1204 Course Work 2

Daniel Elmar (dje1g20, ID: 31820638)

May 13, 2021

0.1 The Relational Model

0.1.1 EX1

dataset(
dateRep TEXT,
day INTEGER,
month INTEGER,
year INTEGER,
cases INTEGER,
deaths INTEGER,
countriesAndTerritories TEXT,
geoId TEXT,
countryterritoryCode TEXT,
popData2019 INTEGER,
continentExp TEXT
)

0.1.2 EX2

dateRep → day
dateRep → month
dateRep → year
day, month, year → dateRep
geoId → countriesAndTerritories
countriesAndTerritories → geoId
countriesAndTerritories → countryterritoryCode
countriesAndTerritories → continentExp

I have assumed that population, although maybe be unique in the data, cant be functionally determined in the domain, as 2 countries could have the same population, or population change over time. This also goes for populating determining other attributes. If I did not make this assumption the FD

countryterritoryCode → popData2019
popData2019 → countryterritoryCode

Would also be in the minimal set.

I have also assumed that cases nor deaths can functionally determine, or be determined by, other attributes in the domain, as multiple entries could have the same number of deaths or cases in the future.

0.1.3 EX3

(dateRep,countriesAndTerritories)
(dateRep,geoId)
(dateRep,countryterritoryCode,continentExp)
(dateRep,popData2019,continentExp)

I have assumed that no more entries will be added with countryterritoryCode or popData2019 of null and a continentExp of either ’other’ or ’Oceania’ on the same day. This would break the candidate keys (dateRep,countryterritoryCode,continentExp) and (dateRep,popData2019,continentExp). Similar entries can be seen in rows with a geoId of ’JPG11668’,

0.1.4 EX4

(cases,deaths,dateRep,geoId)

In my opinion this is the most suitable super-key as uniquely identifies all rows in the data, and is unlikely to be broken as new data is added, for example multiple entities for a given country can be added on the same day, such entries are unlikely to have the same values of cases and deaths as previous entries for the same day.

0.2 Normalisation

0.2.1 EX5

Using candidate key (dateRep,geoId)

day, month and year are dependent on dateRep only. countriesAndTerritories, countryterritoryCode, popData2019 and continentExp are dependent on geoId only.

Additional relations as apart of decomposition.

Countries(geoId,countriesAndTerritories,countryterritoryCode,popData2019,continentExp)
Dates(dateRep,day,month,year)

0.2.2 EX6

I first executed the query

```
”CREATE TABLE Dates(
dateRep TEXT
constraint Dates_pk
primary key,
day INTEGER,
month INTEGER,
year INTEGER
);”
```

This created my new Dates table as described. I then added the data from the dataset to this new Table using.

”INSERT into Dates(dateRep, day, month, year) select distinct dateRep, day, month, year from dataset;”

Then I made the Countries Table using the query

```
”CREATE TABLE Countries(
geoId TEXT
```

constraint Countries_pk
primary key,
countriesAndTerritories TEXT,
countryterritoryCode TEXT,
popData2019 INTEGER,
continentExp TEXT
);”

This created my new Countries table as described. I then added the data from the dataset to this new Table using.

”INSERT into Countries(geoId, countriesAndTerritories, countryterritoryCode, popData2019,continentExp)
select distinct geoId, countriesAndTerritories, countryterritoryCode, popData2019,continentExp
from dataset;”

I then created a new dataset_tmp table to remove the redundant data from dataset, aswell as set the primary keys using the query.

”CREATE TABLE dataset_tmp(dateRep TEXT, cases INTEGER, deaths INTEGER, geoId TEXT, PRIMARY KEY(dateRep,geoId))”

I then populated this new table with the relevant data form the old dataset table using the query.

”INSERT into dataset_tmp(dateRep, cases, deaths, geoId) select distinct dateRep, cases, deaths, geoId from dataset;”

I then dropped the table dataset and renamed dataset_tmp to dataset using the queries.

”DROP TALE dataset;”

”ALTER TABLE dataset_tmp rename to dataset;”

This resulted in the dataBase relations of.

dataset(dateRep,cases,deaths,geoId)
Countries(geoId,countriesAndTerritories,countryterritoryCode,popData2019,continentExp)
Dates(dateRep,day,month,year)

0.2.3 EX7

For the Countires Table.

geoid → (countriesAndTerritories,countryterritoryCode,popData2019,continentExp)
countriesAndTerritories → (geoid,countryterritoryCode,popData2019,continentExp)
countryterritoryCode → popData2019 &&
popData2019 → countryterritoryCode

So the transitive dependencies for the Countries Table are:

countriesAndTerritories → popData2019
countriesAndTerritories → countryterritoryCode

geoid → countryterritoryCode
geoid → popData2019

There are no other transitive relations in my other Tables.

0.2.4 EX8

The table relations:

dataset(dateRep,cases,deaths,geoId)
Dates(dateRep,day,month,year)

Are unchanged as there are no non-prime attributes that are transversely dependent on the primary key. ie all non prime attributes are only dependent on candidate keys

However the relation:

Countries(geoId,countriesAndTerritories,countryterritoryCode,popData2019,continentExp)

Has now been decompossed into he following 2 relations:

Countries(geoId,countriesAndTerritories,countryterritoryCode,continentExp)
CountryPopulation(countriesAndTerritories,popData2019)

Now no non-prime attributes are transversely dependent on the primary key for all table relations.

I used the following queries to create all the new tables

```
CREATE TABLE CountryPopulation( countriesAndTerritories TEXT PRIMARY KEY, popData2019 INTEGER )
```

```
CREATE TABLE Counties_tmp( geoId TEXT PRIMARY KEY, countriesAndTerritories TEXT, countryterritoryCode TEXT, continentExp TEXT )
```

I then used the following queries to population theses tables with there respective data from the table Countries.

```
INSERT into CountryPopulation(geoId, popData2019) select distinct geoId, popData2019 from Countries;
```

```
INSERT into Countries_tmp(geoId,countriesAndTerritories,countryterritoryCode,continentExp) select distinct geoId,countriesAndTerritories,countryterritoryCode,continentExp from Countries;
```

I then used the following query to DROP the old Countries Table and rename the new new:

```
DROP TABLE Countries;
ALTER TABLE Countries_tmp RENAME TO Countries;
```

0.2.5 EX9

My database is in Boyce-Codd Normal Form, this is because every determinate of any attribute in a table is a super key for that table. ie every functional dependency X → Y, X is a super key of the table.

0.3 Modelling

0.3.1 EX10

1. First I opened sqlite3.exe
2. I then entered the comand ”.open coronavirus.db”
3. Next I entered ”.mode csv”
4. Then ”.import dataset.csv dataset”
5. Then ”.output dataset.sql”
6. Then ”.dump”
7. Then ”.exit”

0.3.2 EX11

This is the contents of ex11.sql

```
CREATE TABLE CountryPopulation(
countriesAndTerritories TEXT PRIMARY KEY,
popData2019 INTEGER
);

CREATE TABLE Countries(
geoId TEXT PRIMARY KEY,
countriesAndTerritories TEXT,
countryterritoryCode TEXT,
continentExp TEXT,
FOREIGN KEY(countriesAndTerritories) REFERENCES CountryPopulation(countriesAndTerritories)
);

CREATE TABLE Dates(
dateRep TEXT PRIMARY KEY,
day INTEGER,
month INTEGER,
year INTEGER
);
```

To run this on the database I executed the following steps.

1. First I opened sqlite3.exe
2. I then entered the comand ”.open coronavirus.db”

3. Next I entered ”.read sql11.sql”
4. Then ”.output dataset2.sql”
5. Then ”.dump”
6. Then ”.exit”

I used Foreign Keys in my Countries table, I set the (countriesAndTerritories) column to be a foreign key in for the (countriesAndTerritories) column in the CountryPopulation table. This is to ensure that every entry in the Countries table has a popData2019 assigned to it in the CountryPopulation table, even if it is just a null value.

I will also use multiple foreign keys in my datasets table but this is not included in this question.

I would have included an Index for the dateRep column in the dataset table but we weren't to modify it yet. I would have used the following query to do this.

```
CREATE UNIQUE INDEX dateRep_index ON dataset(dateRep);
```

0.3.3 EX12

This is the contents of ex12.sql

```
INSERT INTO CountryPopulation(
countriesAndTerritories,
popData2019)
SELECT distinct
countriesAndTerritories,
popData2019
FROM dataset;

INSERT INTO Countries(
geoId,
countriesAndTerritories,
countryterritoryCode,
continentExp)
SELECT distinct
geoId,
countriesAndTerritories,
countryterritoryCode,
continentExp
FROM dataset;

INSERT INTO Dates(
dateRep,
day,
month,
year)
SELECT distinct
dateRep,
day,
month,
year
FROM dataset;
```

To run this on the database I executed the following steps.

1. First I opened sqlite3.exe
2. I then entered the comand ”.open coronavirus.db”
3. Next I entered ”.read sql12.sql”
4. Then ”.output dataset3.sql”
5. Then ”.dump”
6. Then ”.exit”

0.3.4 EX13

To confirm that the previous files work correctly as intended on a fresh database I executed the following steps.

1. First I opened sqlite3.exe
2. I then entered the comand ”.open test.db”
3. Next I entered ”.read dataset.sql”
4. Then ”.read ex11.sql”
5. Then ”.read ex12.sql”
6. I then examined all the tables to ensure that they had been correctly created and the data populateds.
7. Then ”.exit”

I can confirm that I am able to execute dataset.sql, ex11.sql, ex12.sql on a fresh database and successfully populate it.

0.4 Querying

0.4.1 EX14

```
SELECT sum(cases) AS ”total cases”, sum(deaths) AS ”total deaths” from dataset;
```

Here I used the sum aggregate function to add up all the cases over the entire dataset, I also used the AS operator to rename the column in the result.

0.4.2 EX15

```
SELECT dataset.dateRep AS date,
sum(cases) AS ”number of cases”
FROM dataset INNER JOIN Dates ON dataset.dateRep=Dates.dateRep
WHERE geoId=’UK’ group by dataset.dateRep
order by Dates.year asc, Dates.month asc, Dates.day asc;
```

Here I used the sum aggregate function to add up all the cases in the UK. I used a INNER JOIN to connect the dataset table to the Dates table, this gave the query access to the day,month and year columns allowing for correct ordering.

I also used the AS operator to rename the columns in the result.

0.4.3 EX16

```
SELECT Countries.continentExp AS continent,
dataset.dateRep AS date,
sum(cases) AS ”number of cases”,
sum(deaths) AS ”number of deaths”
FROM dataset
INNER JOIN Dates ON dataset.dateRep=Dates.dateRep
INNER JOIN Countries on dataset.geoId = Countries.geoId
GROUP BY Countries.continentExp, dataset.dateRep
ORDER BY Countries.continentExp, Dates.year asc, Dates.month asc, Dates.day asc;
```

Here I used the sum aggregate function to add up all the cases and deaths for each continent. I used a INNER JOIN to connect the dataset table to the Dates table, this gave the query access to the day,month and year columns allowing for correct ordering. I also used INNER JOIN between Countries this gave the query access to continentExp which was used for grouping.

I also used the AS operator to rename the columns in the result.

0.4.4 EX17

```
SELECT Countries.countriesAndTerritories AS country,
cast(sum(cases) as real) / cast (CountryPopulation.popData2019 as real) AS ”% cases of population”,
cast(sum(deaths) as real) / cast (CountryPopulation.popData2019 as real) AS ”% deaths of
population”
FROM dataset
INNER JOIN Countries on dataset.geoId = Countries.geoId
INNER JOIN CountryPopulation on CountryPopulation.countriesAndTerritories = Countries.countries
AndTerritories
GROUP BY Countries.countriesAndTerritories;
```

Here I used the sum aggregate function to add up all the cases and deaths for each Country. I used a INNER JOIN to connect the dataset table to the Countries table. This gave the query access to countriesAndTerritories which was used for grouping and to retrieve the population for each Country, this was also achieved by a 2nd INNER JOIN on CountryPopulation to give the query access to the popData2019 column.

I also used cast(... as real) to ensure the mathematical operator ’/’ wouldn’t fail

I also used the AS operator to rename the columns in the result.

0.4.5 EX18

```
SELECT Countries.countriesAndTerritories AS ”country name”,
cast(sum(deaths) as real) / cast(sum(cases) as real) AS ”% deaths of country cases”
FROM dataset
INNER JOIN Countries on dataset.geoId = Countries.geoId
GROUP BY Countries.countriesAndTerritories
ORDER BY ”% deaths of country cases” desc
LIMIT 10;
```

Here I used the sum aggregate function to add up all the deaths and cases for each Country. I used a INNER JOIN to connect the dataset table to the Countries table. This gave the query access to countriesAndTerritories which was used for grouping and the ’country name’ column in the result.

I also used cast(... as real) to ensure the mathematical operator ’/’ wouldn’t fail

I also used LIMIT to control the number of rows that were present in the result, this was set to 10

I also used the AS operator to rename the columns in the result.

0.4.6 EX19

```
SELECT dataset.dateRep AS ”date”,

SUM(deaths) OVER (
ORDER BY Dates.year, Dates.month, Dates.day
ROWS BETWEEN
UNBOUNDED PRECEDING
AND CURRENT ROW
) AS ”cumulative UK deaths”,

SUM(cases) OVER (
ORDER BY Dates.year, Dates.month, Dates.day
ROWS BETWEEN
UNBOUNDED PRECEDING
AND CURRENT ROW
) AS ”cumulative UK cases”

FROM dataset
INNER JOIN Dates on dataset.dateRep = Dates.dateRep
Where dataset.geoId=”UK”
GROUP BY dataset.dateRep
ORDER BY Dates.year, Dates.month, Dates.day;
```

Here I used the sum aggregate function to add up all the deaths and cases for each date. I used a INNER JOIN to connect the dataset table to the Dates table, this gave the query access to the day,month and year columns allowing for correct ordering.

I also used cast(... as real) to ensure the mathematical operator '/' wouldn't fail

I also used LIMIT to control the number of rows that were present in the result, this was set to 10

I also used Window Functions to sum over only specific rows in the table for each row in the result. I implemented the function to sum over all PRECEDING rows to, and including the current one, effectively implementing a cumulatively growing column in the result.

I also used the AS operator to rename the columns in the result.