COMP1204: Data Management
Coursework Two: COVID Cases Database

Daniel Braghis
32161204

May 19, 2022

1 The Relational Model

1.1 EX1

Relation	Type	Relation	Type
dateRep - day	One-many	year - cases	Many-many
dateRep - month	One-many	year - deaths	Many-many
dateRep - year	One-many	year - countriesAndterritories	Many-many
dateRep - cases	Many-many	year - geoId	Many-many
dateRep - deaths	Many-many	year - countryterritoryCode	Many-many
dateRep - countriesAndTerritories	Many-many	year - popData2020	Many-many
dateRep - geoId	Many-many	year - continentExp	One-many
geoId	Many-many	cases - deaths	Many-many
dateRep - countryterritoryCode	Many-many	cases - countriesAndTerriotories	Many-many
dateRep - popData2020	Many-many	cases - geoId	Many-many
dateRep - continentExp	One-many	cases - countryterritoryCode	Many-many
day - month	Many-many	cases - popData2020	Many-many
day - year	Many-many	cases - continentExp	One-many
day - cases	Many-many	deaths - countrieAndTerritories	Many-many
day - deaths	Many-many	deaths - geoId	Many-many
day - countriesAndTerritories	Many-many	deaths - contryterritoryCode	Many-many
day - geoId	Many-many	deaths - popData2020	Many-many
day - contryterritoryCode	Many-many	deaths - continentExp	One-many
day - popData2020	Many-many	countAndTerr - geoId	One-one
day - continentExp	One-many	countAndTerr - contryterritoryCode	One-one
month - year	Many-many	countAndTerr - popData2020	One-one
month - cases	Many-many	countAndTerr - continentExp	One-many
month - deaths	Many-many	geoId - countryterritoryCode	One-one
month - countriesAndTerritories	Many-many	geoId - popData2020	One-one
month - geoId	Many-many	geoId - continentExp	One-many
month - countryterritoryCode	Many-many	countterrCode - popData2020	One-one
month - popDat2020	Many-many	countterrCode - continentExp	One-many
month - continentExp	One-many	popData2020 - continentExp	One-many

Table 1: Attribute data types for dataset

Attribute	Data type
dateRep	TEXT
day	INTEGER
month	INTEGER
year	INTEGER
cases	INTEGER
deaths	INTEGER
countriesAndTerritories	TEXT
geoId	TEXT
countryterritoryCode	TEXT
popData2020	INTEGER
continentExp	TEXT

1.2 EX2

The table of functional dependencies below assumes the following:

- The day, month and year attributes are not NULL.
- The popData2020 attribute can't be a determinant because we assume the country populations could be the same at some point, thus making the population count unable to uniquely identify a country.
- Determinant attributes are assumed to be unique.
- The cases and deaths attributes are either null or a natural number value.
- The countriesAndTerritories attribute contains only countries and we assume there will be no territories stored in it later so it can be determined by geoId or countryterritoryCode attributes.
- There can be more than one continent.

Functional Dependency	Functional Dependency
day,month,year -> dateRep	dateRep -> day
dateRep,countriesAndTerritories -> cases	dateRep -> month
dateRep,countriesAndTerritories -> deaths	dateRep -> year
dateRep,countryterritoryCode -> cases	dateRep,geoId -> cases
dateRep,countryterritoryCode -> deaths	dateRep, geoId -> deaths
coutriesAndTerritories -> countryterritoryCode	geoId -> countriesAndTerritories
countriesAndTerritories -> continentExp	geoId -> countryterritoryCode
countryterritoryCode -> continentExp	countryterritoryCode -> geoId
countriesAndTerritories -> popData2020	contryterritoryCode -> popData2020
geoId -> continentExp	geoId -> popData2020

1.3 EX3

Table 2: Candidate keys from FDs

Candidate keys
day, month, year, countriesAndTerritories
day, month, year, countryterritoryCode
day, month, year, geoId
dateRep, countriesAndTerritories
dateRep, geoId
dateRep, countryterritoryCode

1.4 EX4

Our candidate keys are essentially made of two parts. The date and the country representation through different attributes. For the first part of the key we picked the dateRep attribute because in our case it is more convenient to use a single attribute instead of day, month and year separately. However, using the atomic attributes of dateRep would make the querying easier when working with external systems.

For the second part of the key we opted for the countriesAndTerritories attribute because it is the most descriptive and unambiguous one compared to the geoId and countryterritoryCode attributes. Finally, we ended with the dateRep, countriesAndTerritories primary key for our database.

Table 3: Selected primary key from candidate keys

Primary key
dateRep, countriesAndTerritories

2 Normalisation

2.1 EX5

The primary key selected has two partial dependencies:

- dateRep -> day, month, year
- countriesAndTerritories -> geoId, countryterritoryCode, popData2020, continentExp

We will separate these two relations and have the third resulting one:

- dateRep, countriesAndTerritories -> cases, deaths

2.2 EX6

Additionally to the process above, we will introduce 2 surrogate keys of INTEGER type for dateRep and countriesAndTerritories relations we outlined the relations for above - date_id and country_id. This will ensure database integrity in the case of changing date formatting, changes in country naming scheme and simplify querying. The surrogate keys will now be the prime keys of their respective relations.

Thus, we can now replace the primary key attributes in the third relation with the surrogate keys to have the same benefits of using surrogate keys in the third relation. This will result in a table with these relations:

- date_id -> dateRep, day, month, year
- country_id -> countriesAndTerritories, geoId, countryterritoryCode, popData2020, continentExp
- date_id, country_id -> cases, deaths

2.3 EX7

Transitive dependencies identified:

- geoId -> countriesAndTerritories, geoId -> countryterritoryCode, coutriesAndTerritories -> countryterritoryCode
- coutriesAndTerritories -> countryterritoryCode, countriesAndTerritories -> continentExp, countryterritoryCode -> continentExp
- geoId -> countriesAndTerritories, geoId -> continentExp, countriesAndTerritories -> continentExp

2.4 EX8

The transitive dependencies identified fill not affect our relations being in 3NF because all of the attributes are prime attributes so dependencies between them won't cause anomalies.

2.5 EX9

We are also in BCNF because no prime attribute is transitively dependent on a key. All attributes dependent of date_id are dependent on it and it is not dependent on any key. The key attributes popCode2020 and continentExp can't transitively define the prime attributes related to country attributes. Finally, cases and deaths are not key attributes.

3 Modelling

3.1 EX10

1. Start an sql instance with the sqlite3 command in terminal.
2. Enter CSV mode: .mode csv
3. Create a table called dataset with the dataset attributes shown at **EX1** and their respective types and specify that they have to be non null.
4. Import the dataset: .import dataset.csv dataset
5. Set the output file: .output dataset.sql
6. Dump the dataset to file: .dump

3.2 EX11

Creating the tables with the relations from **EX6** with the attributes and their respective types. geoId and countryCode can be undefined but unique while all the other attributes have to be non null. The ids in Dates and Countries tables are the surrogate keys as prime attributes that autoincrement on assignment. CasesAndDeaths has a composite primary key with the two ids as foreign keys. When the keys are deleted, the rows associated with them are also removed from CasesAndDeaths. Nothing happens on updates.

```
CREATE TABLE IF NOT EXISTS Countries (
  id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  countriesAndTerritories TEXT NOT NULL UNIQUE,
  geoId TEXT UNIQUE,
  countryCode TEXT UNIQUE,
  popData2020 INTEGER NOT NULL,
  continentExp TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS Dates (
  id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  dateRep TEXT NOT NULL UNIQUE,
  day INTEGER NOT NULL,
  month INTEGER NOT NULL,
  year INTEGER NOT NULL
);
CREATE TABLE IF NOT EXISTS CasesAndDeaths (
  date_id INTEGER,
  country_id INTEGER,
  cases INTEGER,
  deaths INTEGER,
  PRIMARY KEY (date_id, country_id),
  FOREIGN KEY (date_id) REFERENCES Dates(id)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  FOREIGN KEY (country_id) REFERENCES Countries(id)
    ON DELETE CASCADE
    ON UPDATE NO ACTION
);
```

3.3 EX12

```
/* Select distinct entries from the dataset except the first row
   containing the head. Provide NULL as the surrogate key. */
INSERT INTO Dates SELECT DISTINCT NULL,dateRep,day,month,year FROM
   dataset LIMIT -1 OFFSET 1;
/* Same approach as the above table*/
INSERT INTO Countries SELECT DISTINCT NULL,countriesAndTerritories,
   geoId,countryterritoryCode,popData2020,continentExp FROM dataset
   LIMIT -1 OFFSET 1;
/* Insert contents into the CasesAndDeaths by inner joining the 2
   populated tables above with the dataset and selecting the ids, cases
    and deaths.*/
INSERT INTO CasesAndDeaths
SELECT
  Dates.id,
```

```
  Countries.id,
  dataset.cases,
  dataset.deaths
FROM dataset
INNER JOIN Dates ON Dates.dateRep = dataset.dateRep
INNER JOIN Countries ON Countries.countriesAndTerritories = dataset.
   countriesAndTerritories;
```

3.4 EX13

```
/* Run the terminal commands. */
sqlite3 coronavirus.db < dataset.sql
sqlite3 coronavirus.db < ex11.sql
sqlite3 coronavirus.db < ex12.sql
```

4 Querying

4.1 EX14

Aggregate the sum of cases and deaths from CasesAndDeaths and cast to integer to remove the floating point zero.

```
SELECT CAST(SUM(cases) AS INTEGER),CAST(SUM(deaths) AS INTEGER) FROM
   CasesAndDeaths;
```

4.2 EX15

Inner join the deaths and countries with CasesAndDeaths by id. keep only the rows with United Kingdom as their country and order them by year first, then month, then day. Show the dates and cases.

```
SELECT dateRep,cases
FROM CasesAndDeaths
INNER JOIN Countries ON country_id=Countries.id
INNER JOIN Dates ON date_id=Dates.id
WHERE countriesAndTerritories="United_Kingdom"
ORDER BY year,month,day;
```

4.3 EX16

Inner join the deaths and countries with CasesAndDeaths by id. Order the rows by year first, then month, then day. Show the dates, countries, cases and deaths for each row.

```
SELECT dateRep,countriesAndTerritories,cases,deaths
FROM CasesAndDeaths
INNER JOIN Countries ON country_id=Countries.id
INNER JOIN Dates ON date_id=Dates.id
ORDER BY year,month,day;
```

4.4 EX17

Inner join just the Countries table as we are not interested in the dates. Group the rows by country with their id to compute the next operations on each group. Aggregate the sum of cases and deaths, convert it to float by multiplying by 1.0, divide by the population for the country, multiply by 100 to get the percentage and round to 2 non-significant figures.

```
SELECT
  countriesAndTerritories,
  ROUND(SUM(cases)*1.0/popData2020*100, 2),
  ROUND(SUM(deaths)*1.0/popData2020*100, 2)
FROM CasesAndDeaths
INNER JOIN Countries ON country_id=Countries.id
GROUP BY country_id;
```

4.5 EX18

Inner join just the Countries table as we are not interested in the dates. Group the rows by country with their id to compute the next operations on each group. Compute the percentages like we did above and save the reference as percent. Order the results in descending order and take the top 10 with LIMIT.

```
SELECT
  countriesAndTerritories,
  ROUND(SUM(deaths)*1.0/SUM(cases)*1.0*100, 2) AS percent
FROM CasesAndDeaths
INNER JOIN Countries ON country_id=Countries.id
GROUP BY country_id ORDER BY percent DESC LIMIT 10;
```

4.6 EX19

Inner join the deaths and countries with CasesAndDeaths by id. Keeping only the rows that have United Kingdom as the country with the WHERE condition. Ordering the dates by year, month and day to organize the rows chronologically. Aggregating the sum for deaths and cases but the OVER command with the options below will computed the running total for the previous rows plus the current one.

```
SELECT
  dateRep,
  SUM(deaths) OVER (ROWS UNBOUNDED PRECEDING),
  SUM(cases) OVER (ROWS UNBOUNDED PRECEDING)
FROM CasesAndDeaths
INNER JOIN Countries ON country_id=Countries.id
INNER JOIN Dates ON date_id=Dates.id
WHERE countriesAndTerritories="United_Kingdom"
ORDER BY year,month,day;
```

5 Extension

5.1 EX20

```
#!/bin/bash

# Group by country and compute the total amount of deaths for each.
# Sort descending and take the top 10 with LIMIT.
# Get the ids of the countries found.
# Replace newlines with spaces to be read by gnuplot.
IDS=$(sqlite3 coronavirus.db "SELECT country_id FROM CasesAndDeaths
   INNER JOIN Countries ON country_id=Countries.id GROUP BY country_id
   ORDER BY SUM(deaths) DESC LIMIT 10;" | tr '\n' ' ')

# Same approach as above but keeping the name of the countries.
# Replacing "_" with "-" for proper formatting in the gnuplot key.
NAMES=$(sqlite3 coronavirus.db "SELECT countriesAndTerritories FROM
   CasesAndDeaths INNER JOIN Countries ON country_id=Countries.id GROUP
    BY country_id ORDER BY SUM(deaths) DESC LIMIT 10;" | tr '\n' ' ' |
    tr '_' '-')

# Running a gnuplot instance inside the bash instance.
gnuplot -persist <<-EOFMarker
  set key top left autotitle columnheader #Key positioning
  set key reverse Left #Reverse the key titles and line color
   positions in key
  set title 'Cumulative COVID-19 Deaths Top 10' #Graph title
  set ylabel 'Deaths' #Y label
  set xlabel 'Date' #X label
  set grid #Show the graph grid
  set xdata time #Declare the data type of x axis values
  set datafile separator "|" #Declare the data separator
  set format x '%d/%m/%Y' #Declare the time format of X axis
  set timefmt "%d/%m/%Y" #Declare the time format to expect
  set xtics mirror rotate by -45 #Rotate the xtics labels
  set rmargin at screen 0.94 #Resize the graph to fit the png
  set term png #Format of output
  set terminal png size 1024,768 #Size of image
  set output "graph.png" #Output file name
  titles = "$NAMES" #Saving the bash variable into the gnuplot
   variable
  ids = "$IDS" #Saving the bash variable into the gnuplot variable
  ttl(n) = sprintf("%s", word(titles, n)) #Parsing the string from
   bash to a gnuplot array
```

```
# Plot loop that will run for each of the 10 countries and plot the data on the same graph
# For each country run a sqlite query that will request the date and the aggregated death like in EX19
# The country of interest is identified with the id previously saved in the ids variable
# In the same way the title is taken from the ttl variable.
# The options take the data columns to plot, sets the graph to be made with lines of width 2.
plot for [i=1:10] '< sqlite3 coronavirus.db "SELECT dateRep,SUM(deaths) OVER (ROWS UNBOUNDED PRECEDING) FROM CasesAndDeaths INNER JOIN Countries ON country_id=Countries.id INNER JOIN Dates ON date_id=Dates.id WHERE country_id='.word(ids, i).' ORDER BY year, month,day;"' using 1:2 title ttl(i) w l lw 2
EOFMarker
```

Listing 1: plot.sh script

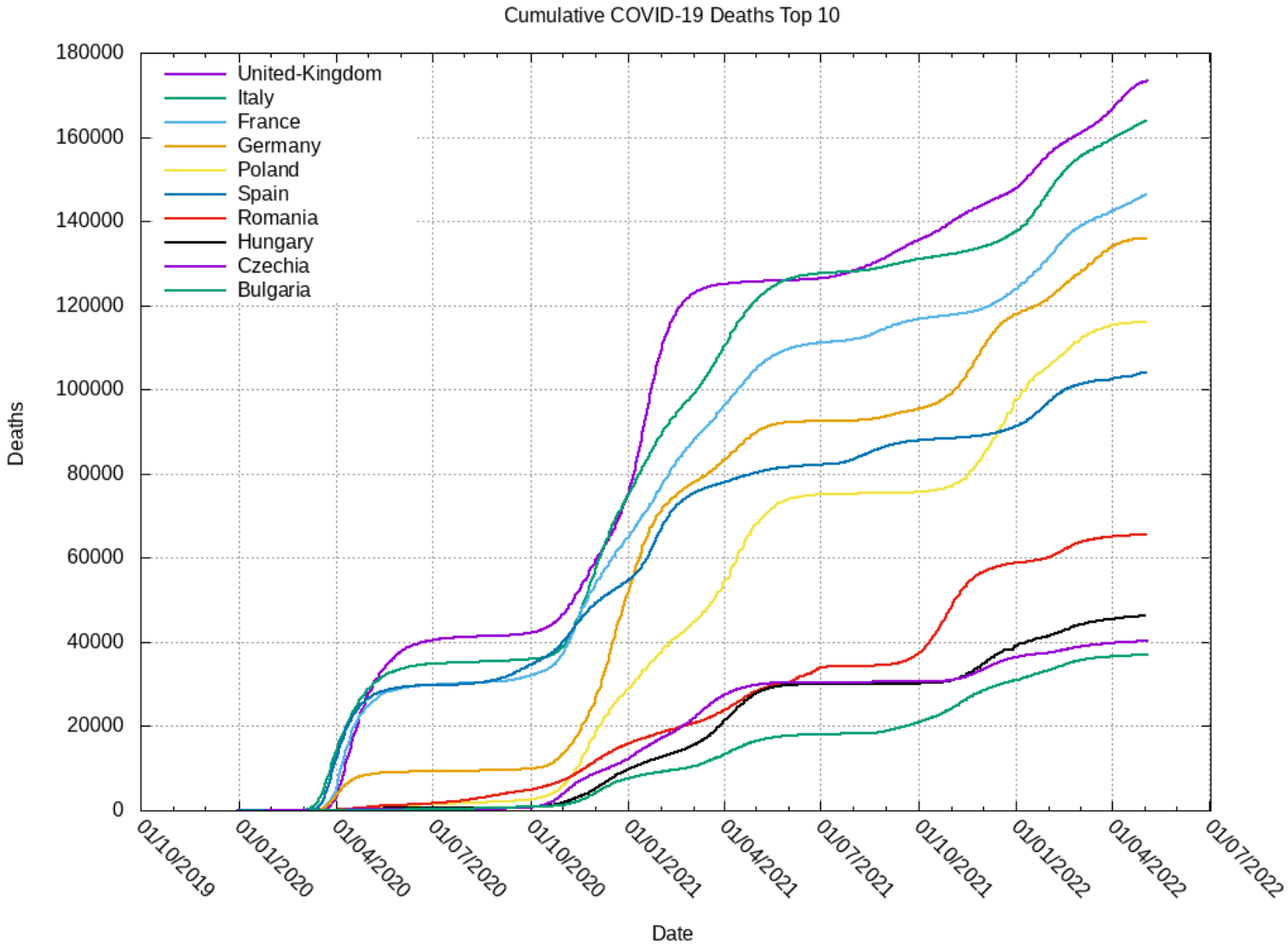

Figure 1: plot.sh resulting graph