# COMP1204 - Data Management
# Database Coursework Report

Christos Mousoulides

Student ID: 31225551
Email: cm10g19@soton.ac.uk

May 12, 2021

## 0.1 The Relational Model

### 0.1.1 EX1

This is the relation represented in the dataset file:

COVIDDataset(NUMERIC: dateRep, INTEGER: day, INTEGER: month, INTEGER: year, INTEGER: cases, INTEGER: deaths, BLOB: countriesAndTerritories, TEXT: geoId, TEXT: countryterritoryCode, INTEGER: popData2019, TEXT: continentExp)

### 0.1.2 EX2

This is the minimal set of Functional Dependencies:

(day,month,year) $->$ dateRep,
countriesAndTerritories $->$ geoId,
countryterritoryCode $->$ popData2019,
geoId $->$ countryterritoryCode,
geoId $->$ popData2019,
geoId $->$ continentExp,
(dateRep, geoId) $->$ cases,
(dateRep,cases,geoId $->$ deaths)

### 0.1.3 EX3

All of the possible candidate keys are listed below:

(dateRep, geoId)
(dateRep, countriesAndTerritories )
(dateRep, cases, deaths, countriesAndTerritories)
(dateRep, cases, deaths, geoId)
(day, month, year, geoId)
(day, month, year, countriesAndTerritories )
(day,month,year, cases, deaths, countriesAndTerritories)
(day,month,year, cases, deaths, geoId)

### 0.1.4 EX4

The candidate key I identify to be a suitable the Primary Key is:

(dateRep,geoId)

Any candidate keys that had day,month and year instead of dateRep are excluded as all three can be derived from dateRep. Also, the geoId column was

chosen over countriesAndTerritories, because it's a unique code given to each territory. Furthermore, cases and deaths were redundant as they could be removed and the outcome would still be the same.

## 0.2 Normalisation

### 0.2.1 EX5

The Partial Dependencies of this relation are:

dateRep $->$ (day,month,year)
geoId $->$ (countriesAndTerritories, countryTerritoryCode, popData2019, continentExp)

### 0.2.2 EX6

CountryAndTerritoryInfo(
TEXT: geoId (Primary Key),
BLOB: countriesAndTerritories,
TEXT: countryterritoryCode,
INTEGER: popData2019,
TEXT: continentExp)

COVIDStats(
NUMERIC: dateRep (Foreign Key),
TEXT: geoId (dateRep and geold = Primary Key),
INTEGER: cases,
INTEGER: deaths)

Firstly the attributes day, month and year were removed from the dataset as they could be derived from dateRep, so the first partial dependency was eliminated. Then, a relation containing only data related to each country/territory was created, so that relevant country information can be grouped together, with its primary key being "geoId", because all of its other attributes can be derived from it and this eliminated the second partial dependency of the dataset. After that, a second relation "COVIDStats" was created that includes all of the data related to the virus. "dateRep" and "geold" together act as a Primary Key because they are both needed for the attributes of cases and deaths to be displayed uniquely for each territory.

### 0.2.3 EX7

The only Transitive Dependency that exists is:

geoId $->$ countryTerritoryCode $->$ popData2019

### 0.2.4 EX8

CountryAndTerritoryInfo(
TEXT: geoId (Primary Key and Foreign Key),
BLOB: countriesAndTerritories,
TEXT: countryterritoryCode,
TEXT: continentExp)

PopInfo(
TEXT geoId (Primary Key),
INTEGER: popData2019)

COVIDStats(
NUMERIC: dateRep (Foreign Key),
TEXT: geoId (dateRep and geold = Primary Key),
INTEGER: cases,
INTEGER: deaths)

To achieve the 3NF the first relation "CountryAndTerritoryInfo" had its attribute "popData2019" removed and added into a new relation that contains information only about the population of each country, thus removing the transitive dependency present. Also the primary key of the first relation "geoId" now acts also as a foreign key and is related to the relation "PopInfo".

### 0.2.5 EX9

The relations listed in the previous section are all in Boyce-Codd Normal Form, because all of their dependencies can be identified as superkeys or are trivial functional dependncies, thus all of the primary keys can determine each of the rows in their respective relation.

## 0.3 Modelling

### 0.3.1 EX10

For this part of the coursework I firstly used file transferring software to bring the dataset file in my working directory. Then, I used the command "sqlite3 coronavirus.db" to create a databse with the required name. After that, I created a table that could hold the values from the dataset file. The code used for the table is listed below:

```
CREATE TABLE dataset(
"dateRep" NUMERIC,
"day" INTEGER,
"month" INTEGER,
```

```
"year" INTEGER,
"cases" INTEGER,
"deaths" INTEGER,
"countriesAndTerritories" BLOB,
"geoId" TEXT,
"countryterritoryCode" TEXT,
"popData2019" INTEGER,
"continentExp" TEXT);
```

After the table was created the commands ".mode csv" followed by ".import dataset.csv dataset" were used to populate the new table created with the contents of the dataset.csv file. Lastly, I exported the dataset onto a file called "dataset.sql" by using the commands ".output dataset.sql" and ".dump".

### 0.3.2 EX11

For this exercise I used the following commands to create the schema for my normalised relation

```
CREATE TABLE PopInfo(
"geoId" TEXT PRIMARY KEY ,
"popData2019" INTEGER
);

CREATE TABLE CountryAndTerritoryInfo(
"geoId" TEXT PRIMARY KEY,
"countriesAndTerritories" BLOB,
"countryterritoryCode" TEXT,
"continentExp" TEXT,
FOREIGN KEY (geoId) REFERENCES PopInfo(geoId)
);

CREATE TABLE COVIDStats(
"dateRep" NUMERIC,
"geoId" TEXT,
"cases" INTEGER,
"deaths" INTEGER,
PRIMARY KEY(dateRep,geoId)
FOREIGN KEY (geoId) REFERENCES CountryAndTerritoryInfo(geoId)
);
```

### 0.3.3 EX12

For this exercise I used the following commands to populate my normalised relation

```
INSERT INTO CountryAndTerritoryInfo(geoId,
```

```
countriesAndTerritories,
countryterritoryCode,continentExp)

  SELECT DISTINCT geoId,
   countriesAndTerritories, countryterritoryCode,
   continentExp
  FROM dataset;

INSERT INTO PopInfo(geoId,popData2019)
  SELECT DISTINCT geoId,popData2019
  FROM dataset;

 INSERT INTO COVIDStats(dateRep,geoId,cases, deaths)
  SELECT dateRep, geoId,cases,deaths
  FROM dataset;
```

### 0.3.4 EX13

In this exercise to test If all three of the previous exercises were working in unison correctly I manually deleted the tables from my database by using "DROP TABLE". After that I used the commands listed below and successfully ended up with a populated database.

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

## 0.4 Querying

### 0.4.1 EX14

In this exercise the results of the query had to be displayed when run so the first two commands used where to make sure that the table displayed is easy to follow.

```
.mode columns
.headers ON
SELECT SUM(cases) AS [totalCases],
  SUM(deaths) AS [totalDeaths]
  FROM COVIDStats;
```

### 0.4.2 EX15

The code listed below reveals the number of cases by date in increasing date order for the United Kingdom.

```
.mode columns
.headers ON
SELECT dateRep AS [Date] ,cases AS [NumberOfCases]
 FROM COVIDStats
 WHERE geoId = 'UK'
 ORDER BY dateRep ASC;
```

### 0.4.3 EX16

The code listed below reveals The number of cases and deaths by date, in increasing date order, for each continent.

```
.mode columns
.headers ON
SELECT CountryAndTerritoryInfo.continentExp AS [Continent],
 COVIDStats.dateRep AS [Date] ,
 COVIDStats.cases AS [NumberOfCases], COVIDStats.deaths AS [NumberOfDeaths]
 FROM COVIDStats
 INNER JOIN CountryAndTerritoryInfo
 ON COVIDStats.geoId = CountryAndTerritoryInfo.geoId
 WHERE COVIDStats.cases != 'cases'
 ORDER BY dateRep ASC;
```

### 0.4.4 EX17

The code listed below calculates and displays the total number of cases and deaths as a percentage of the population for each country.

```
.mode columns
.headers ON
SELECT CountryAndTerritoryInfo.countriesAndterritories AS [Country],
 (SUM(COVIDStats.cases * 1.0) / (PopInfo.popData2019 * 1.0)) * 100
 AS [TotalCase%],
 (SUM(COVIDStats.deaths * 1.0) / (PopInfo.popData2019 * 1.0)) * 100
 AS [TotalDeath%]
 FROM COVIDStats
 LEFT JOIN CountryAndTerritoryInfo
 ON COVIDStats.geoId = CountryAndTerritoryInfo.geoId
 LEFT JOIN PopInfo
 ON CountryAndTerritoryInfo.geoId = PopInfo.geoId
 WHERE COVIDStats.geoId != 'geoId'
 GROUP BY COVIDStats.geoId;
```

### 0.4.5 EX18

the code listed below ranks the top 10 countries, by percentage total deaths out of total cases in that country in descending order.

```
.mode columns
.headers ON
SELECT CountryAndTerritoryInfo.countriesAndterritories AS [Country],
 (SUM(COVIDStats.deaths * 1.0 ) / SUM(COVIDStats.cases * 1.0 )) * 100
 AS [DeathPerCase%]
 FROM COVIDStats
 LEFT JOIN CountryAndTerritoryInfo
 ON COVIDStats.geoId = CountryAndTerritoryInfo.geoId
 WHERE COVIDStats.geoId != 'geoId'
 GROUP BY COVIDStats.geoId
 ORDER BY 2 DESC
 LIMIT 10;
```

### 0.4.6 EX19

The code listed below is intended to give a cumulative sum of both the cases and deaths in the United Kingdom, increasing by the day. This is done with the use of Window Functions.

```
.mode columns
.headers ON
SELECT dateRep AS [Date],
 SUM(deaths) OVER(ORDER BY dateRep) AS [cumulativeUkDeaths],
 SUM(cases) OVER(ORDER BY dateRep) AS [cumulativeUkDeaths]
 FROM COVIDStats WHERE geoId = 'UK'
 ORDER BY dateRep ASC;
```