1 Preliminaries

In this lab, you will work with a Real Estate database schema similar to the schema that you used in Lab2. We've provided a lab3_create.sql script for you to use (which is similar to but quite the same as the create.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 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.

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'll also provide a lab3_data_loading.sql script that will load data into your tables. You'll 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, but none of them should be difficult. Lab3 will be discussed during the Lab Sections before the due date, Sunday, November 22. The due date of Lab3 is 3 weeks after the due date of Lab2 because of the Midterm, and also because we didn't cover all of the Lab3 topics before the Midterm.

2. Description

2.1 Tables with Primary Keys for Lab3

The primary key for each table is underlined. There's one table that you haven't seen before.

Houses(houseID, address, ownerID, mostRecentSaleDate)

Persons(personID, personName, houseID)

Brokers(brokerID, brokerLevel, companyName, soldCount)

Offers(houseID, offererID, offerDate, offerPrice, isACurrentOffer)

ForSaleHouses(houseID, forSaleDate, brokerID, forSalePrice, isStillForSale)

SoldHouses(houseID, forSaleDate, soldDate, buyerID, soldPrice)

SoldHouseChanges (houseID, forSaleDate, buyerID, soldPrice)

In the lab3_create.sql file that we've provided under Resources→Lab3, the first 6 tables are the same as they were in our Lab2 solution, except that the NULL and UNIQUE constraints from Lab2 are **not** included.

In practice, primary keys, unique constraints and other constraints are almost always entered when tables are created, not added later. lab3_create.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, SoldHouseChanges, in lab3_create.sql that has most (but not all) of the attributes in the SoldHouses table. As with SoldHouses, any (houseID, forSaleDate) that's in a SoldHouseChanges row must appear as a (houseID, forSaleDate) in the ForSaleHouses table. In other words, (houseID, forSaleDate) in SoldHouseChanges is a Foreign Key referencing the Primary Key of ForSaleHouses, as you can observe by looking at lab3_create.sql. We'll say more about SoldHouseChanges below.

Under Resources→Lab3, you'll also be given a load script named lab3_data_loading.sql that loads tuples into the tables of the schema. **You must run both lab3_create.sql and lab3_data_loading.sql before you run the parts of Lab3 that are described below.**

2.2 Combine Data

Write a file, *combine.sql* (which should have multiple SQL statements that are in a Serializable transaction) that will do the following. For each tuple in SoldHouseChanges, there might already be a tuple in the SoldHouses that has the same primary key (that is, the same value for houseID and forSaleDate). If there **isn't** a tuple in SoldHouses with the same primary key, then this is a new "sold house" that should be inserted into SoldHouses. If there already **is** a tuple in SoldHouses with that primary key, then this is an update of information about that "sold house". So here are the actions that you should take.

- If there **isn't** already a tuple in the SoldHouses table which has that houseID and forSaleDate, then insert a tuple into the SoldHouses table corresponding to that SoldHouseChanges tuple. Use buyerID and soldPrice, as provided in the SoldHouseChanges tuple. Set soldDate to NULL

- If there already **is** a tuple in the SoldHouses table which has that houseID and forSaleDate, then update the tuple in SoldHouses that has that houseID and forSaleDate. Update buyerID and soldPrice for that existing SoldHouses tuple based on the value of those attributes in the SoldHouseChanges tuple, and set soldDate to today. (An announcement on Piazza tells you how to set a DATE attribute value to "today" using CURRENT_DATE.)

Your transaction may have multiple statements in it. The SQL constructs that we've already discussed in class are sufficient for you to do this part (which is one of the hardest parts of Lab3).

2.3 Add Foreign Key Constraints

Important: Before running Sections 2.3, 2.4 and 2.5, recreate the Lab3 schema using the *lab3_create.sql* script, and load the data using the script *lab3_data_loading.sql*. That way, any database changes that you've done for Combine won't propagate to these other parts of Lab3.

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.sql file that we've provided for Lab3 includes the same Referential Integrity constraints that were in the Lab2 solution, but you're asked to make DDL changes (using ALTER) that will add additional constraints to the Real Estate schema.

The load data that you're provided with should not cause any errors when you add these constraint. 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 every broker (brokerID) in the Brokers table must appear in the Persons table, that means that the brokerID attributes of the Brokers table is a Foreign Key referring to the Primary Key of the Persons table.

- Each broker (brokerID) in the Brokers table must appear in the Persons table as a personID. (Explanation of what that means appear in the above paragraph.) If a Persons tuple is deleted and there is a Broker that refers to that person, then the deletion should be rejected. Also, if the Primary Key of a Persons tuple is updated, and there is a broker corresponding to that person, then the update should be rejected.

- Each offerer (offererID) that's in the Offers table must appear in the Persons table as a personID. If a tuple in the Persons table is deleted, then all Offers for the corresponding offerID should also be deleted. If the Primary Key of a Persons tuple is updated, then all Offers tuples for the corresponding offererID should be updated the same way.

- Each buyer (buyerID) that's in the SoldHouses table must appear in the Persons table as a personID (or be NULL. If a tuple in the Persons table is deleted, then all tuples in the SoldHouses table for the corresponding buyerID should also be set to NULL. If the Primary Key of a Persons tuple is updated, then all SoldHouses tuples for the corresponding buyerID should be updated the same way.

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 for Lab3 are:

1.In SoldHouses, soldPrice must be greater than zero. Please give a name to this constraint when you create it. We recommend that you use the name positive_soldPrice, but you may use another name. The other general constraints don't need names, but you may name them if you'd like.

2. In Brokers, the value of brokerLevel must be either 'A', 'B', 'C', 'D' or NULL.

3. In Offers, if offerPrice is NULL, then isACurrentOffer must also be NULL.

Write commands to add general constraints in the order the constraints are described above, and save your commands to the file *general.sq*l. (Note that UNKNOWN for a Check constraint is okay, but FALSE isn't.)

2.5 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.

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

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

Also, for each of the 3 general constraints, write 2 unit tests:

- 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 given above, in the file *unittests.sql*.

2.6 Working with Views

2.6.1 Create a View

Although the Brokers table has a soldCount attribute, there's another way that we can calculate the total number of houses sold by a broker. Each "for sale house" tuple in ForSaleHouses has a brokerID, which tells you which broker was selling the house. The ForSaleHouses for which there is a corresponding tuple (same value of houseID and forSaleDate) in SoldHouses actually were sold. So the ComputedSoldCount for a broker can be calculated by counting the number of "for sale houses" of that broker were sold.

Create a view called viewComputedSoldCount that has 2 attributes, brokerID and computedSoldCount. This view should have a tuple for each brokerID in the Brokers table that gives the computedSoldCount for that brokerID. Your view should have no duplicates in it.

As you've probably already deduced, you'll need to use a GROUP BY in your view. But there's an additional requirement: Don't include a brokerID in your view unless its computedSoldCount is at least 2.

Save the script for creating the viewComputedSoldCount view in a file called *createview.sql*

2.6.2 Query a View

For this part of Lab3, you'll write a script called *queryview.sql* that contains a query that uses viewComputedSoldCount (and possibly some tables). In addition to that query, you'll also have to include some comments in the queryview.sql script; we'll describe those comments below.

Write and run a SQL query over the viewComputedSoldCount view to answer the following "Sold Count Misreports for Each Broker" question. You may want to use some tables to write this query, but be sure to use the view.

> For some brokers, the soldCount that appears in the Brokers table is not the same as the computedSoldCount for that broker in viewComputedSoldCount; we'll say that such brokers are **misreported**. For each broker that is misreported, you should output their brokerID, personName, soldCount and computedSoldCount.
>
> Note that there could be a brokerID in the Brokers table that doesn't appear in viewComputedSoldCount, because computedSoldCount for that brokerID is less than 2. We won't consider that brokerID to be misreported, so you shouldn't output a tuple for that broker.

Important: Before running this query, recreate the Lab3 schema once again using the *lab3_create.sql* script, and load the data using the script *lab3_data_loading.sql*. That way, any changes that you've done for previous parts of Lab3 (e.g., Unit Test) won't affect the results of this query. Then write the results of that query in a comment. *The format of that comment is not important; it just has to have the right information in it.*

Next, write commands that deletes just the tuples that have the following Primary Keys from the SoldHouses table:

- the tuple whose Primary Key is (444, DATE '2016-08-16')
- the tuple whose Primary Key is (222, DATE '2020-05-08')

Run the "Sold Count Misreports for Each Broker" query once again after those deletions. Write the output of the query in a second comment. Do you get a different answer?

You need to submit a script named *queryview.sql* containing your query on the views. In that file you must also include:

- the comment with the output of the query on the provided data before the deletions,
- the SQL statements that delete the 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 won't be penalized if you do).

Aren't you glad that you had the viewComputedSoldCount view? It probably was easier to write this query using that view than it would have been if you hadn't had that view.

2.7 Create an Index

Indexes are data structures used by the database to improve query performance. Locating the tuples in the Members table that have a particular name and address might be slow if the database system has to search the entire Members table. To speed up that search, create an index named FindPersons over the personName and houseID columns (in that order) of the Persons 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. But this index could make it faster to determine if any person has a particular personName and houseID, or find all persons who have a particular personName. Also, the Database Management System (DBMS) could use the FindPersons index to help ensure that (personName, houseID) is UNIQUE (i.e., that there can't be two tuples that have both the same personName and the same houseID, if both attributes aren't NULL).

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/11/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

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 Lab3 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 22. Late submissions will not be accepted, and there will be no make-up Lab assignments.