom ID	Seats	Type
BoschLT3	300	tiered
BoschLT4	300	tiered
CheLT1	300	tiered
CheLT3	300	tiered
EAA	500	sloping
yukuai	500	tiered
yu	500	tiered
lucy	500	tiered
321	321	312
hklh	467	

The website testing is done by manually checking every page and if their return values are sensible.

State Tests

yu	500	tiered
lucy	500	tiered
123	1	123
321	321	312
12 21	12	213
啦啦啦	123	321
hklh	467	

When testing under different states, first, we will insert some new values using one “add” functionalities that require inserting to insert some new values into the database. Then, we will use other functions that the member has written to test if the newly inserted value is in the result.

Staff Tests

PgAdmin:

Query | Query History

```
1  SELECT id, name, deptid, address FROM unidb.academicstaff;
2
```

Data output | Messages | Notifications

	id [PK] character (9)	name character varying (20)	deptid character (3)	address character varying (50)
1	6339103	Uwe Roehm	SIT	Cremorne
2	1234567	Jon Patrick	SIT	Glebe
3	7891234	Sanjay Chawla	SIT	Neutral Bay
4	1237890	Joseph Davis	SIT	[null]
5	4657890	Alan Fekete	SIT	Cameray
6	0987654	Simon Poon	SIT	Sydney
7	1122334	Irena Koprinska	SIT	Glebe

Query | Query History

```
1  SELECT id, name, address FROM unidb.academicstaff WHERE deptid = 'SIT';
2
```

Data output | Messages | Notifications

	id [PK] character (9)	name character varying (20)	address character varying (50)
1	6339103	Uwe Roehm	Cremorne
2	1234567	Jon Patrick	Glebe
3	7891234	Sanjay Chawla	Neutral Bay
4	1237890	Joseph Davis	[null]
5	4657890	Alan Fekete	Cameray
6	0987654	Simon Poon	Sydney
7	1122334	Irena Koprinska	Glebe

(The above deptid = ‘SIT’ would be replaced by deptid = %s where %s is replaced by user input)

Query | Query History

```
1  SELECT deptid, COUNT(*) FROM unidb.academicstaff GROUP BY deptid
2
```

Data output | Messages | Notifications

	deptid character (3)	count bigint
1	SIT	7

Query Query History

```
1  INSERT INTO unidb.academicstaff VALUES(1111111, 'John Doe', 'SIT', 'password', 'address', 123);
2
```

Data output Messages Notifications

```
INSERT 0 1

Query returned successfully in 80 msec.
```

Query Query History

```
1  SELECT id, name, deptid, address FROM unidb.academicstaff;
2
```

Data output Messages Notifications

	id [PK] character (9)	name character varying (20)	deptid character (3)	address character varying (50)
1	6339103	Uwe Roehm	SIT	Cremorne
2	1234567	Jon Patrick	SIT	Glebe
3	7891234	Sanjay Chawla	SIT	Neutral Bay
4	1237890	Joseph Davis	SIT	[null]
5	4657890	Alan Fekete	SIT	Cameray
6	0987654	Simon Poon	SIT	Sydney
7	1122334	Irena Koprinska	SIT	Glebe
8	1111111	John Doe	SIT	address

(Above shows the effects of inserting a new value)

Website:

All Academic Staff

ID	Name	Deptid	Address
6339103	Uwe Roehm	SIT	Cremorne
1234567	Jon Patrick	SIT	Glebe
7891234	Sanjay Chawla	SIT	Neutral Bay
1237890	Joseph Davis	SIT	None
4657890	Alan Fekete	SIT	Cameray
0987654	Simon Poon	SIT	Sydney
1122334	Irena Koprinska	SIT	Glebe

Search department staff

Department ID

Search department staff

SIT

Staff in Department

ID	Name	Deptid	Address
6339103	Uwe Roehm	SIT	Cremorne
1234567	Jon Patrick	SIT	Glebe
7891234	Sanjay Chawla	SIT	Neutral Bay
1237890	Joseph Davis	SIT	None
4657890	Alan Fekete	SIT	Cameray
0987654	Simon Poon	SIT	Sydney
1122334	Irena Koprinska	SIT	Glebe

Department Staff Count

Department	Count
SIT	7

Add new staff

ID

Name

Department

Password

Address (optional)

Salary

Submit Query

Add new staff

1111111

Guy Smith

ABC

123456

Sydney

1000000

Submit Query

State:

All Academic Staff

ID	Name	Deptid	Address
6339103	Uwe Roehm	SIT	Cremorne
1234567	Jon Patrick	SIT	Glebe
7891234	Sanjay Chawla	SIT	Neutral Bay
1237890	Joseph Davis	SIT	None
4657890	Alan Fekete	SIT	Cameray
0987654	Simon Poon	SIT	Sydney
1122334	Irena Koprinska	SIT	Glebe
1111111	Guy Smith	ABC	Sydney

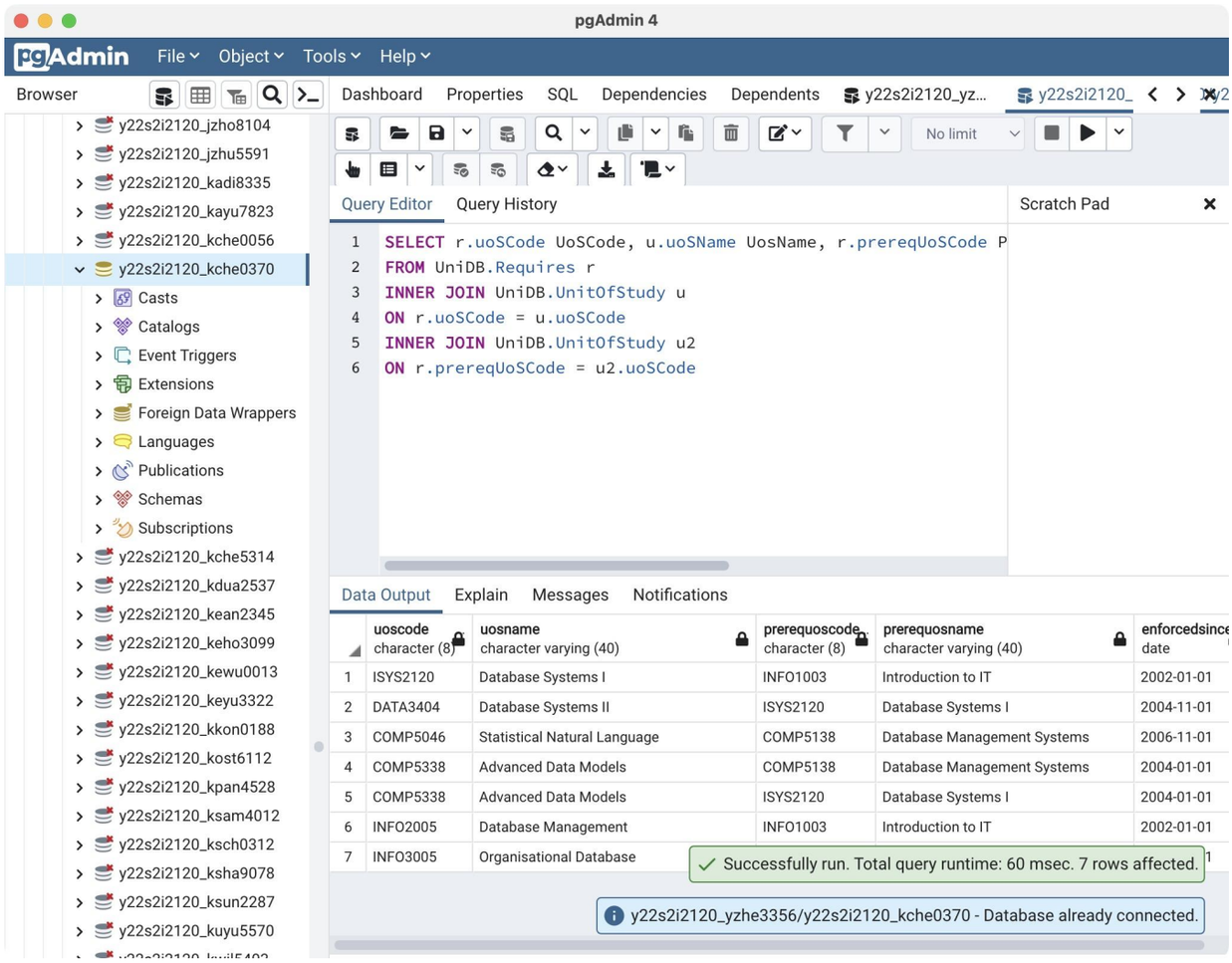
Department Staff Count

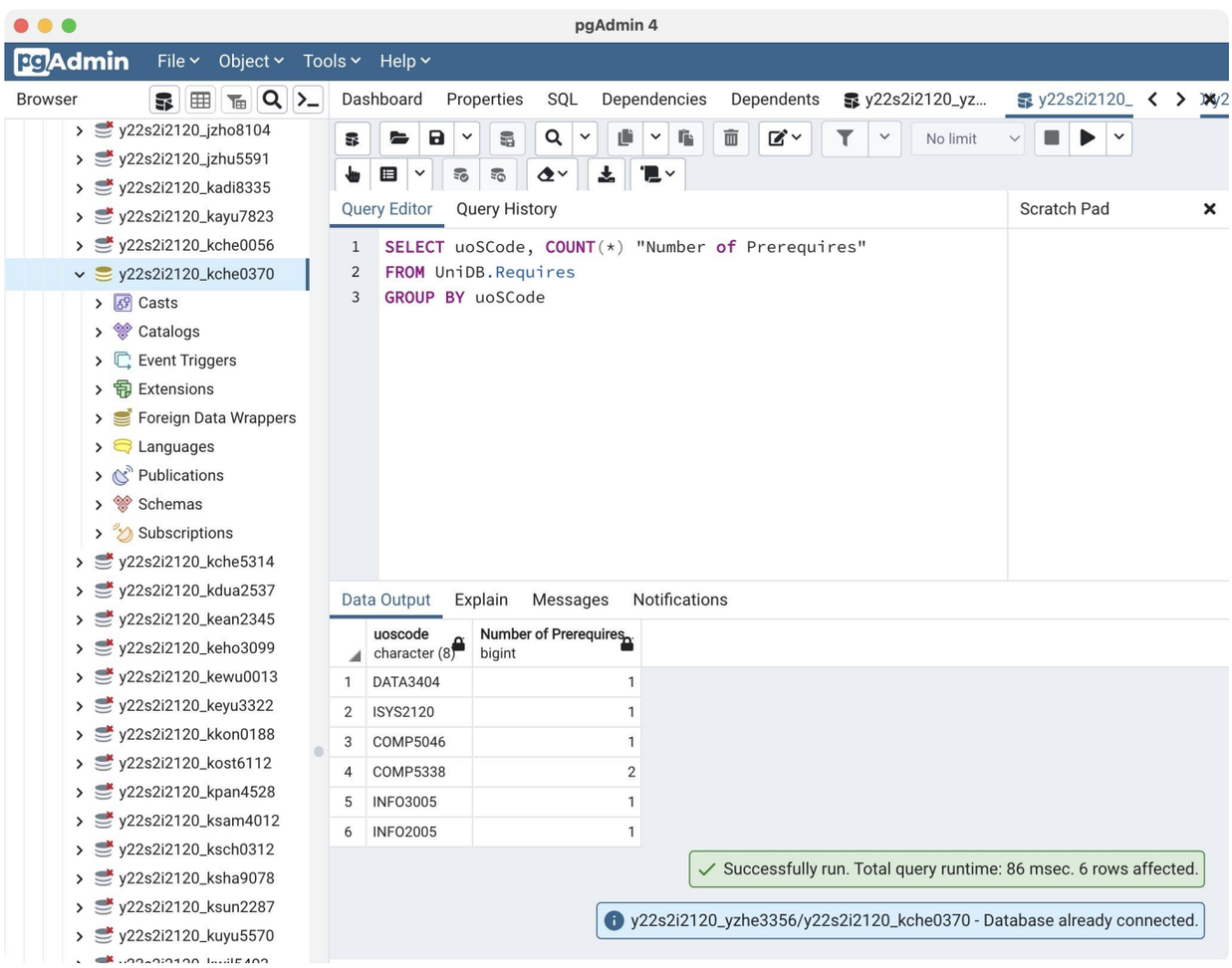
Department	Count
ABC	1
SIT	7

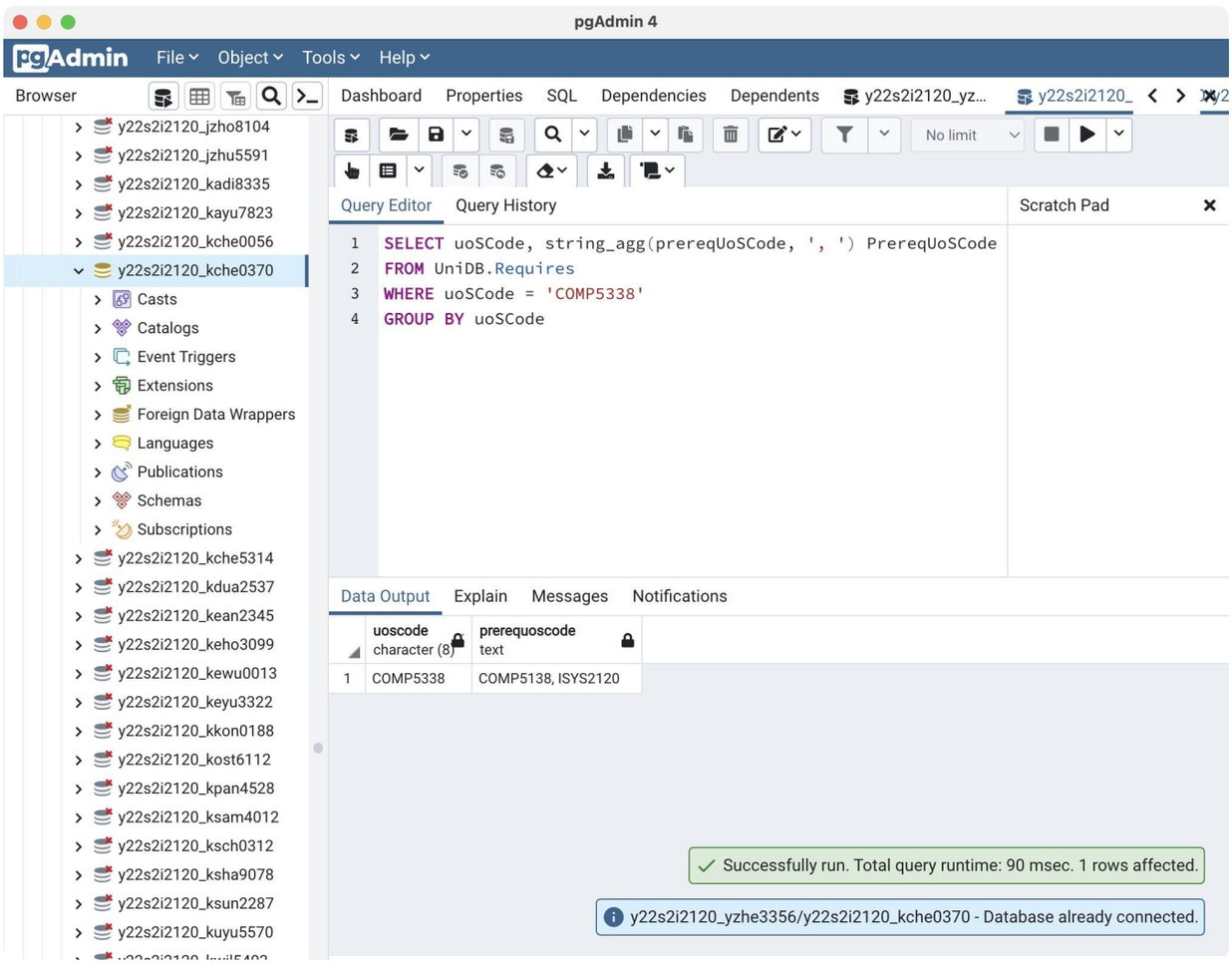
(The above two screenshots show that the added staff entry ‘Guy Smith’ is reflected in the list staff and department staff count pages, meaning they have affected the database)

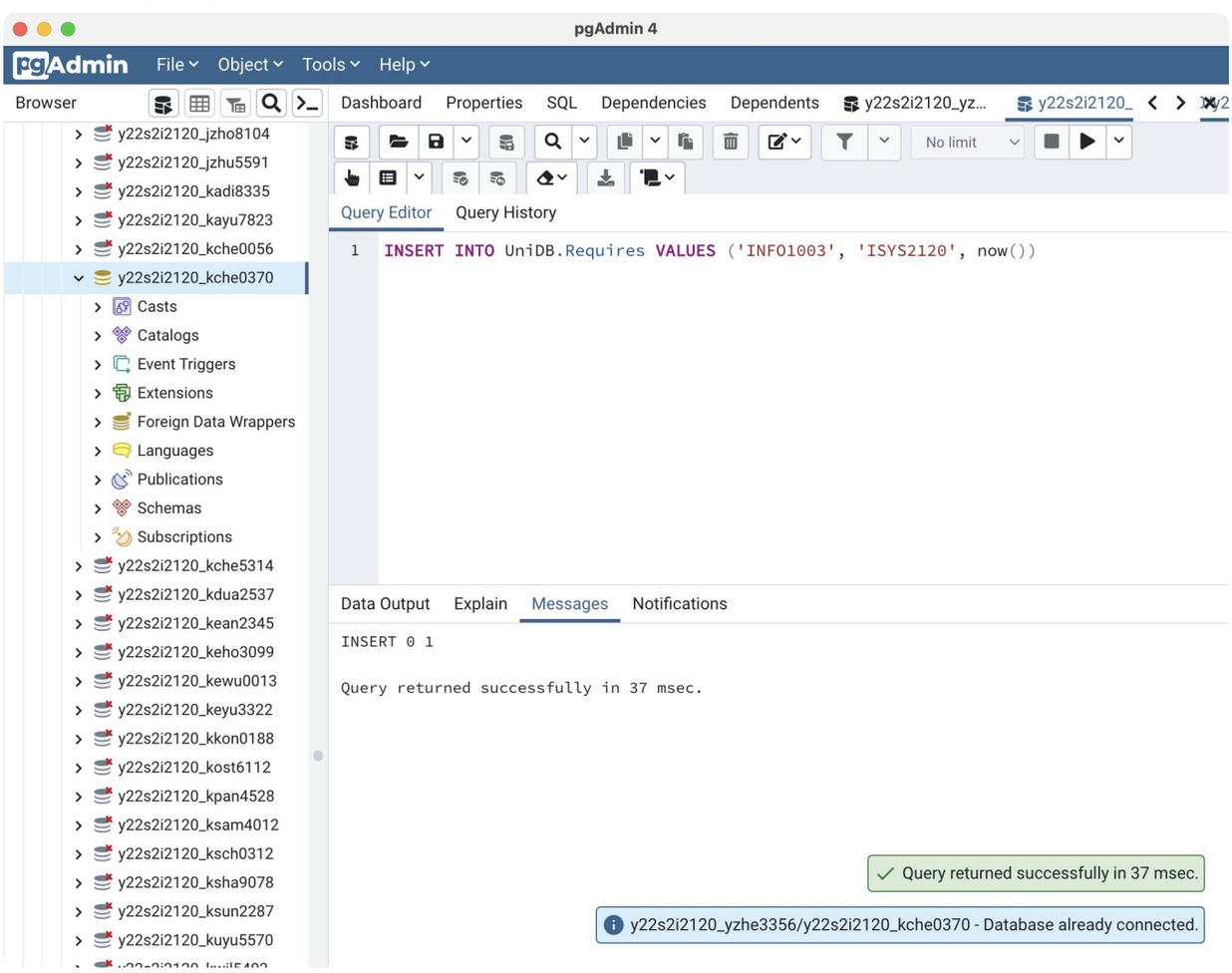
Prerequisites Tests

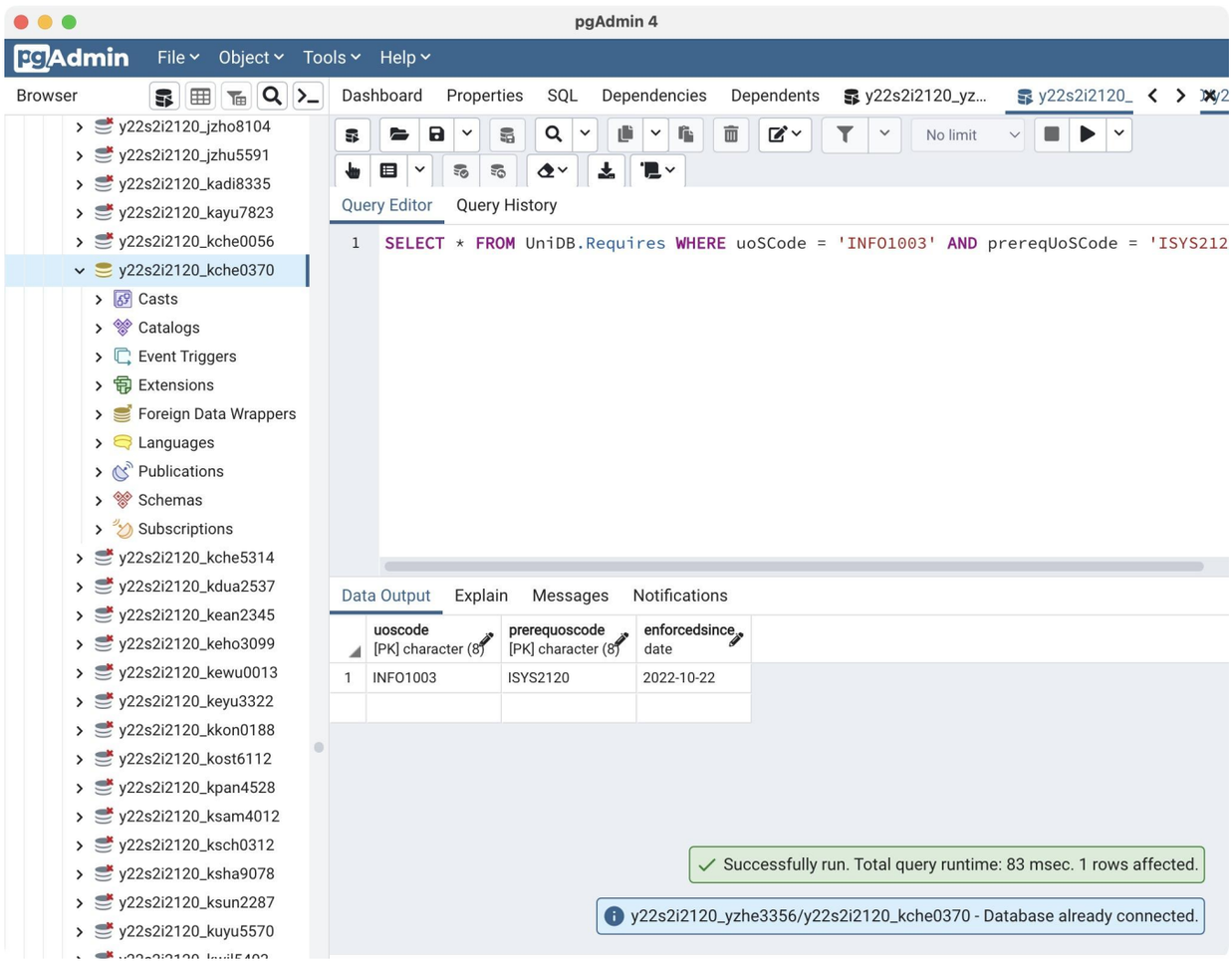
PgAdmin:

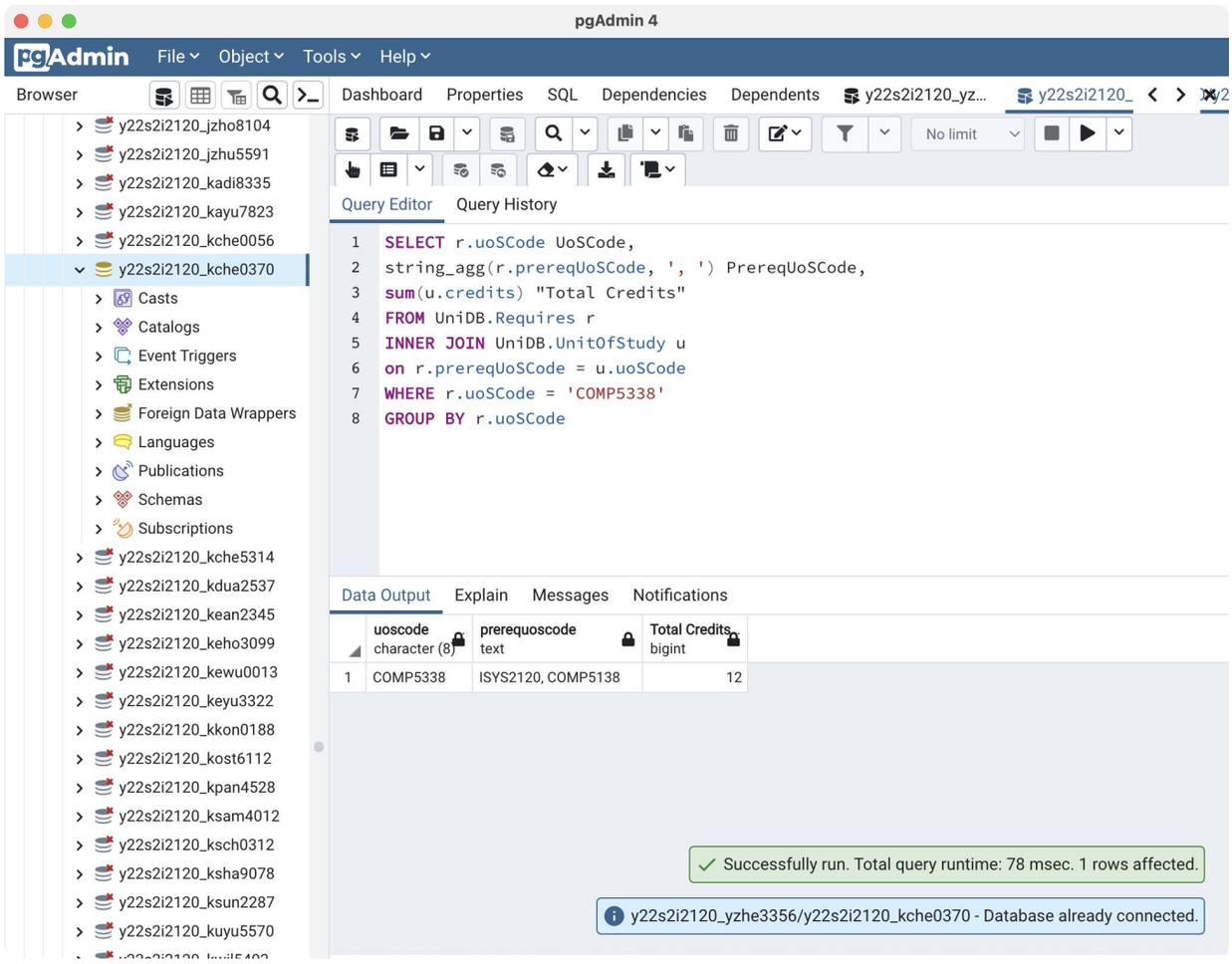





pgAdmin 4
pgAdmin
File
Object
Tools
Help
Browser
Dashboard
Properties
SQL
Dependencies
Dependents
y22s2i2120_yz...
y22s2i2120_
No limit
Query Editor
Query History
Scratch Pad
1 SELECT uoSCode, string_agg(prerequoSCode, ', ') PrereqUoSCode
2 FROM UniDB.Requires
3 WHERE uoSCode = 'COMP5338'
4 GROUP BY uoSCode
Data Output
Explain
Messages
Notifications
uoscode
character (8)
prerequoscode
text
1 COMP5338 COMP5138, ISYS2120
Successfully run. Total query runtime: 90 msec. 1 rows affected.
y22s2i2120_yzhe3356/y22s2i2120_kche0370 - Database already connected.
y22s2i2120_jzho8104
y22s2i2120_jzhu5591
y22s2i2120_kadi8335
y22s2i2120_kayu7823
y22s2i2120_kche0056
y22s2i2120_kche0370
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
y22s2i2120_kche5314
y22s2i2120_kdua2537
y22s2i2120_kean2345
y22s2i2120_keho3099
y22s2i2120_kewu0013
y22s2i2120_keyu3322
y22s2i2120_kkon0188
y22s2i2120_kost6112
y22s2i2120_kpan4528
y22s2i2120_ksam4012
y22s2i2120_ksch0312
y22s2i2120_ksha9078
y22s2i2120_ksun2287
y22s2i2120_kuyu5570


pgAdmin 4
pgAdmin
File
Object
Tools
Help
Browser
Dashboard
Properties
SQL
Dependencies
Dependents
y22s2i2120_yz...
y22s2i2120_
No limit
Query Editor
Query History
1 INSERT INTO UniDB.Requires VALUES ('INFO1003', 'ISYS2120', now())
Data Output
Explain
Messages
Notifications
INSERT 0 1
Query returned successfully in 37 msec.
Query returned successfully in 37 msec.
y22s2i2120_yzhe3356/y22s2i2120_kche0370 - Database already connected.
y22s2i2120_jzho8104
y22s2i2120_jzhu5591
y22s2i2120_kadi8335
y22s2i2120_kayu7823
y22s2i2120_kche0056
y22s2i2120_kche0370
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
y22s2i2120_kche5314
y22s2i2120_kdua2537
y22s2i2120_kean2345
y22s2i2120_keho3099
y22s2i2120_kewu0013
y22s2i2120_keyu3322
y22s2i2120_kkon0188
y22s2i2120_kost6112
y22s2i2120_kpan4528
y22s2i2120_ksam4012
y22s2i2120_ksch0312
y22s2i2120_ksha9078
y22s2i2120_ksun2287
y22s2i2120_kuyu5570


pgAdmin 4
File
Object
Tools
Help
Browser
Dashboard
Properties
SQL
Dependencies
Dependents
y22s2i2120_yz...
y22s2i2120_
No limit
Query Editor
Query History
1 SELECT * FROM UniDB.Requires WHERE uoSCode = 'INFO1003' AND prereqUoSCode = 'ISYS212
Data Output
Explain
Messages
Notifications
uoscode [PK] character (8)
prerequoscode [PK] character (8)
enforcedsince date
1 INFO1003 ISYS2120 2022-10-22
Successfully run. Total query runtime: 83 msec. 1 rows affected.
y22s2i2120_yzhe3356/y22s2i2120_kche0370 - Database already connected.
y22s2i2120_jzho8104
y22s2i2120_jzhu5591
y22s2i2120_kadi8335
y22s2i2120_kayu7823
y22s2i2120_kche0056
y22s2i2120_kche0370
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
y22s2i2120_kche5314
y22s2i2120_kdua2537
y22s2i2120_kean2345
y22s2i2120_keho3099
y22s2i2120_kewu0013
y22s2i2120_keyu3322
y22s2i2120_kkon0188
y22s2i2120_kost6112
y22s2i2120_kpan4528
y22s2i2120_ksam4012
y22s2i2120_ksch0312
y22s2i2120_ksha9078
y22s2i2120_ksun2287
y22s2i2120_kuyu5570


pgAdmin 4
File
Object
Tools
Help
Browser
Dashboard
Properties
SQL
Dependencies
Dependents
y22s2i2120_yz...
y22s2i2120_
No limit
Query Editor
Query History
1 SELECT r.uoSCode UoSCode,
2 string_agg(r.prereqUoSCode, ', ') PrereqUoSCode,
3 sum(u.credits) "Total Credits"
4 FROM UniDB.Requires r
5 INNER JOIN UniDB.UnitOfStudy u
6 on r.prereqUoSCode = u.uoSCode
7 WHERE r.uoSCode = 'COMP5338'
8 GROUP BY r.uoSCode
Data Output
Explain
Messages
Notifications
uoscode character (8)
prerequoscode text
Total Credits bigint
1 COMP5338 ISYS2120, COMP5138 12
Successfully run. Total query runtime: 78 msec. 1 rows affected.
y22s2i2120_yzhe3356/y22s2i2120_kche0370 - Database already connected.
y22s2i2120_jzho8104
y22s2i2120_jzhu5591
y22s2i2120_kadi8335
y22s2i2120_kayu7823
y22s2i2120_kche0056
y22s2i2120_kche0370
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas
Subscriptions
y22s2i2120_kche5314
y22s2i2120_kdua2537
y22s2i2120_kean2345
y22s2i2120_keho3099
y22s2i2120_kewu0013
y22s2i2120_keyu3322
y22s2i2120_kkon0188
y22s2i2120_kost6112
y22s2i2120_kpan4528
y22s2i2120_ksam4012
y22s2i2120_ksch0312
y22s2i2120_ksha9078
y22s2i2120_ksun2287
y22s2i2120_kuyu5570

Website:

Prerequisites pair

UOS Code	UOS Name	Prerequisite UOS Code	Prerequisite UOS Name	Enforeced Since
ISYS2120	Database Systems I	INFO1003	Introduction to IT	2002-01-01
DATA3404	Database Systems II	ISYS2120	Database Systems I	2004-11-01
COMP5046	Statistical Natural Language Processing	COMP5138	Database Management Systems	2006-11-01
COMP5338	Advanced Data Models	COMP5138	Database Management Systems	2004-01-01
COMP5338	Advanced Data Models	ISYS2120	Database Systems I	2004-01-01
INFO2005	Database Management Introductory	INFO1003	Introduction to IT	2002-01-01
INFO3005	Organisational Database Systems	INFO2005	Database Management Introductory	2002-01-01

Prerequisites report

UOS Code	Number of Prerequisites
DATA3404	1
ISYS2120	1
COMP5046	1
COMP5338	2
INFO3005	1
INFO2005	1

Search for prerequisites

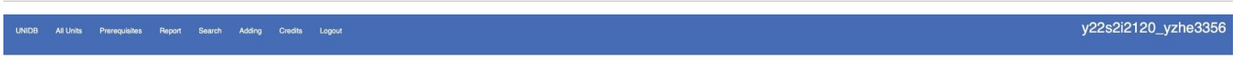
UoS Code

COMP5338

Check !

Prerequisites Table

UOS Code	Prerequisite units



Search for prerequisites

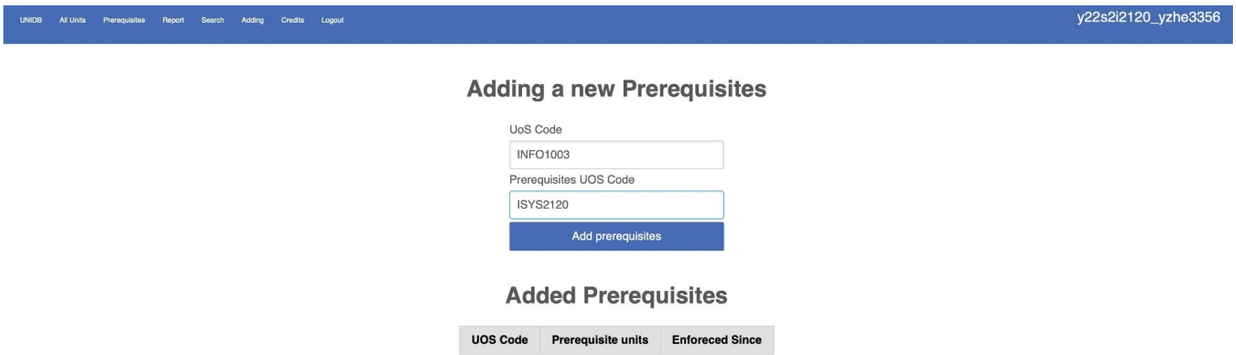
UoS Code

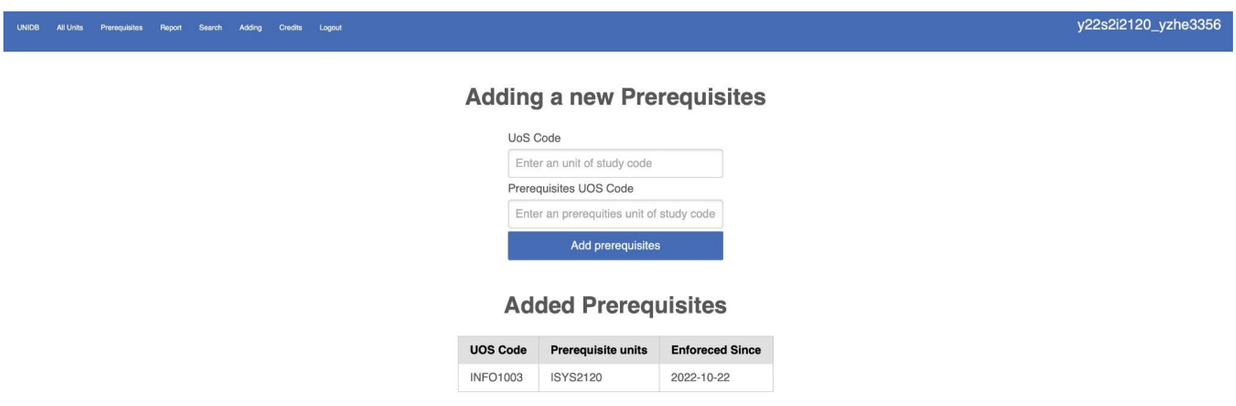
Enter an unit of study code

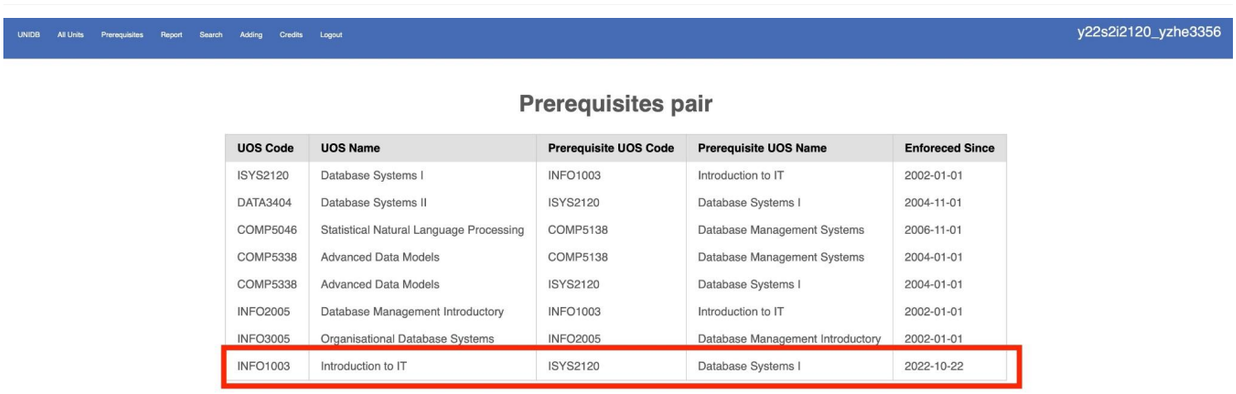
Check !

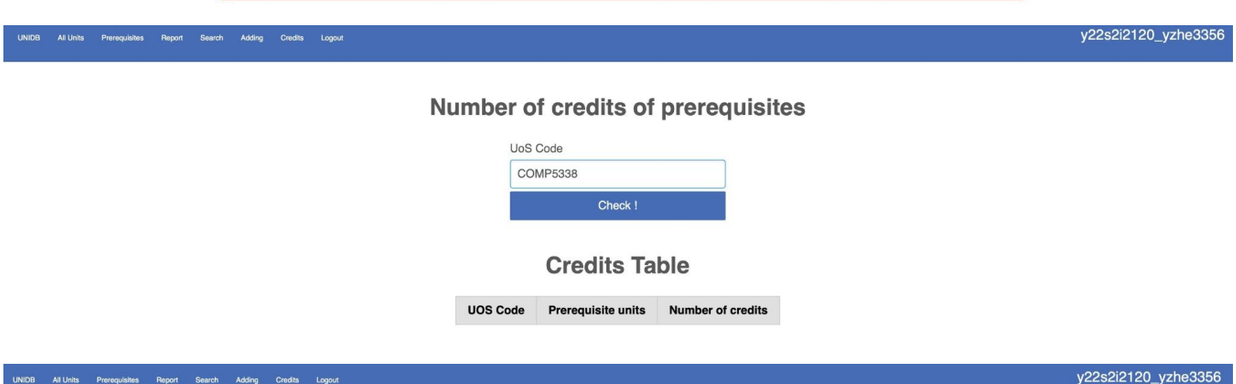
Prerequisites Table

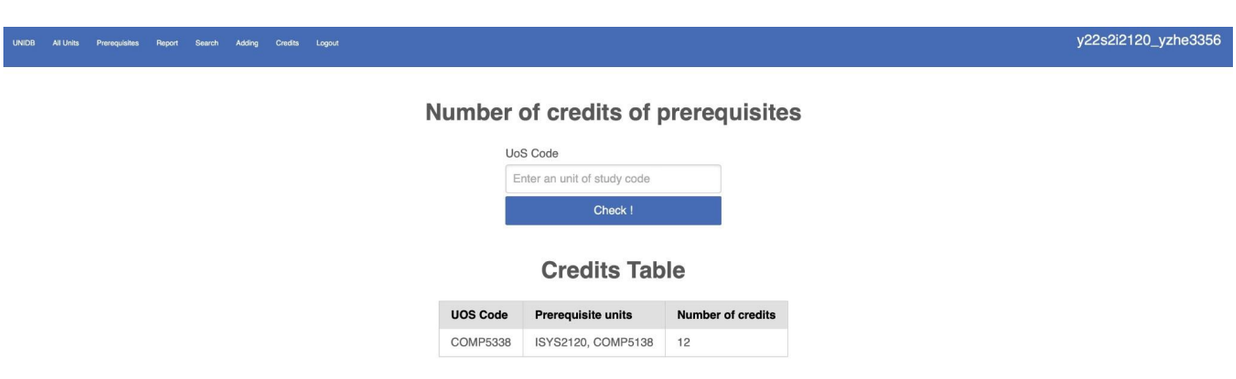
UOS Code	Prerequisite units
COMP5338	COMP5138, ISYS2120


UNIDB All Units Prerequisites Report Search Adding Credits Logout
y22s2i2120_yzhe3356
Adding a new Prerequisites
UoS Code
INFO1003
Prerequisites UOS Code
ISYS2120
Add prerequisites
Added Prerequisites
UOS Code
Prerequisite units
Enforeced Since


UNIDB All Units Prerequisites Report Search Adding Credits Logout
y22s2i2120_yzhe3356
Adding a new Prerequisites
UoS Code
Enter an unit of study code
Prerequisites UOS Code
Enter an prerequities unit of study code
Add prerequisites
Added Prerequisites
UOS Code
Prerequisite units
Enforeced Since
INFO1003
ISYS2120
2022-10-22


UNIDB All Units Prerequisites Report Search Adding Credits Logout
y22s2i2120_yzhe3356
Prerequisites pair
UOS Code
UOS Name
Prerequisite UOS Code
Prerequisite UOS Name
Enforeced Since
ISYS2120 Database Systems I INFO1003 Introduction to IT 2002-01-01
DATA3404 Database Systems II ISYS2120 Database Systems I 2004-11-01
COMP5046 Statistical Natural Language Processing COMP5138 Database Management Systems 2006-11-01
COMP5338 Advanced Data Models COMP5138 Database Management Systems 2004-01-01
COMP5338 Advanced Data Models ISYS2120 Database Systems I 2004-01-01
INFO2005 Database Management Introductory INFO1003 Introduction to IT 2002-01-01
INFO3005 Organisational Database Systems INFO2005 Database Management Introductory 2002-01-01
INFO1003 Introduction to IT ISYS2120 Database Systems I 2022-10-22


UNIDB All Units Prerequisites Report Search Adding Credits Logout
y22s2i2120_yzhe3356
Number of credits of prerequisites
UoS Code
COMP5338
Check !
Credits Table
UOS Code
Prerequisite units
Number of credits


UNIDB All Units Prerequisites Report Search Adding Credits Logout
y22s2i2120_yzhe3356
Number of credits of prerequisites
UoS Code
Enter an unit of study code
Check !
Credits Table
UOS Code
Prerequisite units
Number of credits
COMP5338
ISYS2120, COMP5138
12

Part 3 Security:

The Security Goal

1. Each Student and Academic Staff is the only person who can access that person's own account, and that person's own accounts only.
2. Sign out user can not retrieve any information from the database.
3. The developers can not do anything dangerous to the database, or leak sensitive user information from the database.

The Good Part

SQL Injection

```
        cur.execute(
            """
            insert into unidb.classroom (classroomId, seats, type)
            values (cast(%s as text), %s, cast(%s as text))
            """,
            (classroomId, seats, class_type),
        )
```

Using the client-side pg8080 embedded parameterized queries to prevent SQL Injection. This mechanism helped to achieve goal 1: the staff and students can not use SQL injection to access other person's accounts. Also, goal 2: sign-out users can not use SQL injection to access accounts unauthorizedly.

Login security

```
@app.route("/check-available-classrooms/start_time=<start_time>&end_time=<end_time>" ,
methods=["GET"])
def check_available_classrooms_api(start_time, end_time):
   if "logged_in" not in session or not session["logged_in"]:
       return redirect(url_for("login"))
```

The web server flask uses sessions to manage logins. A user that did not sign in will be redirected to the sign-in page by the server. This mechanism helped to achieve security goal 2: so now a user without sign-in can not access the information by typing the URL for a webpage containing information about the university.

Restricted permissions

```
revoke all on all tables in schema unidb from y22s2i2120_kuyu5570;

-- grant all on all tables in schema unidb to y22s2i2120_kuyu5570;

grant select on unidb.Student, unidb.Transcript, unidb.UnitOfStudy,
unidb.kuyu5570_classroombooking to y22s2i2120_kuyu5570;
grant select, insert on unidb.ClassRoom to y22s2i2120_kuyu5570;
```

Developers are only allowed to affect tables and rows that they have permission to. This prevents people from modifying tables they aren't allowed to, such as a developer doing a student function adding rows to the staff table, or an anonymous user seeing restricted information. This helped to achieve goal 3.

The Not-So-Good Part:

Passwords that are stored in the database are in plain text:

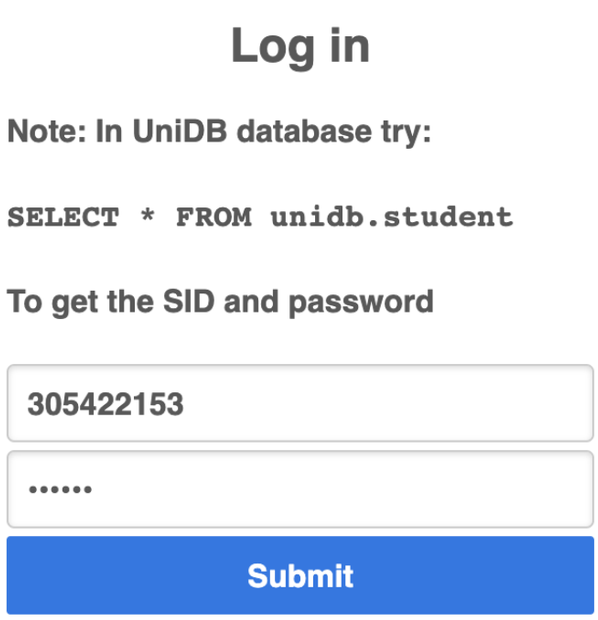
```
/* create the schema */
CREATE TABLE Student (
 studId        INTEGER,
 name          VARCHAR(20) NOT NULL,
 password      VARCHAR(10) NOT NULL,
 address       VARCHAR(50),
 PRIMARY KEY (studId)
);
```

```
/* add some students - the following data is completely arbitrary */
/* any similarities to actual students is purely accidential.     */
INSERT INTO Student VALUES (307088592, 'John Smith', 'Green', 'Newtown');
INSERT INTO Student VALUES (305422153, 'Sally Waters', 'Purple', 'Coogee');
INSERT INTO Student VALUES (305678453, 'Pauline Winters', 'Turkey', 'Bondi');
INSERT INTO Student VALUES (316424328, 'Matthew Long', 'Space', 'Camperdown');
INSERT INTO Student VALUES (309145324, 'Victoria Tan', 'Grapes', 'Maroubra');
INSERT INTO Student VALUES (309187546, 'Niang Jin Phan', 'Robot', 'Kingsford');
```

```
def check_login(sid, pwd):
   # Ask for the database connection, and get the cursor set up
   conn = database_connect()
   if conn is None:
       return None
   cur = conn.cursor()
   try:
       # Try executing the SQL and get from the database
       sql = """SELECT *
                FROM unidb.student
                WHERE studid=%s AND password=%s"""
       cur.execute(sql, (sid, pwd))
       r = cur.fetchone()  # Fetch the first row
       cur.close()  # Close the cursor
       conn.close()  # Close the connection to the db
       return r
   except Exception as e:
       # If there were any errors, return a NULL row printing an error to the debug
       print("Error Invalid Login")
       print(e)
   cur.close()  # Close the cursor
   conn.close()  # Close the connection to the db
   return None
```

Passwords that are stored in the database are in plain text. This creates a potential issue of password leakage if the database is breached. This damaged goal 2. Also, for any developer who has been granted “select” permission to this table (In this case, all group members have this access to this table to achieve the login function.), that developer can read all students’ passwords and potentially, steal account information from that student. This damaged goal 3.

No rate limiting on the login page:



Without rate limiting, the attacker can keep trying different combinations of SID and passwords at a very fast speed until one worked. This damaged the security goal 2.

No isolation on processes of Flask:

Open the webpage using two different browsers, i.e chrome, safari to emulate two different users using the website at the same time. First, we sign in as a student in chrome, then when we refresh the page on safari, the page on safari is now also signed in as the student that has signed in on chrome. This indicated that the Flask did not isolate the session information between different users on different browsers. This is ok if the web server only serves one user. But it is not realistic that a webserver only servers one user. This has defeated goals 1 and goal 2, as anyone can use the account that has been currently signed in on the server.

Security Conclusion:

Overall, the security of the website and database is not acceptable if the website needs to be deployed in a real-life situation. Store passwords in plain text and no isolation between Flask processes are two huge security risks that need to be improved before the security of this website reaches a minimum standard. To improve the password severity, store the password in a hashed + salted format. To solve the flask no isolation issue, consider using the multi-process module in python, and generate a new process for each access. Note the processes created by the multi-process module have their own memories (Variables such as session information are not shared between processes).

Part 4 Extension:

Completed by: Kuai Yu 500034918

Extension 1: List all booking information about classrooms:

The web page shows all classrooms that are being booked

Booked classrooms

Classroom ID	Start_time	End_time
BoschLT1	2021-05-01 08:00:00	2021-05-01 10:00:00
BoschLT2	2021-05-01 10:00:00	2021-05-01 12:00:00
BoschLT3	2021-05-01 12:00:00	2021-05-01 14:00:00
BoschLT4	2021-05-01 14:00:00	2021-05-01 16:00:00
CheLT1	2021-05-01 16:00:00	2021-05-01 18:00:00
CheLT2	2021-05-01 18:00:00	2021-05-01 20:00:00
CheLT3	2021-05-01 20:00:00	2021-05-01 22:00:00
CheLT4	2021-05-01 22:00:00	2021-05-02 00:00:00
CAR157	2021-05-02 00:00:00	2021-05-02 02:00:00
CAR159	2021-05-02 02:00:00	2021-05-02 04:00:00
CAR173	2021-05-02 04:00:00	2021-05-02 06:00:00
CAR175	2021-05-02 06:00:00	2021-05-02 08:00:00
CAR273	2021-05-02 08:00:00	2021-05-02 10:00:00

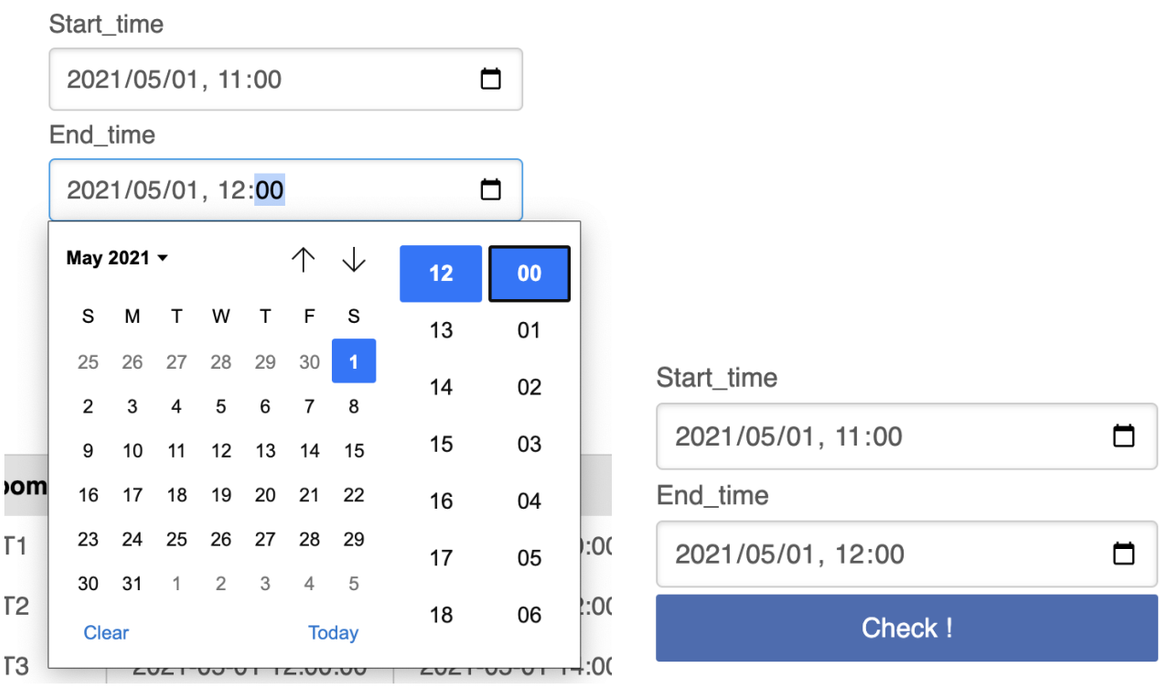
DDL statement that has been used to create the extra table called kuyu5570_classroombooking

```
CREATE Table unidb.kuyu5570_classroombooking
(
   bookingId  serial,
   classroomId varchar(8) references unidb.classroom(classroomId) deferrable,
   start_time timestamp not null,
   end_time timestamp not null,
   primary key (bookingId)
);

INSERT INTO unidb.kuyu5570_classroombooking (classroomId, start_time, end_time)
```

```
VALUES ('BoschLT1', '2021-05-01 08:00:00', '2021-05-01 10:00:00'),
       ('BoschLT2', '2021-05-01 10:00:00', '2021-05-01 12:00:00'),
       ('BoschLT3', '2021-05-01 12:00:00', '2021-05-01 14:00:00'),
       ('BoschLT4', '2021-05-01 14:00:00', '2021-05-01 16:00:00'),
       ('CheLT1', '2021-05-01 16:00:00', '2021-05-01 18:00:00'),
       ('CheLT2', '2021-05-01 18:00:00', '2021-05-01 20:00:00'),
       ('CheLT3', '2021-05-01 20:00:00', '2021-05-01 22:00:00'),
       ('CheLT4', '2021-05-01 22:00:00', '2021-05-02 00:00:00'),
       ('CAR157', '2021-05-02 00:00:00', '2021-05-02 02:00:00'),
       ('CAR159', '2021-05-02 02:00:00', '2021-05-02 04:00:00'),
       ('CAR173', '2021-05-02 04:00:00', '2021-05-02 06:00:00'),
       ('CAR175', '2021-05-02 06:00:00', '2021-05-02 08:00:00'),
       ('CAR273', '2021-05-02 08:00:00', '2021-05-02 10:00:00');

revoke all on all tables in schema unidb from y22s2i2120_kuyu5570;

-- grant all on all tables in schema unidb to y22s2i2120_kuyu5570;

grant select on unidb.Student, unidb.Transcript, unidb.UnitOfStudy,
unidb.kuyu5570_classroombooking to y22s2i2120_kuyu5570;
grant select, insert on unidb.ClassRoom to y22s2i2120_kuyu5570;
```

Extension 2: List all classrooms that are available (have not been booked) in a given period.:

The web drop-down selection of the start and end time. Will return all available classrooms in this period.



Completed by: Kevin Zheng 510233600

Extention 1: Search the number of credits of the prerequisites needed for a UoS

Number of credits of prerequisites

UoS Code

COMP5338

Check !

Credits Table

UOS Code	Prerequisite units	Number of credits
COMP5338	ISYS2120, COMP5138	12