COMP1204 - Data Management
Database Coursework Report

Giovanni Arcudi

MEng - Computer Science with Artifcial Intelligence
University of Southampton
-
Student ID: 31275206
E-mail: ga1g19@soton.ac.uk

July 11, 2020

1 The Relational Model

1.1 EX1

The relation directly represented in the *dataset.csv* file, with the relative data types (as SQLite Storage Classes), can be expressed as follows:

$$
\begin{array}{ll}
CoronavirusData(& \text{TEXT: } dateRep, \\
 & \text{INTEGER: } day, \\
 & \text{INTEGER: } month, \\
 & \text{INTEGER: } year, \\
 & \text{INTEGER: } cases, \\
 & \text{INTEGER: } deaths, \\
 & \text{TEXT: } countriesAndTerritories, \\
 & \text{TEXT: } geoId, \\
 & \text{TEXT: } countryterritoryCode, \\
 & \text{INTEGER: } popData2018, \\
 & \text{TEXT: } continentExp \\
\quad) &
\end{array}
$$

Note: Data types such as MEDIUMINT or VARCHAR(255) can also be used, nevertheless, SQLite would convert them to the aforementioned SQLite Storage Classes.

1.2 EX2

Below, a list of all of the **non-trivial** functional dependencies that exist in the dataset.

$$
\begin{array}{rcl}
\{\textit{dateRep}\} & \rightarrow & \{\textit{day, month, year}\} \\
\{\textit{countriesAndTerritories}\} & \rightarrow & \{\textit{geoId, countryterritoryCode, popData2018,} \\
 & & \textit{continentExp}\} \\
\{\textit{geoId}\} & \rightarrow & \{\textit{countriesAndTerritories, countryterritoryCode,} \\
 & & \textit{popData2018, continentExp}\} \\
\{\textit{countryterritoryCode}\} & \rightarrow & \{\textit{popData2018}\} \\
\{\textit{dateRep, countriesAndTerritories}\} & \rightarrow & \{\textit{cases, deaths}\} \\
\{\textit{dateRep, geoId}\} & \rightarrow & \{\textit{cases, deaths}\} \\
\{\textit{day, month, year}\} & \rightarrow & \{\textit{dateRep}\} \\
\{\textit{day, month, year, countriesAndTerritories}\} & \rightarrow & \{\textit{cases, deaths}\} \\
\{\textit{day, month, year, geoId}\} & \rightarrow & \{\textit{cases, deaths}\}
\end{array}
$$

From this list, some functional dependencies (most of which related to *cases* and *deaths* attributes) have been excluded because it is not guaranteed that they will still hold with future database expansions.

1.3 EX3

Below, a list of the potential Candidate Keys.

$$\{\textit{dateRep, countriesAndTerritories}\},\ \{\textit{dateRep, geoId}\},\ \{\textit{day, month, year, countriesAndTerritories}\},$$
$$\{\textit{date, month, year, geoId}\}$$

1.4 EX4

The most reasonable way to choose a Primary Key is to pick the Candidate Key with least attributes. For this reason, I identified {*dateRep, countriesAndTerritories*} as a suitable Primary Key.

2 Normalisation

2.1 EX5

Having identified {*dateRep, countriesAndTerritories*} as Primary Key, the list of the partial-key dependencies that exist in the relation as it stands will be:

$$\{\textit{day, month, year}\} \xrightarrow{\text{Partially dependent on}} \textit{dateRep}$$

$$\{\textit{geoId, countryterritoryCode, popData2018, continentExp}\} \xrightarrow{\text{Partially dependent on}} \textit{countriesAndTerritories}$$

Therefore, the additional relations that I would create as part of the decomposition are:

Date(dateRep, day, month, year)
Country(countriesAndTerritories, geoId, countryterritoryCode, popData2018, continentExp)

2.2 EX6

Converting the relation into 2nd Normal Form using the answer to EX5, we obtain the following new relations:

Date(TEXT: *dateRep*, - **PrimKey**
INTEGER: *day*,
INTEGER: *month*,
INTEGER: *year*
)

Country(TEXT: *countriesAndTerritories*, - **PrimKey**
TEXT: *geoId*,
TEXT: *countryterritoryCode*,
INTEGER: *popData*2018,
TEXT: *continentExp*
)

Obviously, with the introduction of these two new relations, we can tidy up the main *CoronavirusData* relation, which will therefore be reduced to:

CoronavirusData(TEXT: *dateRep*, - **Foreign Key**
INTEGER: *cases*,
INTEGER: *deaths*,
TEXT: *countriesAndTerritories* - **Foreign Key**
)

with {*dateRep, countriesAndTerritories*} being a **(Primary) Compound Key**.

2.3 EX7

As a result, our new relations will have just one transitive dependency:

$$\textit{countriesAndTerritories} \rightarrow \textit{countryterritoryCode} \rightarrow \textit{popData2018}$$

2.4 EX8

While converting the relation into 3rd Normal Form using the answer to EX7, I noticed that it was not possible to update the population number of a location whose *countryterritoryCode* was NULL. Therefore, assuming every location (including cruise ships) will be provided with a non-null *geoId*, I decided to use *geoId* rather than *countryterritoryCode* as Primary Key of the following new relation:

Population(INTEGER: *geoId*, - **PrimKey**
INTEGER: *popData*2018
)

Obviously, with the introduction of this new relation, we can tidy up the *Country* relation, which will become:

Country(TEXT: *countriesAndTerritories*, - **PrimKey**
TEXT: *geoId*, - **Foreign Key**
TEXT: *countryterritoryCode*,
TEXT: *continentExp*,
)

2.5 EX9

All of my relations (*CoronavirusData, Date, Country, Poupulation*) are in Boyce-Codd Normal Form. This is because, for each of them, every one of their dependencies $X \rightarrow Y$ is a trivial functional dependency ($Y \subseteq X$) or a superkey.

3 Modelling

3.1 EX10

Using the commands shown below, I imported the raw dataset into a single table called *dataset* in an SQLite database called *coronavirus.db*, and exported this table as *dataset.sql*.

```
sqlite3 coronavirus.db

sqlite> CREATE TABLE dataset(
"dateRep" TEXT,
"day" INTEGER,
"month" INTEGER,
"year" INTEGER,
"cases" INTEGER,
"deaths" INTEGER,
"countriesAndTerritories" TEXT,
"geoId" TEXT,
"countryterritoryCode" TEXT,
"popData2018" INTEGER,
"continentExp" TEXT
);

sqlite> .mode csv
sqlite> .import /home/student/Desktop/comp1204-ga1g19-cwk2/dataset.csv dataset
sqlite> .output /home/student/Desktop/comp1204-ga1g19-cwk2/dataset.sql
sqlite> .dump dataset
```

3.2 EX11

As requested, in *ex11.sql* is possible to see in detail the SQL statements that I used to create the full normalised representation, with no data. In summary, I created four new tables according to the four relations specified in EX6 and EX8 defining, for each table, Primary and Foreign Keys, and Foreign Key constraints.

I also defined the following indexes, to speed up frequent queries and sorting operations that users may want to perform on the main *CoronavirusData* table:

```
CREATE INDEX ix_CoronavirusData_cases
ON CoronavirusData(cases);

CREATE INDEX ix_CoronavirusData_deaths
ON CoronavirusData(deaths);

CREATE INDEX ix_CoronavirusData_countriesAndTerritories_cases
ON CoronavirusData(countriesAndTerritories, cases);

CREATE INDEX ix_CoronavirusData_countriesAndTerritories_deaths
ON CoronavirusData(countriesAndTerritories, deaths);

CREATE INDEX ix_CoronavirusData_dateRep_cases
ON CoronavirusData(dateRep, cases);

CREATE INDEX ix_CoronavirusData_dateRep_deaths
ON CoronavirusData(dateRep, deaths);
```

3.3 EX12

Below, the INSERT statements used to populate the new tables from the *dataset* table.

As you can see, I included *WHERE dateRep != 'dateRep'* in order to exclude the first line of the dataset (containing the attribute names) from each importing process.

```
INSERT INTO CoronavirusData(dateRep, countriesAndTerritories, cases, deaths)
SELECT dateRep, countriesAndTerritories, cases, deaths
FROM dataset WHERE dateRep != 'dateRep';

INSERT INTO Date(dateRep, day, month, year)
SELECT DISTINCT dateRep, day, month, year
FROM dataset WHERE dateRep != 'dateRep';

INSERT INTO Country(countriesAndTerritories, geoId, countryterritoryCode,
continentExp)
SELECT DISTINCT countriesAndTerritories, geoId, countryterritoryCode,
continentExp
FROM dataset WHERE dateRep != 'dateRep';

INSERT INTO Population(geoId, popData2018)
SELECT DISTINCT geoId, popData2018
FROM dataset WHERE dateRep != 'dateRep';
```

3.4 EX13

After a test, it is confirmed that on a clean SQLite database it is possible to execute *dataset.sql*, followed by *ex11.sql*, followed by *ex12.sql*, to successfully populate the database.

4 Querying

4.1 EX14

In order to get the worldwide total number of cases and deaths (with total cases and total deaths as columns), I used the following SQL statement:

```
SELECT SUM(cases) AS 'Total_Cases',
       SUM(deaths) AS 'Total_Deaths'
FROM CoronavirusData;
```

4.2 EX15

In order to get the number of cases and the date, by increasing date order, for the United Kingdom (with date and number of cases as columns) I used the following SQL statement:

```
SELECT CoronavirusData.dateRep AS 'Date',
       cases AS 'Number_Of_Cases'
FROM CoronavirusData
   INNER JOIN Date ON CoronavirusData.dateRep = Date.dateRep
WHERE countriesAndTerritories = 'United_Kingdom'
ORDER BY year ASC, month ASC, day ASC;
```

4.3 EX16

In order to get the number of cases, deaths and the date (by increasing date order) for each continent (with continent, date, number of cases and number of deaths as columns), I used the following SQL statement:

```
SELECT Country.continentExp AS 'Continent',
       CoronavirusData.dateRep AS 'Date',
       SUM(cases) AS 'Number_Of_Cases',
       SUM(deaths) AS 'Number_Of_Deaths'
FROM CoronavirusData
   INNER JOIN Date ON CoronavirusData.dateRep = Date.dateRep
   INNER JOIN Country ON CoronavirusData.countriesAndTerritories = Country.
    countriesAndTerritories
GROUP BY CoronavirusData.dateRep, Country.continentExp
ORDER BY year ASC, month ASC, day ASC;
```

4.4 EX17

In order to get the number of cases and deaths as a percentage of the population for each country (with country, % cases of population, % deaths of population as columns), I used the following SQL statement:

```
SELECT CoronavirusData.countriesAndTerritories AS 'Country',
   ((SUM(cases) * 1.0) / (Population.popData2018 * 1.0)) * 100 AS '%_Cases',
   ((SUM(deaths) * 1.0) / (Population.popData2018 * 1.0)) * 100 AS '%_Deaths'
FROM CoronavirusData
   INNER JOIN Country ON CoronavirusData.countriesAndTerritories = Country.
    countriesAndTerritories
   INNER JOIN Population ON Country.geoId = Population.geoId
GROUP BY CoronavirusData.countriesAndTerritories;
```

4.5 EX18

In order to get a descending list of the the top 10 countries, by percentage deaths out of the cases in that country (with country name and % deaths of country cases as columns), I used the following SQL statement:

```
SELECT CoronavirusData.countriesAndTerritories AS 'Country',
       (SUM(deaths) * 1.0) / (SUM(cases) * 1.0) AS '%_Deaths_of_Country_Cases'
FROM CoronavirusData
GROUP BY countriesAndTerritories
ORDER BY 2 DESC
LIMIT 10;
```

4.6 EX19

In order to get the date against a cumulative running total of the number of deaths by day and cases by day for the United Kingdom (with date, cumulative UK deaths and cumulative UK cases as columns), I used the following SQL statement:

```
SELECT CoronavirusData.dateRep AS 'Date',
 SUM(deaths) OVER(ORDER BY year ASC, month ASC, day ASC) AS 'Cumulative_UK_deaths',
 SUM(cases) OVER(ORDER BY year ASC, month ASC, day ASC) AS 'Cumulative_UK_cases'
FROM CoronavirusData
   INNER JOIN Date ON CoronavirusData.dateRep = Date.dateRep
WHERE countriesAndTerritories = 'United_Kingdom'
ORDER BY year ASC, month ASC, day ASC;
```