1. Preliminaries

First, download the file *Lab4.zip* from Piazza. That zip file contains the schema and data for the database we will use for this assignment. The schema is the same as the one we used for Lab3 but without the NewRentPayments table. The zip file also contains the PostgreSQL JDBC library, as well as other files, including the *HousesApplication.java* and *RunHousesApplication.java* files mentioned in later sections of this document.

The *create_Lab4.sql* script will create all tables within the schema Lab4, and *load_values_Lab4.sql* will load data into those tables, just as for Lab3. Alter your search path so that you can work with the tables without qualifying them with the schema name:
ALTER ROLE <username> SET SEARCH_PATH TO Lab4;

You must log out and log back in for this to take effect. To verify your search path, use:

SHOW SEARCH_PATH;

Note: It is important that you do not change the names of the tables. Otherwise, your application may not pass our tests, and you will not get any points for this assignment.

2. Instructions to compile and run JDBC code

Once you place all files of the Lab4.zip file into your working folder, unzip the included jar file with the command:
> unzip postgresql-9.3-1103.jdbc3.jar

Modify *RunHousesApplication.java* with your own database credentials. Compile the Java code, and ensure it runs correctly. It will not do anything useful with the database yet, except for logging in and disconnecting, but it should execute without errors.

If you have changed your password for your database account with the "ALTER ROLE username WITH PASSWORD <new_password>;" command in the past, and you now use a sensitive password (e.g. the same password as your Blue or Gold UCSC password, or your personal e-mail password), make sure that you do not include this in the *RunHousesApplication.jav*a file that you submit to us, as this information will be unencrypted.

You can compile the *RunHousesApplication.java* program with the following command:
> javac RunHousesApplication.java

To run the compiled file, issue the following command:
> java RunHousesApplication

Note that if you do not modify the username and password to match those of your PostgreSQL account in the program, the execution will return an authentication error.
If the program uses methods from the *HousesApplication* class and both programs are located in the same folder, any changes you make to *HousesApplication.java* can also be

compiled with a javac command similar to the one above.

You may get ClassNotFound exceptions if you attempt to run your programs locally and there is no JDBC driver on the classpath, or unsuitable driver errors if you already have a different version of JDBC locally that is incompatible with the class server cmps180-db.lt.ucsc.edu. To avoid such complications, we advise that you use the provided *postgresql-9.3-1103.jdbc3.jar* file that contains a compatible JDBC library.

3. Goal

The fourth lab project puts the database you have created to practical use. You will implement part of an application front-end to the database.

4. Description of methods in the HousesApplication class

HousesApplication.java contains a skeleton for the *HousesApplication* class, which contains methods that interact with the database using JDBC. The methods are not implemented, and one of your tasks is to implement these methods.

The methods in the *HousesApplication* class are the following:

- *getLeaseTenantsWithApartmentsInManyHouses:* This method has an integer argument called *numberOfHouses* and returns the SSNs of Tenants who lease apartments in at least *numberOfHouses* different Houses.

  This means that if the value of *numberOfHouses* is 3, and there is a tenant who leases 3 apartments in 3 different houses, then the method should return the SSN of that tenant. But if there is another tenant who leases 2 apartments in the same house and leases a third apartment in a different house, the method should not return the SSN of that tenant, since he/she leases apartments in only 2 different houses.

- *raiseSalary*: This method takes a name and a raise as arguments, and raises the salary of the persons who have that specific name, incrementing the salary of each by the amount specified in the argument. Since name is not UNIQUE in persons, there may be multiple people with the same name. If there are no people with that name, *raiseSalary* should do nothing. The update should be performed as a single SQL statement.

- *movePersonToApartment*: This method takes an SSN, an address and an apartment number as arguments, and invokes a stored function movePerson (that you will need to implement and store in the database according to the description in section 5) that moves a person to the specified house and apartment. The movePersonToApartment method returns the HouseID of the address/apartment where the person moved if the move is successful, otherwise it retuns -1. This method should only invoke the stored function movePerson, which does all of the moving work, including checking whether the move is possible. Do not implement the move with SQL statements through JDBC

Each method is annotated with comments in the HousesApplication.java file with a description indicating what it is supposed to do (repeating most of the descriptions above). Your task is to implement methods that match the descriptions. The default constructor is already implemented.

A brief guide to using JDBC with PostgreSQL can be found here on the PostgreSQL site. This guide should be useful when implementing the methods.

5. Stored Function

You should write a stored function called *movePerson*, whose inputs are an SSN, an address and an ApartmentNumber. The stored function should move the person with that SSN to that ApartmentNumber in the house that is located at that address. The move should happen according to the following rules:

- If there is no person with that SSN, then movePerson should do nothing.
- If there is no house with that address, then movePerson should do nothing.
- If the ApartmentNumber is impossible, then movePerson should do nothing. (An ApartmentNumber is impossible if it's less than 1, or if it's more than the ApartmentCount for its house.)
- If the SSN, the address and the ApartmentNumber are valid, then the move succeeds, and the person with the SSN should be moved to that ApartmentNumber in the house that is located at that address.

The stored function must return the HouseID of the house where the person moved if the move was successful, and return -1 if the move could not proceed (either because the SSN does not exist, or the address is not in the database, or the apartment number is invalid). Note that if we attempt to move a person to the house that the person is already living in, it should be considered as a valid move.

Write the code to create the stored function, and save it to a text file named *moveperson.pgsql*. Execute the creation of movePerson by issuing the command:

```
\i moveperson.pgsql
```

at the server prompt. If the creation goes through successfully, then the server should respond with the message "CREATE FUNCTION". You will need to call the stored function with the movePersonToApartment method through JDBC (as described in the previous section). You should include the moveperson.pgsql source file in the zip file of your submission, along with your versions of the Java source files *HousesApplication.java* and *RunHousesApplication.java* described in Section 4.

A guide for defining stored procedures/functions with PostgreSQL can be found here on the PostgreSQL site. Note that PostgreSQL stored procedures/functions have some syntactic differences from the PSM stored procedures/functions described in class.

6. Testing

The file *RunStoresApplication.java* contains sample code on how to set up the database connection and call application methods **for a different database and different queries**. It is provided only for illustrative purposes, since it can give you an idea of how to invoke the methods that you want to test in this assignment.

RunHousesApplication.java is the program that you will need to modify in ways that are similar to the content of the *RunStoreApplication.java* program, but for this assignment. You should write tests to ensure that your methods work as expected. In particular, you should:

- Write one test of the *getLeaseTenantsWithApartmentsInManyHouses* method with the numberOfHouses argument set to 4. Your code should print the results. Remember that your method should run correctly for any other value of numberOfHouses.

  You should also include the output you got with the numberOfHouses argument set to 4 in a comment directly in the Java source of *RunHousesApplication*. The comment should be placed in your code immediately after the invocation of the method for numberOfHouses set to 4. The comment should be as follows:

  ```
  /*
  *Output of getLeaseTenantsWithApartmentsInManyHouses
  *when the parameter numberOfHouses is 4.
  output here
  */
  ```

- Write one test for the raiseSalary method that raises the salary of persons named "John Smith" by 10,000.

- Write two tests for the *movePersonToApartment* method. The first test should attempt to move the person with SSN 631952071 to apartment 1 at 401 Heller Drive. The first test should go through, as there is a person with the said SSN and an apartment with this number in that address. The second test should move the person with SSN 563960185 to apartment 10 at 401 Heller Drive. This second attempt should fail as the house at 401 Heller Drive has only 2 apartments. Your code should print the results returned by the tests.

7. Submitting

1. Remember to add comments to your Java code so that the intent is clear.
2. Place the java programs HousesApplication.java and RunHousesApplication.java, and the stored procedure declaration code moveperson.pgsql on your working directory at unix.ucsc.edu.
3. Zip the files to a single file with name Lab4_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 Lab4_1234567.zip. To create the zip file you can use the Unix command:

zip Lab4_1234567 HousesApplication.java RunHousesApplication.java moveperson.pgsql

4. Lab4 is due on Canvas by 11:59pm on Sunday, March 12. Late submissions will not be accepted, and there will be no make-up Lab assignments.