COMP1204 Coursework 2

Flynn Law

May 2024

1 Relational Model

1.1 EX1

Attribute	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

1. **dateRep → day, month, year**
2. **countriesAndTerritories → geoId**
3. **countriesAndTerritories → countryterritoryCode**
4. **countriesAndTerritories → popData2020**
5. **countriesAndTerritories → continentExp**
6. **dateRep, geoId → cases**
7. **dateRep, geoId → deaths**
8. **geoId → countriesAndTerritories**
9. **countryterritoryCode → countriesAndTerritories**

1.3 EX3

Note: continentExp is assumed as data originates from the EU Open Data Portal, which only includes information regarding COVID-19 in EU/EEA countries.

1. **dateRep, geoId**
2. **dateRep, countryterritoryCode**
3. **dateRep, cases, deaths**
4. **dateRep, countriesAndTerritories**

5. **day, month, year, geoId**
6. **day, month, year, countryterritoryCode**
7. **day, month, year, cases, deaths**
8. **day, month, year, countriesAndTerritories**

1.4 EX4

Primary Key selected: **dateRep, countryterritoryCode**
This is because the dateRep and countryterritoryCode format is suitable for searching through data clearly, as they are clearly defined formats. (dd/mm/yyyy and XX respectively). Additonally, the country code format is widely known which makes it suitable for searching

2 Normalisation

2.1 EX5

dateRep $\rightarrow$ day, month year but
geoId $\nrightarrow$ day, month, year

2.2 EX6

In 2nd Normal Form, the relation would be split as follows:

dateRep	countryterritoryCode	cases	deaths

dateRep	day	month	year

countryterritoryCode	countriesAndTerritories	geoID	popData2020	continentExp

This is because day, month and year do not determine the countryterritoryCode, so must be apart of a different relation. Furthermore, countriesAndTerritories, geoId, popData2020 and continentExp do not determine the dateRep Finally, cases and deaths rely both on geoId and countryterritoryCode, therefore must be moved to a different relation.

2.3 EX7

geoId $\rightarrow$ countriesAndTerritories
geoId $\rightarrow$ popData2020
geoId $\rightarrow$ continentExp
countriesAndTerritories $\rightarrow$ geoID
countriesAndTerritories $\rightarrow$ popData2020
countriesAndTerritories $\rightarrow$ continentExp

2.4 EX8

In 3rd Normal Form, the relation would be as follows:

dateRep	countryterritoryCode	cases	deaths

dateRep	day	month	year

countryterritoryCode	geoID

geoID	countriesAndTerritories

countriesAndTerritories	popData2020	continentExp

This is because of the transitive dependencies expressed in **EX7**. Therefore, the attributes have been placed in new tables, where there are no new non-key attributes that determine other ones.

2.5 EX9

The relations are already in Boyce-Codd Normal Form as day, month and year rely fully on dateRep, as do cases and deaths fully relying on the key dateRep, countryterritoryCode. Furthermore, in the relations countriesAndTerritories $\rightarrow$ $popData2020 and countriesAndTerritories \rightarrow continentExp, countriesAndTerritories is a superkey$.

3 Modelling

3.1 EX10

1. Run sqlite3 in terminal (or run in DataGrip)
2. Run `.mode csv` to switch to csv import
3. Run `.import dataset.csv dataset` to import the dataset file into a table called dataset
4. Run `.output dataset.sql` to set the output file 5. Run `.dump` to dump the contents of the sql file into the working directory
6. Run the created `.sql` file with a blank database to populate the databases with the tables and data.

3.2 EX11

Create tables that correspond to the BCNF relation detailed in **EX8**, with primary and foreign keys for each table.

```
CREATE TABLE IF NOT EXISTS DATES(
    dateRep TEXT NOT NULL UNIQUE,
    day TEXT NOT NULL,
    month TEXT NOT NULL,
    year TEXT NOT NULL,
    PRIMARY KEY(dateRep)
);

CREATE TABLE IF NOT EXISTS COUNTRYIDS(
    countryterritoryCode TEXT NOT NULL UNIQUE,
    geoID TEXT NOT NULL UNIQUE,
    PRIMARY KEY(countryterritoryCode),
    FOREIGN KEY(geoID) REFERENCES COUNTRYNAMES(geoID)
);
CREATE TABLE IF NOT EXISTS COUNTRYNAMES(
    geoID TEXT NOT NULL UNIQUE,
    countriesAndTerritories TEXT NOT NULL UNIQUE,
    PRIMARY KEY(geoID),
    FOREIGN KEY (countriesAndTerritories) REFERENCES POPULATION(countriesAndTerritories)
);
CREATE TABLE IF NOT EXISTS POPULATION(
    countriesAndTerritories TEXT NOT NULL UNIQUE,
    popData2020 TEXT NOT NULL,
```

```
    continentEXP TEXT NOT NULL,
    PRIMARY KEY(countriesAndTerritories)
);

CREATE TABLE IF NOT EXISTS COVID19CASESANDDEATHS(
    dateRep TEXT NOT NULL,
    countryterritoryCode TEXT NOT NULL,
    cases TEXT NOT NULL,
    deaths TEXT NOT NULL,
    PRIMARY KEY(dateRep, countryterritoryCode),
    FOREIGN KEY (dateRep) REFERENCES DATES(dateRep),
    FOREIGN KEY (countryterritoryCode) REFERENCES COUNTRYIDS(countryterritoryCode)
);
```

The primary key of the dates table is the dateRep attribute, and the primary key of the countries table is the countryterritoryCode. The tables are the same as detailed in **EX6**. The cases and deaths table has the dateRep and countryterritoryCode as foreign keys, linking this table with the other two tables.

3.3 EX12

From the dataset table (where the imported dataset is stored) populate all the tables created with appropriate data. If data is already entered, ignore (such as multiple dates in the DATES table). Compile all the data from the other tables into the final table, COVID19CASESANDDEATHS, by using inner joins to select data from other tables.

```
INSERT OR IGNORE INTO DATES(dateRep, day, month, year)
SELECT DISTINCT dateRep, day, month, year
FROM dataset;
INSERT OR IGNORE INTO COUNTRYIDS
SELECT DISTINCT countryterritoryCode, geoId
FROM dataset;
INSERT OR IGNORE INTO COUNTRYNAMES
SELECT DISTINCT geoId, countriesAndTerritories
FROM dataset;
INSERT OR IGNORE INTO POPULATION
SELECT DISTINCT countriesAndTerritories, popData2020,continentExp
FROM dataset;
INSERT INTO COVID19CASESANDDEATHS (dateRep, countryterritoryCode, cases, deaths)
SELECT dt.dateRep, c.countryterritoryCode, d.cases, d.deaths
FROM dataset AS d
INNER JOIN DATES AS dt ON d.dateRep = dt.dateRep
INNER JOIN COUNTRYIDS AS c ON d.countryterritoryCode = c.countryterritoryCode;
```

3.4 EX13

Running the following code:

```
sqlite3 coronavirus.db < dataset.sql
sqlite3 coronavirus.db < ex11.sql
sqlite3 coronavirus.db < ex12.sql
```

generates a fully populated database.

4 Querying

4.1 EX14

Add all the cases, and add all the deaths and present them.

```
SELECT SUM(CAST(cases AS INTEGER)) as total_cases, SUM(CAST(deaths AS INTEGER)) as total_deaths
FROM COVID19CASESANDDEATHS;
```

4.2 EX15

Present the dates (in dd/mm/yyyy format), cases and country names (will only be UK cases) and order them to present in order. Full country names (from the COUNTRYNAMES table) will be presented instead of the code stored in the table.

```
SELECT COVID19CASESANDDEATHS.dateRep, cases, countriesAndTerritories AS country FROM COVID19CASESANDDEATHS
INNER JOIN COUNTRYIDS ON COVID19CASESANDDEATHS.countryterritoryCode = COUNTRYIDS.countryterritoryCode
INNER JOIN COUNTRYNAMES ON COUNTRYIDS.geoID = COUNTRYNAMES.geoID
INNER JOIN DATES ON COVID19CASESANDDEATHS.dateRep = DATES.dateRep
WHERE countriesAndTerritories = 'United_Kingdom'
ORDER BY year, month, day;
```

4.3 EX16

Ordered by date, presents the cases and deaths for every country on a particular day.

```
SELECT
    COUNTRYNAMES.countriesAndTerritories AS country_name,
    COVID19CASESANDDEATHS.dateRep AS date,
    COVID19CASESANDDEATHS.cases AS number_of_cases,
    COVID19CASESANDDEATHS.deaths AS number_of_deaths
FROM
    COVID19CASESANDDEATHS
JOIN
    COUNTRYIDS ON COVID19CASESANDDEATHS.countryterritoryCode = COUNTRYIDS.countryterritoryCode
JOIN COUNTRYNAMES ON COUNTRYIDS.geoID = COUNTRYNAMES.geoID
ORDER BY
    substr(dateRep, 7, 4) || '/' || substr(dateRep, 4, 2) || '/' || substr(dateRep, 1, 2),
    COUNTRYNAMES.countriesAndTerritories;
```

4.4 EX17

Calculates percentage of cases in terms of entire population of a country, and does the same for deaths. Presents this for every country in the database.

```
SELECT COUNTRYNAMES.countriesAndTerritories AS country, ROUND(SUM(cases)*1.0/popData2020*100, 2)
    AS casespercentageofpop, ROUND(SUM(deaths)*1.0/popData2020*100, 2) AS deathaspercentageofpop
FROM COVID19CASESANDDEATHS
JOIN COUNTRYIDS ON COVID19CASESANDDEATHS.countryterritoryCode = COUNTRYIDS.countryterritoryCode
JOIN COUNTRYNAMES ON COUNTRYIDS.geoID = COUNTRYNAMES.geoID
JOIN POPULATION ON COUNTRYNAMES.countriesAndTerritories = POPULATION.countriesAndTerritories
GROUP BY
    COUNTRYNAMES.countriesAndTerritories;
```

4.5 EX18

Calculates the deaths out of the population for each country, orders the percentages in descending order, and presents the top 10.

```
SELECT COUNTRYNAMES.countriesAndTerritories AS countries, ROUND(SUM(CAST(deaths AS
    FLOAT))/SUM(CAST(cases AS FLOAT))*100,2) AS percentdeathsofcountrycases
FROM COVID19CASESANDDEATHS
LEFT JOIN COUNTRYIDS ON COVID19CASESANDDEATHS.countryterritoryCode =
    COUNTRYIDS.countryterritoryCode
LEFT JOIN COUNTRYNAMES ON COUNTRYIDS.geoID = COUNTRYNAMES.geoID
GROUP BY COUNTRYNAMES.countriesAndTerritories
ORDER BY percentdeathsofcountrycases DESC
LIMIT 10;
```

Gets the cumulative total of cases and deaths for the UK only by adding the count to the row before.

4.6 EX19

```
SELECT dateRep,
SUM(deaths) OVER (ROWS UNBOUNDED PRECEDING) AS cumulative_UK_deaths,
SUM(cases) OVER (ROWS UNBOUNDED PRECEDING) AS cumulative_UK_cases
FROM COVID19CASESANDDEATHS
WHERE countryterritoryCode = 'GBR';
```