Lab 2 – DD1334 – SQL, Datascience, Python, Transactions, Views

We will use the same dataset as in Lab1, so please revise the instructions there for the setup.
Important: You and your lab partner will be expected to be able to answer questions about your code and if you understand the results.
Goals: This lab focusses on some more advanced problems involving SQL in a Data Science setting.

Before you start, please make sure you have read the relevant chapters of the book pertaining to SQL material and transactions that we have covered in the lectures.

Question 1 – Getting Data Ready in PostgreSQL
a) Create a MATERIALIZED VIEW called **PopData** about city population developments where we augment the yearly population data in citypops with additional information about cities and the economy of the countries within which they lie. We will use this view as the basis for our data science exploration.

Each row (one per year of data and city in a country) should contain

from citypops:

- **year**: year of population data sample
- **name**: city name
- **population**: population for that year
- **country**: country code

Additionally add to each row, associated data from city:

- **longitude:** of city in question
- **latitude:** of city in question
- **elevation:** of city in question

And add most recent information about country economy in which city lies from economy relation:

- **agriculture, service, industry:** percentages of gdp-composition of these sectors
- **inflation:** percentage

Sample query and output on the view, note that there are multiple cities called 'Santiago', but one per country:

```
select * from popdata where name like 'Santiago%';
 country | year |        name         | population | longitude | latitude | elevation | agriculture | service | industry | inflation
---------+------+---------------------+------------+-----------+----------+-----------+-------------+---------+----------+-----------
 PA      | 1990 | Santiago            |      60959 |    -80.97 |     8.11 |       101 |         3.7 |    78.4 |     17.9 |       4.1
 PA      | 2000 | Santiago            |      74679 |    -80.97 |     8.11 |       101 |         3.7 |    78.4 |     17.9 |       4.1
 PA      | 2010 | Santiago            |      88997 |    -80.97 |     8.11 |       101 |         3.7 |    78.4 |     17.9 |       4.1
 C       | 1994 | Santiago de Cuba    |     440084 |    -75.81 |    20.02 |        82 |         3.8 |    73.9 |     22.3 |         6
 C       | 2002 | Santiago de Cuba    |     423392 |    -75.81 |    20.02 |        82 |         3.8 |    73.9 |     22.3 |         6
 C       | 2007 | Santiago de Cuba    |     446345 |    -75.81 |    20.02 |        82 |         3.8 |    73.9 |     22.3 |         6
 C       | 2009 | Santiago de Cuba    |     446233 |    -75.81 |    20.02 |        82 |         3.8 |    73.9 |     22.3 |         6
 DOM     | 1981 | Santiago            |     278638 |     -70.7 |    19.47 |       175 |           6 |    64.9 |     29.1 |         5
 DOM     | 1993 | Santiago            |     365463 |     -70.7 |    19.47 |       175 |           6 |    64.9 |     29.1 |         5
 DOM     | 2002 | Santiago            |     507418 |     -70.7 |    19.47 |       175 |           6 |    64.9 |     29.1 |         5
 DOM     | 2010 | Santiago            |     550753 |     -70.7 |    19.47 |       175 |           6 |    64.9 |     29.1 |         5
 RA      | 1987 | Santiago del Estero |     148000 |    -64.27 |   -27.78 |       187 |         9.3 |      61 |     29.7 |      20.8
 RA      | 1991 | Santiago del Estero |     189947 |    -64.27 |   -27.78 |       187 |         9.3 |      61 |     29.7 |      20.8
 RA      | 2001 | Santiago del Estero |     230424 |    -64.27 |   -27.78 |       187 |         9.3 |      61 |     29.7 |      20.8
 RCH     | 1987 | Santiago            |    4318000 |    -70.67 |   -33.45 |       521 |         3.6 |      61 |     35.4 |       1.7
 RCH     | 2002 | Santiago            |    4659048 |    -70.67 |   -33.45 |       521 |         3.6 |      61 |     35.4 |       1.7
(16 rows)
```

b) Why did we choose a Materialized View instead of a Virtual View for data science applications?

Python Interface Tips for next question:

Look into **figure.py** (download from Canvas) to understand how to extract basic (x,y) data points from a relation using python. Note in particular how **try … except … pass** is used to process potential SQL errors. Look into **menu.py** (download from Canvas) to see an example of a simple text menu interface to specify parameters for a particular SQL query. You can use the ideas in the above files to create a little menu with options for parts of your solutions to questions below.

Question 2 – Data Exploration in Python
In the following be careful to correctly handle/filter NULL missing values and to cast data to python numeric values correctly. Try to make your code well-documented and robust to errors.

a) Generate function to create a scatterplot of x=year against y=total population of all cities in PopData. Also plot the number of cities for which we have data per year. What can you say about this? Try to also create an SQL query and accompanying plot that aggregates information by decade and displays the mean population and standard deviations per decade using an matplotlib errorbar plot

A hint: group by years/10 and round appropriately).

Are city populations on the rise on average? What about large vs small cities?

A hint: check out the aggregation functions in https://www.postgresql.org/docs/9.4/functions-aggregate.html **there are some for mean, standard deviations, covariance, etc.**

b) Let us attempt to predict a city's population by fitting a line **y = ax + b** to the y=population, x=year data per city.

PostgreSQL has aggregation functions to find a best-fitting line approximation to a set of x coordinates X and y coordinates Y: regr_slope(Y, X) (gives **a**), regr_intercept(Y, X) (gives **b**) and the degree of determination r2 measures how closely the data follows a linear trend r2 = regr_r2(Y, X) (1 = data follows linear trend, 0 does not follow linear trend at all). Documentation: https://www.postgresql.org/docs/9.4/functions-aggregate.html

Create a new virtual view LinearPrediction(cityname, country a, b, r2, nsamples, yearfrom, yearto, minpop, maxpop) with these quantities for each city as well as:
nsamples: the number of population years in which data exists per city
yearfrom,yearto: min and max year for which we have data
minpop, maxpop: observed min and max population for the city

Sample query:

```
select * from linearprediction where name like 'Santi%';
        name          | country |        a         |         b          |         r2         | nsamples |yearfrom|yearto| minpop  | maxpop
----------------------+---------+------------------+--------------------+--------------------+----------+--------+------+---------+---------
 Santiago de Cuba     | C       | 586.791044776119 | -736328.962686567 |  0.13143276227465 |        4 |  1994 | 2009 |  423392 |  446345
 Santiago             | RCH     | 22736.5333333333 | -40859491.7333333 |                  1 |        2 |  1987 | 2002 | 4318000 | 4659048
 Santiago             | PA      |           1401.9 | -2728921.66666667 | 0.999848392319604 |        3 |  1990 | 2010 |   60959 |   88997
 Santiago             | DOM     | 9952.59139784946 | -19444780.7258065 |  0.96822488638924 |        4 |  1981 | 2010 |  278638 |  550753
 Santiago del Estero | RA      | 5533.63461538462 | -10839076.7884615 | 0.937412794310795 |        3 |  1987 | 2001 |  148000 |  230424
(5 rows)
```

c) Now create a text menu in python that asks a user a city and associated country and creates a figure that displays all the population,year datapoints for the city as a scatterplot with x-axis = year, y-axis = population as well as predicted behavior of population as a line y = ax+b on a graph.

Also display the yearly predicted population for 2020 – 2030 in the terminal and the r2 value and nsamples based on which the analysis is done to quantify the amount of trust one may have in this prediction. Note that some cities only have very few year, population samples.

Using SQL queries, identify cities for which we have sufficient data, for which r2 is close to 1 and which are having either a strongly declining or inclining population trend. What are some examples? Plot the result.

d) Create a hypothesis about what factors may be correlated to population changes and generate a python program that allows the user to explore this hypothesis in terms of a few

sample queries and visualizations with some user specified query parameters that confirm or disprove the hypothesis. Your program should correctly process errors (use try … except … pass error handling) that may be thrown if data is not found or the input is invalid. You can use all the information from Popdata and create new relations/views as desired.

Some ideas: You could group the cities into quartiles according to a size, growth-trend etc and study them. You could ask if there are correlations between (longitude, latitude) position of a city and population growth. What are the mean and standard deviation properties of city size over different longitude-latitude rectangles? What can we say about the cities with rapidly declining populations? Is there a connection to the type economy in which they lie in terms of service vs agriculture percentages of gdp?
Fun extra idea if you enjoy this investigation: Use the python machine learning libraries scikit-lean to see if gaussian process regression can give you a better prediction of population data and test with current 2018 data for some cities you can find on the web. https://scikit-learn.org/stable/modules/gaussian_process.html

e) The file transactions.py shows how to open two simultaneous connections to a database in a single file, simulating two independent users. Sequentially executing SQL statements over the two connections allows us to specify the execution order like in the simple case discussed in lectures.
Create a small educational python demonstration program using a real database connection that shows the user in a concrete case:

- a phantom tuple that occurs in Nonrepeatable Read and then disappears when we switch to Serializable.
- How nonrepeatable reads can affect the outcome of a query in Read Commited vs Repeatable Read
- How a transaction with two insert statements of which the second fails can be rolled back without affecting another user's queries.
- Note that dirty reads do actually not occur in PostgreSQL as READ UNCOMMITTED is not implemented, but could appear in other SQL DBMS.

Consider R(a, b) with tuples

```
 a | b
---+---
 0 | 1
 0 | 2
 1 | 10
 1 | 20
```

with steps:

a1: xval = result of SELECT SUM(b) FROM R WHERE a = 0;
b1: INSERT INTO R Values(1, xval)
a2: xval2 = SELECT SUM(b) FROM R WHERE a = 1;
b2: INSERT INTO R Values(0, xval2)

Suppose a1, b1 form one transaction t1 of user 1 and a2, b2 another transaction t2 by user 2. If the transactions are executed serially in the two possible orders (t1)(t2) and (t2)(t1), what are the final tuples in R in either case? What should be the result if we could execute a1 a2 b1 b2 in this order instead in serializable isolation node, where we still assume a1, b1 is part of transaction 1 by user 1 and a2 b2 part of transaction 2 by user 2? Note that this result is not equal to either of the possible serial execution results! This contradicts the assumptions of a serializable transaction. Try to see what happens if you run the two transactions with this

interleaving in time in PostgreSQL. What is the result if we change to other isolation modes?