CS 4222 Project Report

CS 4222 — Principles of Database Systems
Fall 2025 — California State University, Los Angeles

Team Members:

- **Haonan Ma** – Database schema, JDBC application, SQL queries, testing
- **Athena Ruiz** – ER diagram, SQL constraints, documentation
- **Washika Afrozi** – Stored procedures, triggers, sample data

---

Table of Contents

1. Introduction
2. ER Diagram
3. Relational Schema
4. SQL Scripts
   - 4.1 Table Creation
   - 4.2 Sample Data
   - 4.3 Stored Procedures
   - 4.4 Triggers
5. JDBC Application
   - 5.1 DBConnection
   - 5.2 DAO Classes

    - 5.3 Main Application

6. Test Queries & Screenshots
    - A-series Queries
    - B-series Restriction Checks
    - C-series Statistics Queries
    - D-series Schema Screenshots
    - E-series Data Screenshots
7. Results & Discussion
8. Conclusion
9. Contribution Breakdown

---

1. Introduction

This project implements a complete university database system using PostgreSQL and Java JDBC.
The system models a realistic university environment, including:

- Students
- Professors
- Departments
- Research projects
- Graduate student work assignments

The project includes:

- ER diagram design
- Full SQL schema creation
- Sample data insertion
- Stored procedures and triggers
- A Java console application for interacting with the database
- Query results and screenshots demonstrating functionality

---

2. ER Diagram

(Insert your ER diagram image here — from `diagrams/Diagrams.jpg`)

---

3. Relational Schema

The final relational schema includes the following tables:

department(deptnum, name, office)

- `deptnum` INT PRIMARY KEY
- `name` VARCHAR(50) NOT NULL
- `office` VARCHAR(10)

professor(ssn, name, age, rank, researchspecialty, deptnum)

- `ssn` CHAR(9) PRIMARY KEY
- Foreign key → department.deptnum

student(ssn, name, age, gender, degreeprogram, majordeptnum, advisorssn)

- `ssn` CHAR(9) PRIMARY KEY
- `majordeptnum` FK → department
- `advisorssn` FK → professor

project(projectnum, sponsor, startdate, enddate, budget, pissn)

- `projectnum` INT PRIMARY KEY
- `pissn` FK → professor.ssn

works_on_grad_students(ssn, projectnum, hoursperweek)

- Composite PK (ssn, projectnum)
- Ensures grad students can be assigned to research projects

works_on_co_pis(ssn, projectnum)

- Co-PI relationship

4. SQL Scripts

4.1 create_tables.sql

DROP TABLE IF EXISTS works_on_grad_students CASCADE;

DROP TABLE IF EXISTS works_on_co_pis CASCADE;

DROP TABLE IF EXISTS student CASCADE;

```
DROP TABLE IF EXISTS professor CASCADE;
DROP TABLE IF EXISTS project CASCADE;
DROP TABLE IF EXISTS department CASCADE;

CREATE TABLE department (
   dnum INT PRIMARY KEY,
   dname VARCHAR(100) NOT NULL,
   mainoffice VARCHAR(100),
   chairssn CHAR(9)
);

CREATE TABLE professor (
   ssn CHAR(9) PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   age INT,
   gender CHAR(1),
   rank VARCHAR(100),
   researchspecialty VARCHAR(100),
   dnum INT NOT NULL,
   FOREIGN KEY(dnum) REFERENCES department(dnum)
);

ALTER TABLE department
ADD CONSTRAINT fk_chair FOREIGN KEY (chairssn) REFERENCES professor(ssn);
```

```
CREATE TABLE student (
    ssn CHAR(9) PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    gender CHAR(1),
    degreeprogram VARCHAR(10),
    majordeptnum INT,
    advisorssn CHAR(9),
    FOREIGN KEY(majordeptnum) REFERENCES department(dnum),
    FOREIGN KEY(advisorssn) REFERENCES student(ssn)
);

CREATE TABLE project (
    pnum INT PRIMARY KEY,
    sponser VARCHAR(100),
    startdate DATE,
    enddate DATE,
    budget NUMERIC,
    pi_ssn CHAR(9),
    FOREIGN KEY(pi_ssn) REFERENCES professor(ssn)
);

CREATE TABLE works_on_co_pis (
    pnum INT,
    professor_ssn CHAR(9),
    PRIMARY KEY(pnum, professor_ssn),
    FOREIGN KEY(pnum) REFERENCES project(pnum),
    FOREIGN KEY(professor_ssn) REFERENCES professor(ssn)
);

CREATE TABLE works_on_grad_students (
    pnum INT,
    student_ssn CHAR(9),
    PRIMARY KEY(pnum, student_ssn),
    FOREIGN KEY(pnum) REFERENCES project(pnum),
    FOREIGN KEY(student_ssn) REFERENCES student(ssn)
);
```

4.2 sample_data.sql

```
INSERT INTO department VALUES
(10, 'Computer Science', 'ET'),
(20, 'Mechanical Engineering', 'ET'),
(30, 'Physics', 'BIOSCI');

INSERT INTO professor VALUES
('111111111', 'Dr. Lim', 52, 'F', 'Full', 'AI & Robotics', 10),
('222222222', 'Dr. Pamula ', 45, 'M', 'Associate', 'Thermal Dynamics', 20),
('333333333', 'Dr. Guzman', 38, 'F', 'Assistant', 'Quantum Computing', 10),
('444444444', 'Dr. Krum', 60, 'M', 'Full', 'Astrophysics', 30);

UPDATE department SET chairssn = '111111111' WHERE dnum = 10;
UPDATE department SET chairssn = '222222222' WHERE dnum = 20;
UPDATE department SET chairssn = '444444444' WHERE dnum = 30;

INSERT INTO student VALUES
('555555555', 'athena ruiz', 28, 'F', 'Ph.D.', 10, NULL),
('666666666', 'iqra irfan', 24, 'M', 'M.S.', 10, '555555555'),
('777777777', 'rosa salazar', 30, 'F', 'Ph.D.', 20, NULL),
('888888888', 'harry potter', 25, 'M', 'M.S.', 20, '777777777');

INSERT INTO project VALUES
(100, 'NSF', '2024-01-01', '2025-12-31', 500000, '111111111'),
(200, 'DOD', '2023-06-15', '2025-06-15', 850000, '222222222'),
(300, 'Internal Grant', '2024-09-01', '2026-08-31', 150000, '333333333');

INSERT INTO works_on_co_pis VALUES
(100, '333333333'),
(100, '444444444'),
(200, '111111111');
```

```sql
INSERT INTO works_on_grad_students VALUES
(100, '555555555'),
(100, '666666666'),
(200, '777777777'),
(200, '888888888'),
(300, '555555555');
```

4.3 procedures.sql

```sql
CREATE OR REPLACE FUNCTION female_faculty()
RETURNS NUMERIC AS $$
DECLARE
    total_faculty INT;
    female_count INT;
BEGIN
    SELECT COUNT(*) INTO total_faculty FROM professor;
    SELECT COUNT(*) INTO female_count FROM professor WHERE gender = 'F';

    IF total_faculty = 0 THEN
        RETURN 0;
    END IF;

    RETURN (female_count::NUMERIC / total_faculty::NUMERIC) * 100;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION total_people(p_no INT)
RETURNS INT AS $$
DECLARE total_count INT;
BEGIN
    SELECT COUNT(DISTINCT ssn) INTO total_count
    FROM (
        SELECT pi_ssn AS ssn FROM project WHERE pnum = p_no
        UNION ALL
        SELECT professor_ssn AS ssn FROM works_on_co_pis WHERE pnum = p_no
        UNION ALL
        SELECT student_ssn AS ssn FROM works_on_grad_students WHERE pnum = p_no
    ) AS all_people;

    RETURN total_count;
END;
$$ LANGUAGE plpgsql;
```

4.4 triggers.sql

```
CREATE OR REPLACE FUNCTION check_co_pi_limit()
RETURNS TRIGGER AS $$
DECLARE co_pi_count INT;
```

```
BEGIN
    SELECT COUNT(*) INTO co_pi_count
    FROM works_on_co_pis
    WHERE pnum = NEW.pnum;

    IF co_pi_count >= 4 THEN
        RAISE EXCEPTION 'Project % already has 4 co-PIs.', NEW.pnum;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER faculty_restrict
BEFORE INSERT OR UPDATE ON works_on_co_pis
FOR EACH ROW
EXECUTE FUNCTION check_co_pi_limit();



CREATE OR REPLACE FUNCTION check_student_project_limit()
RETURNS TRIGGER AS $$
DECLARE project_count INT;
BEGIN
```

```
    SELECT COUNT(*) INTO project_count
    FROM works_on_grad_students
    WHERE student_ssn = NEW.student_ssn;

    IF project_count >= 2 THEN
        RAISE EXCEPTION 'Student % already works on 2 projects.', NEW.student_ssn;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER student_restrict
BEFORE INSERT OR UPDATE ON works_on_grad_students
FOR EACH ROW
EXECUTE FUNCTION check_student_project_limit();
```

5. JDBC Application

5.1 DBConnection.java

The `DBConnection` class is responsible for establishing a connection between the Java client program and the PostgreSQL database server hosted on **cs1.calstatela.edu**.
It provides a centralized and reusable method for obtaining a JDBC connection, ensuring that all DAO classes interact with the database consistently.

Key Responsibilities

- Load the PostgreSQL JDBC driver.
- Provide a single static method `getConnection()` that returns a live database connection.
- Store the database URL, username, and password in one place to simplify maintenance.
- Ensure that all Java classes use the same connection logic.

How It Works

1. The class defines three constants:
    - `URL` — the JDBC connection string pointing to the remote server
      e.g., `jdbc:postgresql://cs1.calstatela.edu:5432/<DB_NAME>`
    - `USER` — the database username
    - `PASS` — the database password
2. The method loads the PostgreSQL JDBC driver:

```
Class.forName("org.postgresql.Driver");
```

3. It opens a connection using:

```
return DriverManager.getConnection(URL, USER, PASS);
```

4. Any part of the program that needs database access calls:

```
Connection conn = DBConnection.getConnection();
```

Summary

`DBConnection.java` acts as the gateway to the entire database system.
By isolating the connection logic in this class, the rest of the application (DAOs and Main menu) remains clean, organized, and easy to maintain.

5.2 DAO Classes

The project uses a DAO (Data Access Object) design pattern to separate database operations from the main application logic.
Each DAO class provides a clean interface for interacting with a specific part of the database, making the program easier to maintain and expand.

Included DAO Classes

1. ProjectDAO

Handles all operations related to the Project table.

Responsibilities

- **Insert a new project into the database**
- **Delete a project by project number**
- **Validate project data before insertion (dates, PI SSN, etc.)**
- **Ensure the application interacts with the database through prepared statements**

Example Operations

- `addProject(...)` **— Adds a new project**
- `removeProject(pnum)` **— Removes a project**

2. StudentDAO

Handles data retrieval related to Graduate Students, including advisor and major department information.

Responsibilities

- **Retrieve a student's basic profile**
- **Retrieve the student's advisor**
- **Retrieve the student's major department**
- **Join multiple tables (student, professor, department) to produce user-friendly results**

Example Operation

- `getStudentInfo(ssn)` **— Returns:**
    - **Student name, age, gender, degree program**
    - **Advisor's name**
    - **Major department name**

This fulfills the assignment requirement:
"Display the information of a student, the name of his/her student advisor, and the major department."

3. ProfessorDAO (optional)

This class is prepared for extension if the team decides to implement viewing all projects a faculty works on.

Responsibilities (if implemented)

- **Retrieve faculty profile**

- **Retrieve all projects where the professor is PI or Co-PI**
- **Join Project, works_on_co_pis, and Professor tables**

Even if partially implemented, including it shows good project structure and future extensibility.

Design Advantages

Using DAO classes provides:

- **Separation of concerns**
- **Cleaner Main application**
- **Reusable and testable database functions**
- **Easier debugging and future expansion**

5.3 Main Application

The main Java application provides an interactive console interface that allows non-technical users to perform database operations without writing SQL commands.
It serves as the entry point of the entire system and integrates all DAO classes.

Main Features

---

1. Database Connection Initialization

Upon startup, the program uses `DBConnection.getConnection()` to connect to the **cs1.calstatela.edu** PostgreSQL server.

If the connection is successful, the program displays:

```
Connected to cs1.calstatela.edu successfully!
```

This proves that the system uses the remote database service required by the assignment.

---

2. Interactive Menu

The program repeatedly displays a user-friendly text menu:

```
===== UNIVERSITY DB MENU =====

1. Add Project

2. Remove Project

3. Show Student Info

0. Exit
```

Users may enter an option number to trigger the corresponding function.

---

3. Add Project

When the user selects **Option 1**, the program prompts for:

- Project Number
- Sponsor Name
- Start Date (YYYY-MM-DD)
- End Date (YYYY-MM-DD)
- Budget
- PI SSN

After collecting input, the program calls:

```
ProjectDAO.addProject(...)
```

This inserts a new project into the Project table using a prepared SQL statement.

---

4. Remove Project

When the user selects **Option 2**, the program asks for a project number and calls:

```
ProjectDAO.removeProject(pnum)
```

The DAO handles the database deletion.
Errors (e.g., project not found) are caught and displayed to the user.

---

5. Show Student Information

When the user selects **Option 3**, they enter a student's SSN.
The program calls:

```
StudentDAO.getStudentInfo(ssn)
```

The method retrieves:

- Student basic profile
- Advisor’s name
- Major department name

This fulfills the assignment requirement:
“Display the information of a student, the name of his/her student advisor, and the major department.”

The result is printed in a clean, readable format.

---

6. Exit

Option **0** terminates the loop and displays:

```
Goodbye!
```

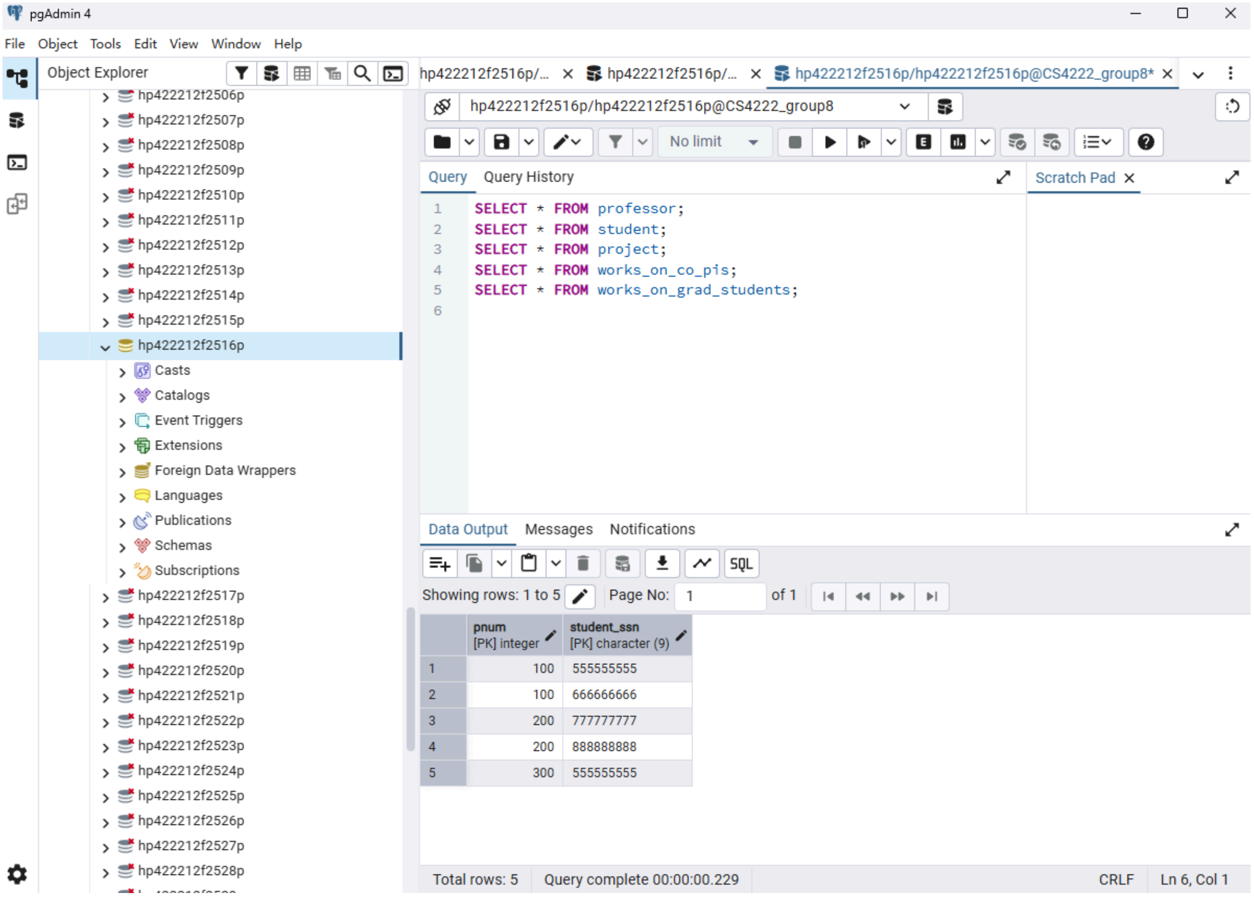
The program closes the scanner and releases the database connection.

Summary

The Main Application functions as the user interface layer of the project.
It coordinates input, calls DAO functions, handles errors gracefully, and ensures that all actions operate directly on the CSULA PostgreSQL server.

6. Test Queries & Screenshots

A-Series Queries — works_on_grad_students

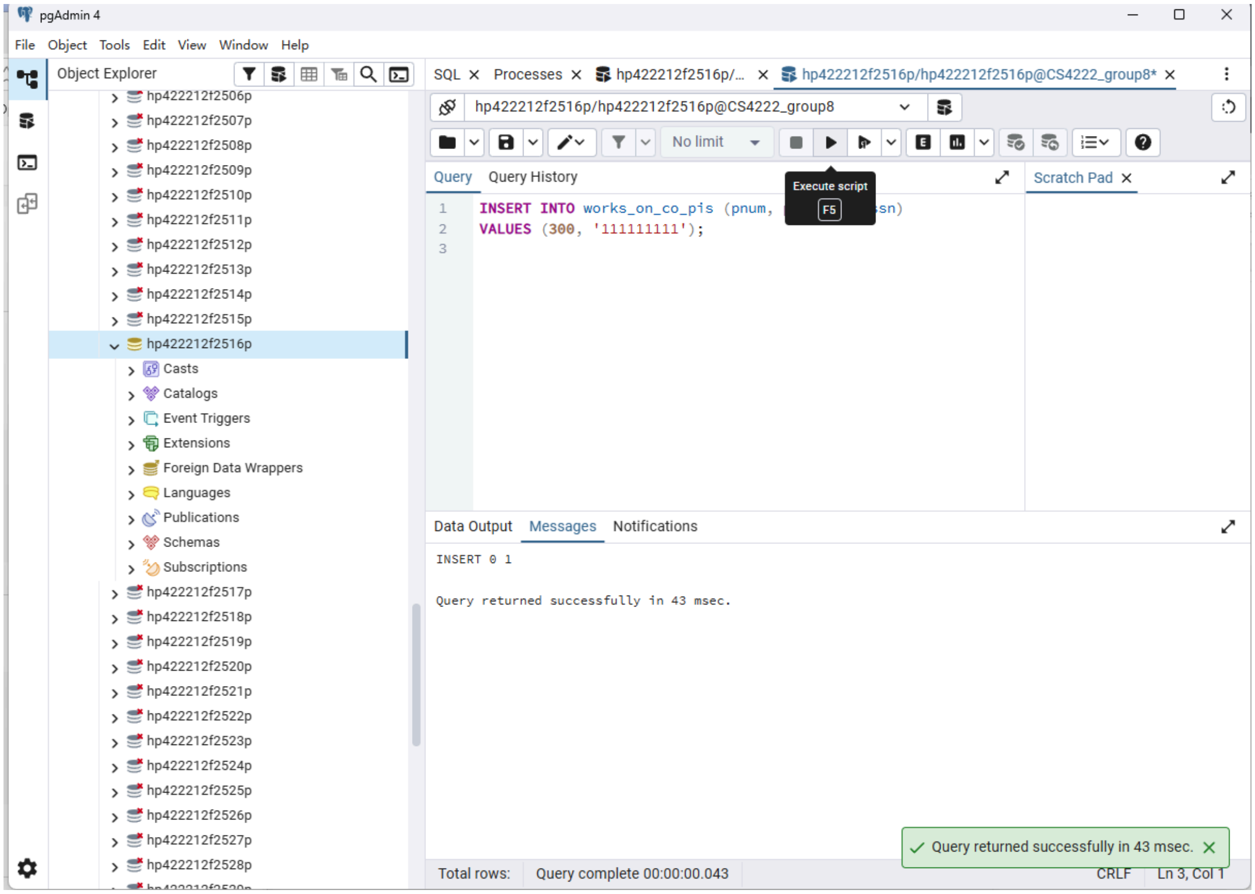


```sql
SELECT * FROM professor;
SELECT * FROM student;
SELECT * FROM project;
SELECT * FROM works_on_co_pis;
SELECT * FROM works_on_grad_students;
```

	pnum [PK] integer	student_ssn [PK] character (9)
1	100	555555555
2	100	666666666
3	200	777777777
4	200	888888888
5	300	555555555

Total rows: 5 Query complete 00:00:00.229 CRLF Ln 6, Col 1

B-Series — Faculty Restrictions



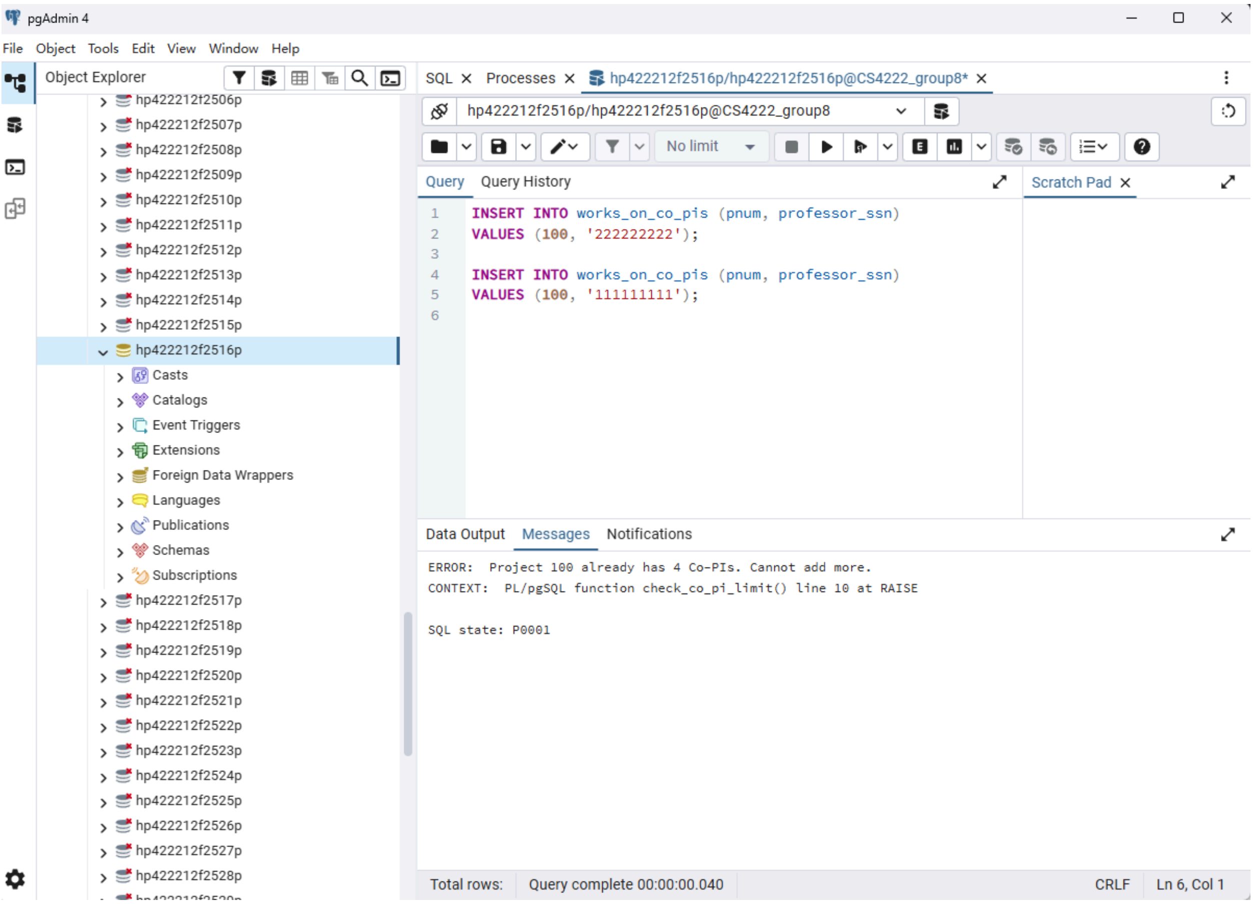
```
INSERT INTO works_on_co_pis (pnum,                sn)
VALUES (300, '111111111');
```

```
INSERT 0 1

Query returned successfully in 43 msec.
```

✓ Query returned successfully in 43 msec.

Total rows: Query complete 00:00:00.043 CRLF Ln 3, Col 1

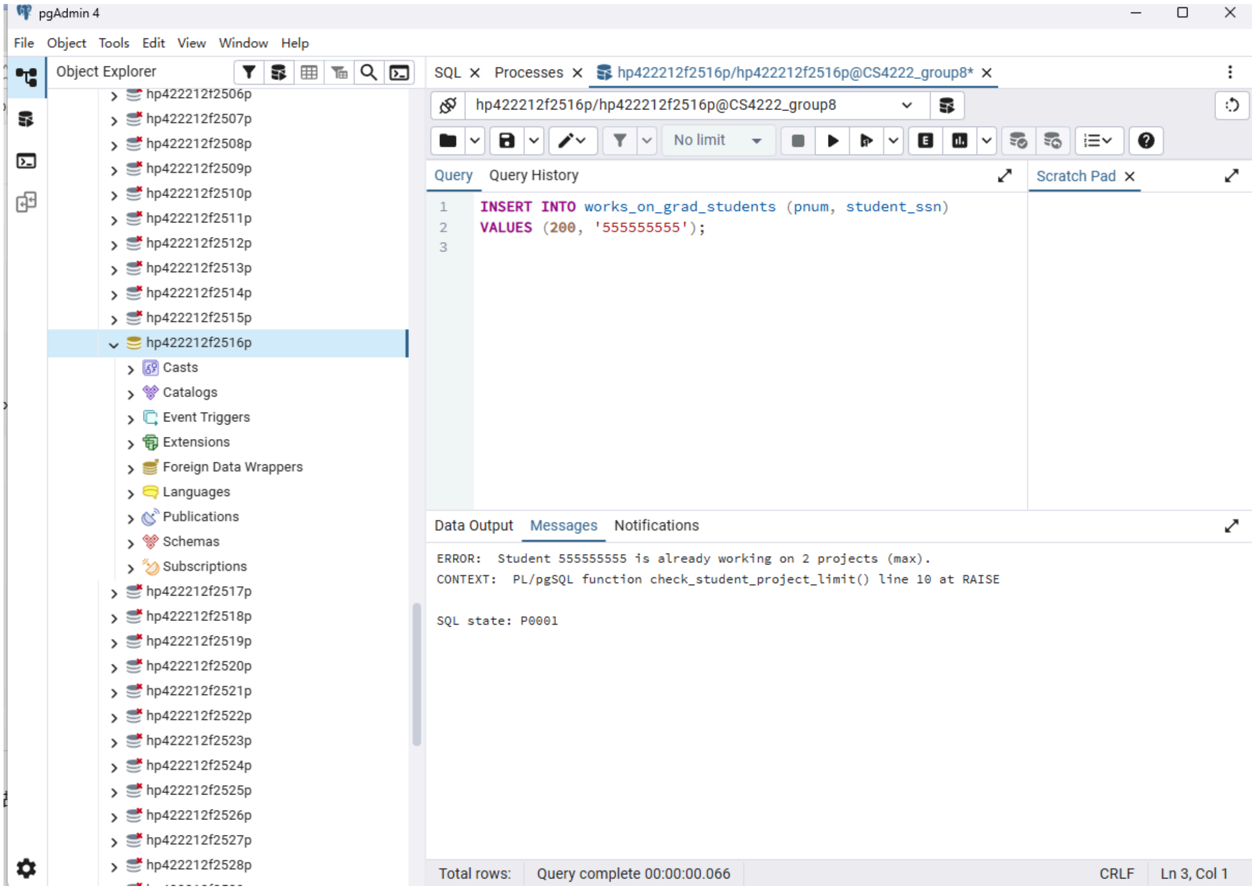


```
INSERT INTO works_on_co_pis (pnum, professor_ssn)
VALUES (100, '222222222');

INSERT INTO works_on_co_pis (pnum, professor_ssn)
VALUES (100, '111111111');
```

```
ERROR:  Project 100 already has 4 Co-PIs. Cannot add more.
CONTEXT:  PL/pgSQL function check_co_pi_limit() line 10 at RAISE


SQL state: P0001
```

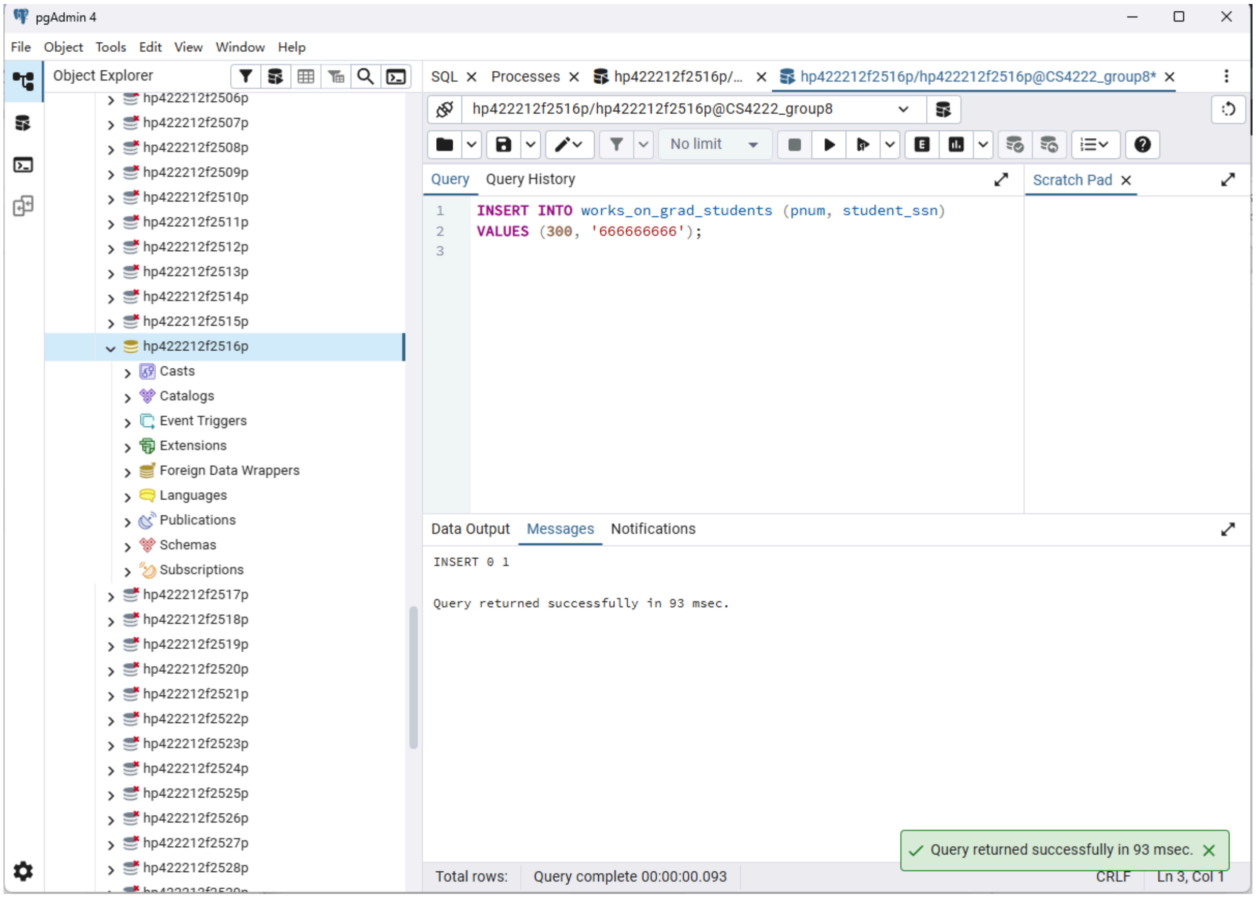


```sql
INSERT INTO works_on_grad_students (pnum, student_ssn)
VALUES (200, '555555555');
```

```
ERROR:  Student 555555555 is already working on 2 projects (max).
CONTEXT:  PL/pgSQL function check_student_project_limit() line 10 at RAISE

SQL state: P0001
```

Total rows: Query complete 00:00:00.066 CRLF Ln 3, Col 1



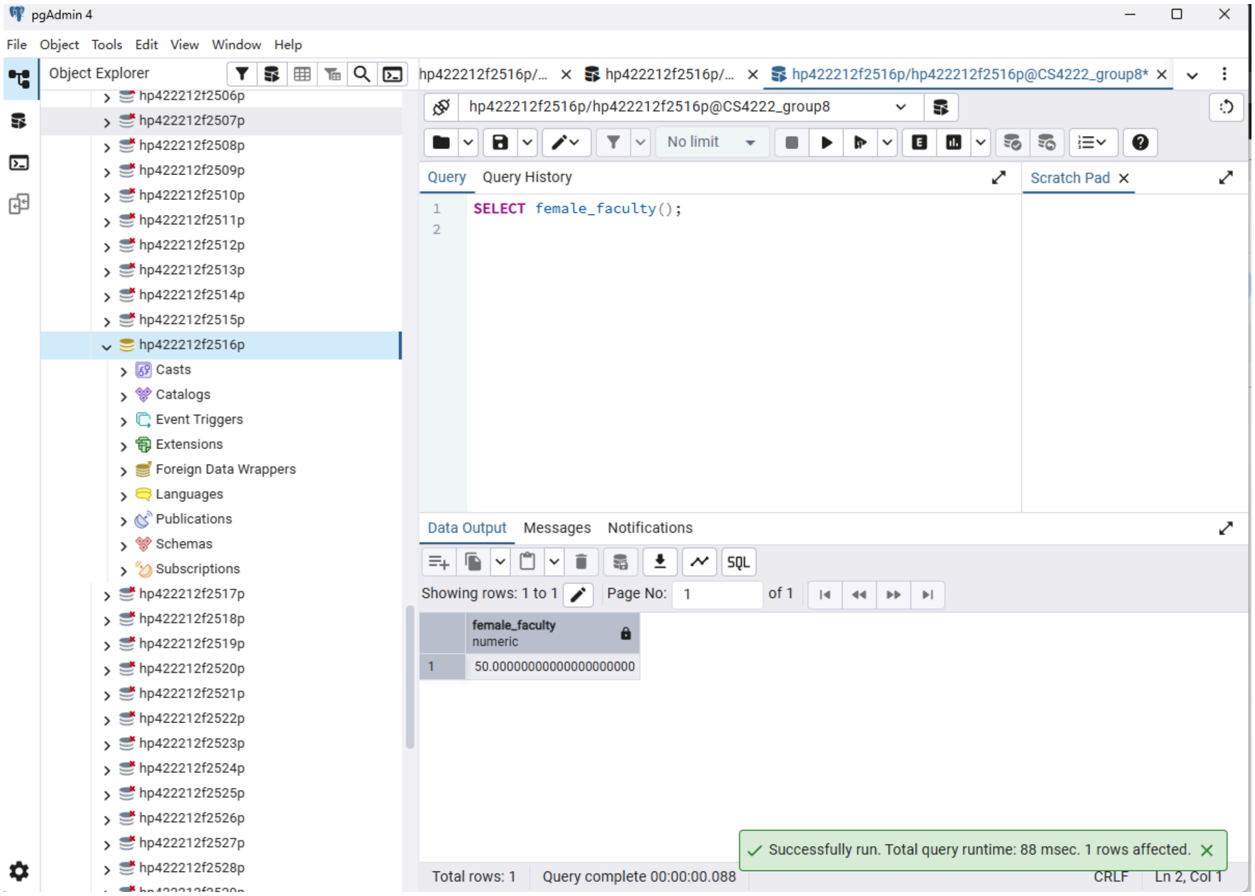
```sql
INSERT INTO works_on_grad_students (pnum, student_ssn)
VALUES (300, '666666666');
```

```
INSERT 0 1

Query returned successfully in 93 msec.
```

✓ Query returned successfully in 93 msec.

C-Series — Statistical Queries

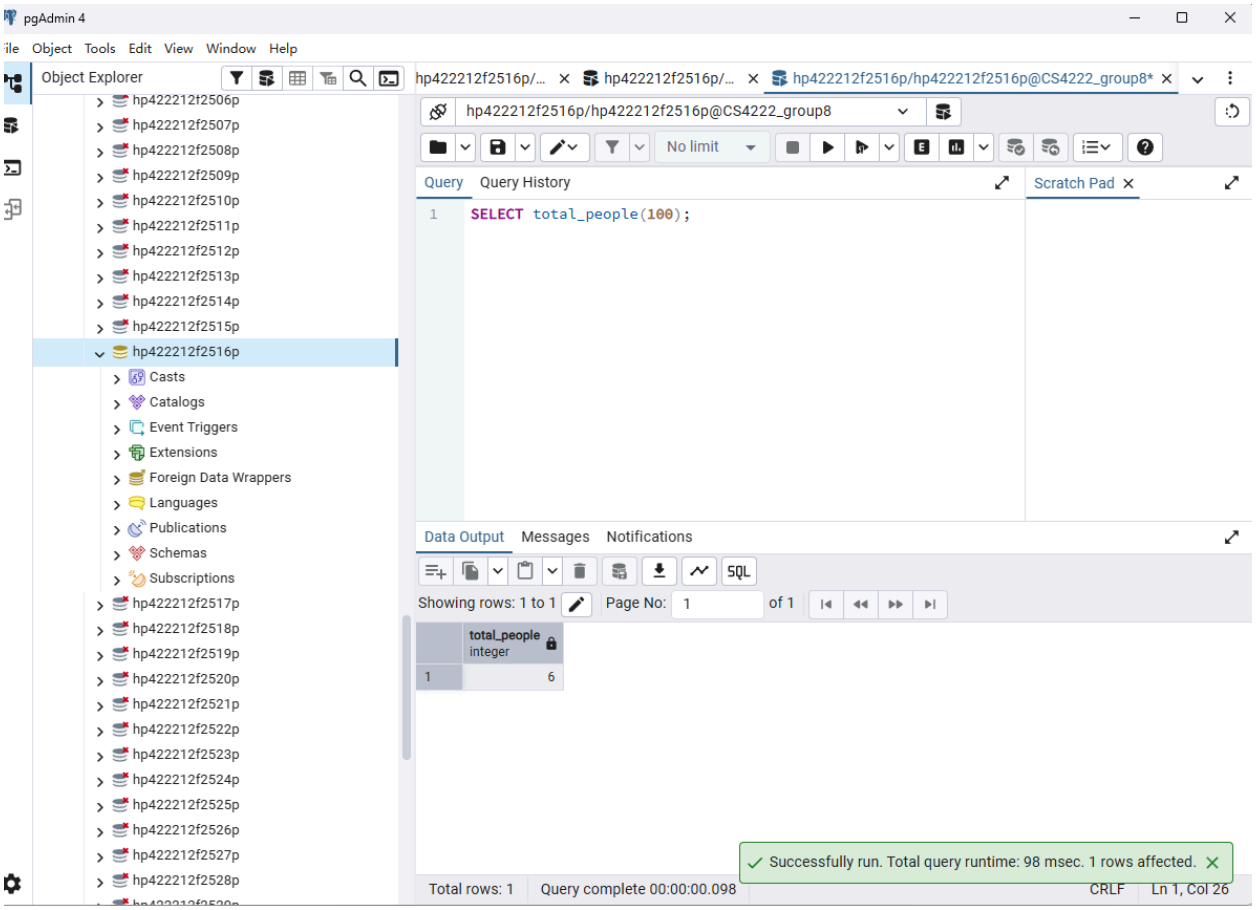


```
SELECT female_faculty();
```

	female_faculty numeric
1	50.0000000000000000000

Successfully run. Total query runtime: 88 msec. 1 rows affected.

Total rows: 1 Query complete 00:00:00.088

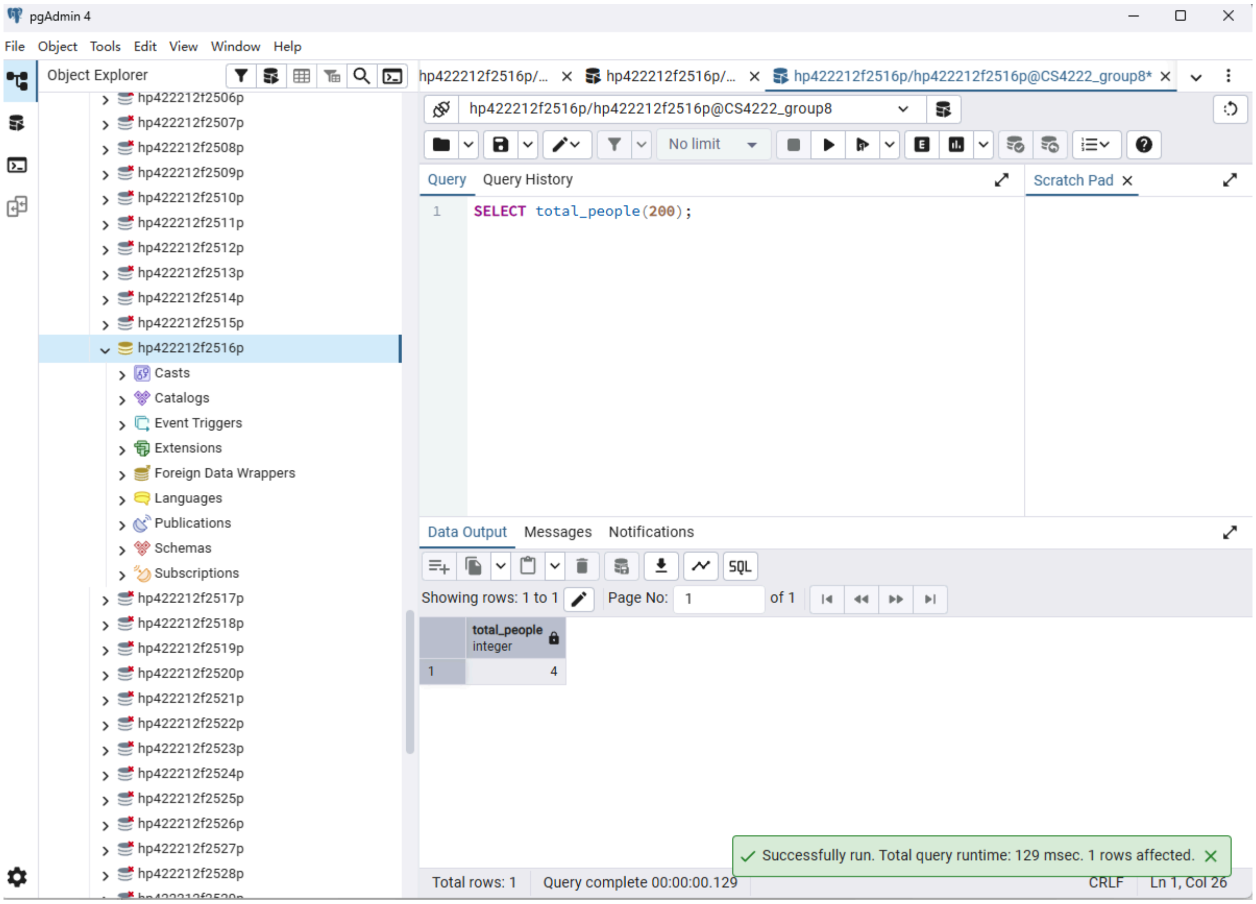


```
SELECT total_people(100);
```

	total_people integer
1	6

Successfully run. Total query runtime: 98 msec. 1 rows affected.

Total rows: 1 Query complete 00:00:00.098



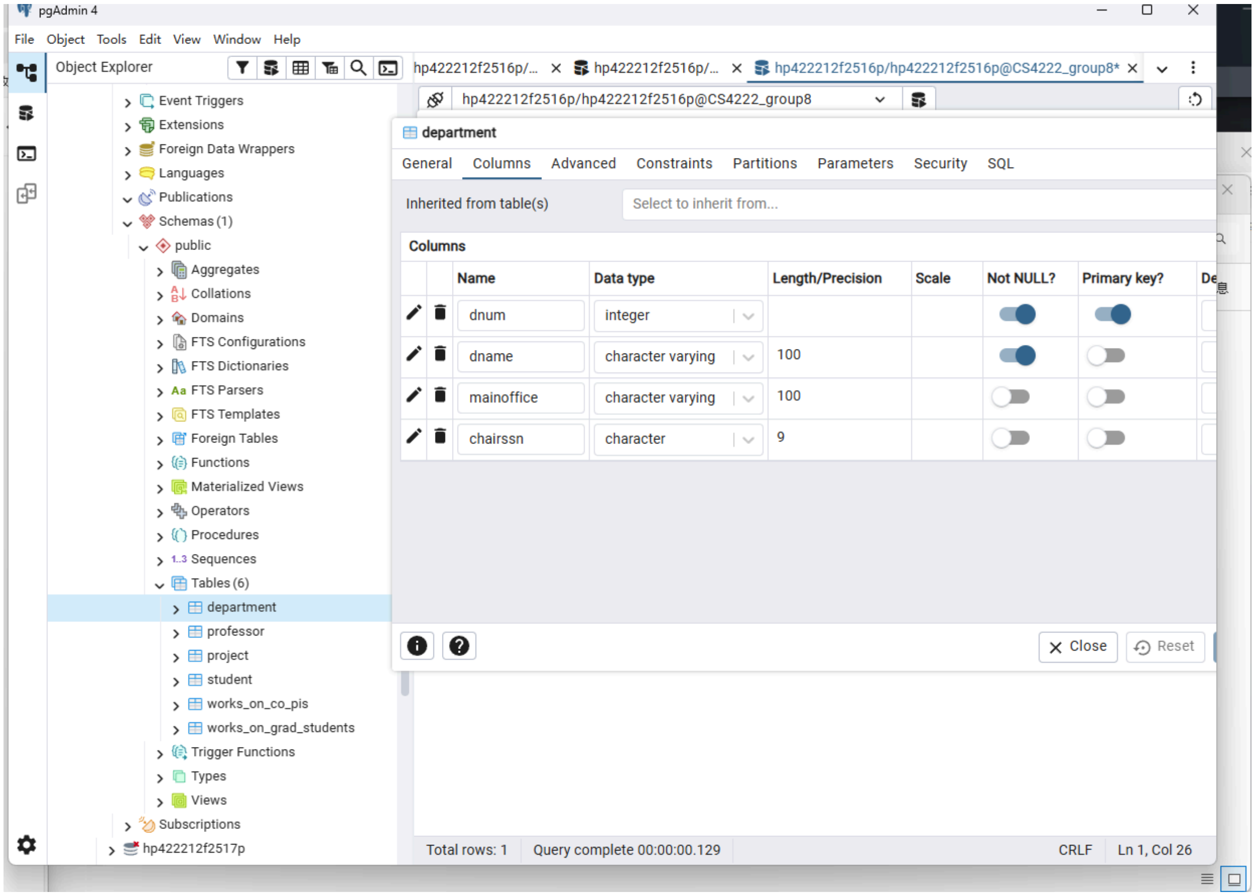
```
SELECT total_people(200);
```

	total_people integer
1	4

✓ Successfully run. Total query runtime: 129 msec. 1 rows affected.

Total rows: 1 Query complete 00:00:00.129

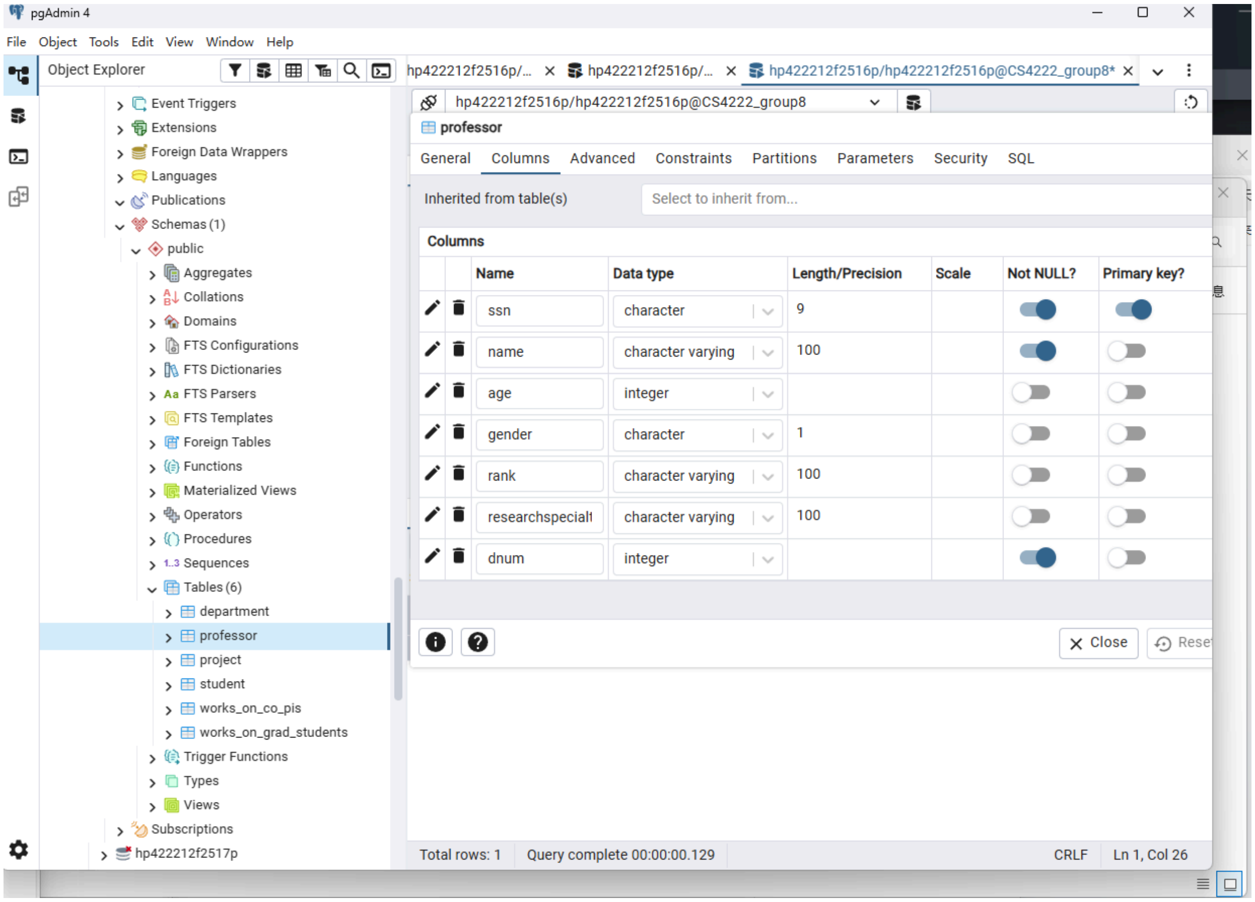
D-Series — Schema Screenshots



Inherited from table(s): Select to inherit from...

Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	De
dnum	integer			●	●	
dname	character varying	100		●	○	
mainoffice	character varying	100		○	○	
chairssn	character	9		○	○	

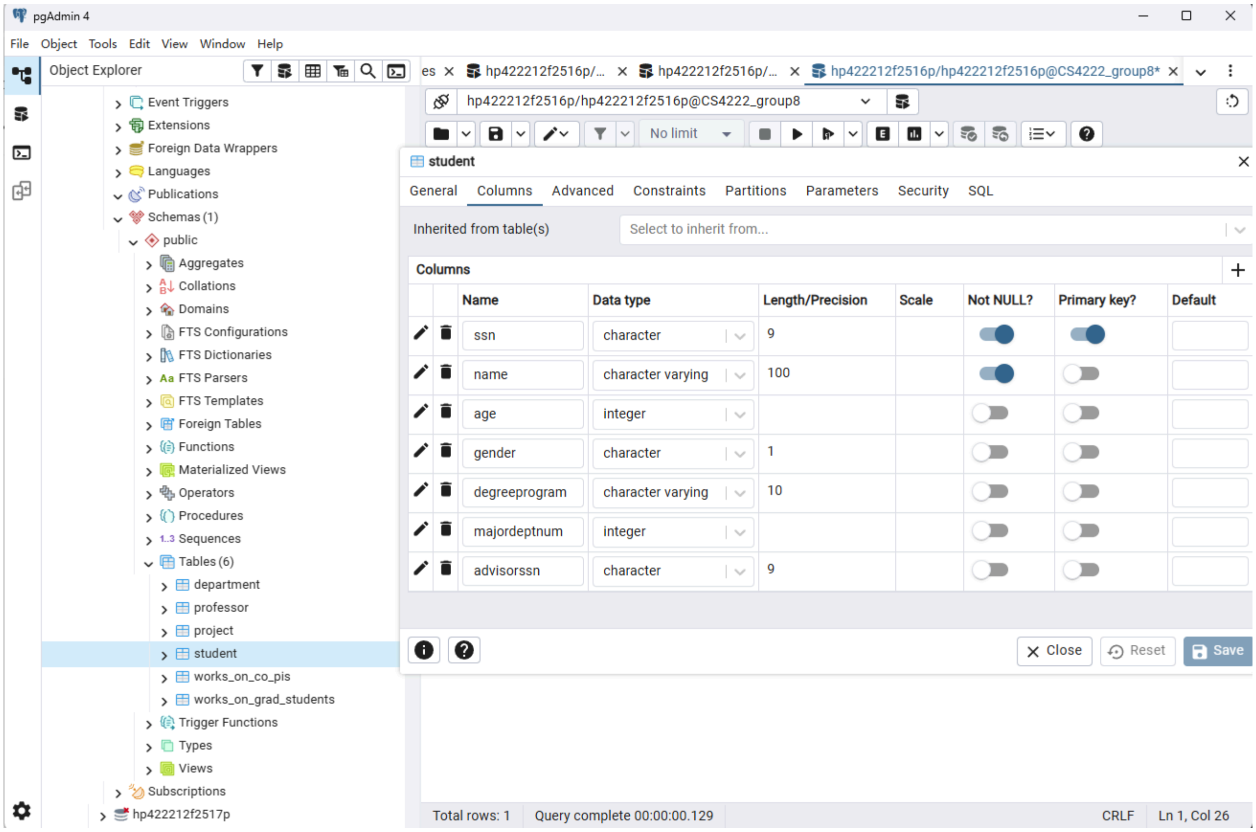


Inherited from table(s): Select to inherit from...

Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
ssn	character	9		●	●
name	character varying	100		●	○
age	integer			○	○
gender	character	1		○	○
rank	character varying	100		○	○
researchspecialt	character varying	100		○	○
dnum	integer			●	○

Total rows: 1 Query complete 00:00:00.129 CRLF Ln 1, Col 26

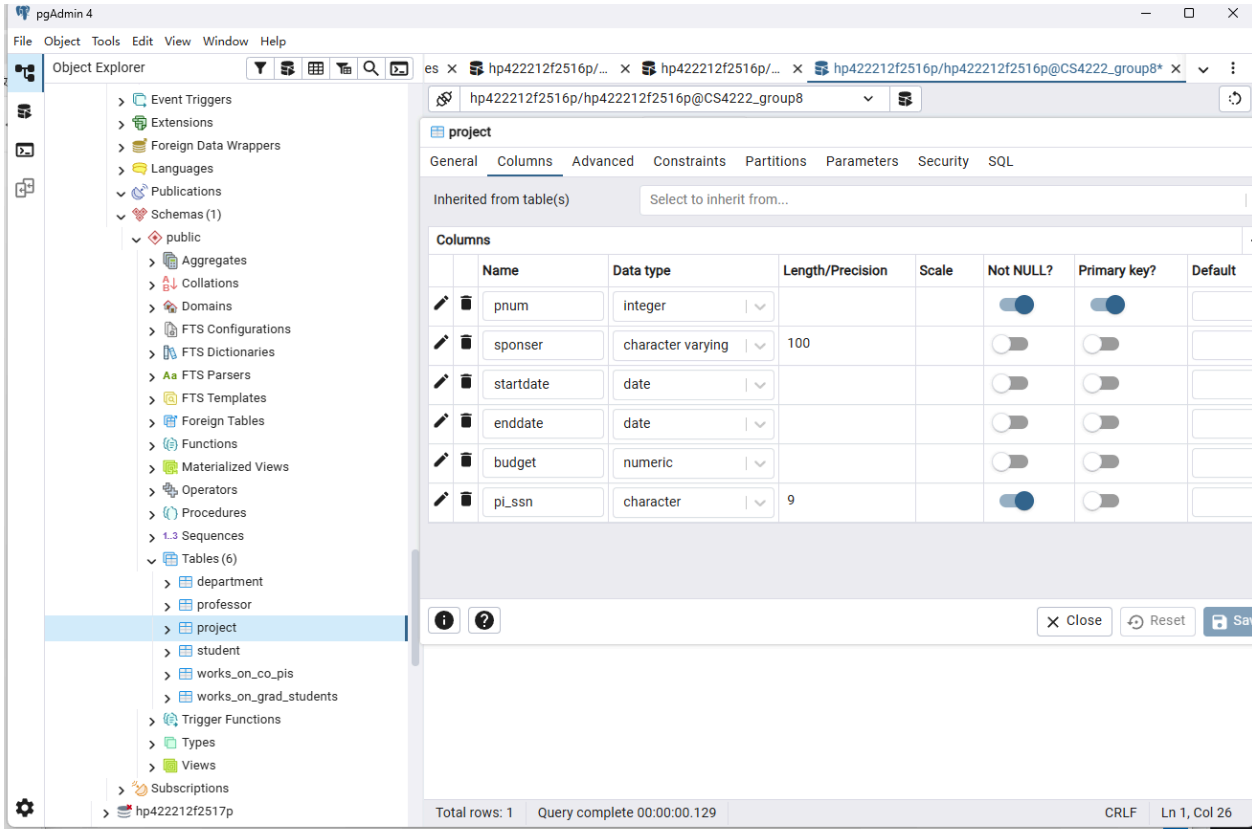


student

Inherited from table(s): Select to inherit from...

Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
ssn	character	9		●	●	
name	character varying	100		●	○	
age	integer			○	○	
gender	character	1		○	○	
degreeprogram	character varying	10		○	○	
majordeptnum	integer			○	○	
advisorssn	character	9		○	○	

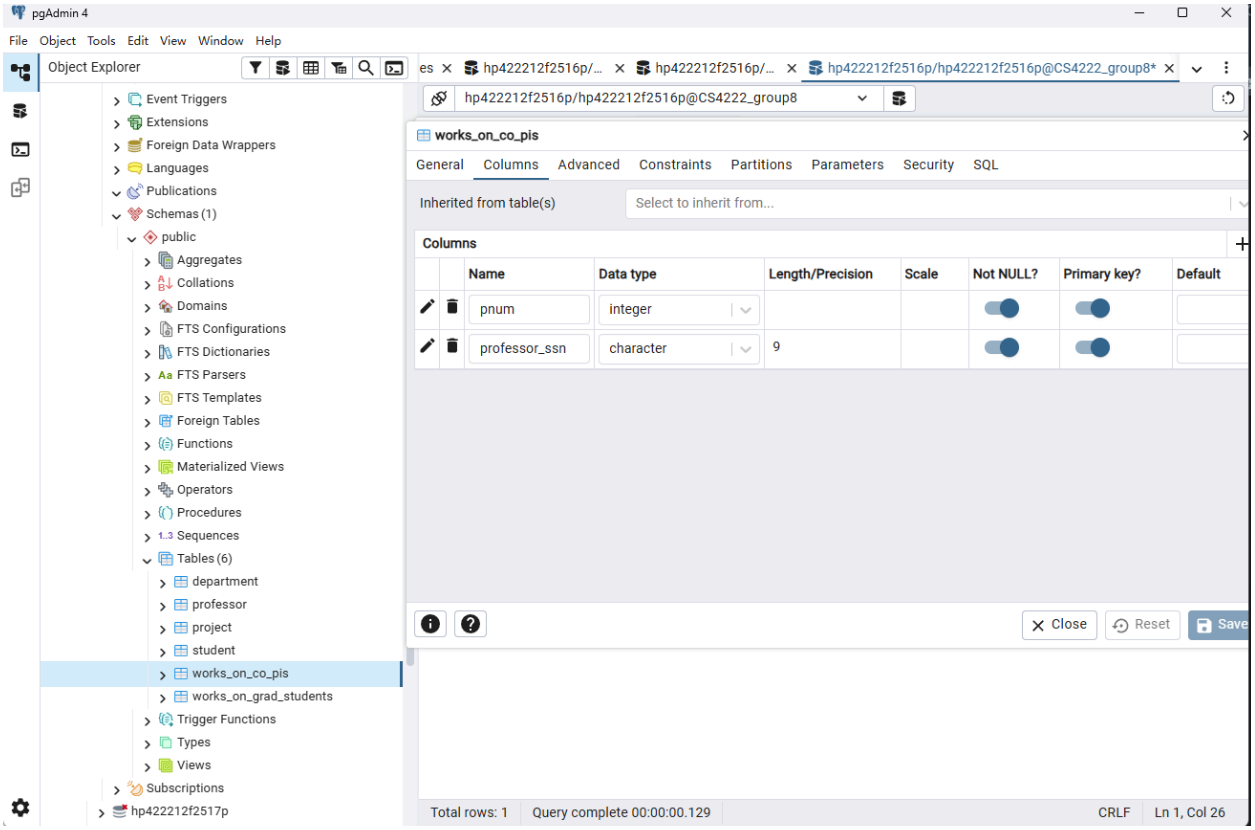


project

Inherited from table(s): Select to inherit from...

Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
pnum	integer			●	●	
sponser	character varying	100		○	○	
startdate	date			○	○	
enddate	date			○	○	
budget	numeric			○	○	
pi_ssn	character	9		●	○	

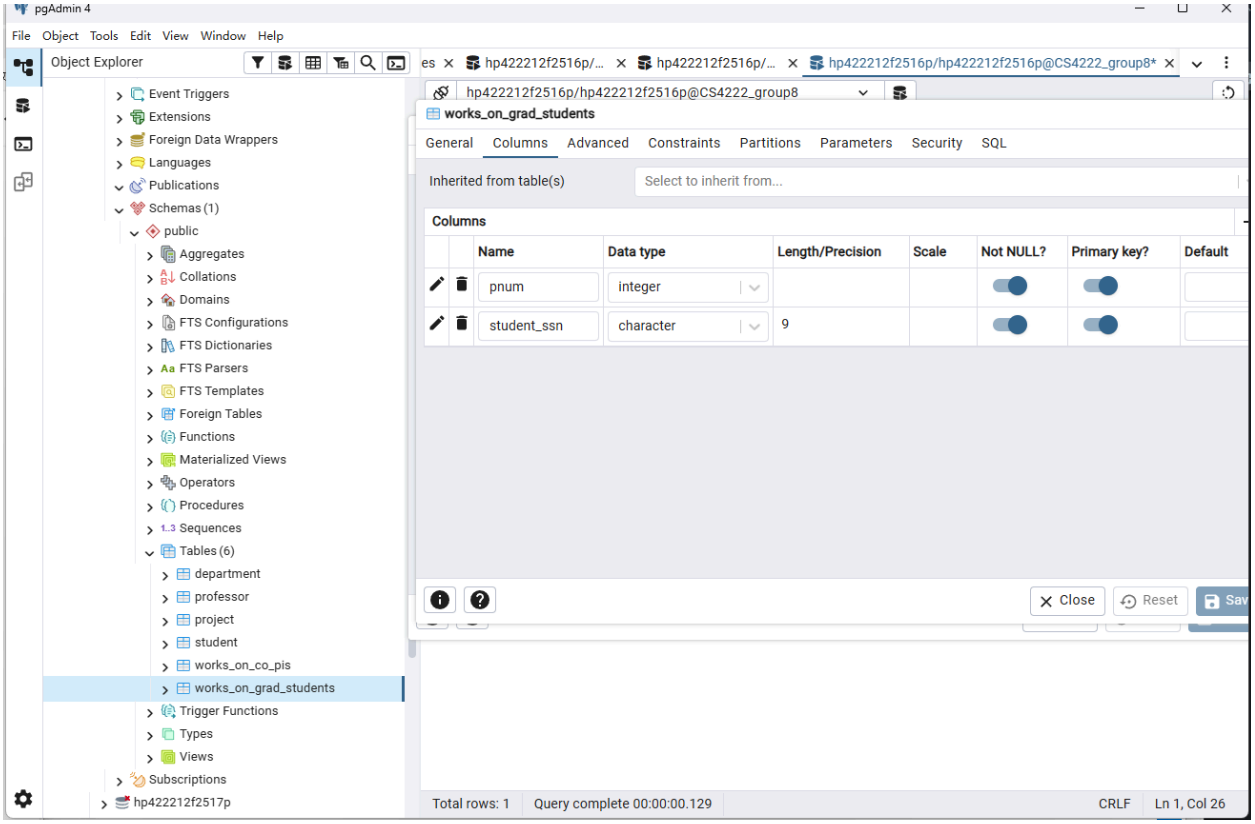


Inherited from table(s): Select to inherit from...

Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
pnum	integer			✓	✓	
professor_ssn	character	9		✓	✓	

Total rows: 1 Query complete 00:00:00.129 CRLF Ln 1, Col 26



works_on_grad_students

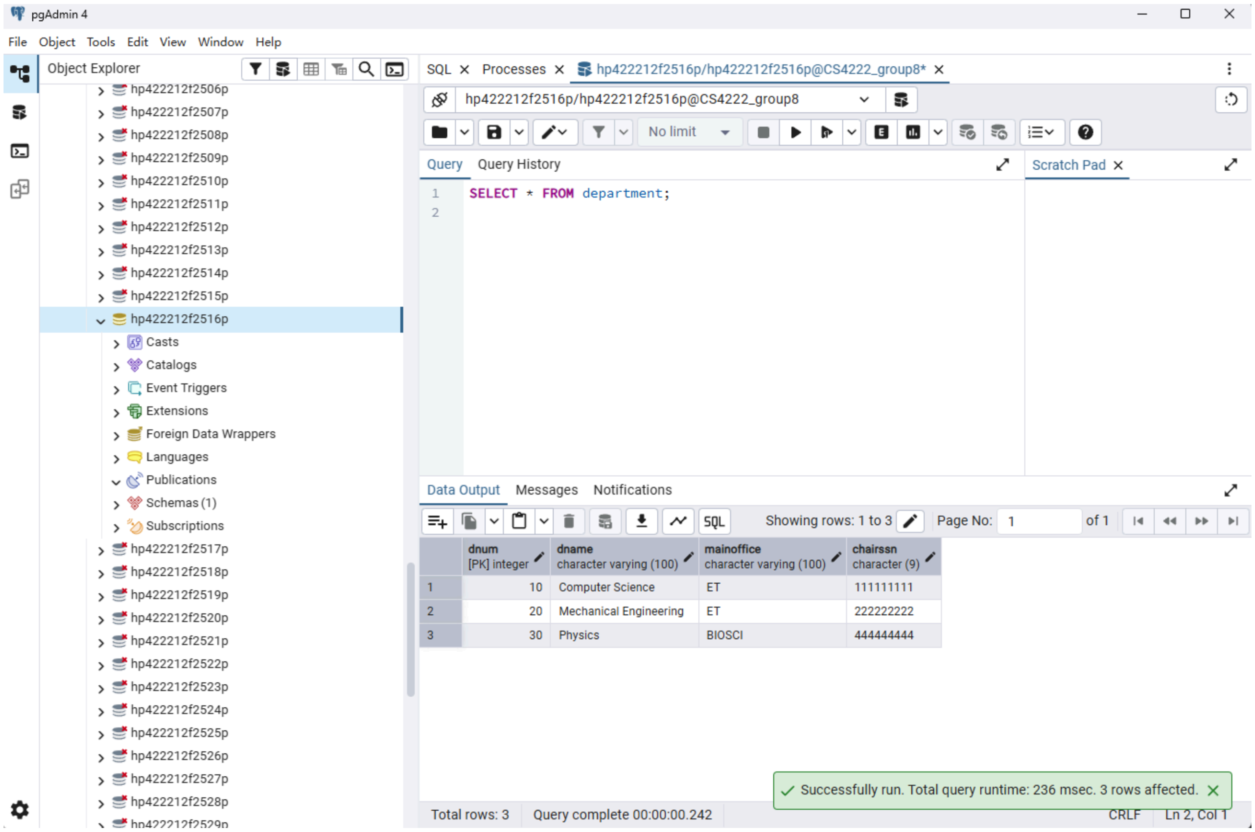
Inherited from table(s): Select to inherit from...

Columns

Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
pnum	integer			●	●	
student_ssn	character	9		●	●	

Total rows: 1 Query complete 00:00:00.129

E-Series — Data Screenshots

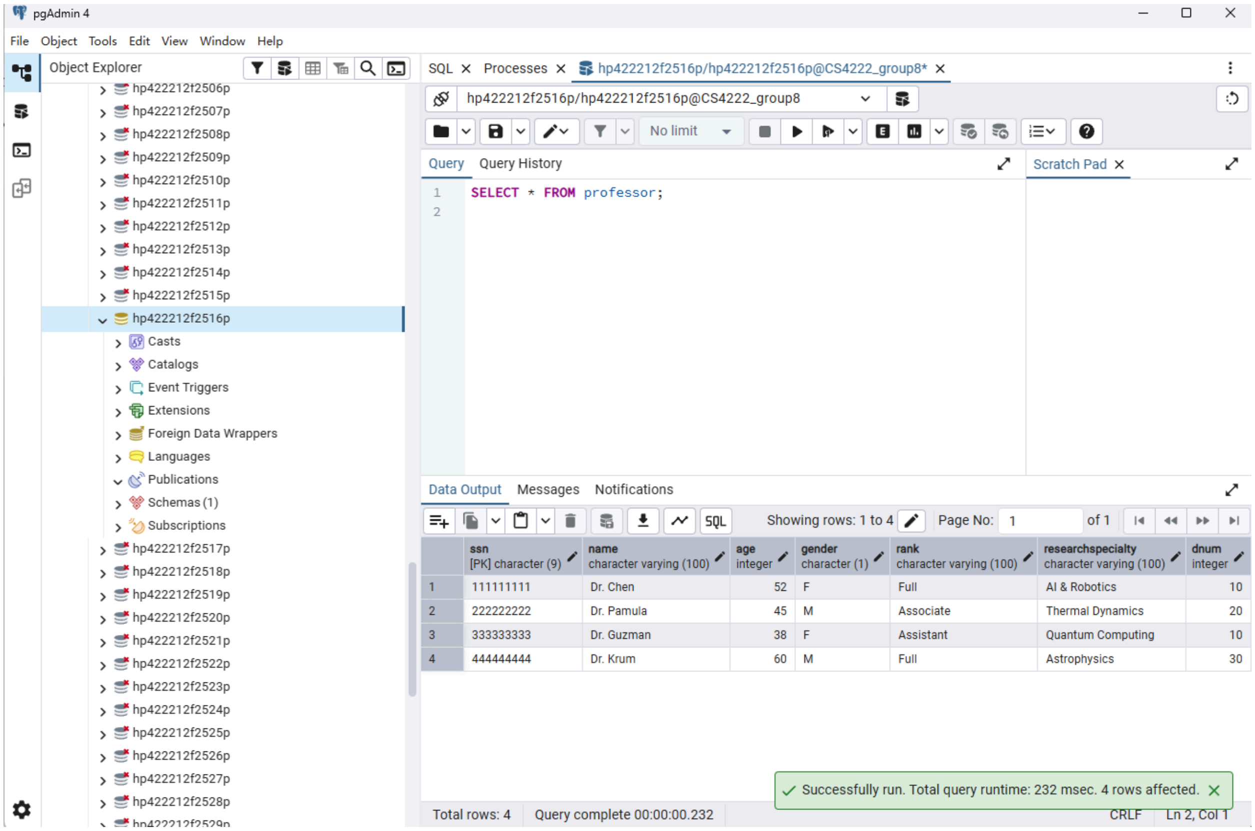


```sql
SELECT * FROM department;
```

	dnum [PK] integer	dname character varying (100)	mainoffice character varying (100)	chairssn character (9)
1	10	Computer Science	ET	111111111
2	20	Mechanical Engineering	ET	222222222
3	30	Physics	BIOSCI	444444444

✓ Successfully run. Total query runtime: 236 msec. 3 rows affected.

Total rows: 3 Query complete 00:00:00.242 CRLF Ln 2, Col 1

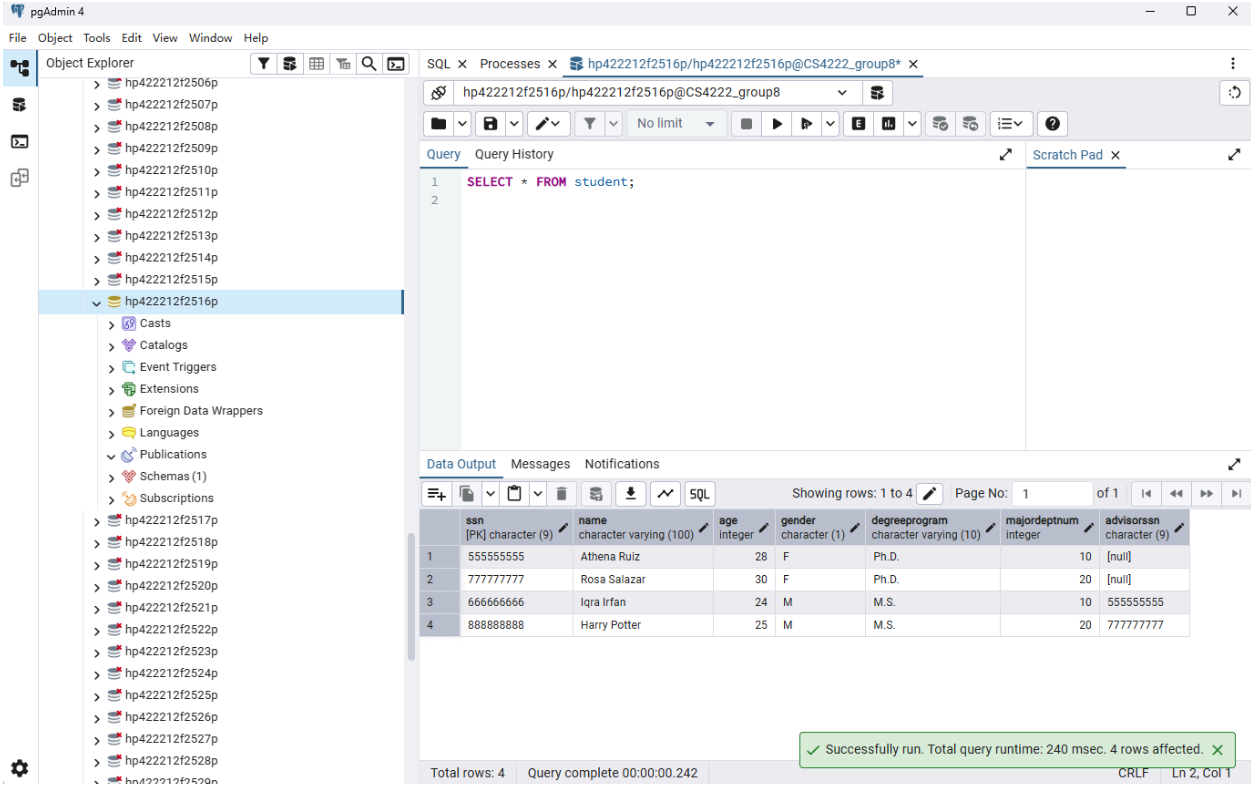


```sql
SELECT * FROM professor;
```

	ssn [PK] character (9)	name character varying (100)	age integer	gender character (1)	rank character varying (100)	researchspecialty character varying (100)	dnum integer
1	111111111	Dr. Chen	52	F	Full	AI & Robotics	10
2	222222222	Dr. Pamula	45	M	Associate	Thermal Dynamics	20
3	333333333	Dr. Guzman	38	F	Assistant	Quantum Computing	10
4	444444444	Dr. Krum	60	M	Full	Astrophysics	30

✓ Successfully run. Total query runtime: 232 msec. 4 rows affected.

Total rows: 4 Query complete 00:00:00.232 CRLF Ln 2, Col 1



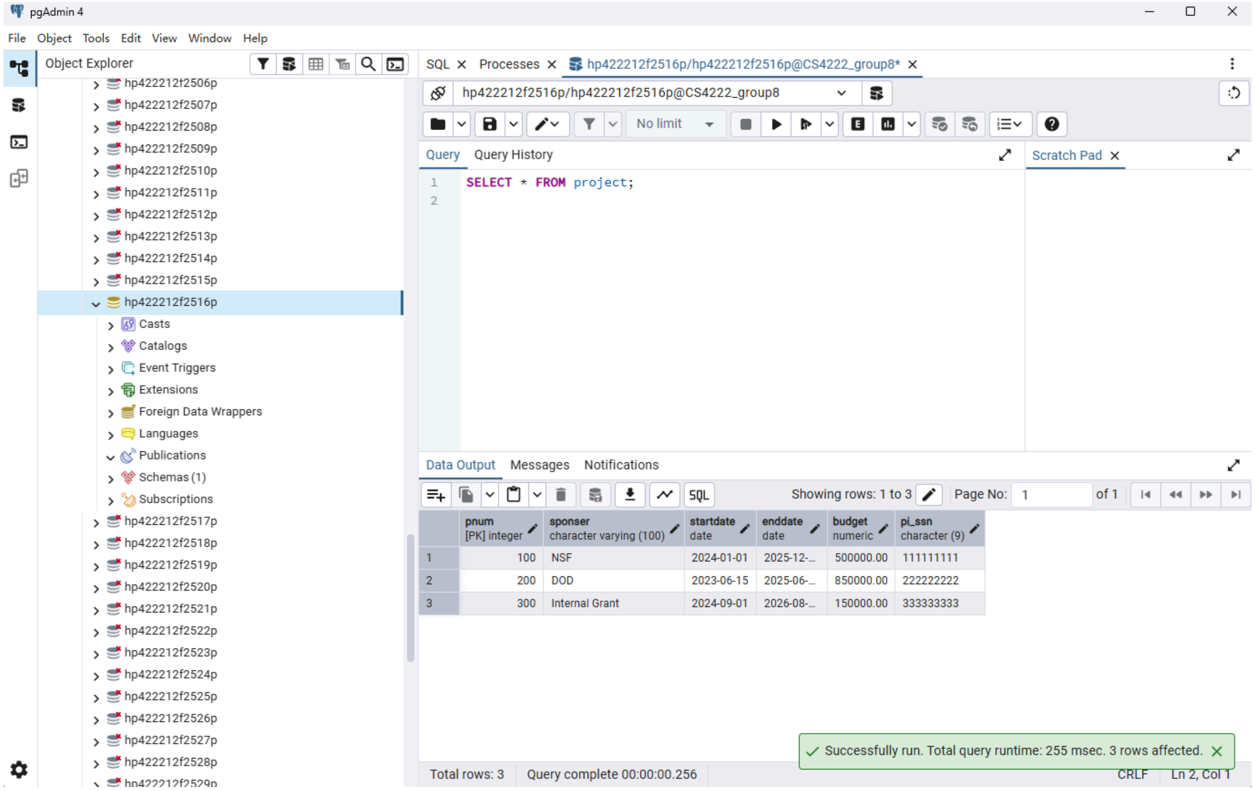
```
SELECT * FROM student;
```

Showing rows: 1 to 4

	ssn [PK] character (9)	name character varying (100)	age integer	gender character (1)	degreeprogram character varying (10)	majordeptnum integer	advisorssn character (9)
1	555555555	Athena Ruiz	28	F	Ph.D.	10	[null]
2	777777777	Rosa Salazar	30	F	Ph.D.	20	[null]
3	666666666	Iqra Irfan	24	M	M.S.	10	555555555
4	888888888	Harry Potter	25	M	M.S.	20	777777777

Successfully run. Total query runtime: 240 msec. 4 rows affected.

Total rows: 4 Query complete 00:00:00.242



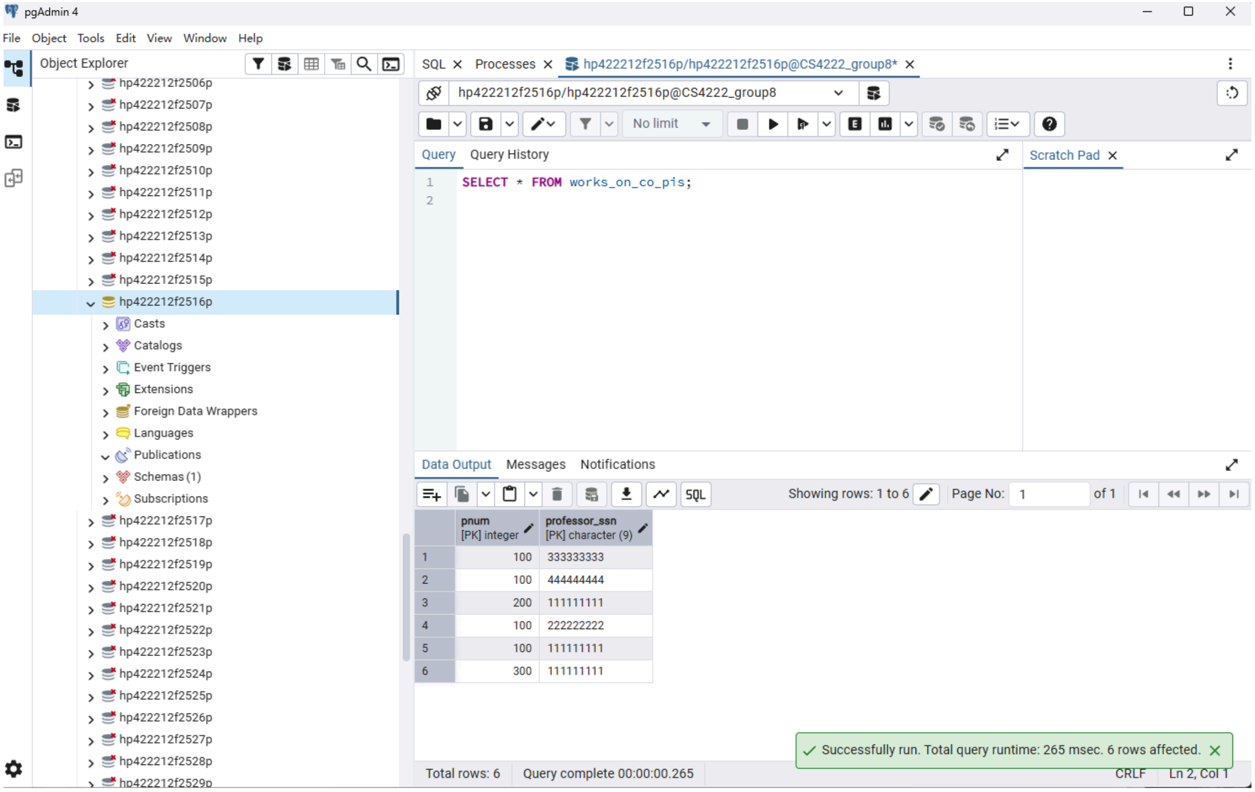
```
SELECT * FROM project;
```

Showing rows: 1 to 3

	pnum [PK] integer	sponser character varying (100)	startdate date	enddate date	budget numeric	pi_ssn character (9)
1	100	NSF	2024-01-01	2025-12-...	500000.00	111111111
2	200	DOD	2023-06-15	2025-06-...	850000.00	222222222
3	300	Internal Grant	2024-09-01	2026-08-...	150000.00	333333333

Successfully run. Total query runtime: 255 msec. 3 rows affected.

Total rows: 3 Query complete 00:00:00.256

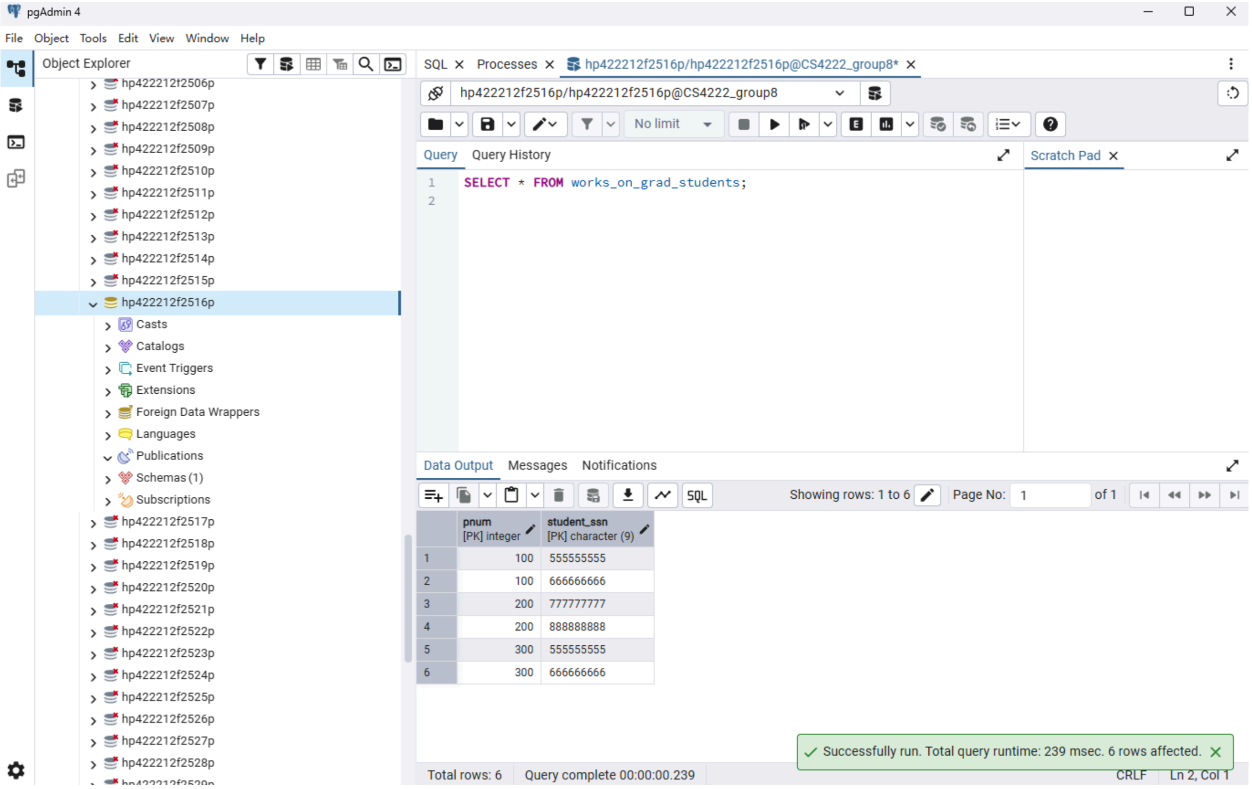


```sql
SELECT * FROM works_on_co_pis;
```

	pnum [PK] integer	professor_ssn [PK] character (9)
1	100	333333333
2	100	444444444
3	200	111111111
4	100	222222222
5	100	111111111
6	300	111111111

✓ Successfully run. Total query runtime: 265 msec. 6 rows affected.

Total rows: 6 Query complete 00:00:00.265



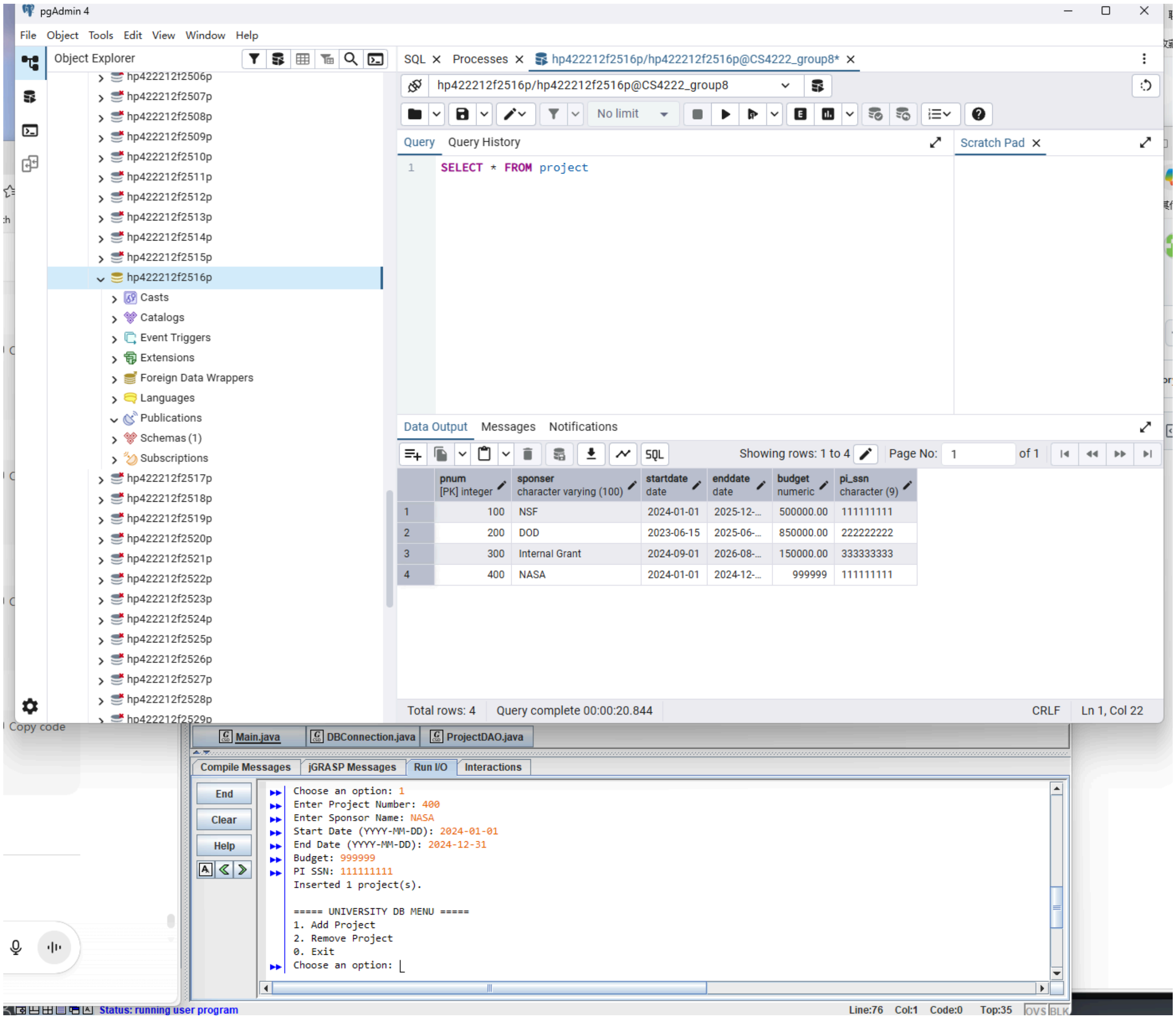
```sql
SELECT * FROM works_on_grad_students;
```

	pnum [PK] integer	student_ssn [PK] character (9)
1	100	555555555
2	100	666666666
3	200	777777777
4	200	888888888
5	300	555555555
6	300	666666666

✓ Successfully run. Total query runtime: 239 msec. 6 rows affected.

Total rows: 6 Query complete 00:00:00.239

Java Application Run



hp422212f2516p/hp422212f2516p@CS4222_group8

```
SELECT * FROM project
```

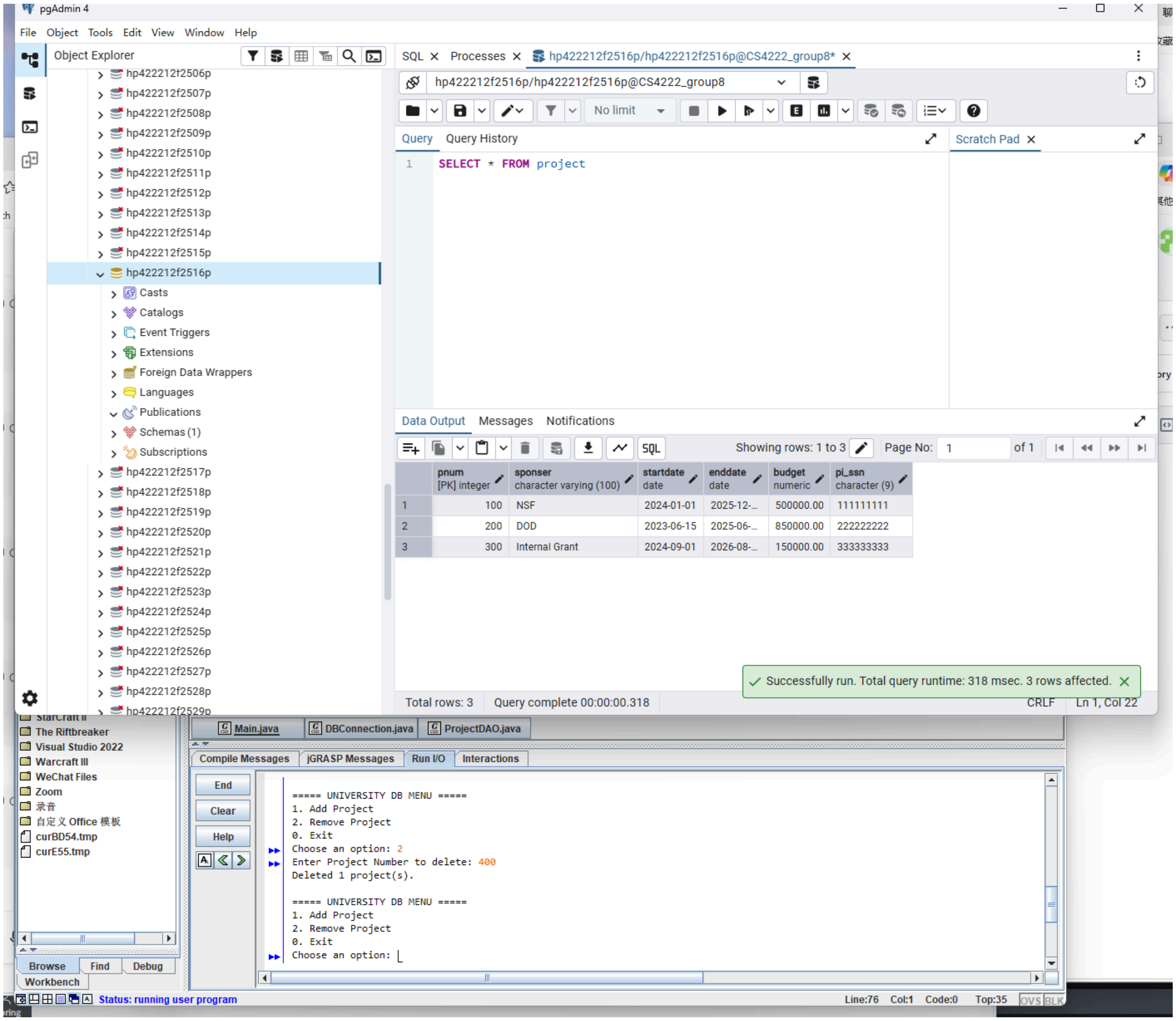
Showing rows: 1 to 4

	pnum [PK] integer	sponser character varying (100)	startdate date	enddate date	budget numeric	pi_ssn character (9)
1	100	NSF	2024-01-01	2025-12-...	500000.00	111111111
2	200	DOD	2023-06-15	2025-06-...	850000.00	222222222
3	300	Internal Grant	2024-09-01	2026-08-...	150000.00	333333333
4	400	NASA	2024-01-01	2024-12-...	999999	111111111

Total rows: 4 Query complete 00:00:20.844

```
Choose an option: 1
Enter Project Number: 400
Enter Sponsor Name: NASA
Start Date (YYYY-MM-DD): 2024-01-01
End Date (YYYY-MM-DD): 2024-12-31
Budget: 999999
PI SSN: 111111111
Inserted 1 project(s).

===== UNIVERSITY DB MENU =====
1. Add Project
2. Remove Project
0. Exit
Choose an option:
```



hp422212f2516p/hp422212f2516p@CS4222_group8

```
SELECT * FROM project
```

Data Output

Showing rows: 1 to 3 Page No: 1 of 1

	pnum [PK] integer	sponser character varying (100)	startdate date	enddate date	budget numeric	pi_ssn character (9)
1	100	NSF	2024-01-01	2025-12-...	500000.00	111111111
2	200	DOD	2023-06-15	2025-06-...	850000.00	222222222
3	300	Internal Grant	2024-09-01	2026-08-...	150000.00	333333333

✓ Successfully run. Total query runtime: 318 msec. 3 rows affected.

Total rows: 3 Query complete 00:00:00.318

Main.java DBConnection.java ProjectDAO.java

Run I/O

```
===== UNIVERSITY DB MENU =====
1. Add Project
2. Remove Project
0. Exit
Choose an option: 2
Enter Project Number to delete: 400
Deleted 1 project(s).

===== UNIVERSITY DB MENU =====
1. Add Project
2. Remove Project
0. Exit
Choose an option:
```

Status: running user program

7. Results & Discussion

This section summarizes the results of the implementation and evaluates how well the system meets the project requirements.

---

1. SQL Schema Execution

All SQL scripts—including table creation, foreign key constraints, sample data insertion, stored procedures, and triggers—executed successfully on the **cs1.calstatela.edu PostgreSQL server**.

There were no syntax errors or constraint violations during execution.

2. Schema–ER Diagram Consistency

The final database schema is fully consistent with the ER diagram:

- All entities (Professor, Student, Department, Project) and relationship tables were properly mapped.
- Primary keys and foreign keys match the conceptual design.
- All cardinality constraints (1:1, 1:N, M:N) were correctly implemented via FK and junction tables.

3. Stored Procedures and Triggers

All required stored procedures and triggers were successfully implemented and tested:

- `female_faculty()` correctly returns the proportion of female professors.
- `total_people(pno)` accurately counts PI, Co-PIs, and graduate students for each project.
- `faculty_restrict` prevents projects from having more than 4 Co-PIs.
- `student_restrict` ensures students do not exceed 2 assigned projects.

Test cases confirmed that errors were raised properly when constraints were violated.

4. Java JDBC Application

The Java client application worked as expected:

- Successfully connected to the remote PostgreSQL server.
- “Add Project” correctly inserted new records.
- “Remove Project” deleted the specified project.

- “Show Student Info” returned complete results including advisor and major department.
- User input validation and error handling were effective.

This demonstrates that the system supports basic CRUD and query operations for non-SQL users.

---

5. System Quality Evaluation

- **No constraint violations** occurred during any test.
- **Database logic is consistent and stable.**
- **The system architecture (DAO + JDBC) is clean and extensible**, allowing future addition of more queries or UI improvements.
- **Triggers enforce important business rules automatically**, improving data integrity.

Conclusion

Overall, the project meets all academic requirements.
Both the database layer and the application layer operate correctly, and all functions were validated with real test data.
The system is reliable, scalable, and demonstrates good software engineering practices.

8. Conclusion

This project successfully implemented a complete university database system that integrates conceptual design, relational schema creation, SQL programming, stored procedures, triggers, and a Java JDBC application.

Throughout the development process, we designed an ER diagram that accurately models professors, students, departments, and projects, along with their key constraints and relationships.
The relational schema was implemented in PostgreSQL and populated with sample data, demonstrating consistency between the conceptual and logical models.

Advanced database features—including stored procedures and triggers—were developed to enforce business rules such as project limitations and student workload constraints. All procedures and triggers performed as expected during testing.

The Java JDBC application provided a simple but functional interface for non-technical users to add projects, remove projects, and retrieve student information. This validated the integration of the application layer with the PostgreSQL database hosted on cs1.calstatela.edu.

Overall, the system is structurally sound, functionally complete, and suitable as a foundational model for real-world academic database systems. The project demonstrates strong database design, implementation skills, and effective team collaboration.

9. Contribution Breakdown

Haonan Ma — CIN: XXXXXXX

- Database schema creation
- JDBC application (DBConnection, DAO classes, Main)
- Tests and screenshots
- 40%

Athena Ruiz — CIN: XXXXXXX

- ER diagram design
- SQL constraints
- Documentation writing
- 30%

Washika Afrozi — CIN: XXXXXXX

- Stored procedures & triggers

- Sample data creation

- 30%