COMP1204: Data Management
Coursework Two

Lukas Kakogiannos
lk1u20, 32158998

May 17, 2022

Abstract

The aim of this coursework is to create an SQLite based database to represent data of the ongoing Coronavirus outbreak from an Open Data Source **dataset.csv**. Moreover, the database model should be fully normalised, facilitating querying.

1 The Relational Model

1.1 EX1

The table below shows the relations directly represented in the dataset.csv file, and their respective data types.

Table 1: Relations Within the Dataset

CovidData	SQLite Attribute
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

After examining the dataset, the following functional dependencies can be observed:

1. $dateRep \rightarrow day$
2. $dateRep \rightarrow month$
3. $dateRep \rightarrow year$
4. $day, month, year \rightarrow dateRep$
5. $countriesAndTerritories \rightarrow geoId$
6. $geoId \rightarrow countryterritoryCode$
7. $geoId \rightarrow popData2020$
8. $geoId \rightarrow continentExp$
9. $geoId \rightarrow countriesAndTerritories$
10. $countryterritoryCode \rightarrow geoId$
11. $dateRep, countriesAndTerritories \rightarrow cases$
12. $dateRep, countriesAndTerritories \rightarrow deaths$

1.3 EX3

All potential candidate keys:

1. dateRep, countriesAndTerritories
2. dateRep, geoId
3. day, month, year, countriesAndTerritories
4. day, month, year, geoId

1.4 EX4

After comparing all candidate keys, I chose **{dateRep, geoId}** to be an optimal primary key as it has the least attributes.

2 Normalisation

2.1 EX5

The partial-key dependencies present in the relation are listed below:

{day, month, year} partially dependant to {dateRep}
{geoId, countryterritoryCode, popData2020, continentExp} partially dependant to {countriesAndTerritories}

Hence, as part of the decomposition, the following relations could be deduced:

Date{dateRep, day, month, year}
CountryInformation{countriesAndTerritories, geoId, countryterritoryCode, popData2020, continentExp}

2.2 EX6

By using the answer to EX5, we can convert the relation to 2nd Normal Form.

Table 2: Date

CovidData	SQLite Attribute	Key
dateRep	TEXT	-primaryKey
day	INTEGER	
month	INTEGER	
year	INTEGER	

Table 3: CountryInformation

CovidData	SQLite Attribute	Key
countriesAndTerritories	TEXT	-primaryKey
geoId	TEXT	
countryterritoryCode	TEXT	
popData2020	INTEGER	
continentExp	TEXT	

By introducing these new relations, it is possible to reduce the main relation to the following:

Table 4: ReducedCovidData

CovidData	SQLite Attribute	Key
dateRep	TEXT	-foreignKey
cases	INTEGER	
deaths	INTEGER	
countriesAndTerritories	TEXT	-foreignKey

2.3 EX7

By looking at the dataset, it is possible to assume that the population of a country is a constant number, therefore, the new relations contain one transitive dependency:

$$countriesAndTerritories \rightarrow countryterritoryCode \rightarrow popData2020$$

2.4 EX8

Based on the dependency found in EX7, a new relation has to be formed in order to achieve 3rd Normal Form. Assuming every country has a geoId, the following relation can be formed:

Table 5: Population

CovidData	SQLite Attribute	Key
geoId	INTEGER	-primaryKey
popData2020	INTEGER	

Finally, **countryInformation** will change to:

Table 6: ReducedCountryInformation

CovidData	SQLite Attribute	Key
countriesAndTerritories	TEXT	-primaryKey
geoId	TEXT	-foreignKey
countryterritoryCode	TEXT	
continentExp	TEXT	

2.5 EX9

For each one of the four relations (ReducedCovidData, Date, ReducedCountryInformation, Population), each one of their dependencies X $\rightarrow$ Y is either a superkey or a trivial functional dependency ($Y \subseteq X$), consequently, all of them are already in Boyce-Codd Normal Form.

3 Modelling

3.1 EX10

To start modelling the database physically, I first had to import the raw data from **dataset.csv** into a table called **dataset** in a SQLite database called **coronavirus.db**. To achieve that, the following commands were used:

Listing 1: CMD commands for dataset.sql

```
sqlite3 coronavirus.db

CREATE TABLE dataset( "dateRep" TEXT, "day" INTEGER,
    "month" INTEGER, "year" INTEGER, "cases" INTEGER,
    "deaths" INTEGER, "countriesAndTerritories" TEXT,
    "geoId" TEXT, "countryterritoryCode" TEXT,
    "popData2020" INTEGER, "continentExp" TEXT );

sqlite> .mode csv
sqlite> .import dataset.csv dataset
sqlite> .output dataset.sql
sqlite> .dump dataset
```

3.2 EX11

After creating **dataset.sql**, I wrote the SQL to represent the four relations mentioned in EX9(ReducedCovidData, Date, ReducedCountryInformation, Population) as additional tables. This was achieved with the use of **CREATE** statements and **CONSTRAINT** statements for the primary/foreign keys. Furthermore, the six following indexes were added to help querying.

Listing 2: Indexes for querying

```
CREATE INDEX idx_ReducedCovidData_cases
ON ReducedCovidData(cases);

CREATE INDEX idx_ReducedCovidData_deaths
ON ReducedCovidData(deaths);

CREATE INDEX idx_ReducedCovidData_dateRep_cases
ON ReducedCovidData(dateRep, cases);

CREATE INDEX idx_ReducedCovidData_dateRep_deaths
ON ReducedCovidData(dateRep, deaths);

CREATE INDEX
    idx_ReducedCovidData_countriesAndTerritories_cases
ON ReducedCovidData(countriesAndTerritories, cases);
```

```sql

CREATE INDEX
    idx_ReducedCovidData_countriesAndTerritories_deaths
ON ReducedCovidData(countriesAndTerritories, deaths);
```

3.3 EX12

By using **INSERT INTO** and **SELECT DISTINCT** statements, the new tables were populated with values from the 'dataset' table. To exclude the first line of the database, the following statement was used:

WHERE dateRep != 'dateRep'

Find below a full example of a final set of statements to populate the **Date** table:

Listing 3: Date Table Statements

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

3.4 EX13

After several tests and tables gone wrong, the desired result was achieved, a fully populated database and its respective tables. Moreover, the import and dump functions were used similarly to EX10 in order to dump the database unto **dataset2.sql** and **dataset3.sql**, after running **EX11.sql** and **EX12.sql**, respectively.

4 Querying

4.1 EX14

To get the worldwide total number of cases and deaths, the following statement was used:

Listing 4: Querying worldwide total number of cases and deaths

```sql
SELECT SUM(cases) AS 'total cases', SUM(deaths) AS '
    total deaths'
FROM ReducedCovidData;
```

4.2 EX15

To get the number of cases by date, in increasing date order, for the United Kingdom, the following statement was used:

Listing 5: Querying number of cases for the UK by increasing date

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

4.3 EX16

To get the number of cases and deaths by date, in increasing date order, for each country, the following statement was used:

Listing 6: Querying number of cases and deaths for all countries by increasing date

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

It can be seen from the **GROUP BY** statement that the result is ordered by date first.

4.4 EX17

To get the total number of cases and deaths as a percentage of the population rounded to three decimal cases, for each country, the following statement was used:

Listing 7: Querying total number of cases and deaths of a country(percentage of population)

```sql
SELECT ReducedCovidData.countriesAndTerritories AS 'Country', ROUND((SUM(cases) * 100.0) / (Population.
```

```
    popData2020), 3) AS '%_Cases_of_Population', ROUND
    ((SUM(deaths) * 100.0) / (Population.popData2020),
    3)AS '%_Deaths_of_Population'
FROM ReducedCovidData
        INNER JOIN ReducedCountryInformation ON
            ReducedCovidData.countriesAndTerritories =
            ReducedCountryInformation.
            countriesAndTerritories
        INNER JOIN Population ON
            ReducedCountryInformation.geoId =
            Population.geoId
GROUP BY ReducedCovidData.countriesAndTerritories;
```

4.5 EX18

To get a descending list of the the top 10 countries, by percentage total deaths out of total cases in that country rounded to three decimal cases, the following statement was used:

Listing 8: Querying 10 countries with highest deaths to cases ratio percentage

```
SELECT ReducedCovidData.countriesAndTerritories AS '
    Country', ROUND((SUM(deaths) * 100.0) / (SUM(cases)
     * 1.0), 3) AS '%_Deaths_of_Country_Cases'
FROM ReducedCovidData
GROUP BY countriesAndTerritories
ORDER BY 2 DESC
LIMIT 10;
```

4.6 EX19

To get the date against a cumulative running total of the number of deaths by day and cases by day for the United Kingdom, the following statement was used:

Listing 9: Querying cumulative number of deaths and cases over time for the UK

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