COMP1204: Data Management
Coursework two

Mohammad Rayes
35316004

May 2024

1 The Relational Model

1.1 EX1

Attribute	SQLite Data Type
dateRep	TEXT
day	INTEGER
month	INTEGER
year	INTEGER
cases	INTEGER
deaths	INTEGER
countriesAndTerritories	TEXT
geoId	TEXT
countryterritoryCode	TEXT
popData2020	INTEGER
continentExp	TEXT

Table 1: Attributes and SQLite Data Types in the Dataset

1.2 EX2

1.2.1 Minimal Set of Functional Dependencies

The minimal functional dependencies for the dataset, assuming unique identification of records by combinations of date and geographical identifiers, are as follows:

1. dateRep $\rightarrow$ day
2. dateRep $\rightarrow$ month
3. dateRep $\rightarrow$ year
4. geoId $\rightarrow$ countriesAndTerritories
5. geoId $\rightarrow$ countryterritoryCode
6. geoId $\rightarrow$ continentExp
7. geoId $\rightarrow$ popData2020
8. (dateRep, geoId) $\rightarrow$ cases
9. (dateRep, geoId) $\rightarrow$ deaths

Assumptions and Justifications:

- *Uniqueness of geoId:* Assumed that each `geoId` uniquely identifies a set of static attributes about a location (country, population, continent), which is common in datasets containing geographical information.
- *Stability of dateRep:* The representation of a date as `dateRep` is assumed to uniquely determine its day, month, and year components, typical for date formatting in data systems.
- *Combination keys for dynamic data:* The combination of `dateRep` and `geoId` is assumed necessary and sufficient to determine the counts of cases and deaths, reflecting the nature of reporting in epidemiological data where each entry represents a unique event/day/location.
- *Treatment of nulls and blanks:* It is assumed that all entries are complete and any nulls would be addressed prior to applying these dependencies, as incompleteness would violate the principles of functional dependency.

1.3 EX3

Based on the provided functional dependencies, several combinations of attributes are considered to explore their eligibility as candidate keys. Each candidate key must be minimal and able to uniquely identify every tuple in the dataset.

1.3.1 Explored Combinations

1. (dateRep, geoId)
2. (dateRep, countryterritoryCode)
3. (day, month, year, geoId)
4. (geoId, day, month, year, cases, deaths)

1.4 EX4

After evaluating potential candidate keys, (**dateRep**, **geoId**) is chosen as the primary key for the dataset for the following reasons:

- **Simplicity and Minimality**
- **Comprehensive Coverage**
- **Practicality in Usage**

This key not only fulfills the requirements of a candidate key but also aligns with the practical needs of database management and data integrity maintenance.

2 Normalisation

2.1 EX5

Identified partial-key dependencies from established functional dependencies illustrate that specific attributes rely solely on parts of a potential composite key. This necessitates additional relations for normalization adherence:

- dateRep → day, month, year: Suggests a separate relation for date components derived directly from the reporting date.
- geoId → countryterritoryCode, countriesAndTerritories,continentExp, popData2020: Indicates the need for a dedicated geographical information table.

Resulting relations:

1. **Date Information Table**: Includes dateRep, day, month, year.
2. **Geographic Details Table**: Contains geoId, countryterritoryCode, countriesAndTerritories, continentExp, popData2020.

This decomposition aligns with the principles of the Second Normal Form (2NF).

2.2 EX6

To achieve 2NF, we decomposed the original dataset into three tables, eliminating partial-key dependencies and ensuring full dependency on primary keys:

- **Date Information**: Holds date-related attributes, isolated to prevent partial-key dependencies. Fields include dateRep (TEXT), day (INTEGER), month (INTEGER), year (INTEGER). Primary Key: dateRep.
- **Geographic Information**: Consolidates geographic details, ensuring clean structure. Fields include geoId (TEXT), countryterritoryCode (TEXT), countriesAndTerritories (TEXT),continentExp (TEXT), popData2020 (INTEGER). Primary Key: geoId.
- **Case and Death Records**: Captures cases and deaths linked to dates and locations. Fields include dateRep (TEXT), geoId (TEXT), cases (INTEGER), deaths (INTEGER). Primary Key: (dateRep, geoId).

These tables ensure data integrity by reducing redundancy.

2.3 EX7

No transitive dependencies are present in the new relations, complying with 2NF and 3NF principles.

2.4 EX8

Our relations inherently satisfy the requirements of 3NF as they are free from transitive dependencies and each attribute is fully functionally dependent on the primary key(s).

2.5 EX9

The relations conform to Boyce-Codd Normal Form (BCNF), ensuring no anomalies or redundancy:

- **Date Information Table**: All attributes are functionally dependent solely on dateRep, the primary key.
- **Geographic Information Table**: Attributes are functionally dependent solely on geoId, the primary key.
- **Case and Death Records Table**: cases and deaths depend only on the composite key (dateRep, geoId).

Thus, the relations are in BCNF, ensuring robust data integrity.

3 Modelling

3.1 EX10

To import a raw dataset into SQLite and create a dump file, follow these streamlined steps:

1. **Start SQLite**: Launch SQLite with your database.

```
sqlite3 coronavirus.db
```

2. **Create the Table**: Set up the dataset schema.

```
CREATE TABLE dataset (
    dateRep TEXT,
    day INTEGER,
    month INTEGER,
    year INTEGER,
    cases INTEGER,
    deaths INTEGER,
    countriesAndTerritories TEXT,
    geoId TEXT,
    countryterritoryCode TEXT,
    popData2020 INTEGER,
    continentExp TEXT);
```

3. **Set CSV Mode**: Configure SQLite for CSV data.

   ```
   .mode csv
   ```

4. **Import the Dataset**: Ensure CSV columns align with your table schema.

   ```
   .import dataset.csv dataset
   ```

5. **Prepare for Dumping**: Specify the output file for the SQL dump.

   ```
   .output dataset.sql
   ```

6. **Dump the Dataset**: Export the schema and data to an SQL file.

   ```
   .dump
   ```

3.2 EX11

```sql
-- Table: Date Information
CREATE TABLE IF NOT EXISTS Date_Information (
    dateRep TEXT PRIMARY KEY NOT NULL,
    day INTEGER NOT NULL,
    month INTEGER NOT NULL,
    year INTEGER NOT NULL
);

-- Table: Geographic Information
CREATE TABLE IF NOT EXISTS Geographic_Information (
    geoId TEXT PRIMARY KEY NOT NULL,
    countryterritoryCode TEXT NOT NULL,
    countriesAndTerritories TEXT NOT NULL,
    continentExp TEXT NOT NULL,
    popData2020 INTEGER
);

-- Table: Case and Death Records
CREATE TABLE IF NOT EXISTS Case_Death_Records (
    dateRep TEXT NOT NULL,
    geoId TEXT NOT NULL,
    cases INTEGER,
    deaths INTEGER,
    PRIMARY KEY (dateRep, geoId)
);
```

3.3 EX12

```sql
-- Insert data into Date Information Table
INSERT OR IGNORE INTO Date_Information (dateRep, day, month, year)
SELECT dateRep, day, month, year
FROM dataset;

-- Insert data into Geographic Information Table
INSERT OR IGNORE INTO Geographic_Information (geoId,countriesAndTerritories
↪  ,countryterritoryCode, continentExp, popData2020)
```

```
SELECT geoId,countriesAndTerritories ,countryterritoryCode, continentExp,
↪  popData2020
FROM dataset;

-- Insert data into Case and Death Records Table
INSERT OR IGNORE INTO Case_Death_Records (dateRep, geoId, cases, deaths)
SELECT dateRep, geoId, cases, deaths
FROM dataset;
```

3.4 EX13

To verify the functionality of the SQL scripts, the following commands were executed:

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

All of these tests were successful and executed without errors.

4 Querying

4.1 EX14

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

4.2 EX15

```
    SELECT Date_Information.dateRep AS Date_Reported,
       Case_Death_Records.cases AS Number_of_Cases
FROM Case_Death_Records
INNER JOIN Date_Information ON Case_Death_Records.dateRep =
↪  Date_Information.dateRep
INNER JOIN Geographic_Information ON Case_Death_Records.geoId =
↪  Geographic_Information.geoId
WHERE Geographic_Information.countryterritoryCode = 'GBR'
ORDER BY Date_Information.year, Date_Information.month, Date_Information.day ASC;
```

4.3 EX16

```
  SELECT
    g.countriesAndTerritories AS CountryName,
    d.dateRep AS Date,
    cd.cases AS Cases,
    cd.deaths AS Deaths
FROM
    Case_Death_Records cd
JOIN
    Geographic_Information g ON cd.geoId = g.geoId
```

```
JOIN
    Date_Information d ON cd.dateRep = d.dateRep
ORDER BY
    d.year ASC, d.month ASC, d.day ASC, g.countriesAndTerritories ASC;
```

4.4 EX17

```
   SELECT
    g.countriesAndTerritories AS CountryName,
    ROUND(SUM(cd.cases) * 100.0 / g.popData2020, 2) AS Cases_Percentage,
    ROUND(SUM(cd.deaths) * 100.0 / g.popData2020, 2) AS Deaths_Percentage
FROM
    Case_Death_Records cd
JOIN
    Geographic_Information g ON cd.geoId = g.geoId
GROUP BY
    g.countriesAndTerritories
```

4.5 EX18

```
    SELECT
    g.countriesAndTerritories AS "Country Name",
    ROUND((SUM(cd.deaths) * 100.0 / SUM(cd.cases)), 2) AS "Percentage Deaths of
    ↪  Country Cases (%)"
FROM
    Case_Death_Records cd
JOIN
    Geographic_Information g ON cd.geoId = g.geoId
GROUP BY
    g.countriesAndTerritories
HAVING
    SUM(cd.cases) > 0  -- To prevent division by zero
ORDER BY
    "Percentage Deaths of Country Cases (%)" DESC
LIMIT 10;
```

4.6 EX19

```
SELECT
    d.dateRep AS "Date",
    SUM(cd.cases) OVER (ORDER BY d.year, d.month, d.day) AS "Cumulative UK
    ↪  Cases",
    SUM(cd.deaths) OVER (ORDER BY d.year, d.month, d.day) AS "Cumulative UK
    ↪  Deaths"
FROM
    Case_Death_Records cd
JOIN
    Geographic_Information g ON cd.geoId = g.geoId
JOIN
    Date_Information d ON cd.dateRep = d.dateRep
WHERE
    g.countriesAndTerritories = 'United_Kingdom'
ORDER BY
    d.year, d.month, d.day;
```