COMP1204: Data Management
Coursework Two: Databases

Patrick Jfremov-Kustov
pjk1g22
33461104

1 The Relational Model

1.1 EX1

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

Column Name	**Data 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

- dateRep → day
- dateRep → month
- dateRep → year
- day, month, year → dateRep
- countriesAndTerritories → geoId
- geoId → countriesAndTerritories
- countriesAndTerritories → countryterritoryCode
- countryterritoryCode → countriesAndTerritories
- geoId → continentExp
- geoId → popData2020
- dateRep, countriesAndTerritories → cases
- dateRep, countriesAndTerritories → deaths

  Assumptions:

  Values not available for cases and deaths and null values are shown as zero, instead of leaving them blank. Nonetheless, this must be checked to ensure consistency. This could be influenced by future data such as reporting methods changing or virus variations happening.

1.3 EX3

Potential candidate keys:

- dateRep, countriesAndTerritories
- dateRep, geoId
- dateRep, countryterritoryCode
- day, month, year, countriesAndTerritories
- day, month, year, geoId
- day, month, year, countryterritoryCode

1.4 EX4

The composite key of dateRep and geoId, because out of countriesAndTerritories, geoId and countryterritoryCode, geoId represents the unique identifier in the shortest way out of all attributes.

2 Normalisation

2.1 EX5

2.1.1 Partial key dependencies:

- day, month, and year dependent on dateRep
- countriesAndTerritories, continentExp, countryterritoryCode and popData2020 dependent on geoId

2.1.2 Additional relations:

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

2.2 EX6

2.2.1 Date relation

Date (dateRep, day, month, year)

Field Name	Data Type
dateRep	TEXT
day	INTEGER
month	INTEGER
year	INTEGER

2.2.2 Cases relation

Cases (dateRep, geoID, cases, deaths)

Field Name	Data Type
dateRep	TEXT
geoId	TEXT
cases	INTEGER
deaths	INTEGER

2.2.3 Country relation

Country (geoId, countriesAndTerritories, continentExp, countryterritoryCode, popData2020)

Field Name	Data Type
geoId	TEXT
countriesAndTerritories	TEXT
continentExp	TEXT
countryterritoryCode	TEXT
popData2020	INTEGER

2.2.4 Explanation:

The original relation had partial dependencies listed above, where the attributes were dependent on only part of the primary key. I decomposed the relation into three new relations: Date and Country.

Then, I identified the new relations with their composite key, fields, and types. This allowed us to achieve 2NF by removing both redundancy and inconsistency from the original relation (dataset.csv).

2.3 EX7

There is one transitive dependency in the Country table whereby the popData2020 is dependent on any of the other fields in the relation because you won't find a unique record using that field alone.

2.4 EX8

In order to make the data set from 2NF to 3NF, we must remove the transitive dependency in the Country table where the difference is shown below.

2.4.1 Original Country relation

Country (geoId, countriesAndTerritories, continentExp, countryterritoryCode, popData2020)

Field Name	Data Type
geoId	TEXT
countriesAndTerritories	TEXT
continentExp	TEXT
countryterritoryCode	TEXT
popData2020	INTEGER

2.4.2 Amended Country relation

Country (geoId, countriesAndTerritories, continentExp, countryterritoryCode, popData2020)

Field Name	Data Type
geoId	TEXT
countriesAndTerritories	TEXT
continentExp	TEXT
countryterritoryCode	TEXT
popData2020	INTEGER

2.5 EX9

This is already in Boyce-Codd Normal Form because there is no longer any redundant data in any of the relations. The reason for this is because every field in the relations can be uniquely identified by other fields.

3 Modelling

3.1 EX10

I downloaded the Precompiled Binaries for Windows from the sqllite website (https://www.sqlite.org/download.html). Then I opened Command Prompt on Windows and located to the file, followed by entering "sqlite3" to access the application. Since it doesn't exist yet, I used ".open coronavirus.db" to create a new databse file. I used "mode .csv" so that sqlite will recongise the imported CSV file. To import it, I used ".import dataset.csv dataset". This creates SQL statements (including CREATE and INSERT) that is used to later create the database. Then I used ".output dataset.sql" to output the SQL statements required to create the table and then ".dump" which then fills the file with these statements. Lastly I ran "sqlite3 coronavirus.db < dataset.sql" on the empty coronavirus.db that populates the dataset table.

3.2 EX11

I created a new file by using ".open ex11.sql" to write the set of SQL statements.

Then I wrote this code:

```
CREATE TABLE date (
    dateRep TEXT NOT NULL,
    day INTEGER,
    month INTEGER,
    year INTEGER,
    PRIMARY KEY (dateRep)
);

CREATE TABLE cases (
    dateRep TEXT NOT NULL,
    geoId TEXT NOT NULL,
    cases INTEGER,
    deaths INTEGER,
    PRIMARY KEY (dateRep, geoId),

    FOREIGN KEY (dateRep)
       REFERENCES date (dateRep)
);

CREATE TABLE country (
    geoId TEXT NOT NULL,
    countriesAndTerritories TEXT NOT NULL,
    continentExp TEXT NOT NULL,
    countryterritoryCode TEXT NOT NULL,
    popData2020 INTEGER,
    PRIMARY KEY (geoId, countriesAndTerritories, continentExp, countryterritoryCode),

    FOREIGN KEY (geoId)
       REFERENCES country (geoId)
);
```

I then used .open coronavirus.db" followed by ".output dataset2.sql" followed by ".dump" to insert the values and create the tables. Lastly I ran "sqlite3 coronavirus.db < dataset2.sql" on the coronavirus.db that populates the dataset table alongside having empty tables.

3.3 EX12

I created a new file by using ".open ex12.sql" to write the set of SQL statements.

Then I wrote this code:

```
INSERT OR IGNORE INTO date (dateRep, day, month, year)
SELECT DISTINCT dateRep, day, month, year FROM dataset;

INSERT OR IGNORE INTO cases (dateRep, geoId, cases, deaths)
SELECT DISTINCT dateRep, geoId, cases, deaths from dataset;

INSERT OR IGNORE INTO country (geoId, countriesAndTerritories, continentExp, countryterritoryCode, po
SELECT DISTINCT geoId, countriesAndTerritories, continentExp, countryterritoryCode, popData2020 from
```

I then used .open coronavirus.db" followed by ".output dataset3.sql" followed by ".dump" to insert the values and create the tables. Lastly I ran "sqlite3 coronavirus.db < dataset3.sql" on the coronavirus.db that populates the dataset alongside the different, separate tables.

3.4 EX13

I deleted the coronavirus.db and ran the three commands to populate the database successfully.

4 Querying

4.1 EX14

I created a new file by using ".open ex14.sql" to write the SQL query.

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

Then I ran "sqlite3 coronavirus.db < ex14.sql" that returned the required results.

4.2 EX15

I created a new file by using ".open ex15.sql" to write the SQL query.

```
SELECT cases.dateRep, cases.cases
FROM cases
INNER JOIN country ON cases.geoId = country.geoId
WHERE country.geoId = 'UK'
ORDER BY DATE(cases.dateRep) ASC;
```

Then I ran "sqlite3 coronavirus.db < ex15.sql" that returned the required results.

4.3 EX16

I created a new file by using ".open ex16.sql" to write the SQL query.

```
SELECT cases.geoId, cases.dateRep, cases.cases, cases.deaths
FROM cases
INNER JOIN country on cases.geoId = country.geoId
```

Then I ran "sqlite3 coronavirus.db < ex16.sql" that returned the required results, organised by date first.

4.4 EX17

I created a new file by using ".open ex17.sql" to write the SQL query.

```
SELECT country.countriesAndTerritories,
       ROUND(ROUND(SUM(cases.cases) * 100, 2) / country.popData2020, 2),
       ROUND(ROUND(SUM(cases.deaths) * 100, 2) / country.popData2020, 2)
FROM cases
INNER JOIN country on cases.geoId = country.geoId
GROUP BY country.geoId
```

Then I ran ”sqlite3 coronavirus.db < ex17.sql” that returned the required results.

4.5 EX18

I created a new file by using ”.open ex18.sql” to write the SQL query.

```
SELECT
    country.countriesAndTerritories,
    ROUND((ROUND(SUM(cases.deaths), 2) / SUM(cases.cases)) * 100, 2) AS percentage_of_death
FROM cases
INNER JOIN country ON cases.geoId = country.geoId
GROUP BY country.countriesAndTerritories
ORDER BY percentage_of_death DESC
LIMIT 10;
```

Then I ran ”sqlite3 coronavirus.db < ex18.sql” that returned the required results.

4.6 EX19

I created a new file by using ”.open ex19.sql” to write the SQL query.

```
SELECT cases.dateRep,
    SUM(cases.cases) OVER (ROWS UNBOUNDED PRECEDING),
SUM(cases.deaths) OVER (ROWS UNBOUNDED PRECEDING)
FROM cases
INNER JOIN date ON date.dateRep = cases.dateRep
INNER JOIN country ON country.geoId = cases.geoId
WHERE cases.geoId = ’UK’
ORDER BY year,month,day;
```

Then I ran ”sqlite3 coronavirus.db < ex19.sql” that returned the required results.