CS 457 / CS 557 – Database Software Design

Assignment 2

GroupA23

Junjia Lin (#002269013)

Luyun Nie (#002268087)

Wentao Lu (#002276355)

Yi Ren (#002269013)

I. Modeling with UML

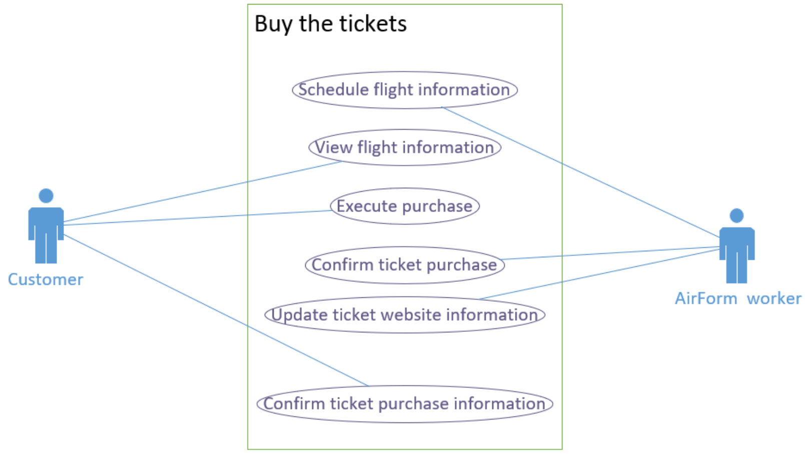
The airline AirForm proposes us to model in UML a reduced version of its information system of reservation of flight tickets. Flights are planned in advance and assigned to an airplane, an airport of departure, an airport of arrival, a departure date and an arrival date. Each airplane has a capacity in maximum number of passengers. Tickets are issued for each flight when planning, there is no overbooking.

Users buy the tickets. This purchase results in a reservation (via ticket) for the flight in question. We keep the last names, first names, addresses and phones of the users who made a reservation, as well as the booking date and the ticket price. Upon check-in (departure), passengers confirm their tickets for the registered flight. We memorize this initial confirmation.

When the flight is over, the reservations associated with it are archived, and they are deleted when the flight is canceled.

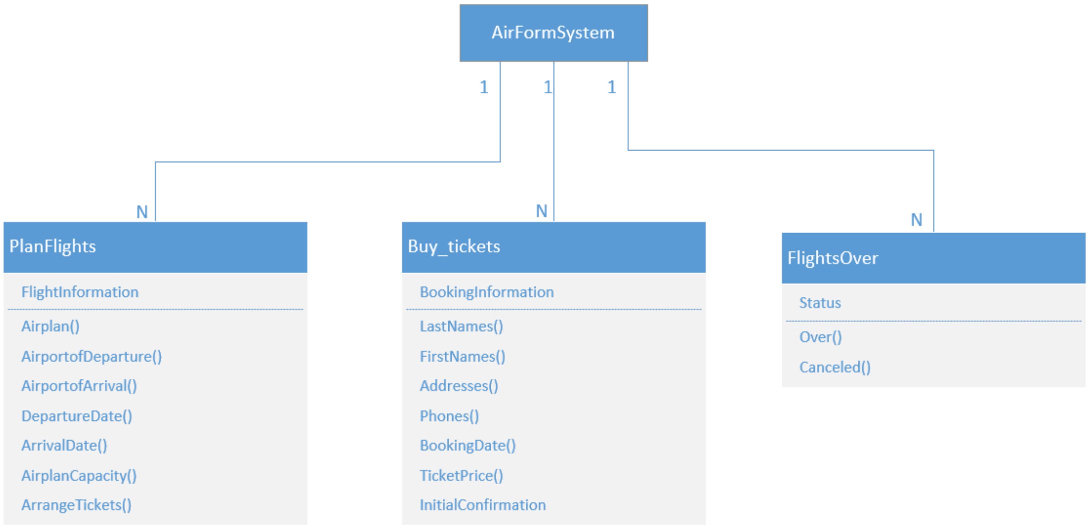
1. Write the high-level use case "Buy the tickets" initiated by a customer and refine if it is possible this high-level use case.

   Answer: The use cases "Buy the tickets" by a customer are view flight information, execute purchase and confirm ticket purchase information. In addition, we also add airform worker use case in buying the tickets that are schedule flight information, confirm ticket purchase and update ticket website information. The figure as shown below.



2. Propose a class diagram that models the AirForm system without representing other elements than those indicated in the statement. Please, specify the attributes for each class.

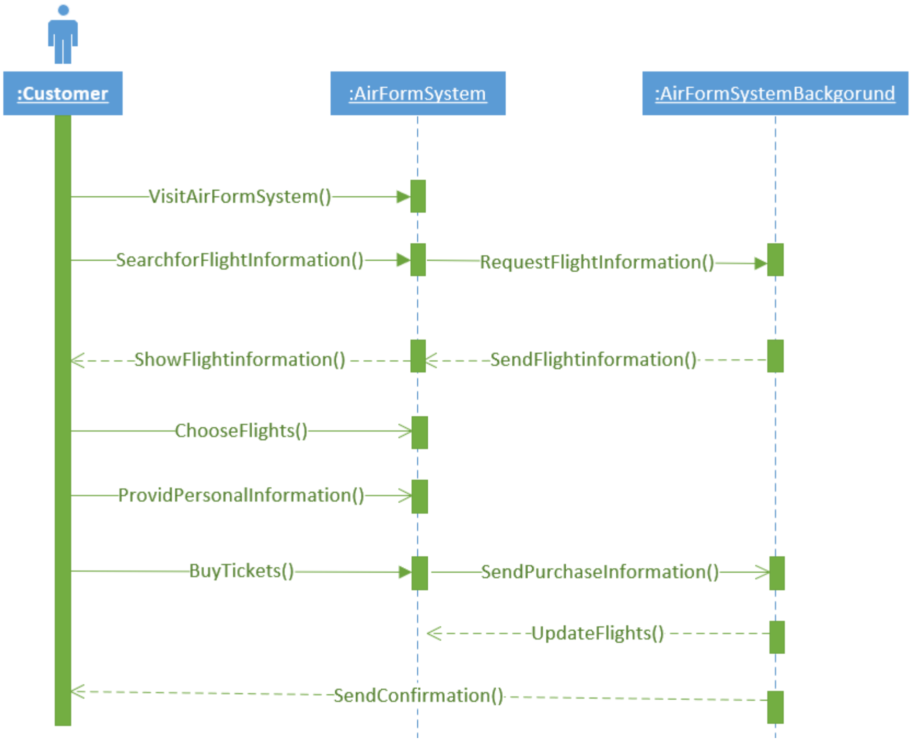
Answer:



3. Model the following treatments with detailed sequence diagrams.

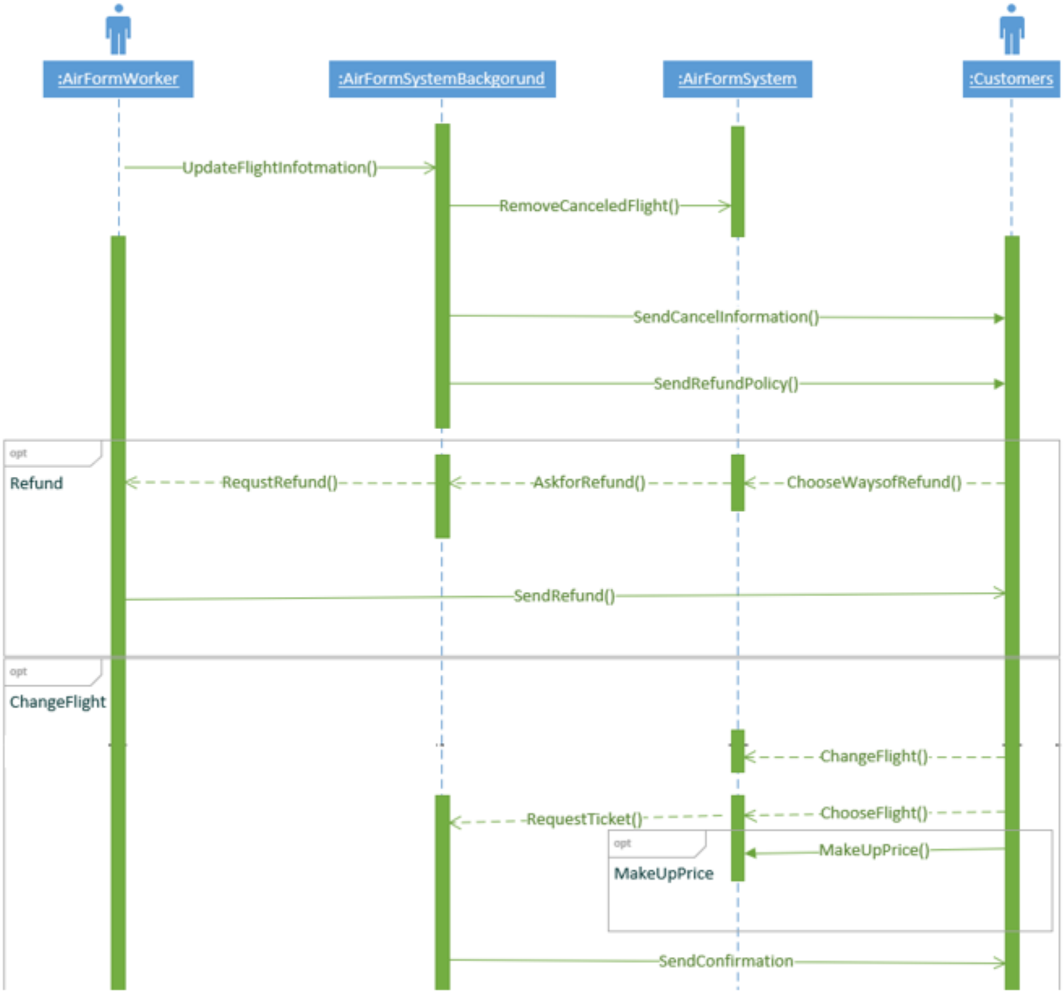
a. Book a ticket on a flight.

Answer:



b. Cancel a flight.

Answer:



4. Suppose that the airline AirForm wants to generalize the system to offer services for high-end tourists, and social group of wealthy people who travel the world to participate in social activities unavailable to ordinary people. Which changes in the AirForm system would you recommend? Justify your answer.

Answer:

If the company want to change the system to face high-end tourists and social group of wealthy people, the membership would be a good solution. People who want to use this system to buy a ticket should register for memberships before their purchase. When they are applying for memberships, their income level and frequency of travel can be taken into consideration of whether issue them memberships or not. In this way, ordinary people can not get access to have memberships, so they cannot use this system.

II. Programming with SQL

Instructions

In this section, you are recommended to install *pgAdmin*. In fact, it is a user interface that already includes *PostgreSQL*. Here is a tutorial that illustrates how to install it. Please, proceed as follow:

- Be sure to terminate every SQL statement with semicolon ";".
- Using your favorite editor, create files named *createall.sql* to create all the tables, *dropall.sql* to drops all the tables, *populateall.sql* to populate all the tables and *Assignment2Px.sql* that contains all the queries of the problem *x*. For example, *Assignment2P1.sql* denotes the queries of the problem 1.
- Include your name (and the name if your teammates if it is applicable) in a header comment at the top of your source file.
- Make sure that the output of each query is distinguishable. Comment your code; if nothing else, mark each query with its number.
- Good luck ☺️.

Problem 1

You are going to use *PostgreSQL* to design a simple University database. You will create tables and implement some queries.

Create the tables described below. Name these tables TEACHER, COURSE, STUDENT, ENROLMENT, COURSE_SCHEDULE.

TEACHER (t_id : number, t_name: text, t_status:text , t_dept: text)

COURSE(c_id : text, c_name: text,c_level :text)

STUDENT (s_id : number, s_name: text, s_status: text)

ENROLMENT (#c_id : text, #s_id : number)

COURSE_SCHEDULE (#c_id : text, #t_id : number)

Primary Keys are underlined, and the foreign keys are preceded by the symbol #.

Populate the tables you created by relying on the data provided in the file *populate.txt*.

Database Answer:

/* Create all tables(createall.sql): */

```
create table TEACHER(
t_id   integer,
t_name    text,
t_states   text,
t_dept   text,
primary key(t_id)
);

create table COURSE(
c_id   text,
c_name    text,
c_level   text,
primary key(c_id)
);

create table STUDENT(
s_id   integer,
s_name   text,
s_status    text,
primary key(s_id)
);

create table ENROLMENT(
c_id   text,
s_id    integer,
foreign key(c_id) references COURSE
   on delete set null,
foreign key(s_id) references STUDENT
   on delete set null
);
create table COURSE_SCHEDULE(
c_id   text,
t_id   integer,
foreign key(c_id)references COURSE
   on delete set null,
foreign key(t_id)references TEACHER
   on delete set null
);
```

/* Drops all the tables (dropall.sql): */

```
DROP TABLE ENROLMENT;
DROP TABLE COURSE_SCHEDULE;
DROP TABLE TEACHER;
DROP TABLE COURSE;
DROP TABLE STUDENT;
```

/* Populate all tables(populateall.sql): */

```
INSERT INTO teacher VALUES
(00111,'John A. Brown','P','CS'),
(00112,'James kareter','P','ECE'),
(00113,'Christopher Lee','AP', 'ECE'),
(00114,'Susanne Hambrusch','L','CS'),
(00115,'Sheron Noel','P','MA'),
(00116,'Kim Basinger','AP','ECE'),
(00117,'Christopher Clifton','P','CS'),
(00118,'Elisa Bertino','P','CS'),
(00119,'Susanne Hambrusch','AP','CS');
```

/* INSERT INTO COURSE VALUES */

```
('CS110','Intro to Computers','F'),
('CS348','Information Systems','S'),
('CS250','Computer Architecture','SP'),
('CS448','Intro to Data Bases','S'),
('MA511','Linear Algebra','GR'),
('CS503','Operating System','GR'),
('MA525','Intro to Complex Analysis','GR'),
('ECE264','Advanced C Programming','S'),
('ECE255','Intro to Electric Analysis & Design','S');
```

/* INSERT INTO STUDENT VALUES */

```
(234,'Anglo Anebal','F'),
(235,'Abram Ace','S'),
(236,'Adelbert Antti','SP'),
(237,'William Walker','GR'),
(238,'Emila Wdyth','GR'),
(239,'Judith Elba','S'),
(240,'Benjamin Bratt','SP'),
(241,'Tawny Kitaen','F');
```

/* INSERT INTO ENROLMENT VALUES */

```
('CS110',    240),
('CS110',    241),
('CS348',   235),
('CS348',     239),
('CS348',     237),
('CS250',     236),
('CS250',     241),
('ECE264',    236),
('ECE264',    237),
('ECE264',    238),
('MA525',     236),
('CS503',     238),
('CS503',     239),
('CS448',     240),
('CS250',     240),
('MA511',     240);
```

/* INSERT INTO COURSE_SCHEDULE VALUES */

```
('CS110',       00114),
('CS348',       00117),
('CS250',       00118),
('CS448',       00114),
('MA511',       00115),
('CS503',       00119),
('MA525',       00115),
('ECE264',      00113),
('ECE255',      00116);
```

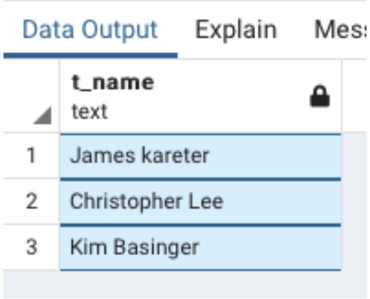
Queries

1. Find the name(s) of all teachers(s) who are from ECE department.

Answer:

```
SELECT t_name FROM public.TEACHER
WHERE t_dept LIKE 'ECE';
```

```
"James kareter"
"Christopher Lee"
"Kim Basinger"
```



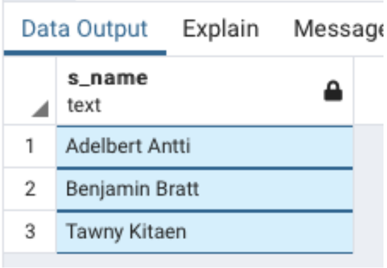
	t_name text
1	James kareter
2	Christopher Lee
3	Kim Basinger

2. Find the name(s) of all student(s) enrolled in CS250

Answer:

```
SELECT s_name FROM public.ENROLMENT
LEFT JOIN public.STUDENT
on ENROLMENT.s_id = STUDENT.s_id
WHERE c_id LIKE 'CS250';
```

```
"Adelbert Antti"
"Benjamin Bratt"
"Tawny Kitaen"
```



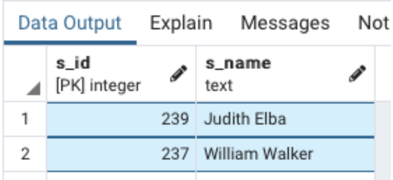
	s_name text
1	Adelbert Antti
2	Benjamin Bratt
3	Tawny Kitaen

3. Find the student id(s) and names(s) of all students enrolled in CS348 and either in ECE264 or in CS503

Answer:

```
SELECT  DISTINCT s_id,s_name FROM STUDENT
WHERE s_id in(
 SELECT s_id FROM ENROLMENT
 WHERE c_id In ('CS348','ECE264')
   GROUP By s_id
   HAVING Count (DISTINCT c_id) = 2)
 OR
 s_id in(
 SELECT s_id FROM ENROLMENT
 WHERE c_id In ('CS348','CS503')
   GROUP BY s_id
   HAVING COUNT (DISTINCT c_id) = 2
 );
```

```
239     "Judith Elba"
237     "William Walker"
```



Data Output   Explain   Messages   Not

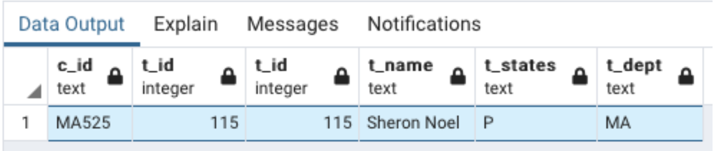
	s_id [PK] integer	s_name text
1	239	Judith Elba
2	237	William Walker

4. Find the name of the teacher teaching MA525

Answer:

```
SELECT * FROM public.COURSE_SCHEDULE
LEFT JOIN public.TEACHER
on TEACHER.t_id = COURSE_SCHEDULE.t_id
WHERE c_ID LIKE 'MA525';
```

```
"MA525"    115    115    "Sheron Noel"    "P"    "MA"
```



Data Output   Explain   Messages   Notifications

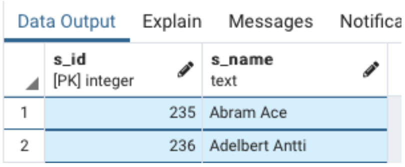
	c_id text	t_id integer	t_id integer	t_name text	t_states text	t_dept text
1	MA525	115	115	Sheron Noel	P	MA

5. Find the name(s) of all students enrolled in one or three courses

Answer:

```
SELECT DISTINCT s_id,s_name FROM STUDENT
WHERE s_id in(
 SELECT s_id FROM ENROLMENT
   Group By s_id
   Having Count (DISTINCT c_id) = 1)
 OR
 s_id in(SELECT s_id FROM ENROLMENT
   GROUP BY s_id
Having Count (DISTINCT c_id) = 3);
```

```
235    "Abram Ace"
236    "Adelbert Antti"
```



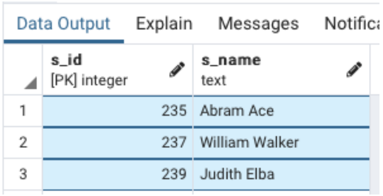
	s_id [PK] integer	s_name text
1	235	Abram Ace
2	236	Adelbert Antti

6. Find the name(s) of all students who are being taught by Prof. Christopher Clifton.

Answer:

```
 SELECT DISTINCT s_id,s_name FROM STUDENT
WHERE s_id in (
 SELECT s_id FROM ENROLMENT
 WHERE c_id in(
       SELECT c_id FROM COURSE_SCHEDULE
       WHERE t_id in (
              SELECT t_id FROM TEACHER
              WHERE t_name LIKE '%Christopher Clifton%'
       )
 )
);
```

```
235    "Abram Ace"
237    "William Walker"
239    "Judith Elba"
```



	s_id [PK] integer	s_name text
1	235	Abram Ace
2	237	William Walker
3	239	Judith Elba

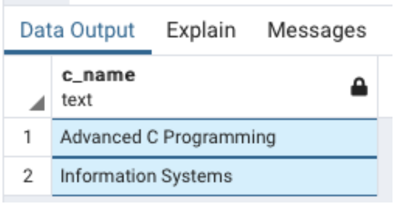
7. Name any undergraduate course(s) being taken by graduate student(s).

Answer:

```
SELECT DISTINCT c_name FROM COURSE
WHERE c_level != 'GR' AND c_id IN(
 SELECT c_id FROM public.STUDENT
 LEFT JOIN public.ENROLMENT
 ON ENROLMENT.s_id = STUDENT.s_id
 WHERE s_status = 'GR');
```

"Advanced C Programming"

"Information Systems"



	c_name text
1	Advanced C Programming
2	Information Systems

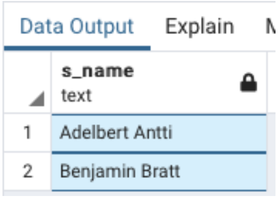
8. Name any undergraduate student(s) who is taking a course with Prof. Sheron Noel

Answer:

```
SELECT DISTINCT s_name FROM STUDENT
LEFT JOIN public.ENROLMENT
ON ENROLMENT.s_id = STUDENT.s_id
WHERE s_status != 'GR' AND c_id IN (
 SELECT c_id FROM COURSE_SCHEDULE
 WHERE t_id in (SELECT t_id FROM TEACHER WHERE t_name LIKE '%Sheron Noel'))
```

"Adelbert Antti"

"Benjamin Bratt"



	s_name text
1	Adelbert Antti
2	Benjamin Bratt

Problem 2

Suppose that you have the following university schema:

```
create table classroom (

        building             varchar(15),

 room_number  varchar(7),   capacity  numeric(4,0),

         primary key (building, room_number)

        );


create table department(
        dept_name            varchar(20),
         building            varchar(15),

         budget                      numeric(12,2) check (budget > 0),
         primary key (dept_name)

        );


create table course (
        course_id            varchar(8),
         title               varchar(50),

         dept_name           varchar(20),

         credits             numeric(2,0) check (credits > 0),
         primary key (course_id),

         foreign key (dept_name) references department

          on delete set null

        );


create table instructor (

        ID               varchar(5),         name

varchar(20) not null,       dept_name

varchar(20),
        salary         numeric(8,2) check (salary > 29000),        primary key
(ID),
        foreign key (dept_name) references department
on delete set null
          );

create table section (
          course_id              varchar(8),
         sec_id   varchar(8),  semester  varchar(6)
                  check (semester in ('Fall', 'Winter', 'Spring', 'Summer')),
          year                   numeric(4,0) check (year > 1701 and year < 2100),
          building               varchar(15),
          room_number                   varchar(7),
          time_slot_id           varchar(4),
        primary key (course_id, sec_id, semester, year),
foreign key (course_id) references course        on
delete cascade,
           foreign key (building, room_number) references classroom
                  on delete set null
          );

create table teaches (
          ID                     varchar(5),
           course_id             varchar(8),
           sec_id                varchar(8),
           semester              varchar(6),
           year                  numeric(4,0),
        primary key (ID, course_id, sec_id, semester, year),      foreign key
(course_id,sec_id, semester, year) references section
```

```
            on delete cascade,
        foreign key (ID) references instructor
            on delete cascade
        );

create table student (
        ID                  varchar(5),
        name                varchar(20) not null,
        dept_name           varchar(20),
        tot_cred            numeric(3,0) check (tot_cred >= 0),
        primary key (ID),
      foreign key (dept_name) references department
on delete set null
        );

create table takes (
        ID            varchar(5),  course_id
         varchar(8),

        sec_id              varchar(8),
        semester            varchar(6),
        year                 numeric(4,0),
        grade             varchar(2),
      primary key (ID, course_id, sec_id, semester, year),      foreign key
(course_id,sec_id, semester, year) references section
            on delete cascade,
        foreign key (ID) references student
            on delete cascade
        );
```

```
create table advisor (
        s_ID  varchar(5),  i_ID  varchar(5),
         primary key (s_ID),
     foreign key (i_ID) references instructor (ID)
on delete set null,
         foreign key (s_ID) references student (ID)
                on delete cascade
        );


create table time_slot (
        time_slot_id            varchar(4),
         day                    varchar(1),
         start_hr               numeric(2) check (start_hr >= 0 and start_hr < 24),
         start_min              numeric(2) check (start_min >= 0 and start_min < 60),
         end_hr                 numeric(2) check (end_hr >= 0 and end_hr < 24),
         end_min                numeric(2) check (end_min >= 0 and end_min < 60),
         primary key (time_slot_id, day, start_hr, start_min)
        );


create table prereq (
        course_id               varchar(8),
         prereq_id              varchar(8),
     primary key (course_id, prereq_id),   foreign
key (course_id) references course
                on delete cascade,
         foreign key (prereq_id) references course
        );
```

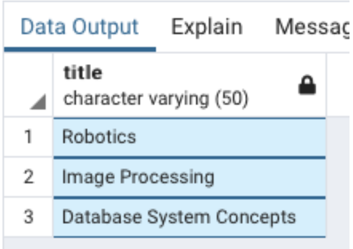
After you create the previous tables in *pgAdmin* by importing the file *createTables.sql*, write the following queries in *SQL*, using the university schema. We suggest you to populate your tables by importing the file *PopulateData.sql* in *pgAdmin*.

1. Find the titles of courses in the Comp. Sci. department that have 3 credits.

   Answer:

```
SELECT title FROM COURSE
WHERE dept_name LIKE 'Comp. Sci.' AND credits = 3;
```

```
"Robotics"
"Image Processing"
"Database System Concepts"
```



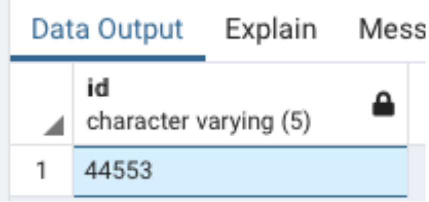
	title character varying (50)
1	Robotics
2	Image Processing
3	Database System Concepts

2. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

   Answer:

```
SELECT DISTINCT TAKES.id FROM TAKES
WHERE TAKES.course_id in (
 SELECT course_id FROM TEACHES
 WHERE TEACHES.id in (
        SELECT INSTRUCTOR.id FROM INSTRUCTOR
        WHERE INSTRUCTOR.name LIKE '%Einstein%'));
```

```
"44553"
```



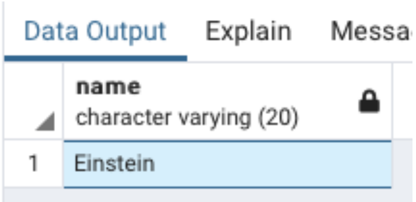
	id character varying (5)
1	44553

3. Find the highest salary of any instructor.

   Answer:

```
SELECT INSTRUCTOR.name FROM INSTRUCTOR
ORDER BY INSTRUCTOR.salary DESC LIMIT 1;
```

```
"Einstein"
```



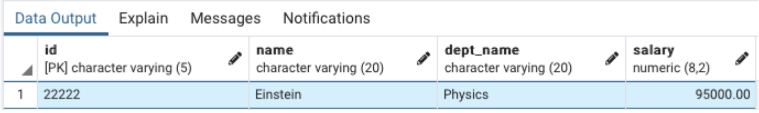
	name character varying (20)
1	Einstein

4. Find all instructors earning the highest salary (there may be more than one with the same salary).

   Answer:

```
SELECT * FROM public.INSTRUCTOR
WHERE INSTRUCTOR.id IN(
 SELECT id FROM INSTRUCTOR
 WHERE salary  = (
        SELECT MAX(salary) FROM INSTRUCTOR));

"22222"      "Einstein"  "Physics"   95000.00
```



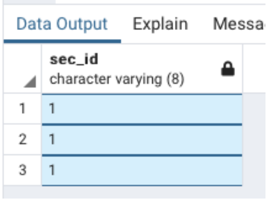
	id [PK] character varying (5)	name character varying (20)	dept_name character varying (20)	salary numeric (8,2)
1	22222	Einstein	Physics	95000.00

5. Find the enrollment of each section that was offered in Autumn 2009.

   Answer:

```
SELECT sec_id FROM public.SECTION
WHERE public.SECTION.year = 2009 AND semester LIKE 'Fall';


"1"
"1"
"1"
```



	sec_id character varying (8)
1	1
2	1
3	1

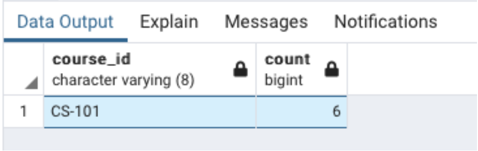
6. Find the maximum enrollment, across all sections, in Autumn 2009.

   Answer:

```
SELECT course_id,COUNT(*) FROM TAKES
WHERE TAKES.year = 2009 AND semester LIKE 'Fall'
GROUP BY course_id
```

```
ORDER BY COUNT(*) DESC LIMIT 1;
```

"CS-101"   6



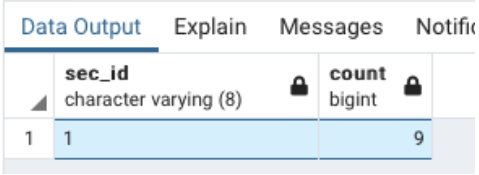
	course_id character varying (8)	count bigint
1	CS-101	6

7. Find the sections that had the maximum enrollment in Autumn 2009.

Answer:

```
SELECT sec_id,COUNT(*) FROM TAKES
WHERE TAKES.year = 2009 AND semester LIKE 'Fall'
GROUP BY sec_id
ORDER BY COUNT(*) DESC LIMIT 1;
```

"1"   9



	sec_id character varying (8)	count bigint
1	1	9

Suppose you are given a relation grade *points(grade, points)*, which provides a conversion from letter grades in the takes relation to numeric scores; for example an "A" grade could be specified to correspond to 4 points, an "A−" to 3.7 points, a "B+" to 3.3 points, a "B" to 3 points, and so on. The grade points earned by a student for a course offering (section) is defined as the number of credits for the course multiplied by the numeric points for the grade that the student received.

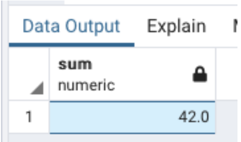
Given the above relation, and our university schema, write each of the following queries in SQL. You can assume for simplicity that no takes tuple has the null value for grade.

8. Find the total grade-points earned by the student with *ID* 12345, across all courses taken by the student.

   Answer:

```
SELECT SUM(gradepoints) FROM(
 SELECT DISTINCT TAKES.id,grade, credits,
 CASE
    WHEN grade = 'A+' THEN 4.3*credits
    WHEN grade = 'A' THEN 4.0*credits
    WHEN grade = 'A-' THEN 3.7*credits
    WHEN grade = 'B+' THEN 3.3*credits
    WHEN grade = 'B' THEN 3.0*credits
    WHEN grade = 'B-' THEN 2.7*credits
    WHEN grade ='C+' THEN 2.3*credits
    WHEN grade ='C' THEN 2.0*credits
     WHEN grade ='C-' THEN 2.3*credits
       WHEN grade ='F' THEN 1*credits
       WHEN grade IS NULL THEN 0
   END AS gradepoints
FROM TAKES,COURSE
WHERE TAKES.id LIKE '12345') AS theresult;
```

42.0



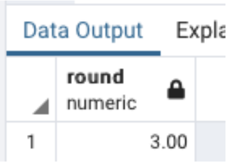
```
  sum
  numeric
1       42.0
```

9. Find the grade-point average (GPA) for the above student, that is, the total gradepoints divided by the total credits for the associated courses.

   Answer:

```
SELECT ROUND(SUM(gradepoints)/SUM(credits),2) FROM(
 SELECT DISTINCT grade, credits,
 CASE
    WHEN grade = 'A+' THEN 4.3*credits
    WHEN grade = 'A' THEN 4.0*credits
    WHEN grade = 'A-' THEN 3.7*credits
    WHEN grade = 'B+' THEN 3.3*credits
    WHEN grade = 'B' THEN 3.0*credits
    WHEN grade = 'B-' THEN 2.7*credits
    WHEN grade ='C+' THEN 2.3*credits
    WHEN grade ='C' THEN 2.0*credits
     WHEN grade ='C-' THEN 2.3*credits
       WHEN grade ='F' THEN 1*credits
       WHEN grade IS NULL THEN 0
   END AS gradepoints
FROM TAKES,COURSE
WHERE TAKES.id = '12345') AS average_gpa;
```

3.00



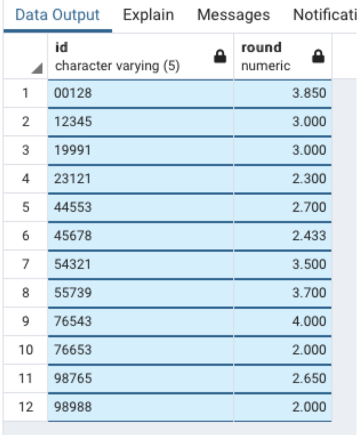
	round numeric
1	3.00

10. Find the ID and the grade-point average of every student.

Answer:

```
SELECT newsheet.id,ROUND(SUM(newsheet.gradepoints)/SUM(credits),3)
FROM(
  SELECT DISTINCT TAKES.id,grade,credits,
 CASE
    WHEN grade = 'A+' THEN 4.3*credits
    WHEN grade = 'A' THEN 4.0*credits
    WHEN grade = 'A-' THEN 3.7*credits
    WHEN grade = 'B+' THEN 3.3*credits
    WHEN grade = 'B' THEN 3.0*credits
    WHEN grade = 'B-' THEN 2.7*credits
    WHEN grade ='C+' THEN 2.3*credits
    WHEN grade ='C' THEN 2.0*credits
      WHEN grade ='C-' THEN 2.3*credits
        WHEN grade ='F' THEN 1*credits
        WHEN grade IS NULL THEN 0
  END AS gradepoints
FROM TAKES,COURSE) AS newsheet
GROUP BY newsheet.id
ORDER BY newsheet.id;
```

```
"00128"    3.850
"12345"    3.000
"19991"    3.000
"23121"    2.300
"44553"    2.700
"45678"    2.433
"54321"    3.500
"55739"    3.700
"76543"    4.000
"76653"    2.000
"98765"    2.650
"98988"    2.000
```



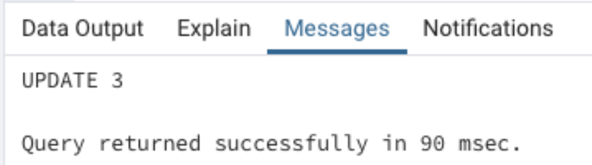
	id character varying (5)	round numeric
1	00128	3.850
2	12345	3.000
3	19991	3.000
4	23121	2.300
5	44553	2.700
6	45678	2.433
7	54321	3.500
8	55739	3.700
9	76543	4.000
10	76653	2.000
11	98765	2.650
12	98988	2.000

11. Increase the salary of each instructor in the Comp. Sci. department by 10%.

Answer:

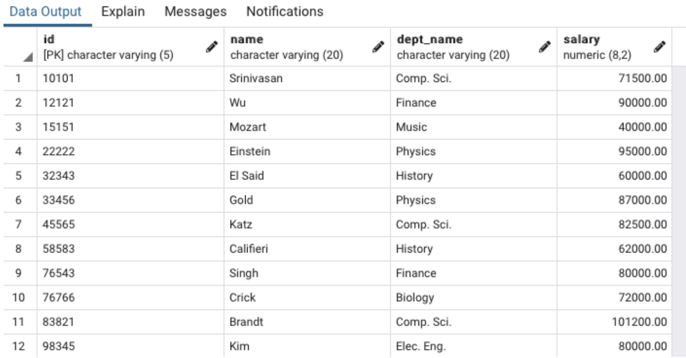
```
UPDATE public.INSTRUCTOR
SET salary = salary*1.1
WHERE dept_name = 'Comp. Sci.';
```

UPDATE 3 Query returned successfully in 90 msec.



```
UPDATE 3

Query returned successfully in 90 msec.
```



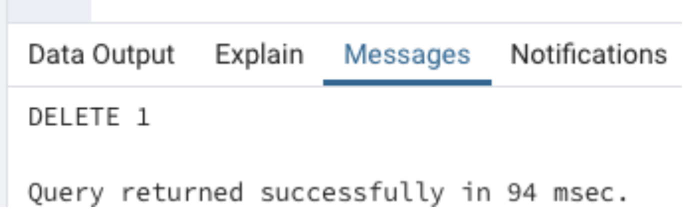
	id [PK] character varying (5)	name character varying (20)	dept_name character varying (20)	salary numeric (8,2)
1	10101	Srinivasan	Comp. Sci.	71500.00
2	12121	Wu	Finance	90000.00
3	15151	Mozart	Music	40000.00
4	22222	Einstein	Physics	95000.00
5	32343	El Said	History	60000.00
6	33456	Gold	Physics	87000.00
7	45565	Katz	Comp. Sci.	82500.00
8	58583	Califieri	History	62000.00
9	76543	Singh	Finance	80000.00
10	76766	Crick	Biology	72000.00
11	83821	Brandt	Comp. Sci.	101200.00
12	98345	Kim	Elec. Eng.	80000.00

12. Delete all courses that have never been offered (that is, do not occur in the section relation).

Answer:

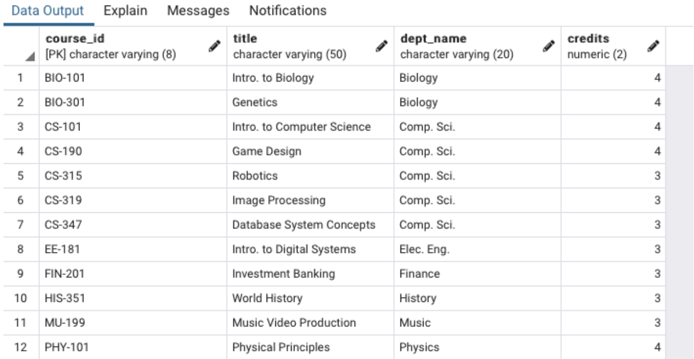
```
DELETE FROM COURSE
WHERE COURSE.course_id NOT IN (SELECT SECTION.course_id FROM SECTION);
```

DELETE 1



```
DELETE 1

Query returned successfully in 94 msec.
```



	course_id [PK] character varying (8)	title character varying (50)	dept_name character varying (20)	credits numeric (2)
1	BIO-101	Intro. to Biology	Biology	4
2	BIO-301	Genetics	Biology	4
3	CS-101	Intro. to Computer Science	Comp. Sci.	4
4	CS-190	Game Design	Comp. Sci.	4
5	CS-315	Robotics	Comp. Sci.	3
6	CS-319	Image Processing	Comp. Sci.	3
7	CS-347	Database System Concepts	Comp. Sci.	3
8	EE-181	Intro. to Digital Systems	Elec. Eng.	3
9	FIN-201	Investment Banking	Finance	3
10	HIS-351	World History	History	3
11	MU-199	Music Video Production	Music	3
12	PHY-101	Physical Principles	Physics	4

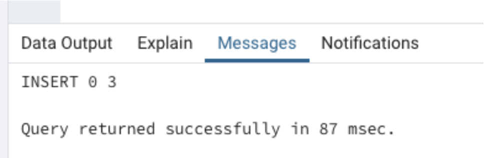
The course:BIO-399 has been deleted.

13. Insert every student whose *tot_cred* attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.

Answer:

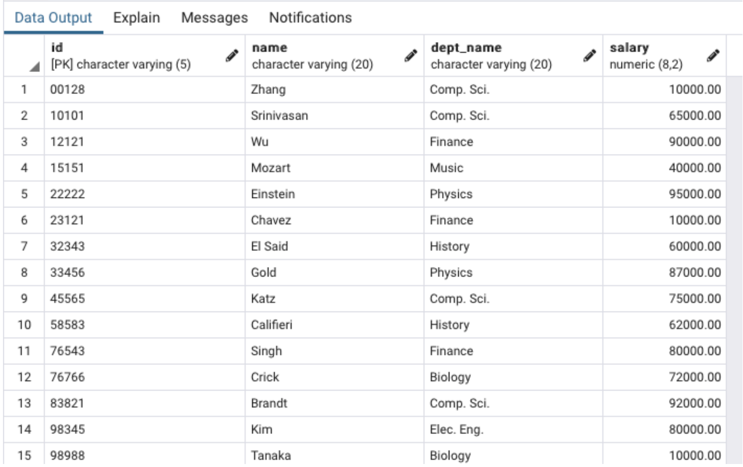
```
INSERT INTO INSTRUCTOR(id,name,dept_name,salary)
SELECT STUDENT.id, STUDENT.name, STUDENT.dept_name,10000
FROM STUDENT WHERE STUDENT.tot_cred > 100 AND STUDENT.id NOT IN(
 SELECT id FROM INSTRUCTOR);
```

INSERT 0 3 Query returned successfully in 87 msec.



```
INSERT 0 3

Query returned successfully in 87 msec.
```



	id [PK] character varying (5)	name character varying (20)	dept_name character varying (20)	salary numeric (8,2)
1	00128	Zhang	Comp. Sci.	10000.00
2	10101	Srinivasan	Comp. Sci.	65000.00
3	12121	Wu	Finance	90000.00
4	15151	Mozart	Music	40000.00
5	22222	Einstein	Physics	95000.00
6	23121	Chavez	Finance	10000.00
7	32343	El Said	History	60000.00
8	33456	Gold	Physics	87000.00
9	45565	Katz	Comp. Sci.	75000.00
10	58583	Califieri	History	62000.00
11	76543	Singh	Finance	80000.00
12	76766	Crick	Biology	72000.00
13	83821	Brandt	Comp. Sci.	92000.00
14	98345	Kim	Elec. Eng.	80000.00
15	98988	Tanaka	Biology	10000.00

There are two problems we need to solve in this question.

The first is that there exists a check constraint in the instructor salary column, which means we cannot add an instructor whose salary is below 29000. To solve this problem, we firstly imply a modification sentence to delete the constraint (to avoid error because of code repeated implementation, we didn’t imply these lines in the final sql file):

```
-- ALTER TABLE INSTRUCTOR
-- DROP CONSTRAINT instructor_salary_check;

-- ALTER TABLE INSTRUCTOR
-- ADD CONSTRAINT instructor_salary_check check(salary >29000);
```

Then we found there is an instructor who shared the same id with a student that we filtered the existed id from the student table.