1 Preliminaries

In this lab, you will work with a database schema similar to the schema that you used in Lab2. You will be given a create_Lab3.sql script to use, so that everyone can start from the same place. You'll also be given a load_values_Lab3.sql script that will load data into your tables. Those two files (for create and load) will be available on the Resources page on Piazza on Monday, Feb 6.

You will be required to merge data (as explained below) into one of the tables. You will also need to add 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 Lab 3:

1. Perform SQL to "merge 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. You will have an extra week to complete this assignment, because of the Midterm. There will be Lab Sections all 3 weeks between Monday, Feb 6 and the Lab3 due date, Sunday, Feb 26.

2. Description

2.1 Tables with Primary Keys for Lab3

Primary key for each table is underlined.

Persons (SSN, Name, HouseID, ApartmentNumber, Salary)
Houses (HouseID, HouseAddress, ApartmentCount, Color)
Landlords (LandlordID, OwnerSSN, LandlordAddress)
Ownerships (LandlordID, HouseID, PurchaseDate, PropertyTax)
Tenants (HouseID, ApartmentNumber, LeaseTenantSSN, LeaseStartDate, LeaseExpirationDate, Rent, LastRentPaidDate, RentOverdue)

NewRentPayments (HouseID, ApartmentNumber, LeaseTenantSSN, Rent, DatePaid)

The first 4 tables are the same as they were in the Lab2 solution, including NULL and UNIQUE constraints. The table Tenants is different from the one of Lab2 as it does not include the UNIQUE(HouseID, LeaseTenantSSN) constraint. We chose to omit that constraint as it would complicate the merge task that you will do (described below).

Note that there is an additional table, NewRentPayments. As the table name suggests, each of its tuples records a rent payment made for an apartment in a house on a specified date. There will be at most one

payment for a particular house/apartment, since (HouseID, ApartmentNumber is the primary key of NewRentPayments.

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. However, we will be adding some additional constraints to these tables, as described below.

You will also be provided with a load script named load_values_Lab3.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.

2.2 Merge Data

Write a file, *merge.sql* that will do the following: For each tuple t in NewRentPayments, either there already is a tuple in Tenants that has the same primary key as t, or there isn't such a tuple in Tenants.

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

  a) If the LeaseTenantSSN and the Rent values in Tenants and NewRentPayments don't both match, then leave the that Tenants tuple as is.

  b) If the LeaseTenantSSN and the Rent values in Tenants and NewRentPayments do both match, then update the Tenants tuple, setting LastRentPaidDate to be DatePaid and RentOverdue to be FALSE.

- If there is no tuple, insert a row corresponding to that NewRentPayments tuple t into Tenants. Make LeaseStartDate be the current date, LeaseExpirationDate be NULL, and RentOverdue be FALSE. Assume that the LeaseTenantSSN and Rent are correct. Please refer to the documentation of PostgreSQL here to learn how to obtain the current date.

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 should run Sections 2.3, 2.4 and 2.5 after running Section 2.2.)

a) The SSN fields in Landlords and Tenants (that is, OwnerSSN and LeaseTenantSSN) should reference the SSN field in Persons.

b) The HouseID fields in Persons, Ownerships and Tenants should reference the HouseID field in Houses.

c) The LandlordID field in Ownerships should reference the LandlordID field in Landlords.

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

2.4 Add General Constraints

General constraints are:

1.PropertyTax in Ownerships must be zero or more.

2. Rent in Tenants must be positive.

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

3. In Tenants, if LeaseExpirationDate isn't NULL then it must be later than LeaseStartDate.

4. If LastRentPaidDate in Tenants is the current date, then RentOverdue must be FALSE.

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

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 HouseID 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 4 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 + 8 = 11 unit tests, in the order given above and grouped together by constraint, in the file *unittests.sql*.

2.6 Working with views

2.6.1 Create a view

Create a view named Home_Landlords that gives the SSN, LandlordID, HouseID, ApartmentNumber for Landlords who live in a house that they own. The SSN should be the Landlord's SSN, and the HouseID and ApartmentNumber should be for the location where the Landlord lives. Save the script for creating the view in a file called *createview.sql*.

2.6.2 Query a view

Write a query over the Home_Landlords view to answer the following question about "Overdue Landlords":

- "Give the name and rent for each landlord who lives in a house that he owns, and who also has overdue rent on a tenancy for which he is the lease tenant. The name should be the landlord's name, and the rent should be the rent for a tenancy that is overdue.

Note that a Landlord might be the lease tenant for more than one house/apartment that has overdue rent.

Before running this query, recreate the Lab3 schema using the *create_Lab3.sql* script, and load the data using the script *load_values_Lab3.sql*. That way, any changes that you've done for other parts of Lab3 won't affect the result. Then run the above "Overdue Landlords" query, and write the results in a comment.

Next, write commands to delete just the tuples that have the following primary keys from the Tenants table:

(1000,2)

(1100,2)

Run the "Overdue Landlords" query once again after those deletions. Note down 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 the person who lives in a particular house and apartment may be slow if we have to search the entire Persons table. To speed up that search, create an index over the HouseID and ApartmentNumber columns of the Persons table. Save the command in the file *createindex.sql*. (You should create this index after completing Section 2.7.)

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 (merge.sql foreign.sql general.sql unittests.sql createview.sql queryview.sql createindex.sql). Before running 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 merge.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 Lab2_1234567 merge.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, February 26. Late submissions will not be accepted, and there will be no make-up Lab assignments.