Coursework Report

Constantinos Psaras

31/3/2024

Personal Details

Username: cp2n23
Student ID: 35350172

Introduction

This report presents the solutions to the coursework exercises.

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

Column Name	Data Type
dateRep	TEXT
day	INTEGER
month	INTEGER
year	INTEGER
cases	INTEGER
deaths	INTEGER
countriesAndTerritories	TEXT
geoId	TEXT
countryterritoryCode	TEXT
popData2019	INTEGER
continentExp	TEXT

Table 1: Schema for the covidStats relation

1.2 EX2

Determinants	Determined
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
countriesAndTerritories	countryterritoryCode
geoId	countriesAndTerritories
countriesAndTerritories	continentExp
geoId	countryterritoryCode
countryterritoryCode	continentExp
countryterritoryCode	geoId
countriesAndTerritories	popData2020
countryterritoryCode	popData2020
geoId	continentExp
geoId	popData2020

Assumptions:

- The population of a country (`popData2020`) can not uniquely identify a country since some countries may have the same population.
- The `cases` and `deaths` attributes are either `null` or an integer value.
- The `day`, `month` and `year` attributes are not null.

1.3 EX3

Candidate Keys

1. **{dateRep, geoID}**
2. **{dateRep, countriesAndTerritories}**
3. **{dateRep, countriesAndTerritoriesCode}**

1.4 EX4

- Suitable primary keys would be `{dateRep,geoID}` or `{dateRep,countryterritoryCode}` or `{dateRep ,countriesAndTerritories}`. It makes most sense to use the country code, geographical ID or country name to uniquely identify each country. Out of the three `geoId` is the most suitable, as territories may later be added to `countriesAndTerritories`, making the field not unique for each country.
- For the second part of the key using `dateRep` makes most sense, as the only other viable option would be using `day`, `month` and `year`.
- Chosen Primary Key: `{dateRep,geoID}`

2 Normalisation

2.1 EX5

- day, month, and year are functionally determined by dateRep. This indicates a partial-key dependency where non-prime attributes (day, month, year) are functionally determined by a subset of the candidate key (dateRep).

  `dateRep` → `day, month, year`

  Based on this partial-key dependency, we should decompose the relation into two additional relations(tables):

  - Relation 1: {dateRep, countriesAndTerritories, cases, deaths, geoId, countryterritoryCode, popData2020, continentExp}
  - Relation 2: {dateRep, day, month, year}

- countriesAndTerritories, countryterritoryCode, continentExp and popData2020 are also functionally determined by geoId.

  `geoId` → `countriesAndTerritories, countryterritoryCode, continentExp, popData2020`

  We should therefore decompose this relation into two additional relations(tables):

  - Relation 1: {geoID ,dateRep,cases,deaths}(Assuming we've already performed the first decomposition).
  - Relation 2: {geoID, countriesAndTerritories, countryterritoryCode, continentExp, popData2020}

- Third resulting relation: `geoId, dateRep` → `cases, deaths`

2.2 EX6

- The introduction of 2 surrogate keys for `dateRep` and `geoId` named `date_id` and `country_id` respectively
- The introduction of surrogate keys ensures data integrity against format changing of any of the fields.
- The two new surrogate keys now represent the primary keys of their respective relation.
- The resulting relations are:

  1. `date_id` → `dateRep, day, month, year`
  2. `country_id` → `countriesAndTerritories, countryterritoryCode, geoId, popData2020, continentExp`
  3. `date_id, country_id` → `cases, deaths`

2.3 EX7

- Transitive dependencies occur when a non-prime attribute depends on another non-prime attribute rather than directly on the primary key.
- Identified Dependencies:

  `countriesAndTerritories` → `countryterritoryCode`,`countriesAndTerritories` → `continentExp`, `countryterritoryCode` → `continentExp` `geoId` → `continentExp`, `geoId` → `countriesAndTerritories`, `countriesAndTerritories` → `continentExp`, `geoId` → `countryterritoryCode`, `countriesAndTerritories` → `countryterritoryCode`

2.4 EX8

- 3NF Requirements:
  - Each item in the relation contains only atomic values ✓
  - Each attribute has a unique name ✓
  - Order of rows does not matter ✓
  - Each row must be unique ✓
  - All non-key attributes must be fully functionally dependent on the entire primary key ✓
  - There must exist no transitive dependencies ✓
- All above dependencies contain prime attributes therefore there won't be any anomalies. Thus, the above relations are in 3NF.

2.5 EX9

- Boyce-Codd Normal Form (BCNF) Requirements:
  - Relation must be in 3NF ✓
  - For every non-trivial functional dependency $\mathbf{X} \rightarrow \mathbf{Y}$ in the relation, $\mathbf{X}$ must be a superkey ✓
  - Since no prime attributes are transitively dependent on a key, the above relations are already in BCNF

3 Modelling

3.1 EX10

1. **Navigate to database directory**
2. **Launch SQLite**: Execute `sqlite3 coronavirus.db` to open the SQLite command-line interface and connect to the database.
3. **Import data from csv file**: Use `.mode csv` and then `import dataset.csv dataset` to extract all data from the csv file.
4. **Set output file and dump database**: Use `.output dataset.sql` to specify `dataset.sql` as the file to which the database dump will be written. Run `.dump` to populate `dataset.sql`.

3.2 EX11

1. **Create** `ex11.sql`: Normalize the database by creating additional tables with appropriate schema, indexes, and foreign keys, excluding the `dataset` table.

2. **Dump Database**: Use SQLite command-line interface to dump the full database to `dataset2.sql`, capturing the schema and data. This ensures a complete backup of the database for easy restoration or transfer.

```
--Create countries table
CREATE TABLE IF NOT EXISTS countries (
        id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        countriesAndTerritories TEXT NOT NULL UNIQUE,
        geoId TEXT UNIQUE,
        countryterritoryCode TEXT UNIQUE,
        popData2020 INTEGER NOT NULL,
        continentExp TEXT NOT NULL
);
--Create table dates
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 covidStats table
CREATE TABLE IF NOT EXISTS covidStats (
        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

1. **Create** `ex12.sql`:Populate the `covidStats`, `countries`, and `dates` tables by inserting data from the `dataset` table.

2. **Dump Database**: Use SQLite command-line interface to dump the full database, including the populated tables, to `dataset3.sql`, capturing the schema and data. This ensures a complete backup of the database with the populated tables for easy restoration or transfer.

```
--Populate dates table
INSERT INTO dates SELECT DISTINCT NULL,dateRep,day,month,year FROM dataset;
--Populate countries table
INSERT INTO countries
SELECT DISTINCT NULL,
countriesAndTerritories,geoId,countryterritoryCode,popData2020,continentExp
FROM dataset;
--Populate covidStats table
-- Populate covidStats table, using COALESCE to handle NULL deaths
INSERT INTO covidStats (date_id, country_id, cases, deaths)
```

```
SELECT
    dates.id,
    countries.id,
    dataset.cases,
    CASE WHEN dataset.deaths = ''
        THEN 0
        ELSE CAST(dataset.deaths AS INTEGER)
        END AS deaths
FROM dataset
    INNER JOIN dates
        ON dates.dateRep = dataset.dateRep
    INNER JOIN countries
        ON countries.countriesAndTerritories = dataset.countriesAndTerritories;
```

3.4 EX13

Running `sqlite3 coronavirus.db < dataset.sql`, `sqlite3 coronavirus.db <ex11.sql`, `sqlite3 coronavirus.db < ex12.sql` succesfully re-generated the database.

4 Querying

4.1 EX14

- Approach: Use `SUM` function to calculate total number of cases and deaths across all records in the covidStats table. Alias the sum as `total_cases` and `total_deaths` accordingly

- ```
  SELECT
      SUM(cases) AS total_cases,
      SUM(deaths) AS total_deaths
  FROM covidStats;
  ```

4.2 EX15

- Approach: Use `SELECT` statement to retrieve dateRep and cases columns from the covidStats table, filtering the records where geoId is `'UK'`. Order the results by dateRep in ascending order.

- ```
  SELECT dateRep,cases
  FROM covidStats
      INNER JOIN countries ON country_id=countries.id
      INNER JOIN dates ON date_id=dates.id
  WHERE countries.countriesAndTerritories= 'United_Kingdom'
  ORDER BY year,month,day;
  ```

4.3 EX16

- Approach: `INNER JOIN countries` , `dates` and `covidStats` tables based on the surrogate keys.

- Select the date, cases, deaths and country name and order the results first by date and then by counttry name.

- ```
  SELECT dateRep AS date,countriesAndTerritories AS countryName,cases,deaths
  FROM covidStats
      INNER JOIN countries ON country_id=countries.id
      INNER JOIN dates ON date_id=dates.id
  ORDER BY year,month,day, countryName;
  ```
  ```

4.4 EX17

- Approach: Use `INNER JOIN` to join the `countries` and `covidStats` tables based on the `country_id`.
- Use the `SUM` function to aggregate the totals of both cases and deaths.
- Wrap the `SUM` with `ROUND` function to get the percentage in 2 decimal places.
- ```
  SELECT
      countriesAndTerritories AS countryName,
      ROUND(SUM((cases * 100.0) / popData2020), 2)
          AS percent_cases_of_population,
      ROUND(SUM((deaths * 100.0) / popData2020), 2)
          AS percent_deaths_of_population
  FROM
      covidStats
  INNER JOIN
      countries ON country_id = countries.id
  GROUP BY
      country_id;
  ```

4.5 EX18

- Approach: Use a `INNER JOIN` to joint `countries` and `covidStats` tables. Group the entries by `countryName`
- Order the results in descending order based on `percentDeaths` and `LIMIT` the results to 10 rows.

```
SELECT countriesAndTerritories AS countryName,
       ROUND((CAST(SUM(deaths) AS REAL) / NULLIF(SUM(cases), 0)) * 100, 2)
           AS percentDeaths
FROM covidStats
    INNER JOIN countries ON country_id = countries.id
GROUP BY countryName
ORDER BY percentDeaths DESC
LIMIT 10;
```

4.6 EX19

- `INNER JOIN` the `countries`, `dates` and `covidStats` tables based on the corresponding surrogate keys.
- Use `WHERE` to limit results to the UK.
- Order the results chronologically

```
SELECT dateRep AS date,
       SUM(deaths) OVER (ROWS UNBOUNDED PRECEDING) AS cumulativeDeaths_UK,
       SUM(cases) OVER (ROWS UNBOUNDED PRECEDING) AS cumulativeCases_UK
FROM covidStats
    INNER JOIN countries ON country_id = countries.id
    INNER JOIN dates ON date_id = dates.id
WHERE countriesAndTerritories = 'Cyprus'
ORDER BY dateRep;
```
```

5 Extension

5.1 EX20

Script:

```
#!/bin/bash

# Fetch the top 10 countries' IDs with the most deaths
IDS=$(sqlite3 coronavirus.db "SELECT country_id
                              FROM covidStats
                                  INNER JOIN countries ON country_id=countries.id
                              GROUP BY country_id
                              ORDER BY SUM(deaths) DESC
                              LIMIT 10;" | tr '\n' ' ')

# Fetch the names of the top 10 countries
NAMES=$(sqlite3 coronavirus.db "SELECT countriesAndTerritories
                                FROM covidStats
                                    INNER JOIN countries ON country_id=countries.
                                       id
                                GROUP BY country_id
                                ORDER BY SUM(deaths) DESC
                                LIMIT 10;" | tr '\n' ' ' |  tr '_' '-')

# Use gnuplot to generate the plot
gnuplot -persist <<-EOFMarker
  # Set plot settings
  set key top left autotitle columnheader
  set key reverse Left
  set title 'Cumulative COVID-19 Deaths Top 10'
  set ylabel 'Deaths'
  set xlabel 'Date'
  set grid
  set xdata time
  set datafile separator "|"
  set format x '%d/%m/%Y'
  set timefmt "%d/%m/%Y"
  set xtics mirror rotate by -45
  set rmargin at screen 0.94
  set term png
  set terminal png size 1024,768
  set output "graph1.png"

  # Set variables for country names and IDs
  titles = "$NAMES"
  ids = "$IDS"

  # Define function to retrieve country name from the titles variable
  ttl(n) = sprintf("%s", word(titles, n))

  # Plot data for each of the top 10 countries
  plot for [i=1:10] \
  '< sqlite3 coronavirus.db "SELECT dateRep,SUM(deaths) OVER (ROWS UNBOUNDED
     PRECEDING) FROM covidStats 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
```

Result:

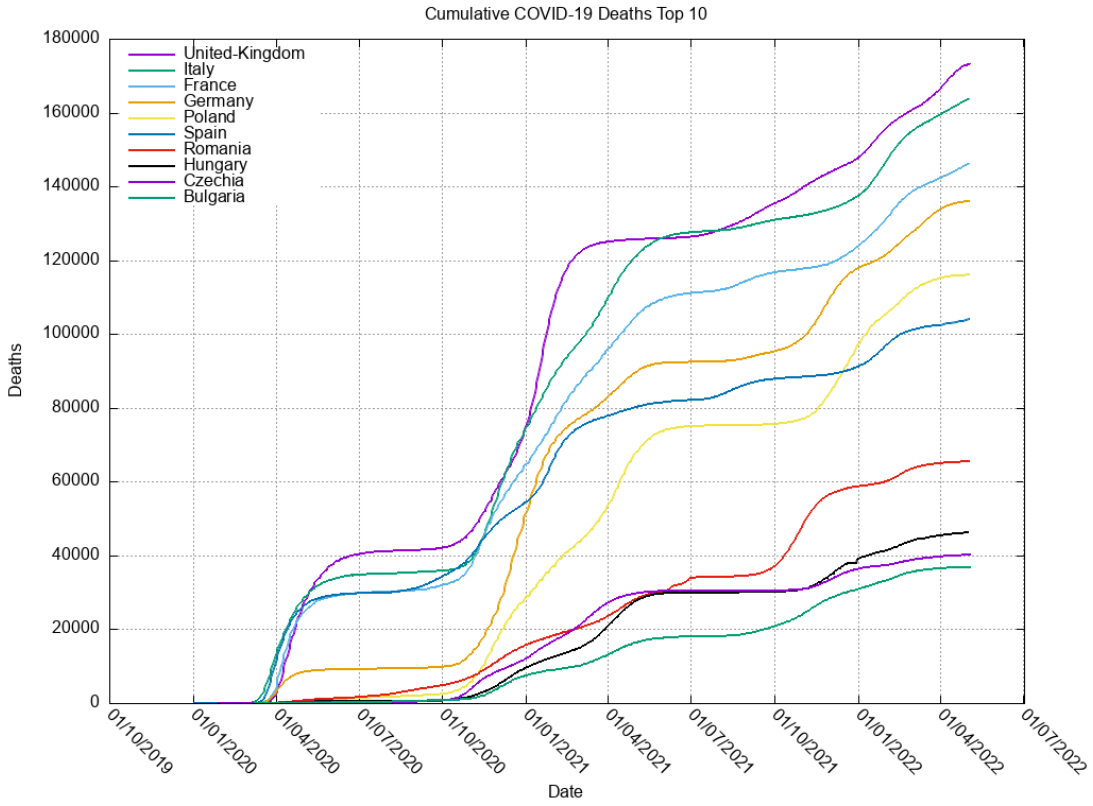

Figure 1: Top 10 countries in terms of cumulative deaths