1 Preliminaries

In this lab, you will work with a database schema similar to the schema that you used in Lab2. We've provided a create_lab3.sql script for you to use, 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 yourlogin SET SEARCH_PATH TO lab3;
```

so that you'll always be using the lab3 schema without having to mention it whenever you refer to a table. We've also provided a lab3_data_loading.sql script that will load data into your tables. You'll need to run that before executing Lab3.

You will be required to combine new data (as explained below) into one of the tables. You will also 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, but none of them should be difficult. Lab3 will be discussed during the Lab Sections during the weeks before Sunday, November 19.

2. Description

2.1 Tables with Primary Keys for Lab3

Primary key for each table is underlined.

Airlines (<u>AirlineID</u>, AirlineName)
Airports (<u>AirportID</u>, City, State)
Flights (<u>AirlineID, FlightNum</u>, Origin, Destination, DepartureTime, ArrivalTime)
Customers (<u>CustID</u>, CustName, Status)
Tickets (<u>TicketID</u>, CustID, AirlineID, FlightNum, SeatNum, Cost, Paid)

NewTickets (<u>TicketID</u>, CustID, AirlineID, FlightNum, SeatNum)

The first 5 tables are almost the same as they were in the Lab2 solution, including NULL and UNIQUE constraints. One difference: We deleted the attribute FlightDate from Tickets; that was unnecessary (and confusing), since we're assuming that all flights are on the same day.

Note that there is an additional table, NewTickets. As the table name suggests, each of its tuples records either a new ticket or something new about either an existing ticket. We say more about NewTickets below.

You're given a script named create_lab3.sql as part of Lab3, which creates all 6 of these tables in a schema named **Lab3.** As in the previous assignments, in order to avoid mentioning the schema every time you refer to the tables, you can make **Lab3** the default schema in the search path by issuing the following command:

ALTER ROLE your_user_id SET SEARCH_PATH TO Lab3;

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

You will also be provided with a load script named lab3_data_loading.sql that will load a set of values to the tables of the schema. That file will be available soon on the Resources page on Piazza under Labs. You must run both create_lab3.sql and lab3_data_loading.sql before you run the following parts of Lab3.

2.2 Combine Data

Write a file, *combine.sql* (which may have multiple sql statements in it in a serializable transaction) that will do the following: For each "new ticket" tuple t that's in NewTickets, either there already is a matching tuple in Tickets that has the same primary key as t, or there isn't a matching tuple in Tickets.

- If there already is a matching tuple in Tickets that has the same primary key, then do the following:

  a) If the CustID, AirlineID and FlightNum values in Tickets and NewTickets <u>don't</u> both match, then leave the existing Tickets tuple as is. (We're treating the new ticket as an error.)

  b) If the CustID, AirlineID and FlightNum values in Tickets and NewTickets <u>do</u> both match, then update the Tickets tuple, setting SeatNum to be the value that is in the new ticket. Leave Cost (and other attributes) as they are, but set Paid to be FALSE. (Yes, that does seem a little unfair.)

- If there is no matching tuple in Tickets that has the same primary key, then insert a tuple corresponding to the NewTickets tuple t into Tickets. For that tuple, set Cost to be NULL and Paid to be FALSE.

2.3 Add Foreign Key Constraints

Here's a description of the Foreign Keys that you need to add for this assignment. The default for referential integrity should be used in all cases. The data that you're provided with should not cause any errors. You must run Sections 2.3, 2.4 and 2.5 after running Section 2.2.

a) The CustID field in Tickets should reference the CustID primary key in Customers.

b) The AirlineID field in Flights should reference the AirlineID primary key in Airlines.

c) (Two attribute foreign key): The (AirlineID, FlightNum) fields in Tickets should reference the (AirlineID, FlightNum) primary key in Flights.

Write commands to add foreign key constraints in the same order that the foreign keys are described above. Save your commands to the file *foreign.sql*

2.4 Add General Constraints

General constraints are:

1. Cost in Tickets must be positive. (Note that UNKNOWN for a Check constraint is okay.)

2. ArrivalTime in Flights must be greater than DepartureTime

Note: Please give a name to this positive rent constraint when you create it. We recommend that you use the name flights_take_time, but you may use another name. The other constraints don't need names.

3. In Airports, if State is 'TN' then City is 'Knoxville'.

4. In Flights, the Origin and Destination must be different.

Write commands to add general constraints in the order the constraints are described above, and save your commands to the file *general.sq*l.

2.5 Write unit tests

Unit tests are important for verifying that your constraints are working as you expect. We will write just a few for the common cases, but there are many more possible tests we could write.

For each of the 3 foreign key constraints specified in section 2.3, write <u>one</u> unit test:

- o An INSERT command that violates the foreign key constraint (and elicits an error).

Also, for each of the 4 general constraints, write <u>2</u> unit tests:

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

Save these 3 + 8 = 11 unit tests, in the order given above, grouped by constraint, in the file *unittests.sql*.

2.6 Working with views

2.6.1 Create a view

Create a view named CA_NY_Passengers that gives the CustID, AirlineID and FlightNum for each customer that has a ticket on a flight from an airport that's in California (CA) to an airport that's in New York State (NY). AirlineID and FlightNum identify the flight from CA to NY, and a passenger may have several different flights. Save the script for creating this view in a file called *createview.sql*.

2.6.2 Query a view

Write a query over the CA_NY_Passengers view to answer the following question about "Passenger Flight Total":

- o For each customer that's in CA_NY_Passengers, give that person's CustID, name and the number of flights that they have between California airports and New York State airports. In your result, that last attribute should appear as CA_NY_Total.

Important: Before running this query, recreate the Lab3 schema using the *create_lab3.sql* script, and load the data using the script *lab3_data_loading.sql*. That way, any changes that you've done for other parts of Lab3 (e.g., Unit Test) won't affect the result of this query. Then write the results of that query in a comment.

Next, write commands that delete just the tuples that have the following primary TicketID values from the Tickets table:

202

204

Run the ""Passenger Flight Total" query once again after those deletions. Write the output of the query in a second comment. Do you get a different answer?

You will need to submit a script named *queryview.sql* containing your query on the view. In the submitted file *queryview.sql*, apart from the SQL query on the view, **include the comment with the output of the query on the provided data before the deletions, the SQL statements that delete the two tuples indicated above, and a second comment with the second output of the same query after the deletions**. You do not need to replicate the query twice in the *queryview.sql* file (but you will not be penalized if you do).

2.7 Create an index

Indexes are data structures used by the database to improve query performance. Locating all the Tickets on a particular flight may be slow if we have to search the entire Tickets table. To speed up that search, create an index named LookUpFlightTickets over the AirlineID and FlightNum columns of the Tickets table. Save the command in the file *createindex.sql*.

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 here.

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). Make sure that you run the query on the view make sure you recreate the schema and reload the data, as the updates you choose to do for testing the general constraints may change the initial data in a way that changes the output of the query on the view. The command to execute a script is: \i <filename>.

4 Submitting

1. Save your scripts indicated above as combine.sql foreign.sql general.sql unittests.sql createview.sql queryview.sql createindex.sql. You may add informative comments inside your scripts if you want (the server interprets lines that start with two hyphens as comment lines).

2. 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 Labe should be named Lab3_1234567.zip To create the zip file you can use the Unix command:

   ```
   zip Lab3_1234567 combine.sql foreign.sql general.sql unittests.sql createview.sql queryview.sql
   createindex.sql
   ```

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

3. You should already know how to transfer the files from the UNIX timeshare to your local machine before submitting to Canvas.

4. Lab3 is due on Canvas by 11:59pm on Sunday, November 19. Late submissions will not be accepted, and there will be no make-up Lab assignments.