COMP1204: Data Management
Coursework Two: Representing Covid-19 data

PROKOPIS GEORGIOU
STUDENT ID: 34077499

CORONAVIRUS DATA FROM THE EU OPEN DATA PORTAL. RELATIONAL MODEL, NORMALISATION, MODELLING, QUERYING

Introduction

The COVID-19 pandemic has presented unprecedented challenges to global health systems and societies, prompting the need for innovative data management strategies. In their COMP1204: Data Management coursework, the students have been tasked with addressing this demand by organizing COVID-19 data sourced from the EU Open Data Portal into a SQLite database. Their objective is to establish a structured framework that facilitates insightful analysis of pandemic trends. Throughout this report, the students will outline their methodology for designing the database. This entails conceptualizing a relational model, ensuring data organization adheres to normalization principles, and constructing queries to extract meaningful insights. Each step will be thoroughly explained, from identifying relationships between different data points to optimizing the database's efficiency through normalization. Furthermore, the report will demonstrate how the students translated their plan into action. This includes setting up the database, importing the COVID-19 dataset, and executing queries to uncover valuable insights. The aim is to showcase how a well-designed database can streamline data management processes and empower informed analysis of COVID-19 statistics.

1 The Relational Model

EX1: Express the relation directly represented in the dataset file. Assign relevant SQLite data types to each column.

dataset(dateRep, day, month, year, cases, deaths, countriesAndTerritories, geoId, countryterritoryCode, popData2020, continentExp)

INTEGER	TEXT
day	dateRep
month	countriesAndTerritories
year	geoId
cases	countryterritoryCode
deaths	continentExp
popData2020	

Table 1: Attributes and types

This represents the dataset as a relational table with each column having the appropriate SQLite data type.

EX2: List the minimal set of Functional Dependencies (FDs).

Functional Dependency
(day, month, year) → dateRep
dateRep, countriesAndTerritories → cases
dateRep, countriesAndTerritories → deaths
dateRep, countryterritoryCode → cases
dateRep, countryterritoryCode → deaths
dateRep, geoId → cases
dateRep, geoId → deaths

Functional Dependency
countriesAndTerritories → countryterritoryCode
countriesAndTerritories → continentExp
countryterritoryCode → continentExp
countriesAndTerritories → popData2020
geoId → countriesAndTerritories
geoId → countryterritoryCode
geoId → popData2020

Assumptions: 1) dateRep can represent day, month and year. Thus, it would be redundant to write them separately. 2) countriesAndTerritories, geoId, countryterritoryCode, popData2020 and continentExp are all redundant since information is unnecessarily repeated.

EX3: From your minimal set of functional dependencies, list the potential candidate keys.

From these FDs, the potential candidate keys are: (dateRep, geoId), (dateRep, countriesAndTerritories), (dateRep, countryterritoryCode).

EX4: Identify a suitable primary key, and justify your decision.

The composite key (dateRep, geoId) appears to be the most suitable primary key. It provides a unique identifier for each record in the dataset and offers a high level of granularity by combining the date of reporting with the unique country or territory. This primary key ensures that each record is uniquely identifiable and allows for efficient data retrieval and manipulation.

2 Normalisation

EX5: List any partial-key dependencies in the relation as it stands and any resulting additional relations you should create as part of the decomposition.

1. dateRep → day
2. dateRep → month
3. dateRep → year
4. geoId → popData2020
5. geoId → continentExp
6. geoId → countryterritoryCode
7. geoId → countriesAndTerritories

Additional Tables:

1. (dateRep, geoId) → (cases,deaths)
2. dateRep → (day, month, year)
3. geoId → (countriesAndTerritories, countryterritoryCode, popData2020, continentExp)

EX6: Use decomposition and your answer to the above to achieve 2nd Normal Form, introducing appropriate new relations. List the new relations and their fields, types and keys. Explain the process you took.

Additional Relations:

1. Condition(dateRep, geoId, cases, deaths)
2. Date(dateRep, day, month, year)
3. Country(geoId, countriesAndTerritories, popData2020, continentExp)

The primary keys for the relation Condition are (dataRep and geoId). For the relation Date it is dataRep and fro relation Country, the primary key is geoId. The types are the same as in Table 1. To achieve 2nd Normal Form (2NF), I decomposed the original relation into smaller relations to eliminate partial-key dependencies. The decomposition process involved identifying attributes that were functionally dependent on only part of the primary key and separating them into new relations.

Condition Relation

This relation includes attributes that are functionally dependent on both the date of reporting (`dateRep`) and the unique identifier for the country or territory (`geoId`). The primary key for this relation is a composite key consisting of `dateRep` and `geoId`.

Date Relation

To address the partial-key dependencies related to the date of reporting (`dateRep`), I created a separate relation named `Date`. This relation includes attributes specific to the date, namely `day`, `month`, and `year`. The primary key for this relation is `dateRep`.

Country Relation

Similarly, I created a separate relation named `Country` to handle attributes related to countries or territories. This relation includes attributes such as `countriesAndTerritories`, `popData2020`, and `continentExp`, which are functionally dependent on the unique identifier for the country or territory (`geoId`). The primary key for this relation is `geoId`.

EX7: List transitive dependencies, if any, in your new relations.

All new relations are in 2NF as they do not contain any transitive dependencies. Therefore, no further decomposition is needed.

EX8: Convert your relations into 3rd Normal Form using your answers to the above. List the new relations and their fields, types and keys. Explain the process you took.

The relations Condition, Date, and Country are in 3rd Normal Form (3NF) because they meet the following criteria:

1. **Atomic Values**: All attributes contain atomic values, meaning they cannot be further divided. In each relation, the attributes represent single, indivisible pieces of information. For example, in the Condition relation, cases and deaths represent single counts of COVID-19 cases and deaths, respectively.
2. **No Transitive Dependencies**: There are no transitive dependencies in any of the relations. A transitive dependency occurs when a non-prime attribute depends on another non-prime attribute, rather than directly on the primary key. However, in all three relations, each non-prime attribute is fully functionally dependent on the primary key, and there are no indirect dependencies between non-prime attributes. For example, in the Country relation, countriesAndTerritories, popData2020, and continentExp are all directly dependent on the primary key geoId.
3. **Every Non-Prime Attribute Fully Depends on the Primary Key**: Each non-prime attribute in all three relations is fully functionally dependent on the primary key. This means that the value of any non-prime attribute is uniquely determined by the primary key alone, without any further dependencies. For instance, in the Date relation, day, month, and year are all fully dependent on the primary key dateRep.

EX9: Convert your relations into Boyce-Codd Normal Form. Justify and explain how your relations are in BCNF.

To convert the relations into Boyce-Codd Normal Form (BCNF), we need to ensure that every determinant is a candidate key. Here's how we can justify that the relations Condition, Date, and Country are in BCNF:

Condition Relation:

- Primary key: (dateRep, geoId)
- Non-prime attributes: cases, deaths
- Justification: The primary key (dateRep, geoId) uniquely identifies each tuple in the relation. Both cases and deaths are fully functionally dependent on the primary key. Since the primary key is a composite key, it cannot be decomposed further.

Date Relation:

- Primary key: dateRep
- Non-prime attributes: day, month, year
- Justification: The primary key dateRep uniquely identifies each tuple in the relation. All non-prime attributes (day, month, year) are fully functionally dependent on the primary key. Since the primary key is a candidate key, and there are no dependencies on other non-prime attributes.

Country Relation:

- Primary key: geoId
- Non-prime attributes: countriesAndTerritories, popData2020, continentExp
- Justification: The primary key geoId uniquely identifies each tuple in the relation. All non-prime attributes (countriesAndTerritories, popData2020, continentExp) are fully functionally dependent on the primary key. Since the primary key is a candidate key, and there are no dependencies on other non-prime attributes.

The relations Condition, Date, and Country are in BCNF because every determinant is a candidate key, ensuring that there are no non-trivial functional dependencies on attributes that are not candidate keys.

3 Modelling

EX10

```
sqlite3 coronavirus.db
sqlite> .open coronavirus.db
sqlite> .mode csv
sqlite> .import dataset.csv dataset
sqlite> .output dataset.sql
\\Open DataGrip and set the data source and drivers
sqlite3 coronavirus.db .dump > database.sql
```

EX11

```
CREATE TABLE Date (
    dateRep TEXT NOT NULL PRIMARY KEY,
    day     INTEGER NOT NULL,
    month   INTEGER NOT NULL,
    year    INTEGER NOT NULL
);
CREATE TABLE Country (
    geoId                   TEXT NOT NULL CONSTRAINT Country_pk PRIMARY KEY,
    countriesAndTerritories TEXT NOT NULL,
    countryterritoryCode    TEXT NOT NULL,
    popData2020             INTEGER NOT NULL,
    continentExp            TEXT NOT NULL
);
CREATE TABLE Condition (
    dateRep TEXT NOT NULL,
    geoId   TEXT NOT NULL,
    deaths  INTEGER,
    cases   INTEGER,
    CONSTRAINT Condition_pk PRIMARY KEY (dateRep, geoId)
);
```

EX12

```
-- Populate the Date table
INSERT INTO Date (dateRep, day, month, year)
SELECT DISTINCT dateRep, day, month, year
FROM dataset;

-- Populate the Country table
INSERT INTO Country (geoId, countriesAndTerritories, countryterritoryCode, popData2020, continentExp)
SELECT DISTINCT geoId, countriesAndTerritories, countryterritoryCode, popData2020, continentExp
FROM dataset;

-- Populate the Condition table
INSERT INTO Condition (dateRep, geoId, deaths, cases)
SELECT DISTINCT dateRep, geoId, deaths, cases
FROM dataset;
```

EX13

```
sqlite3 coronavirus.db < dataset.sql
sqlite3 coronavirus.db < ex11.sql
sqlite3 coronavirus.db < ex12.sql
//Confirmimg that I have run these commands and they worked.
```

4 Querying

EX14: The worldwide total number of cases and deaths (with total cases and total deaths as columns)

```
SELECT
    SUM(cases) AS total_cases,
    SUM(deaths) AS total_deaths
FROM
    Condition;
```

EX15: The number of cases by date, in increasing date order, for the United Kingdom (with the date reported and number of cases as columns)

```
SELECT
    Date.dateRep,
    Condition.cases
FROM
    Date
JOIN
    Condition ON Date.dateRep = Condition.dateRep
JOIN
    Country ON Condition.geoId = Country.geoId
WHERE
    Country.countriesAndTerritories = 'United_Kingdom'
ORDER BY
    Date.year ASC,
    Date.month ASC,
    Date.day ASC;
```

EX16: The number of cases and deaths by date, in increasing date order, for each country (with country name, date, number of cases and number of deaths as columns)

```
SELECT
    Country.countriesAndTerritories,
    Condition.dateRep,
    Condition.cases, Condition.deaths
FROM
    Country
JOIN
    Condition ON Country.geoId = Condition.geoId
JOIN
    Date ON Condition.dateRep = Date.dateRep
GROUP BY
    Country.countriesAndTerritories,
    Condition.dateRep
ORDER BY
    Country.countriesAndTerritories ASC,
    Date.year ASC,
    Date.month ASC,
    Date.day ASC;
```

EX17: The total number of cases and deaths as a percentage (to 2DP) of the population, for each country

```
SELECT
    Country.countriesAndTerritories,
    ROUND((SUM(Condition.cases) * 100.0 / Country.popData2020), 2) AS "% cases",
    ROUND((SUM(Condition.deaths) * 100.0 / Country.popData2020), 2) AS "% deaths"
FROM
    Condition
JOIN
    Country ON Condition.geoId = Country.geoId
GROUP BY
    Country.countriesAndTerritories;
```

EX18: A descending list of the the top 10 countries by name, by percentage (to 2DP) total deaths out of total cases in that country.

```
SELECT
    Country.countriesAndTerritories AS Country,
    ROUND((SUM(Condition.deaths) * 100.0 / SUM(Condition.cases)), 2) AS "% deaths of country cases"
FROM
    Condition
JOIN
    Country ON Condition.geoId = Country.geoId
GROUP BY
    Country.countriesAndTerritories
ORDER BY
    "% deaths of country cases" DESC
LIMIT
    10;
```

EX19: The date against a cumulative running total of the number of deaths by day and cases by day for the united kingdom.

```
SELECT Date.dateRep AS "date",
       SUM(Condition.deaths) OVER win1 AS "cumulative UK deaths",
       SUM(Condition.cases) OVER win1 AS "cumulative UK cases"
FROM Date
JOIN Condition ON Date.dateRep = Condition.dateRep
WHERE Condition.geoId = 'UK'
WINDOW win1 AS (ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
ORDER BY Date.year, Date.month, Date.day ASC;
```