ECS 165A Database Systems — Fall 2013
Dr. Bertram Ludäscher — November, 2013

Group Project: Mondial/GM (GapMinder)

Due Date: Friday, December 6 (Submission details: see class site and mailing list)

The goal of the first phase of the project was to get GapMinder datasets into Postgres. Now our focus will shift to data quality, data integration, and other practical issues. First some terminology: The GapMinder (short: GM) data files provide data about **indicators**, e.g., *life expectancy* (short: LE), *population* (POP), *income* (GDP per capita), *health spending*, *electricity use*, etc. This data is given for different countries and years. Thus, we can think of the GM data organized into multiple tables as follows:

- GM_LE(country, year, LE), GM_POP(country, year, POP), GM_GDP(...), ... (*DECOMP*)

Alternatively, instead of this decomposed schema, we can think of these tables as combined into one large table:

- GMH(country, year, LE, POP, GDP, HS, EL, ...) (*Horizontal*)

Last not least, instead of the "wide, horizontal" table, we can create a "vertical" one:

- GMV(country, year, indicator, value) (*Vertical*)

Other structures are possible as well, e.g., GMA(country, indicator, value_array), but we will focus on DECOMP, GMH, and GMV, above. Our goal is to use these three representations and see how they compare, e.g., with respect to extensibility, ease of query formulation, and efficiency of query execution. Since it would be tedious and error-prone to maintain all three variants independently, your task will be to pick one **base representation** (which should be one that is easy to extend, e.g., by adding new values for existing indicators, or by adding new indicators), and then to implement at least one other variant as a **materialized view** over the base relation(s). That is, you should write triggers that automatically update the materialized view tables in response to updates to the base relations.

Deliverable D3 (GM Base) Pick one of the above three representations, i.e., *DECOMP*, *Horizontal*, or *Vertical* as your ***GM base***, using the table names provided above. Then populate your table(s) with the GM data given earlier on the class site. NOTE: You may have already picked one of those in the first project phase. You can use this as D3 (and modify as needed, to adjust table names).

Deliverable D4 (GM View) Pick one of the remaining two representations and create:

(a) a (conventional) **view** from the GM base, or

(b) a **system-level materialized view** (PostgreSQL 9.3) from the GM base,

(c) a **user-defined materialized view** (prior PostgreSQL versions).

If you choose (b), use `REFRESH MATERIALIZED VIEW` to maintain the view as the base tables change. If you use (c), then updates to your GM base should be automatically propagated via **triggers** to your GM view. In particular, inserts into the GM base should automatically trigger inserts into your GM view. In the `README.txt` file that goes with your submission, explain how your solution works.

Deliverable D4-EC (GM View, Extra Credit)
Pick the remaining third representation and solve D4 (GM View/Materialized View) with it.

Deliverable D5 (Exploring GM Countries) GM data is often (and inherently) incomplete, e.g., because the corresponding indicators are not available for a given country and year. To explore what GM countries are there, formulate the following as SQL queries:

1. How many distinct GM countries are present in the given GM files?
2. For each indicator, how many GM countries have data for that indicator?
3. What are the countries that have at least *some* (non-NULL) data for *all* the indicators?
4. How many (non-NULL) data points are there per year (i.e., across all GM countries)?

Deliverable D6 (Mondial++: Mondial + GM Integration) Mondial uses country codes to uniquely identify a country, along with the country names, where GM uses only country names. Data integration challenges arise, e.g., due to possible different spellings between Mondial and GM country names, as well as the transient or evolving nature of countries (USSR, Germany, Korea, etc.), their official names, etc.

Create and populate a "bridge table" that associates Mondial country codes with GM country names. Take into account the GM documentation[1] and think what additional columns you might need to describe an association between a Mondial code and a GM country name. For example, you could have a comment field, explaining possible "issues" with the association, or you could have one or more columns describing in what years this association is valid, etc. As part of this deliverable, document:

1. how you populated the table, and
2. what "problem cases" you have detected and how you are dealing with them.

For example, you can populate the bridge table directly using Postgres and/or use manual or script-based means to deal with problem cases. In addition you might want to try out Google-Refine[2] which can be a good way to deal with spelling and other data cleaning issues. If you choose to use Google-Refine, mention it in your `README.txt` documentation.

Deliverable D7 (Mondial + GM Queries) Implement the following queries in PostgreSQL:

1. For each (Mondial++) country return the minimal, maximal, and average latitude of all cities in that country. The result should be ordered by continent (first) and country (second).
2. Extend the previous query to also return the Life Expectancy, Income, and Electricity for each country in the year 2000.
3. For each continent, compute the average income (GDP/capita) in the year 2000 ...
   (a) ... by simply averaging over the country income data in the year 2000 (unweighted average), and
   (b) ... by taking into account the population of the country (weighted average).

Deliverable D8-EC (EXTRA CREDIT: Skyline Queries) In the following, your task is to compute the names of countries which are in the *skyline* of a certain query. Here, a skyline is defined by giving two attributes (e.g. Life Expectancy and Health Spending) and indicating for each whether the min or max is used. Details and examples of expressing skyline queries in SQL will be given in class or during discussion sections.

(a) Compute the country skyline for (Life_Expectancy MAX, Health_Spending MAX) for the year 2000.

(b) Same as (a) but for (Life_Expectancy MAX, Health_Spending MIN).

Deliverable D9-EC (EXTRA CREDIT: MyGapMinder) In a visualization tool of your choosing, display the XY-plot for all years and all (or some) countries ...

(a) ... where X is the Year and Y is the Life_ Expectancy, and

(b) ... where X is the Health_Spending and Y is the Life_Expectancy

For simplicity, you can "loosely-couple" Postgres with your visualization tool, by exporting the answers of your SQL queries to a file.

[1]`http://www.gapminder.org/data/countries-territories-in-gapminder-world/`
[2]`http://code.google.com/p/google-refine/`