CPSC 321: Homework 3 *Due: Wednesday, Oct. 1, 2025*

Instructions. This homework involves both a write-up and an SQL coding component. You must submit both parts to Canvas by the end of the day (11:59 PM) on the due date. Your write-up must be typed (not handwritten), organized, and clearly marked (i.e., with respect to which question and part is being answered). Additionally, include your first and last name, the course (CPSC-321, Fall 2025), and the assignment (HW-2) at the top of your write-up. Your write-up must be submitted as a single PDF file called `hw-3.pdf`. For the SQL coding part, you must complete the starter scripts (`hw3_1.sql` and `hw3_2.sql`), which will be provided via Piazza. Your scripts must include the information described below and also have the file header comments filled out. Failing to fill out the file header comments (and not following directions in general) will result in an overall reduction in points for the assignment.

1. [5pts] Implement the following schema using `CREATE TABLE` statements for your PostgreSQL database. The schema is loosely based on the CIA World Factbook.[1] Your statements must go into a script file, with appropriate comments, called `hw3_1.sql`. You must pick suitable attribute data types and include primary keys, foreign keys, and constraints as specified below.

`country(country_code, country_name, gdp, inflation)`

- A country has a two-letter country code (e.g., "US"), a full name (e.g., "United States of America"), a gross domestic product per capita (e.g., 46,900 dollars per person), and inflation rates (e.g., 3.8 percent), where `country_code` is the primary key. Assume that two countries with different country codes can have the same country name. All attributes are required.

`Province(province_name, country_code, area)`

- A province (which is referred to as a "state" in the US) consists of a name (e.g., "Washington"), the country code the province is located in, and the total province area in km^2. The `province_name` and `country_code` together form the primary key, with `country_code` a foreign key to the `Country` table. Assume it is possible for two countries to have a province with the same name (e.g., Montana exists in both the US and Bulgaria).

`City(city_name, province_name, country_code, population)`

- A city is identified by its name, province, and country, and has a total population. The `province_name` and `country_code` together define a foreign key to the `Province` table. Assume it is possible for two provinces to have a city with the same name (e.g., Portland is a city in both Oregon and Maine). All attributes are required.

`Border(country_code_1, country_code_2, border_length)`

[1] https://www.cia.gov/the-world-factbook/

- A border defines a connection between two countries, with a corresponding border length in km. Both `country_code_1` and `country_code_2` are separately foreign keys to the `Country` table. Assume there is only one row in the table for a given border between two countries (i.e., the table does not store a symmetric closure over the border relation). All attributes are required.

Populate your tables using `INSERT INTO` statements with enough data to test your table constraints. At a minimum, you must include at least four different countries, four different provinces per country, and four different cities per province. You must also include at least two borders. Include your insert statements in your `hw3_1.sql` file. Note you do not have to use "real" data when populating your tables. If you use real data, it does not have to be "comprehensive", e.g., you do not need to include all provinces within a country, and you do not need to include all cities within a province.

2. [10pts] Each question below asks you to write an SQL query against your world factbook tables from Part 1 above. For some queries you may need to add additional rows to your tables. In general, you must add enough rows to ensure your queries are working correctly (which may take some thought). A good rule of thumb is that each query should return multiple rows and there should be some rows in the corresponding tables that do not match the query conditions. Place all of your SQL queries to the questions below in the file `hw3_2.sql`. It must be possible for the grader to run both your `hw3_1.sql` and `hw3_2.sql` files to recreate your tables and queries. For each query in `hw3_1.sql`, you must provide a *comment* giving:

> *(i) the question number (Query 1–10), (ii) a description of the purpose of the query (e.g., based on the question), and (iii) any other relevant comments or assumptions made.*

Note that some queries below ask for "specific values" (e.g., a high GDP, low inflation, small area, etc.). For these cases, pick a value that makes sense for your data. Finally, for full credit, when writing your queries you must:

> *(i) use as few joins as possible (i.e., do not include unnecessary tables in your* `FROM` *clauses), (ii) only use* `DISTINCT` *and* `ORDER BY` *when needed, and (iii) only use the SQL constructs/syntax we have covered so far in class.*

Finally, be sure you give each answer below in the order given, and in general, try to make grading your work easy for the grader.

1. Write an SQL query using a "comma join" (i.e., without join syntax) to find all provinces that have a small total area that are in a country with high inflation. Your query should return the country code, country name, inflation, province name, and area and results should be sorted from highest to lowest inflation, then alphabetically by country code (for countries with the same inflation), and then by smallest to largest area.

2. Rewrite your query in (1) to use appropriate JOIN syntax instead of a comma join. Since the result should be the same as for the previous query, you do not need to provide an example input that doesn't satisfy the query.

3. Write an SQL query using "comma joins" that finds the unique set of all provinces that have at least one city with a population greater than a specific value (note that a province is identified by both the province name and the country it is in). Return the country code, country name, province name, and province area. Your query must use comma joins and should only return one row per matching province.

4. Rewrite your query from (3) using JOIN syntax for all of the joins. Since the result should be the same as for the previous query, you do not need to provide an example input that does not satisfy the query.

5. Write an SQL query that finds the unique set of all provinces with at least two cities having a population greater than a specific value. Return the country code, country name, province name, and province area. Your query must use comma joins, and must return only one row per matching province.

6. Rewrite your query from (5) using JOIN syntax for all of the joins. Since the result should be the same as for the previous query, you do not need to provide an example input that does not satisfy the query.

7. Write an SQL query that finds unique pairs of different cities with the same population. A city is considered to be different if it has a different name, is in a different province, and/or is in a different country. As examples, Portland and Salem are different cities, and Portland Oregon and Portland Maine are also different cities. Return the city name, province name, and country code for each city along with the population of both cities (so seven attributes in total). If city A and B have the same population (and are different cities), your query should only return the pair once (i.e., it should not return both (A, B) and (B, A)). Your query must use JOIN syntax.

8. Write an SQL query using only "comma joins" that finds all countries with a high GDP and low inflation that border a country with a low GDP and high inflation. Your query should return the country code and country name. Your query should only return unique countries (i.e., one row per matching country). Note that as specified in Part 1, the border table is not assumed to be symmetric.

9. Rewrite your query from (9) using JOIN syntax for all of the joins. Since the result should be the same as for the previous query, you do not need to provide an example input that does not satisfy the query.

10. Develop an interesting query over the database that involves multiple joins, multiple value comparisons, returns a subset of the attributes, returns only unique rows, and requires

sorting. In your comment for the query include a sentence or two describing the purpose of the query (similar to how the questions are phrased above).

3. Your `hw-3.pdf` file must include:

   - A brief description of any challenges you encountered and how you dealt with them.
   - A brief description of the data you used for parts 1 and 2, including how many rows are in each table and whether you ended up having to add any addition rows for part 2.
   - For each query in part 2, include: (i) the result of running the query (e.g., as a screen shot or the result copy-pasted from PostgreSQL); and (ii) an example of a relevant input (row or rows) to the query that do not match the query conditions (unless stated otherwise in the query). The goal of the latter is to get you thinking about how to test and verify whether your queries are actually working correctly.