COMP1204: Data Management
Coursework Two: Coronavirus Data Analysis

Shaho Zagrosi
33381836

May 11, 2023

Contents

1 The Relational Model

1.1 EX1

The relation of the coronavirus dataset being studied in this report (`dataset.csv`) is as follows:

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

Each attribute and their respective relations are presented in the table below.

Relation	Relation Type
dateRep → day	One → Many
dateRep → month	One → Many
dateRep → year	One → Many
dateRep → cases	Many → Many
dateRep → deaths	Many → Many
dateRep → countriesAndTerritories	Many → Many
dateRep → geoId	Many → Many
dateRep → countryterritoryCode	Many → Many
dateRep → popData2020	Many → Many
dateRep → continentExp	Many → Many
day → month	Many → Many
day → year	Many → Many
day → cases	Many → Many
day → deaths	Many → Many
day → countriesAndTerritories	Many → Many
day → geoId	Many → Many
day → countryterritoryCode	Many → Many
day → popData2020	Many → Many
day → continentExp	One → Many
month → year	Many → Many
month → cases	Many → Many
month → deaths	Many → Many
month → countriesAndTerritories	Many → Many
month → geoId	Many → Many
month → countryterritoryCode	Many → Many
month → popData2020	Many → Many
month → continentExp	One → Many
year → cases	Many → Many
year → deaths	Many → Many
year → countriesAndTerritories	Many → Many
year → geoId	Many → Many
year → countryterritoryCode	Many → Many
year → popData2020	Many → Many
year → continentExp	One → Many
cases → deaths	Many → Many
cases → countryterritoryCode	Many → Many
cases → geoId	Many → Many
cases → popData2020	Many → Many
cases → continentExp	One → Many
deaths → countriesAndTerritories	Many → Many
deaths → geoId	Many → Many
deaths → countryterritoryCode	Many → Many

Table 1 continued from previous page

Relation	Relation Type
deaths → popData2020	Many → Many
deaths → continentExp	One → Many
countriesAndTerritories → geoId	One → One
countriesAndTerritories → countryterritoryCode	One → One
countriesAndTerritories → continentExp	One → Many
geoId → countryterritoryCode	One → One
geoId → popData2020	One → One
geoId → continentExp	One → Many
countryterritoryCode → popData2020	One → One
countryterritoryCode → continentExp	One → Many
popData2020 → continentExp	One → Many

SQLite data types in this dataset are presented in the table below.

Attribute	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

1.2.1 Functional Dependencies

The following functional dependencies are identified within the dataset:

FD1. Attribute *dateRep*, Attribute *countriesAndTerritories* → Attribute *cases* (The combination of Attributes *dateRep* and *countriesAndTerritories* functionally determines Attribute *cases*)

FD2. Attribute *dateRep*, Attribute *countriesAndTerritories* → Attribute *deaths* (The combination of Attributes *dateRep* and *countriesAndTerritories* functionally determines Attribute *deaths*)

FD3. Attribute *dateRep*, Attribute *countryterritoryCode* → Attribute *cases* (The combination of Attributes *dateRep* and *countryterritoryCode* functionally determines Attribute *cases*)

FD4. Attribute *dateRep*, Attribute *countryterritoryCode* → Attribute *deaths* (The combination of Attributes *dateRep* and *countryterritoryCode* functionally determines Attribute *deaths*)

FD5. Attribute *date*, Attribute *month*, Attribute *year* → Attribute *dateRep* (The combination of Attributes *date*, *month*, and *year* functionally determines Attribute *dateRep*)

FD6. Attribute *dateRep*, Attribute *geoId* → Attribute *cases* (The combination of Attributes *dateRep* and *geoId* functionally determines Attribute *cases*)

FD7. Attribute *dateRep*, Attribute *countryterritoryCode* → Attribute *deaths* (The combination of Attributes *dateRep* and *countryterritoryCode* functionally determines Attribute *deaths*)

FD8. Attribute *dateRep* → Attribute *day* (The combination of Attributes *dateRep* functionally determines Attribute *day*)

FD9. Attribute *dateRep* → Attribute *month* (The combination of Attributes *dateRep* functionally determines Attribute *month*)

FD10. Attribute *dateRep* → Attribute *year* (The combination of Attributes *dateRep* functionally determines Attribute *year*)

FD11. Attribute *countriesAndTerritories* → Attribute *continentExp* (Attribute *countriesAndTerritories* functionally determines Attribute *continentExp*)

FD12. Attribute *countriesAndTerritories* → Attribute *popData2020* (Attribute *countriesAndTerritories* functionally determines Attribute *popData2020*)

FD13. Attribute *countryterritoryCode* → Attribute *continentExp* (Attribute *countryterritoryCode* functionally determines Attribute *continentExp*)

FD14. Attribute *countryTerritoryCode* → Attribute *geoId* (The combination of Attributes *countryTerritoryCode* functionally determines Attribute *geoId*)

FD15. Attribute *countryTerritoryCode* → Attribute *popData2020* (The combination of Attributes *countryTerritoryCode* functionally determines Attribute *popData2020*)

FD16. Attribute *geoId* → Attribute *continentExp* (Attribute *geoId* functionally determines Attribute *continentExp*)

FD17. Attribute *geoId* → Attribute *popData2020* (The combination of Attributes *geoId* functionally determines Attribute *popData2020*)

1.2.2 Assumptions

Below is a list of assumptions about the dataset and its functional dependencies:

A1. Each attribute in the dataset has a defined domain, all values adhering to the domain constraints.

A2. Missing values are explicitly marked as NULL.

A3. Each tuple in the dataset is unique.

A4. Attributes cannot be further decomposed into smaller meaningful attributes.

1.3 EX3

The candidate keys identified in this dataset are presented in the table below.

Candidate Keys
dateRep, geoId
dateRep, countryterritoryCode
dateRep, countriesAndTerritories
day, month, year, geoId
day, month, year, countryterritoryCode
day, month, year, countriesAndTerritories

1.4 EX4

A suitable primary key for this table would be the combination of `dateRep` and `countryterritoryCode` as it uniquely identifies each record by combining the date of the report (`dateRep`) with the standardized country territory code (`countryterritoryCode`). This ensures that each row in the table corresponds to a specific report for a particular country on a given date, providing a unique identifier for each entry.

Primary Key
dateRep, countryterritoryCode

2 Normalisation

2.1 EX5

2.1.1 Partial Dependencies

D1. dateRep → day, month, year

D2. countryterritoryCode → countriesAndTerritories, geoId, popData2020, continentExp

Thus forming the final relation; `dateRep`, `countryterritoryCode` → cases, deaths.

2.2 EX6

The surrogate keys `intDate` and `intCountry` serve as unique identifiers for their respective entities. This decomposition process has successfully achieved 2nd Normal Form (2NF), as every non-key attribute is now fully dependent on the primary key.

Primary Key	Surrogate Key	Key Type
dateRep	intDate	INTEGER
countryterritoryCode	intCountry	INTEGER

2.3 EX7

In the newly decomposed relations, there are no transitive dependencies; each non-key attribute is directly dependent on the primary key of its respective relation.

2.4 EX8

T1. intDate → dateRep, day, month, year

T2. intCountry → countryterritoryCode, countriesAndTerritories, geoId, popData2020, continentExp

T3. intDate, intCountry → cases, deaths

There are no transitive dependencies in the new relations, indicating that the decomposed relations are in 3rd Normal Form (3NF).

2.5 EX9

For the dates relation; the primary key, `intDate`, is the only determinant. All other attributes are functionally dependent on `intDate`, as `intDate` is a super-key, the Dates relation is in BCNF. In the countries relation; `intCountry` is the only determinant, all other attributes are functionally dependent on `intCountry`. As `intCountry` is a super-key, the countries relation is in BCNF. As for the final relation, the composite primary key (`intDate`, `intCountry`) is the only determinant, and both cases and deaths are functionally dependent on this composite key. As (`intDate`, `intCountry`) is a super-key, the final relation is in BCNF. As all three relations satisfy the BCNF conditions, it is possible to conclude that the decomposed relations are in Boyce-Codd Normal Form.

3 Modelling

3.1 EX10

The raw dataset from `dataset.csv` was imported into an SQLite database into a single table called '`dataset`' within '`coronavirus.db`', exporting the table structure and data as SQLite statements into '`dataset.sql`'. This can later be used to recreate and populate the 'dataset' table in a fresh database.

```
sqlite3
.mode csv
.import dataset.csv dataset
.output dataset.sql
.dump
sqlite3 coronavirus.db
sqlite3 coronavirus.db < dataset.sql
```

3.2 EX11

Appropriate data types, indexes, and foreign key constraints were included. Data from `dataset` were filtered into `filteredDataset` to ensure only entries without `null` fields were selected.

```
CREATE TABLE IF NOT EXISTS CountryInfo (
    intCountry INTEGER PRIMARY KEY,
    countriesAndTerritories TEXT NOT NULL UNIQUE,
    geoId TEXT UNIQUE,
    countryterritoryCode TEXT UNIQUE,
    popData2020 INTEGER NOT NULL,
    continentExp TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS DateInfo (
    intDate INTEGER PRIMARY KEY,
    dateRep TEXT NOT NULL UNIQUE,
    day INTEGER NOT NULL,
    month INTEGER NOT NULL,
    year INTEGER NOT NULL
);

CREATE TABLE IF NOT EXISTS CaseInfo (
    intDate INTEGER,
    intCountry INTEGER,
    cases INTEGER,
    deaths INTEGER,
    PRIMARY KEY (intDate, intCountry),
    FOREIGN KEY (intDate) REFERENCES DateInfo(intDate)
        ON DELETE CASCADE
        ON UPDATE NO ACTION,
    FOREIGN KEY (intCountry) REFERENCES CountryInfo(intCountry)
        ON DELETE CASCADE
        ON UPDATE NO ACTION
);
```

The following commands were executed in terminal:

```
sqlite3 coronavirus.db < ex11.sql
.output dataset2.sql
.dump
```

Indexes were created on the surrogate keys `intDate` and `intCountry`. Foreign key constraints were also defined for tables referencing these surrogate keys.

3.3 EX12

The comments in the code below explain respective processes in populating each of the tables.

```
-- Populate the DateInfo table
INSERT OR IGNORE INTO CalendarInfo (dateRep, day, month, year)
SELECT
    filteredDataset.dateRep,
    filteredDataset.day,
    filteredDataset.month,
    filteredDataset.year
FROM (
    SELECT * FROM dataset
    WHERE cases IS NOT NULL AND deaths IS NOT NULL AND cases >= 0 AND deaths >= 0
) AS filteredDataset;

-- Populate the Country table
INSERT OR IGNORE INTO CountryInfo (countriesAndTerritories, geoId, countryterritoryCode,
↪  popData2020, continentExp)
SELECT
    filteredDataset.countriesAndTerritories,
    filteredDataset.geoId,
    filteredDataset.countryterritoryCode,
    filteredDataset.popData2020,
    filteredDataset.continentExp
FROM (
    SELECT * FROM dataset
    WHERE cases IS NOT NULL AND deaths IS NOT NULL AND cases >= 0 AND deaths >= 0
) AS filteredDataset;

-- Populate the CaseInfo table
INSERT INTO CaseInfo (intDate, intCountry, cases, deaths)
SELECT
    CalendarInfo.intDate AS intDate,
    CountryInfo.intCountry AS intCountry,
    filteredDataset.cases,
    filteredDataset.deaths
FROM (
    SELECT * FROM dataset
    WHERE cases IS NOT NULL AND deaths IS NOT NULL AND cases >= 0 AND deaths >= 0
) AS filteredDataset
INNER JOIN CalendarInfo ON CalendarInfo.dateRep = filteredDataset.dateRep
INNER JOIN CountryInfo ON CountryInfo.countriesAndTerritories =
↪  filteredDataset.countriesAndTerritories;
```

3.4 EX13

The following commands were executed, resulting in a fully populated and normalised SQLite database.

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

4 Querying

4.1 EX14

Accumulate cases and deaths in CaseInfo and cast the result as an `INTEGER`.

```
SELECT SUM(cases) AS "sum cases", SUM(deaths) as "sum deaths" FROM CaseInfo;
```

4.2 EX15

Ordering by the date variables (`year`, `month`, `day`), inner joining `intDate` and `intCountry` from their respective tables, filtering only entries with the `geoId` of `UK`.

```
SELECT
    CalendarInfo.dateRep,
    CaseInfo.cases
FROM CaseInfo
INNER JOIN CalendarInfo ON CalendarInfo.intDate = CaseInfo.intDate
INNER JOIN CountryInfo ON CountryInfo.intCountry = CaseInfo.intCountry
WHERE CountryInfo.geoId = 'UK'
ORDER BY year, month, day;
```

4.3 EX16

Ordering by the country variable first (`intCountry`) and the date variable second (`intDate`), inner joining `intDate` and `intCountry`, selecting `countriesAndTerritories`, `dateRep`, `cases`, and `deaths`, to show the number of cases and deaths by date and country.

```
SELECT
    CountryInfo.countriesAndTerritories,
    CalendarInfo.dateRep,
    CaseInfo.cases,
    CaseInfo.deaths
FROM CaseInfo
INNER JOIN CalendarInfo ON CalendarInfo.intDate = CaseInfo.intDate
INNER JOIN CountryInfo ON CountryInfo.intCountry = CaseInfo.intCountry
ORDER BY CaseInfo.intCountry ASC, CaseInfo.intDate ASC;
```

4.4 EX17

Ordering by the country variable first (`countriesAndTerritories`), taking a sum of the cases and deaths, dividing each dataset by the country's respective population variable (`popData2020`). The percentage value is converted to a floating point value in the process due to dividing by `100.0`.

```
SELECT
    CountryInfo.countriesAndTerritories,
    (SUM(CaseInfo.cases) * 100.0 / CountryInfo.popData2020) AS cases_percentage,
    (SUM(CaseInfo.deaths) * 100.0 / CountryInfo.popData2020) AS deaths_percentage
```

```
FROM CaseInfo
INNER JOIN CountryInfo ON CountryInfo.intCountry = CaseInfo.intCountry
GROUP BY CountryInfo.countriesAndTerritories, CountryInfo.popData2020
ORDER BY CountryInfo.countriesAndTerritories;
```

4.5 EX18

Ordering by the whole dataset's death rate (`death_rate`) in descending order and limiting to the top `10` results, a table is produced with the countries and their respective death rates.

```
SELECT
    CountryInfo.countriesAndTerritories,
    (SUM(CaseInfo.deaths) * 100.0 / SUM(CaseInfo.cases)) AS death_rate
FROM CaseInfo
INNER JOIN CountryInfo ON CountryInfo.intCountry = CaseInfo.intCountry
GROUP BY CountryInfo.countriesAndTerritories
ORDER BY death_rate DESC
LIMIT 10;
```

4.6 EX19

Using Window Functions in SQLite to calculate the cumulative totals, ordered by date.

```
WITH UK_Data AS (
    SELECT
        CalendarInfo.dateRep,
        CaseInfo.deaths,
        CaseInfo.cases
    FROM CaseInfo
    INNER JOIN CountryInfo ON CaseInfo.intCountry = CountryInfo.intCountry
    INNER JOIN CalendarInfo ON CaseInfo.intDate = CalendarInfo.intDate
    WHERE CountryInfo.geoId = 'UK'
)
SELECT
    dateRep,
    SUM(deaths) OVER (ORDER BY dateRep) AS cumulative_deaths,
    SUM(cases) OVER (ORDER BY dateRep) AS cumulative_cases
FROM UK_Data;
```