1. Preliminaries

Under Resources→Lab4 on Piazza, there are some files that are discussed in this document. Two of the files are create_lab4.sql and load_lab4.sql. The create_lab4.sql script creates all tables within the schema Lab4; otherwise, it is the same as create_lab3.sql in our solution to Lab3. Lab3's new General constraints and revised Foreign Key constraints are not in this schema. The file load_lab4.sql loads data into those tables, just as similar files did for previous Lab Assignments.

Alter your search path so that you can work with the tables without qualifying them with the schema name:

```
ALTER ROLE cse180 SET SEARCH_PATH TO Lab4;
```

You must log out and log back in for this to take effect. To verify your search path, use:

```
SHOW SEARCH_PATH;
```

Note: It is important that you do not change the names of the tables. Otherwise, your application may not pass our tests, and you will not get any points for this assignment.

2. Instructions for installing Python and psycopg2 in your Docker container

The first step is to install the Python language and the library psycopg2 inside your Docker container. After firing up the container, you can achieve this by executing the following the following three lines from your local environment (i.e., outside the docker container):

docker exec container-psql apt-get update
docker exec container-psql apt-get install -y python3
docker exec container-psql apt-get install python3-psycopg2

3. Instructions for database access from Python

The *runAirlineApplication.py* which you've been given under Resources→Lab4 is not executable as is. You will have to complete it to make it runnable, and that includes writing the three Python functions that are described in Section 4 of this document. You will also have to write a Stored Function that is used by one of those Python functions; that Stored Function is described in Section 5 of this document. We assume that CSE 180 students are familiar with Python 3. *runAirlineApplication.py* will be the only file in your Python program.

Assuming that you are using the Docker container as usual, and *runAirlineApplication.py* is in your current directory, you can execute it with the following command:

docker exec container-psql python3 runAirlineApplication.py

Note: If you have been using a different userid and password than "cse180" and "database4me" for PostgreSQL, then you can specify them as arguments as follows:

docker exec container-psql python3 runAirlineApplication.py <your_userid> <your_password>

4. Goal

The fourth lab project puts the database you have created to practical use. You will implement part of an application front-end to the database, both executing SQL statements that SELECT and UPDATE, and executing a Stored Function.

5. Description of the three Python functions in the runAirlineApplication.py file that interact with the database

The *runAirlineApplication.py* file that you've been given contains skeletons for three Python functions that interact with the database using psycopg2. These three Python functions are described below. The first argument for all of these Python functions is your connection to the database. Your task is to implement functions which match those descriptions.

The following psycopg2-related links appear in Lecture 12, which describes how to access databases from Python. All of these links should be helpful, but none of them are perfect introductory tutorials.

- Python PostgreSQL Tutorial Using psycopg2 on the PYnative site, showing some SELECT, INSERT, DELETE and UPDATE statement. This one uses Python 3, which is why it's the first link on the list.
- psycopg2 Tutorial on the PostgreSQL site. Unfortunately, code is written in Python 2, not Python 3, but it explains use of psycopg2 pretty well.
- psychopg2 usage examples, on the psycopg site. Once again, code is written in Python 2, not Python 3, but it explains use of psycopg2.

- **countNumberOfDepartingPassengers(departureAirport)**:
  - The airline needs to know how many unique customers have departed from a given airport.
  - The Reservation table tells us about reservations made by a passenger(passengerID) for a given flight(flightID), while the Flight table tells us about the flights(flightID) that departed from a given airport(departureAirport).
  - We want to count the number of different passengers that left a particular airport. The function **countNumberOfDepartingPassengers()** should count the number of different passengers who have departed an airport by the parameter departureAirport. The function should return the count.
  - But it's possible that there isn't an airport whose airportCode value equals departureAirport. In that case, **countNumberOfDepartingPassengers()** should be -1. And it is possible that there is an airport whose airportCode value equals departureAirport, but that airport has no departing passengers. In that case, **countNumberOfDepartingPassengers()** should return 0(since the airport has 0 different customers).
  - Note that Python functions can have more than one SQL statement in them. You will probably have more than one SQL statement in your code for **countNumberOfDepartingPassengers()**.

- **updateReservationPayment(departureDate)**:
  - Unexpected current events obligate the airline to cancel certain flights due to depart, it is necessary to update corresponding reservations to reimburse affected passengers.
  - In the Reservation table, the value of paymentMethod corresponds to the method that a passenger used to pay for the reservation. These correspond to credit codes (V, M, A, D). These are the only values in the Lab4 load data.
  - Besides the database connection, the function **updateReservationPayment()** has the departureDate, which is a string formatted as YEAR-MONTH-DAY (ex. '2025-11-10'). The function **updateReservationPayment()** should do the following:
    - If departureDate matches the date of the scheduledDeparture timestamp field in the Flight table, then the paymentMethod field in the Reservation table should have 'REIMBURSED <departureDate>' appended at the end.
      - For example, assuming that departureDate is '2025-11-10', then every Reservation paymentMethod with a Flight whose scheduledDeparture is on that day should be changed to be '<paymentMethod> REIMBURSED 2025-11-10'. For Reservations with a value 'V' in their paymentMethod field this would look like: 'V REIMBURSED 2025-11-10'.
    - If the year in departureDate is less than(<) 2025 or greater than (>) 2025, then no changes should be made to any status values, and **updateReservationPayment** should return -1.
  - But how can you concatenate strings together in SQL? As the Lecture 6 slide whose title is "UPDATE with Subquery" illustrated, you employ the operator || (which is the vertical OR bar written twice) to concatenate two strings together. That slide used:

    ```
    'Pres. ' || execName
    ```

    to append 'Pres. ' to the beginning of an execName.
  - Note:
    - To get a SQL DATE datatype from a TIMESTAMP you can use the DATE(timestamp) function.
      - Ex. DATE(table.timestamp)
    - To get a SQL DATE datatype from a string you can use the TO_DATE(string, string_date_format) function.
      - Ex. TO_DATE(date_as_string, 'YYYY-MM-DD'

- **promoteCrewMembers(crewAssignments, minYearsExperience)**
    - The airlines wants to promote crew members according to new seniority policy reliant on number of assignments and years of experience
    - Besides the database connection, this Python function has two other parameters, **crewAssignments** and **minYearsExperience** which are both integers.
    - **promoteCrewMembers** invokes a Stored Function, **promoteCrewMembersFunction,** that you will need to implement and store in the database according to the description in Section 6. The stored function **promoteCrewMembersFunction** has the same parameters assignments and years as the **promoteCrewMembers** (but the database connection is not a parameters for the stored function), and **promoteCrewMembersFunction** returns an integer.
    - Section 6 tells you how to apply the promotions, and explains the integer value that **promoteCrewMembersFunction** returns. The **promoteCrewMembers** Python function returns the same integer value that the **promoteCrewMembersFunction** stored function returns.
    - **promoteCrewMembersFunction** doesn't print anything. The **promoteCrewMembers** function must only invoke the stored function **promoteCrewMembersFunction,** which does all of the work for this part of the assignment; **promoteCrewMembers** should not do any of the work itself.

6. Stored Function

As Section 5 mentioned, you should write a Stored Function (not a Stored Procedure) called **promoteCrewMembersFunction** that has parameters crewAssignments and minYearsExperience identifying how crew members should be promoted. The database connection is not a parameter for the Stored Function.

If crewAssignments and minYearsExperience summed up are less than or equal to 0, **promoteCrewMembersFunction** should return the value -1. An individual parameter can be 0 but not both of them. Neither crewAssignments or minYearsExperience can be negative, in this case **promoteCrewMembersFunction** should return value -1. Otherwise, proceed as described below.

The FlightCrewAssignment has a primary key composed from foreign keys flightID and crewId from the Flight and CrewMember table accordingly. For each crewMember we can count the number of assignments a crewMember has had by counting the number of rows in the FlightCrewAssignment table. Years of experience is stored as an integer in the CrewMember table as minYearsExperience.

The airline uses tiered requirements to determine whether someone will receive a promotion. The parameters crewAssignments and minYearsExperience determine the minimum values for someone to receive a promotion. For a crew member to be promoted their number of assignments must be greater than or equal to the crewAssignments, AND their years of experience must be greater than or equal to minYearsExperience.

For example: If crewAssignments = 10, and minYearsExperience = 5. Every crew member who has at least 10 crewAssignments AND at least 5 years of experience will be promoted.

The **promoteCrewMembersFunction** should return the number of crew members that have been promoted for the given pair of crewAssignments and minYearsExperience.

To promote a crewMember prepend the string “Senior ” to the crewRole VARCHAR attribute in the CrewMember table. “Senior “ should be prepended with a blank space following it to avoid the title being merged with the role. You can do this using the same technique outlined for the previous function as follows:

- As the Lecture 6 slide whose title is "UPDATE with Subquery" illustrated, you employ the operator || (which is the vertical OR bar written twice) to concatenate two strings together. That slide used:

  'Pres. ' || execName

  to append 'Pres. ' to the beginning of an execName.

The **promoteCrewMembersFunction** should **NOT** promote crewMembers that already have been promoted. This is denoted by having the prefix “Senior ” in their crewRole entry. It is very important that previously promoted crewMembers are not promoted more than once, as it will affect the correctness of your output.

Suppose that the following five crewMembers could be promoted

crewMember	Role	Assignments	Years of Experience

John Miller	Pilot	5	3
Stacy Jones	Senior Pilot	25	10
Sarah Wilson	Pilot	20	3
Mike Davis	Flight Attendent	3	1
Steve Anderson	Flight Attendent	20	10

Let's discuss some examples of the orders that **promoteCrewMembersFunction** should delete based on the example above.

- If crewAssignments is 10, and minYearsExperience is 2, then **promoteCrewMembersFunction** should promote Sarah Wilson, and Steve Anderson. It should not promote John Miller even if his years of experience are greater than 2. It should not promote Stacy Jones as she has already been promoted. The return value for **promoteCrewMembersFunction** should be 2.

- If crewAssignments is 20, and minYearsExperience is 5, then **promoteCrewMembersFunction** should promote Steve Anderson. It should not promote Sarah Wilson even with her years of experience being equal to 20. It should not promote Stacy Jones as she has already been promoted. The return value for **promoteCrewMembersFunction** should be 1.

- If crewAssignments is 5, and minYearsExperience is 3, then **promoteCrewMembersFunction** should promote John Miller, Sarah Wilson, and Steve Anderson. It should not promote Mike Davis as his assignments and years of experience are below crewAssignments and minYearsExperience. It should not promote Stacy Jones as she has already been promoted. The return value for **promoteCrewMembersFunction** should be 3..

Write the code to create the Stored Function, and save it to a text file named **promoteCrewMembersFunction.pgsql**. To create the Stored Function **promoteCrewMembersFunction**, issue the psql command:

```
\i promoteCrewMembersFunction.pgsql
```

at the server prompt. If the creation goes through successfully, then the server should respond with the message “CREATE FUNCTION”. You will need to call the Stored Function from the **promoteCrewMembers** function in your Python program, as described in the previous section, so you’ll need to create the Stored Function before you run your program. You should include the promoteCrewMembersFunction.pgsql source file in the zip file of your Submission, together with your versions of the Python file runAirlineApplication.py that was described in Section 4. See Section 7 for detailed Submission instructions.

A guide for defining Stored Functions for PostgreSQL can be found here on the PostgreSQL site, but there a better description on this PostgreSQL Tutorial site. For Lab4, you should write a Stored Function that has IN parameters.

We've given you some more hints in Lecture 11 and in the Lab4 announcement on Piazza about writing PostgreSQL Stored Functions, including:

- *fireSomePlayersFunction.pgsql*, an example of a PostgreSQL Stored Function from another quarter, and
- *What_Does_fireSomePlayersFunction_Do.pdf*, an explanation of what that Stored Function does. But we won't provide the tables and load data for running *fireSomePlayersFunction.pgsql*.

7. Testing

Within main for *runAirlineApplication.py*, you should write several tests of the Python functions described in Section 4. You might also want to write your own tests, but only the following tests should be included in the *runAirlineApplication.py* file that you submit in your Lab4 solution.

- Write five tests in *runAirlineApplication.py* of the Python function ***countNumberOfDepartingPassengers***.
  - The first test should be departureAirport value LAX.
  - The second test should be for departureAirport value ATL.
  - The third test should be for departureAirport value EWR.
  - The fourth test should be for departureAirport value SFO.
  - The fifth test should be for departureAirport value SJC.

  1) If a test of *countNumberOfDepartingPassengers* returns a value greater or equal to zero, then print the following message:

     Number of passengers for airport <departureAirport> is <number of passengers>

     where <departureAirport> is the parameter value provided, and <number of customers > is the value returned by *countNumberOfDepartingPassengers*.

  2) However, if a test of *countNumberOfDepartingPassengers* returns a negative value, then you should print out the values of its parameter, with a message explaining the error that occurred. You may choose the format yourself, as long as the error explanation is clear. You should continue executing further tests, even if *countNumberOfCustomers* returns a negative value.

     In both cases 1) and 2), the output should be followed by an extra blank line.

- Write four tests in *runAirlineApplication.py* of the Python function ***updateReservationPayment***.
  - The first test should be for departureDate value '2025-10-8'.
  - The second test should be for departureDate value '2025-10-9'.
  - The third test should be for departureDate value '2026-10-10'.
  - The fourth test should be for departureDate value '2025-10-10'.

  1) If a test of *updateReservationPayment* returns a negative value, then you should print out the values of its parameter, with a message explaining the error that occurred. You may choose the format yourself, as long as the error explanation is clear. You should continue executing further tests, even if *updateReservationPayment* returns a negative value.

  2) But if a test of *updateReservationPayment* returns a non-negative value, then you should print out its result (which is the number of Reservation tuples that were updated) using the following format:

  Number of Reservations whose paymentMethod values were updated by updateReservationPayment is <number of paymentMethod values updated>

  where <number of paymentMethod values updated> is the number of paymentMethod values that were updated.

  [The output for each invocation of *updateReservationPayment* should appear on a single line, not split into two lines, and you should print a blank line after each.]

  In both cases 1) and 2), the output should be followed by an extra blank line.

- Also write five tests in *runAirlineApplication.py* of the Python function **promoteCrewMembers**.
    - The first test should be for crewAssignments = 4, and yearsOfExperience = 10.
    - The second test should be for crewAssignments = 5, and yearsOfExperience = 0.
    - The third test should be for crewAssignments = 0, and yearsOfExperience = 0.
    - The fourth test should be for crewAssignments = 2, and yearsOfExperience = 5.
    - The fifth test should be for crewAssignments = 1, and yearsOfExperience = 2.

    1) If a test of *promoteCrewMembers* returns a non-negative value, then you should print out its result (which is the number of reservations that were deleted) using the following format:

    Number of promotions for crewAssignments < crewAssignments > and minYearsExperience < minYearsExperience > is <number of crew members that have been promoted>

    where <crewAssignments> is the value of the crewAssignments parameter provided, where <minYearsExperience> is the value of the minYearsExperience parameter provided, and <number of crew members that have been promoted is the number of crew members in CrewMember that were promoted.

    [The output for each invocation of *promoteCrewMembers* should appear on a single line, not split into two lines, and you should print a blank line after each.]

    2) But if a test of *promoteCrewMembers* returns a negative value, then you should print out the values of its parameter, with a message explaining the error that occurred, followed by a blank line. You may choose the format yourself, as long as the error explanation is clear. You should continue executing further tests, even if *promoteCrewMembers* returns a negative value.

You must run all of these function tests in order, starting with the database provided by our create and load scripts. Some of these functions change the database, so using the load data that we've provided and executing the functions in order is required. You should not reload the data multiple times in Lab4.

Does the order in which these tests are run matter? What do you think?

8. Submitting

1. Remember to add comments to your Python code so that the intent is clear.
2. Save the Python program *runAirlineApplication.py* and the stored procedure declaration code *promoteCrewMembers.pgsql* in your working directory.
3. Zip the files to a single file with name Lab4_XXXXXXX.zip where XXXXXXX is your 7-digit student ID, for example, if a student's ID is 1234567, then the file that this student submits for Lab4 should be named Lab4_1234567.zip. To create the zip file, you can use the Unix command:

   zip Lab4_1234567 runAirlineApplication.py promoteCrewMembers.pgsql

   Please do not include any other files in your zip file, except perhaps for an optional README file, if you want to include additional information about your Lab4 submission.
4. Some students might want to use views to do Lab4. That's not required, but it is permitted. If you do use views, you must put the statements creating those views in a file called createAirlineViews.sql, and include that file in your Lab4 zip file.
5. Lab4 is due on Canvas by 11:59pm on **Thursday, November 4**. Late submissions will not be accepted, and there will be no make-up Lab assignments.