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 lab3_create.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 Wednesday, February 28. (You have an extra week to do this Lab because of the Midterm, which was on Wednesday, February 14.)

2. Description

2.1 Tables with Primary Keys for Lab3

The primary key for each table is underlined.

TaxReturns(taxpayerID, taxYear, kind, dateFiled, income, taxOwed)

Payments(taxpayerID, datePaid, amountPaid)

Individuals(taxpayerID, name, address, spouseID, lastDateFiled, lastDatePaid, totalTaxOwed)

Businesses(taxpayerID, name, address, businessType, lastDateFiled, lastDatePaid, totalTaxOwed)

IRSagents(IRSagentID, taxpayerID, jobLevel, active)

Delinquents(taxpayerID, IRSagentID, collectionAgencyID)

NewPayments(taxpayerID, datePaid, amountPaid)

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, including NULL and UNIQUE constraints. Note that there is an additional table, NewPayments that has the same attributes as Payments. As the table name suggests, each of its tuples records a new payment for an individual. We say more about NewPayments below.

As in previous assignments, 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 lab3_create.sql handles those constraints for you. However, we will be adding some additional constraints to these tables, as described below.

Under Resources→Lab3, you've also been 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 may have multiple sql statements in it in a serializable transaction) that will do the following. For each "new payment" tuple t that's in NewPayments, either a) there isn't a tuple in Payments that has the same primary key, or b) there is a tuple in Payments with the same primary key.

a) If there isn't already a tuple in Payments that has the same primary key, then insert a tuple into the Payments table corresponding to that tuple in the NewPayments table.

b) If there is already a tuple in Payments that has the same primary key, then update Payments, adding to the new amountPaid to the old amountPaid. For example, suppose that (for a particular taxpayerID and date), there already was a Payment tuple whose amountPaid was 500.00. If the new payment tuple for that taxpayerID and date is 200.00, then the amountPaid in the Payments tuple for that taxpayerID and date should become 500.00 + 200.00 = 700.00.

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 probably the hardest part of Lab3). A helpful hint is provided in the initial Lab3 announcement posted on Piazza.

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. The default for referential integrity should be used in all cases. The data that you're provided with should not cause any errors.

a) The taxpayerID field in IRSagents should reference the taxpayerID primary key in Individuals.

b) The IRSagentID field in Delinquents should reference the IRSagentID primary key in IRSagents.

c) The collectionAgencyID field in Delinquents should reference the taxpayerID primary key in Businesses.

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. amountPaid in Payments must be positive. Please give a name to this positive rent constraint when you create it. We recommend that you use the name positive_payments, but you may use another name. The other constraints don't need names.

2. In Businesses, lastDateFiled must be greater than or equal to lastDatePaid. (Do this just for Businesses, not for Individuals.)

3. In IRSagents, if jobLevel is NULL then active must also be NULL.

4. In Individuals, taxpayerID and spouseID 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.sql*. (Note that UNKNOWN for a Check constraint is okay.)

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 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, in the file *unittests.sql*.

2.6 Working with views

2.6.1 Create two views

Create a view named TaxDebts. For each taxpayerID in TaxReturns, this view should provide taxpayerID and the sum of taxOwed in the TaxReturns for that taxpayerID. In your result, the second attribute should be called debt. But only include a tuple for a taxpayerID if that taxpayerID has at least 4 tax returns in TaxReturns.

Create a second view named PaymentCredits. For each taxpayerID in Payments, this view should provide taxpayerID, the biggest value of datePaid for that taxpayerID in Payments, and the total of amountPaid for that taxpayerID in Payments. In your result, the attributes should be called taxpayerID, biggestDatePaid and credit. But only include a tuple for a taxpayerID if that taxpayerID has at least 2 different amountPaid values in Payments.

Save the script for creating these view in a file called *createviews.sql*.

2.6.2 Query views

Write a query over the TaxDebts and PaymentCredits views to answer the following "Delinquent Tax Summary" question:

> Some tax payers have tuples appearing in the Delinquents table and also in the TaxDebts and PaymentCredits views. For each tax payer that has tuples appearing in all 3, output that tax payer's taxpayerID, their debt (from TaxDebts) and their credit (from PaymentCredits).

Important: Before running this query, recreate the Lab3 schema 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 result of this query. Then write the results of that query in a comment.

Next, write commands that delete just the tuples with the following primary keys from the TaxReturns table:

(112, 2016)
(116, 2017)

Run the "Delinquent Tax Summary" 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 *queryviews.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 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 *queryviews.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 TaxReturns for individuals or businesses that were filed on or after a particular date may be slow if the database system has to search the entire TaxReturns table. To speed up that search, create an index named LookUpReturns over the kind and dateFiled columns (in that order) of the TaxReturns table. Save the command in the file *createindex.sql*.

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

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 createviews.sql queryviews.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 createviews.sql queryviews.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 createviews.sql queryviews.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 Wednesday, February 28. Late submissions will not be accepted, and there will be no make-up Lab assignments.