

1 Preliminaries

In this lab, you will work with a database schema similar to the schema that you used in Lab2. We have provided a **create_lab3.sql** script for you to use (which is similar to, but not quite the same as the **create_lab2.sql** in our Lab2 solution), so that everyone can start from the same place. Please remember to DROP and CREATE the Lab3 schema before running that script (as you did in previous labs), and also execute:

```
ALTER ROLE cse180 SET SEARCH_PATH TO Lab3;
```

so that you will always be using the Lab3 schema without having to mention it whenever you refer to a table.

You will need to log out and log back in to the server for this default schema change to take effect. (Students often forget to do this.)

We have provided a **load_lab3.sql** script that will load data into your tables. You will need to run that script before executing Lab3. The command to execute a script is: `\i <filename>`

In Lab3, you will be required to combine new data (as explained below) into one of the tables. You will need to add some new constraints to the database and do some unit testing to see that the constraints are followed. You will also create and query a view, and create an index.

New goals for Lab3:

1. Perform SQL to “combine data” from two tables
2. Add foreign key constraints
3. Add general constraints
4. Write unit tests for constraints
5. Create and query a view
6. Create an index

There are lots of parts in this assignment, so make sure to start early to ensure you have enough time to finish. Lab3 will be discussed during the Lab Sections before the due date, **Tuesday, November 18**.

2. Description

2.1 Tables with Primary Keys for Lab3

The primary key for each table is underlined. The attributes are the same ones from Lab2, but there's one table that you haven't seen before.

Passenger(passengerID, passengerName, email, dateOfBirth, frequentFlyer)

Airport(airportCode, city, country, numRunways, avgDelayMinutes)

Flight(flightID, departureAirport, arrivalAirport, scheduledDeparture, scheduledArrival, actualDeparture, actualArrival, aircraftType)

CrewMember(crewID, crewName, crewRole, yearsExperience)

Reservation(reservationID, passengerID, flightID, bookingDate, seatClass, ticketPrice, paymentMethod)

CheckIn(reservationID, passengerID, checkInTime, bagCount, seatNumber)

FlightCrewAssignment(flightID, crewID, compensation)

UpdateReservation(reservationID, seatClass, ticketPrice)

In the create_lab3.sql file that we have provided under Resources→Lab3, the first 7 tables are similar to the tables that were in our Lab1 create_lab1.sql solution; the NULL and UNIQUE constraints from Lab2 are not included. Moreover, create_lab3.sql is missing the Foreign Key Constraints on Reservation and CheckIn that were in create_lab1.sql. You will create new variations of those constraints in Section 2.3 of Lab3.

In practice, primary keys, unique constraints, and other constraints are almost always entered when tables are created, not added later. create_lab3.sql handles some constraints for you, but you will be adding some additional constraints to these tables in Lab3, as described below.

Note also that there is an additional table, UpdateReservation, in the create_lab3.sql file that has some of the attributes that are in the Reservation table. We will say more about UpdateReservation below.

Under Resources→Lab3, we have provided a load script named load_lab3.sql that loads tuples into the tables of the schema. You must run both create_lab3.sql and load_lab3.sql before you run the parts of Lab3 that are described below.

2.2 Working with a View

For this part, run the `create_lab3.sql` and `load_lab3.sql` scripts as described in the Preliminaries section.

2.2.1 Create a view

A reservation in the Reservation table has a `ticketPrice` paid by the passenger. We want to identify passengers who book flights frequently - specifically, those who have made 2 or more reservations. These "frequent flyers" are valuable customers for the airline.

Create a view called `FrequentFlyerView`, which identifies passengers who have 2 or more reservations. The attributes in the view should be:

- `passengerID`
- `passengerName`
- `totalReservations` (the count of reservations for this passenger)
- `totalSpent` (the sum of all `ticketPrice` values for this passenger's reservations)

Only include a passenger in `FrequentFlyerView` if they have 2 or more reservations.

No duplicates should appear in your view. Save the script for creating that view in a file called `createview.sql`

2.2.2 Query a View

For this part of Lab 3, you will write a script called **queryview.sql** that contains a query which we will refer to as the FrequentFlyerSummary query. (You do not actually need to name the query, or create another view for this.) The FrequentFlyerSummary query uses FrequentFlyerView, and (possibly) some other tables.

In addition to writing the FrequentFlyerSummary query, you must also include some comments in the queryview.sql script; we will describe those necessary comments below:

Write and run a SQL query which calculates the average amount spent per reservation for each frequent flyer. The attributes in your result should be passengerID, passengerName, totalReservations, and avgSpentPerReservation (calculated as totalSpent divided by totalReservations).

Run this file with that view, then write the results of the FrequentFlyerSummary query in a comment. The format of that comment is not important; the comment just has to have all the right information in it.

Next, write a SQL statement that **deletes** the following tuple from the Reservation table:

- The tuple whose Primary Key (reservationID) is 5.

Run the FrequentFlyerSummary query once again after that deletion. Write the output of the query in a second comment. Do you get a different answer? Think about why the results changed.

You need to submit a script named queryview.sql containing:

1. Your FrequentFlyerSummary SQL query.
2. A comment with the output of that query on the load data before the deletion.
3. A SQL statement which performs the deletion described above.
4. Repeat your FrequentFlyerSummary SQL query.
5. A second comment with the output of that query after the deletion.

It probably was a lot easier to write the FrequentFlyerSummary query using the view than it would have been if you did not have the view!

2.3 Combine Data

Important: Before running this section, recreate the Lab3 schema using the `create_lab3.sql` script, and load the data using the script `load_lab3.sql`. That way, any database changes that you've done for the view section won't propagate to this part of Lab 3.

Write a file, `combine.sql` that will do the following:

For each tuple in `UpdateReservation`, there might already be a tuple in the `Reservation` table that has the same Primary Key (that is, the same value for `reservationID`). If there is not a tuple in `Reservation` with the same Primary Key, then this is a new reservation that should be inserted into the `Reservation` table. If there already is a tuple in `Reservation` with that Primary Key, then this is an update of information about that reservation. So here are the effects that your transaction should have:

1. **If there already is a tuple in the Reservation table which has that reservationID**, then update the tuple in `Reservation` which has that `reservationID` using the `seatClass` and `ticketPrice` values in the `UpdateReservation` tuple. Also, update the `paymentMethod` to 'Credit Card'.
2. **If there is not already a tuple in the Reservation table which has that reservationID**, then you should insert a tuple into the `Reservation` table which has the `reservationID`, `seatClass`, and `ticketPrice` values that are in the `UpdateReservation` tuple. Also, the `passengerID` should be set to `NULL`, the `flightID` should be set to `NULL`, the `bookingDate` should be set to `CURRENT_DATE`, and the `paymentMethod` should be set to 'Credit Card'.

Your transaction may have multiple statements in it.

2.4 Add Foreign Key Constraints

Important: Before running this section, recreate the Lab3 schema using the `create_lab3.sql` script, and load the data using the script `load_lab3.sql`. That way, any database changes that you've done for previous sections won't propagate to this part of Lab 3.

Here's a description of the Foreign Keys that you need to add for this assignment. (Foreign Key Constraints are also referred to as Referential Integrity constraints.) The `create_lab3.sql` file that we've provided for Lab 3 includes only some of the Referential Integrity constraints that were in the Lab 2 solution, but you're asked to use `ALTER` to add additional constraints to the Lab3 schema.

The load data that you've been given should not cause any errors when you add these constraints. Just add the constraints listed below, exactly as described, even if you think that additional Referential Integrity constraints should exist. Note that (for example) when we say that each passenger (`passengerID`) that appears in the `Reservation` table must appear in the `Passenger` table, that means that the `passengerID` attribute of the `Reservation` table is a Foreign Key referring to the Primary Key of the `Passenger` table (which also is `passengerID`).

1. Each passenger (`passengerID`) that appears in the `Reservation` table must appear in the `Passenger` table as a Primary Key (`passengerID`). If a tuple in the `Passenger` table is deleted, then the `passengerID` in all affected `Reservation` tuples should be set to `NULL`. If the Primary Key (`passengerID`) of a tuple in the `Passenger` table is updated, then all `Reservation` tuples for that updated passenger should also be updated to the new `passengerID` value.
2. Each flight (`flightID`) that appears in the `Reservation` table must appear in the `Flight` table as a Primary Key (`flightID`). If a tuple in the `Flight` table is deleted and there are `Reservation` tuples which correspond to that `flightID`, then that flight tuple deletion should be rejected. If the Primary Key (`flightID`) of a tuple in the `Flight` table is updated, then all `Reservation` tuples for that updated flight should also be updated to the new `flightID` value.
3. Each reservation (`reservationID`) that appears in the `CheckIn` table must appear in the `Reservation` table as a Primary Key (`reservationID`). If a tuple in the `Reservation` table is deleted and there are `CheckIn` tuples that correspond to that reservation, then all `CheckIn` tuples which correspond to the deleted reservation should also be deleted. If a Primary Key in the `Reservation` table is updated, then all `CheckIn` tuples which correspond to that reservation Primary Key should also be updated the same way.

Write commands to add foreign key constraints in the same order that the foreign keys are described above. Your foreign key constraints should all have names, but you may choose any names that you like. Save your commands to the file **foreign.sql**

2.5 Add General Constraints

The general constraints for Lab 3, which should be written as CHECK constraints, are:

1. In **Airport**, numRunways must be greater than zero. This constraint should be named **airportRunwaysPositive**.
2. In **Flight**, the value of aircraftType must be 'Boeing 737', 'Airbus A320', 'Boeing 777', 'Airbus A380', or NULL. This constraint should be named **validAircraftType**.
3. In **Reservation**, if ticketPrice is NULL, then paymentMethod must also be NULL. This constraint should be named **ifNullPriceThenNullMethod**.

Write commands to add general constraints in the order the constraints are described above, and save your commands to the file **general.sql**. Remember that values TRUE and UNKNOWN are okay for a CHECK constraint, but FALSE is not.

2.6 Write Unit Tests

Unit tests are important for verifying that your constraints are working as you expect. We will require tests for just a few common cases, but there are many more unit tests that are possible. The unit tests you write need to do what's described below on the specific database instance that contains the Lab 3 load data.

For each of the 3 foreign key constraints specified in section 2.4, write one unit test:

- An INSERT command that violates the foreign key constraint (and elicits an error). You must violate that specific foreign key constraint, not any other constraint.

Also, for each of the 3 general constraints, write 2 unit tests. This means that you will write 2 tests for the first general constraint, followed by 2 tests for the second general constraint, followed by 2 tests for the third general constraint.

- An UPDATE command that meets the constraint.
- An UPDATE command that violates the constraint (and elicits an error).

Save these $3 + 6 = 9$ unit tests in the order specified above in the file `unitests.sql`.

2.7 Create an index

Important: Before running this section, recreate the Lab3 schema using the `create_lab3.sql` script, and load the data using the script `load_lab3.sql`. That way, any database changes that you've done for previous sections won't propagate to this part of Lab 3.

Indexes are data structures used by the database to improve query performance. Locating the tuples in the Reservation table for a particular seatClass and ticketPrice might be slow, if the database system has to search the entire Reservation table (if its size was very large). To speed up that search, create an index named **ReservationIndex** over the seatClass and ticketPrice columns (in that order) of the Reservation table. Save the command in the file `createindex.sql`.

Of course, you can run the same SQL statements whether or not this index exists; having indexes just changes the performance of SQL statements.

Bonus: For this assignment, you need not do any searches that use the index, but if you're interested, you might want to do searches with and without the index, and look at query plans using EXPLAIN to see how queries are executed. Please refer to the documentation of PostgreSQL on EXPLAIN that's at <https://www.postgresql.org/docs/15/sql-explain.html>

3 Testing

Before you submit, login to your database via psql and execute the provided database creation and load scripts, and then test your seven scripts (combine.sql foreign.sql general.sql unittests.sql createview.sql queryview.sql createindex.sql). Note that there are two sections in this document (both labeled **Important**) where you are told to recreate the schema and reload the data before running that section, so that updates you performed earlier won't affect that section. Please be sure that you follow these directions, since your answers may be incorrect if you don't.

4 Submitting

Save your scripts indicated above as:

- **createview.sql**
- **queryview.sql**
- **combine.sql**
- **foreign.sql**
- **general.sql**
- **unitests.sql**
- **createindex.sql**

Don't forget to include comments on your view queries.

Zip the files to a single file with name **Lab3_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 Lab 3 should be named Lab3_1234567.zip.

(Of course, you use your own student ID, not 1234567.)

Lab 3 is due on Canvas by 11:59pm on Tuesday, November 18. Late submissions will not be accepted, and there will be no make-up Lab assignments.