Creation of an SQLite Database for Coronavirus Data

Yaqin Kayem Hasan
ID:31426441
username: ykh1e19

May 2021

1 Introduction

This coursework involved creating and normalising an SQLite database to hold the data from Open Data Source's Coronavirus data and constructing queries against the data once it is in a suitable form.

There are 5 parts for this coursework, with the mark contribution of each part bracketed:
1. The Relational Model (20%)
2. Normalisation (25%)
3. Modelling (20%)
4. Querying (30%)
5. Extension (OPTIONAL: 5%)

2 The Relational Model

2.1 EX1

The relation **R** directly represented by the *dataset.csv* file is a set of 11-tuples (tuples with 11 elements).

This can be represented as R = {(dateRep1, day1, month1, year1, cases1, deaths1, countriesAndTerritories1, GeoId1, countryterritoryCode1, popData20191, continentExp1), ... , (dateRepN, dayN, ... , continentExpN).

Using the data dictionary provided, the most appropriate data types for the attributes are as follows:
dateRep: TEXT
day: INT8
month: INT8
year: INT16
cases: INT64
deaths: INT64
countriesAndTerritories: TEXT
geoId: TEXT
countryterritoryCode: TEXT
popData2019: INT64
continentExp: TEXT

2.2 EX2

Using → to denote functional dependency, such that attribute1 → attribute2 represents attribute2 being functionally dependent on attribute1. One possible minimal set of functional dependencies would be:
(day, month, year) → dateRep

(dateRep, countriesAndTerritories) → cases
(dateRep, countriesAndTerritories) → deaths
countriesAndTerritories → geoId
countriesAndTerritories → countryterritoryCode
countriesAndTerritories → popData2019
countriesAndTerritories → continentExp

where (day, month, year) and (dateRep, countriesAndTerritories) are composite keys.

It should also be notes that there are multiple possible minimal sets of functional dependencies since the last 4 functional dependencies can be rearranged, while still following the same rules and not increasing the cardinality of the minimal set. For example, the last 4 functional dependencies with the determinant countriesAndTerritories can be alternatively represented as:
countriesAndTerritories → geoId
geoID → countryterritoryCode
countryterritoryCode → popData2019
countryterritoryCode → continentExp

Though this introduces more transitive dependencies, which increases the amount of work required to normalize into the third normal form.

2.3 EX3

There are multiple candidate keys for this relation. Though they all follow the same pattern where one group of key attributes is picked from either dateRep or (day, month, year). Then, the other group of key attributes is picked from countriesAndTerritories, geoId or countryterritoryCode.

For example:
dateRep, geoId
day, month, year, countryterritoryCode

2.4 EX4

However, using dateRep instead of (day, month, year) is simpler and the domain of countriesAndTerritories is the most human-readable of the options.

Therefore, the most suitable candidate key to make a primary key would be a composite key made up of dateRep and countriesAndTerritories.

3 Normalization

3.1 EX5

A cursory glance at *dataset.csv* reveals the 2 most obvious partial dependencies.
dateRep → day, month, year
countriesAndTerritories → geoId, countryterritoryCode, popData2019, continentExp

With attribute1 → attribute2, attribute3 denoting attribute1 → attribute2 and attribute1 → attribute3.

However, the most important attributes are still unaccounted for. Since all candidate keys can be used to uniquely identify all rows/tuples in the relation, any of the candidate keys identified in **EX4** will work.

Looking at the determinants of the other 2 partial dependencies, it is evident that the composite candidate key identified to be the most suitable primary key would be the composite key made up of dateRep and countriesAndTerritories.

As such we can decompose *dataset.csv* into 3 relations:
CountryDate($\underline{\text{countriesAndTerritories}}$, $\underline{\text{dateRep}}$, cases, deaths)
Countries($\underline{\text{countriesAndTerritories}}$, geoId, countryterritoryCode, popData2019, continentExp)
Dates($\underline{\text{dateRep}}$, day, month, year)

Where the underlined attributes are key attributes. Additionally, Countries.countriesAndTerritories and Dates.dateRep are foreign keys.

3.2 EX6

We can see that for all the three tables formed by the decomposition of *dataset.csv* in **EX5**, all attributes are atomic. Furthermore, all non-key attributes are functionally dependent on the primary key. Therefore, it is in second normal form.

3.3 EX7

We can see that for all the three tables formed by the decomposition of *dataset.csv* in **EX5**, there are no transitive dependencies since countriesAndTerritories → geoId, countryterritoryCode, popData2019, continentExp.

3.4 EX8

As seen in **EX7**, there are no transitive dependencies and therefore it is in third normal form.

3.5 EX9

As seen in **EX5**, all tables are have only have dependencies where the determinants are a super key of the relation. Therefore, it is in Boyce-Codd normal form.

4 Modelling

4.1 EX10

coronavirus.db was created and *dataset.csv* was imported as dataset table. The results were then dumped to *dataset.sql*.

4.2 EX11

ex11.sql was created as follows:

```
CREATE TABLE CountryDate (
  dateRep TEXT,
  countriesAndTerritories TEXT,
  cases INT64,
  deaths INT64,
  PRIMARY KEY (dateRep, countriesAndTerritories)
);

CREATE TABLE Countries(
  countriesAndTerritories TEXT,
  geoId TEXT,
  countryterritoryCode TEXT,
  popData2019 INT64,
  continentExp TEXT,
  FOREIGN KEY (countriesAndTerritories) REFERENCES CountryDate(countriesAndTerritories)
);

CREATE TABLE Dates(
  dateRep TEXT,
  day INT8,
  month INT8,
  year INT16,
  FOREIGN KEY (dateRep) REFERENCES CountryDate(dateRep)
);
```

The results were then dumped to *dataset2.sql*.

4.3 EX12

ex12.sql was created as follows:

```
INSERT OR IGNORE INTO CountryDate(dateRep, countriesAndTerritories, cases, deaths)
SELECT
  dateRep, countriesAndTerritories, cases, deaths
FROM
  dataset;

INSERT INTO Countries(countriesAndTerritories, geoId, countryterritoryCode, popData2019,
SELECT
  countriesAndTerritories, geoId, countryterritoryCode, popData2019, continentExp
FROM
  dataset;

INSERT INTO Dates(dateRep, day, month, year)
SELECT
  dateRep, day, month, year
FROM
  dataset;
```

The results were then dumped to *dataset3.sql*.

4.4 EX13

it was then tested that the following commands successfully produce a fully populated database:

sqlite3 coronavirus.db ¡ dataset.sql
sqlite3 coronavirus.db ¡ ex11.sql
sqlite3 coronavirus.db ¡ ex12.sql

5 Querying

5.1 EX14

ex14.sql was created as follows:

```
SELECT SUM(cases), SUM(deaths)
FROM CountryDate;
```

5.2 EX15

ex14.sql was created as follows:

```
CREATE TABLE CountryDateAlt(
  day INT8,
  month INT8,
  year INT8,
  countriesAndTerritories TEXT,
  cases INT64,
  deaths INT64
);

INSERT OR IGNORE INTO CountryDateAlt(day, month, year, countriesAndTerritories, cases, d
SELECT
  day, month, year, countriesAndTerritories, cases, deaths
FROM
  dataset;

CREATE TABLE UKCasesByIncreasingDate(
  day INT8,
  month INT8,
  year INT8,
  cases INT64
);

INSERT INTO UKCasesByIncreasingDate(day, month, year, cases)
SELECT
  day, month, year, cases
FROM
  CountryDateAlt
WHERE
  countriesAndTerritories='United_Kingdom'
ORDER BY
  year ASC, month ASC, day ASC;

SELECT *
FROM UKCasesByIncreasingDate;

DROP TABLE UKCasesByIncreasingDate;

DROP TABLE CountryDateAlt;
```

CountryDateAlt was used in favor of CountryDate when sorting by date was required. CountryDateAlt could have been a persistent relation within the database. However, this would require modifying *ex11.sql* and *ex12.sql*.

Therefore, it will be created a transient relation whenever sorting by date is required. The SQL for its creation, population and deletion will be reused later with little to no modifications and omitted for brevity and clarity.

5.3 EX16

ex16.sql was created as follows:

```
CREATE TABLE CasesAndDeathsByContinent(
  continent TEXT,
  day INT8,
  month INT8,
  year INT8,
  cases INT64,
  deaths INT64
);

CREATE TABLE CasesAndDeathsByContinentDate(
  continent TEXT,
  day INT8,
  month INT8,
  year INT8,
  cases INT64,
  deaths INT64
);

INSERT INTO CasesAndDeathsByContinent(continent, day, month, year, cases, deaths)
SELECT
  Countries.continentExp, CountryDateAlt.day, CountryDateAlt.month, CountryDateAlt.year,
FROM
  CountryDateAlt, Countries
WHERE
  CountrydateAlt.countriesAndTerritories = Countries.countriesAndTerritories;

INSERT INTO CasesAndDeathsByContinentDate(continent, day, month, year, cases, deaths)
SELECT
  continent, day, month, year, SUM(cases) AS cases, SUM(deaths) AS deaths
FROM CasesAndDeathsByContinent
GROUP BY continent, day, month, year
ORDER BY continent ASC, year ASC, month ASC, day ASC;

SELECT *
FROM CasesAndDeathsByContinentDate;

DROP TABLE CasesAndDeathsByContinent;
DROP TABLE CasesAndDeathsByContinentDate;
```

With CasesAndDeathsByContinent being an intermediary transient relation used to retrieve the relevant Countries.continentExp.

5.4 EX17

ex17.sql was created as follows:

```
CREATE TABLE CountryCaseDeaths(
  country TEXT,
  date TEXT,
  cases INT64,
  deaths INT64
);

CREATE TABLE CountryPopulations(
  country TEXT,
  population INT16,
  cases INT64,
  deaths INT64
);

CREATE TABLE Percentages(
  country TEXT,
  case_per REAL,
  deaths_per REAL
);

INSERT INTO CountryCaseDeaths(country, date, cases, deaths)
SELECT
  countriesAndTerritories, dateRep, SUM(cases) as cases, SUM(deaths) as deaths
FROM
  CountryDate
GROUP BY countriesAndTerritories;

INSERT INTO CountryPopulations(country, population, cases, deaths)
SELECT
  CountryCaseDeaths.country, Countries.popData2019, CountryCaseDeaths.cases, CountryCase
FROM
  CountryCaseDeaths, Countries
Where
  CountryCaseDeaths.country = Countries.countriesAndTerritories;

INSERT INTO Percentages(country, case_per, deaths_per)
SELECT
  country, (((cases * 1.0) / (population * 1.0)) * 100.0), (((deaths * 1.0) / (populatio
FROM
  CountryPopulations
GROUP BY country;

SELECT *
FROM Percentages;

DROP TABLE CountryCaseDeaths;
DROP TABLE CountryPopulations;
DROP TABLE Percentages;
```

With CountryCaseDeaths being an intermediary transient relation used for summing the cases and deaths for each country. Additionally, CountryPopulations is the next intermediary transient relation used to retrieve the relevent Countries.popData2019.

5.5 EX18

ex18.sql was created by duplicating *ex17.sql* and changing the percentages intermediary transient relation population creation and population slightly as well as changing the final selection operations from:

```
INSERT INTO Percentages(country, case_per, deaths_per)
SELECT
  country, (((cases * 1.0) / (population * 1.0)) * 100.0), (((deaths * 1.0) / (populatio
FROM
  CountryPopulations
GROUP BY country;

SELECT *
FROM Percentages;
```

to:

```
INSERT INTO Percentages(country, deaths_per_cases)
SELECT
  country, (((deaths * 1.0) / (cases * 1.0)) * 100.0)
FROM
  CountryPopulations
GROUP BY country;

SELECT country, deaths_per_cases
FROM Percentages
ORDER BY deaths_per_cases DESC
LIMIT 10;
```

5.6 EX19

ex19.sql was created as follows:

```
CREATE TABLE UKCaseDeath(
  country TEXT,
  day INT8,
  month INT8,
  year INT8,
  cases INT64,
  deaths INT64
);
```

```
CREATE TABLE UKCaseDeathTotal(
  day INT8,
  month INT8,
  year INT8,
  cases INT64,
  deaths INT64
);

INSERT INTO UKCaseDeath(country, day, month, year, cases, deaths)
SELECT
  countriesAndTerritories, day, month, year, cases, deaths
FROM
  CountryDateAlt
WHERE
  countriesAndTerritories='United_Kingdom'
ORDER BY
  year ASC, month ASC, day ASC;

INSERT INTO UKCaseDeathTotal(day, month, year, cases, deaths)
SELECT
  day, month, year,
  SUM(UKCaseDeath.cases) OVER (ORDER BY year ASC, month ASC, day ASC) AS cases,
  SUM(UKCaseDeath.deaths) OVER (ORDER BY year ASC, month ASC, day ASC) AS deaths
FROM UKCaseDeath;

SELECT * FROM UKCaseDeathTotal;

DROP TABLE UKCaseDeath;
DROP TABLE UKCaseDeathTotal;
```

With UKCaseDeath being an intermediary transient relation.