Data Management Assignment 2 : SQL

pkll1g19 - Pui Kwan Lauren Lee
Student ID : 30736501

May 2020

1. The Relational Model

EX1 Relation represented in the dataset

The relation in our spreadsheet is an instance of the following relation schema, Covid:
*Covid(Date : dateRep, Integer : day, Integer : month, Integer : year, Integer : cases,
Integer : deaths, String : countriesAndTerritories, String : geoId,
String : countryterritoryCode, Integer : popData2018, String : continentExp)*

EX2 Functional dependencies that exist in the database

List of functional dependencies:

- dateRep → day, month, year
- day, month, year → dateRep
- countriesAndTerritories → geoId, country, popData2018, continentExp
- geoId → countriesAndTerritories, countryTerritoryCode popData2018, continentExp
- countryTerritoryCode → countriesAndTerritories, geoId, popData2018, continentExp
- dateRep, countriesAndTerritories → cases, deaths
- dateRep, countryTerritoryCode → cases, deaths
- dateRep, geoId → cases, deaths

Assumptions include that the same number of cases and the number of deaths can occur on different days in different countries, hence why it is not a determinant in any functional dependencies. Likewise, multiple countries belong to the same continents. Data on different countries are recorded simultaneously.

The 2018 population (popData2018) can be the same for more than one country. Even if the current values are distinct, if we were to add a new country (e.g. North Korea), the population can be the same.

We are making the assumption that there is a one to one mapping between countriesAndTerritories, countryTerritoryCode (though with null values), and geoID as these the latter two seems to be abbreviations of the former.

EX3 Potential candidate keys

Potential candidate keys include the following:

- dateRep, countriesAndTerritories
- dateRep, countryTerritoryCode
- dateRep, geoId

EX4 Identifying a suitable primary key

A suitable primary key will be **dateRep, geoId** as **geoId** is designed to uniquely identify a location; **countryAndTerritories** are more likely to change if the country's name is to be changed. In addtion, **countryTerritoryCode** contains null values.

2. Normalisation

EX5 List any partial-key dependencies in the relation as it stands

The partial key dependencies that exist in the relation
*Covid(**dateRep**, **geoId**, day, month, year, cases, deaths, countriesAndTerritories, countryterritoryCode, popData2018, continentExp)*

Partial key dependencies include:

- dateRep → day, month, year
- geoId → countriesAndTerritories, countryterritoryCode, popData2018, continentExp

EX6 Additional relations created as part of the decomposition

New relations to be added:

- *DateRec(**dateRep**, day, month, year)*
- *Location(**geoId**, countriesAndTerritories, countryterritoryCode, popData2018, continentExp)*

As well as the original relation

- *Covid(**dateRep**, **geoId**, cases, deaths)*

EX6 Converting to 2NF

In order for the relation to be in 2nd Normal Form, it must be in 1st normal form with no partial key dependencies. Since the relation meets the first criteria (there are no nested/multi-values in any of the columns, and no repeating columns/rows), we only have to rid of the partial key dependencies.

1. From part 2.1, we can see that there exists partial key dependencies on dateRep and geoId. Looking at the functional dependency with dateRep, we copy over the determinant and move all the dependent attributes to a new relation. Again, dateRep is made the primary key in this new relation as it functionally determines the rest of the attributes.

   *Covid(**dateRep**, **geoId**, cases, deaths, countriesAndTerritories, countryterritoryCode, popData2018, continentExp)*
   *DateRec(**dateRep**, day, month, year)*

2. Then repeat for the second functional dependency mentioned in 2.1: create a new relation, copy over the determinant geoId, then move all of the dependent attributes to this new dependency.

   *Covid(**dateRep**, **geoId**, cases, deaths)*
   *DateRec(**dateRep**, day, month, year)*
   *Locations(**geoId**, countriesAndTerritories, countryterritoryCode, popData2018, continentExp)*

3. Our relational database schema is the following:

   *Covid(**Date : dateRep**, **String : geoId**, Integer : cases, Integer : deaths)*
   *DateRec(**Date : dateRep**, Integer : day, Integer : month, Integer : year)*
   *Locations(**String : geoId**, String : countriesAndTerritories, String : countryterritoryCode, Integer : popData2018, String : continentExp)*

Primary keys are in bold.

EX7 Transitive dependencies present

There are no transitive dependencies. **Covid** - no transitive dependencies, as all attributes depend on **dateRep, geoId** only
DateRec - no transitive dependencies, all attributes depend on **dateRep**
Locations - no transistive dependencies, as even with the dependencies following the A $\rightarrow$ B $\rightarrow$ C, B $\rightarrow$ A.

- geoId $\rightarrow$ countriesAndTerritories $\rightarrow$ countryterritoryCode $\rightarrow$ popData2018
- geoId $\rightarrow$ countryterritoryCode $\rightarrow$ countriesAndTerritories $\rightarrow$ popData2018
- geoId $\rightarrow$ countriesAndTerritories $\rightarrow$ countryterritoryCode $\rightarrow$ continentExp
- geoId $\rightarrow$ countryterritoryCode $\rightarrow$ countriesAndTerritories $\rightarrow$ continentExp

EX8 Converting to 3NF

As there are no transitive dependencies in any of the tables, it is already in 3NF.

Our relational database schema is still the following:

*Covid(**Date : dateRep**, **String : geoId**, Integer : cases, Integer : deaths)*
*DateRec(**Date : dateRep**, Integer : day, Integer : month, Integer : year)*
*Locations(**String : geoId**, String : countriesAndTerritories, String : countryterritoryCode, Integer : popData2018, String : continentExp)*

EX9: Is the relation in Boyce-Codd Normal Form?

All determinants in the relations apart from **CountryTerritory** are the candidate keys or determines the candidate keys. Thus they are in Boyce-Codd Normal Form. **CountryTerritory** is in 3NF and not BCNF as it violates the requirement for all determinants needing to be the candidate key.

Below shows the functional dependencies in each relation.

- **Covid**
  - dateRep, geoId → cases, deaths (Candidate key)
- **DateRec**
  - dateRep → day, month, year (Candidate key)
  - day, month, year → dateRep (Candidate key)
- **Locations**
  - geoId → countryTerritoryCode, countriesAndTerritories, popData2018, continentExp (Candidate key)
  - countriesAndTerritories → countryTerritoryCode, geoID, popData2018, continentExp (Candidate key)
  - countryTerritoryCode → geoID, countriesAndTerritories, popData2018, continentExp (Candidate key)

3. Modelling

EX10

The dataset.sql file contains the sql statement for creating and populating the dataset table.

EX11

Indexes have been added to the foreign key attributes as they are often used for joins. These include **dateRepIndex** and **geoIDIndex** (these being unique). Additionally, because we may look at individual continent's, we will add an index to **continentExpIndex**.

EX12

The ex12.sql contains the SQL needed.

EX13

Database can be reconstructed by executing dataset.sql, ex11.sql, and ex12.sql.

4. Querying

EX14

```
SELECT SUM(cases) AS "total cases", SUM(deaths) AS "total deaths" FROM Covid;
```

EX15

```
SELECT Covid.dateRep AS date, cases AS "number of cases" FROM Covid
JOIN DateRec ON DateRec.dateRep = Covid.dateRep
WHERE Covid.geoID = "UK"
ORDER BY year, month, day;
```

EX16

```
SELECT Locations.continentExp AS continent, Covid.dateRep,
       SUM(Covid.cases) AS "number of cases", SUM(Covid.deaths) AS "number of deaths"
FROM Locations
INNER JOIN Covid ON Covid.geoID = Locations.geoID
INNER JOIN DateRec ON DateRec.dateRep = Covid.dateRep
GROUP BY Locations.continentExp, DateRec.dateRep
ORDER BY Locations.continentExp, DateRec.year, DateRec.month, DateRec.year;
```

EX17

```
SELECT Locations.countriesAndTerritories AS country,
       SUM(Covid.cases) * 100.0 / Locations.popData2018 AS "% cases of population",
       SUM(Covid.deaths) * 100.0 / Locations.popData2018 AS "% deaths of population"
FROM Locations
INNER JOIN Covid ON Covid.geoID = Locations.geoID
GROUP BY country
ORDER BY country;
```

EX18

```
SELECT Locations.countriesAndTerritories AS 'country name',
SUM(Covid.deaths) * 100.0 / SUM(Covid.cases) AS "% deaths of country cases"
FROM Locations
JOIN Covid ON Locations.geoID = Covid.geoID
GROUP BY Locations.countriesAndTerritories
ORDER BY "% deaths of country cases" DESC
LIMIT 10;
```

EX19

```
SELECT DateRec.dateRep, SUM(Covid.deaths)
OVER (ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS 'cumulative UK deaths',
SUM(Covid.cases)
OVER (ORDER BY DateRec.year, DateRec.month, DateRec.day
      ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS 'cumulative UK cases'
FROM Covid
INNER JOIN DateRec ON Covid.dateRep = DateRec.dateRep
WHERE geoID = "UK"
ORDER BY DateRec.year, DateRec.month, DateRec.day;
```