STAT 19000 Project 12 Examples

For questions (1a) through (1c) in this project, you don't need to learn any new skills in order to solve the problems. With that being said, the questions are written to *try* and show you instances where using SQL can make your life easier, without forcing you to do so.

The goal of question 2 is to get a glance at creating your own database, so the topic is not completely foreign to you. Specifically, we ask you to write sqlite create table statements for the `movies` and `reviews` datasets from https://rottentomatoes.com, identify appropriate sqlite types for each column, as well as identifying 1 primary key and 1 foreign key for our data.

After that, you use R and `RSQLite` to create your new database, `rt.db`.

Creating a table in sqlite

Creating a table is different depending on if you are using `postgresql`, `MySQL`, `sqlite`, or some other rdbms. In this project we will learn only how to do so in `sqlite`. `sqlite` only has 5 storage classes (which you can think of as types), so you only have to categorize each column into 1 of those 5 types.

The 5 types are: `NULL`, `TEXT`, `INTEGER`, `REAL`, and `BLOB`. You can find descriptions here.

Let's say we have the following csv:

id	name	age	cash	resume
1	John Smith	14	16.47	...
2	Anna Dot	27	1000.50	...

If you had to categorize each column into 1 of the 5 types which would they be? Well, the name column clearly contains text, so `TEXT` is appropriate. The age column is a number, so either `INTEGER` or `REAL`, but which one? Well, do we have decimal points? No. So, an `INTEGER` is more appropriate as `INTEGER`'s don't have decimals. The cash column, on the other hand, has decimals, so `REAL` would be more appropriate. When we see a "..." in a column, or some sort of gibberish, we can assume some sort of file or data lives in binary format. For binary data, `BLOB` is the appropriate type. This is really all there is to it!

In an rdbms like `postgresql`, this becomes more difficult as you would need to explicitly define how many decimal points, how much text, and you'd need to learn about a variety of other types, but we won't worry about this. We just want to get an idea on how this works for now.

Great. Now how would we go about defining this table? First, we would use the `CREATE TABLE` statement and define our table's name:

```sql
CREATE TABLE persons (
);
```

This create's a table named persons, pretty simple! Now let's define the columns in the table:

```
CREATE TABLE persons (
    name TEXT,
    age INTEGER,
    cash REAL,
    resume BLOB
);
```

That's not too bad! The pattern is `(column_name) (type),` where the very last row doesn't have a terminating comma. But wait, we are missing a column, id! What type is id? It looks like an `INTEGER` to me. So let's update things:

```
CREATE TABLE persons (
    id INTEGER,
    name TEXT,
    age INTEGER,
    cash REAL,
    resume BLOB
);
```

Great! Are we done? No. It is clear that we have a column that we can use to uniquely identify a person. Whenever you have a column where every single value is (and should be) absolutely unique, it is a candidate to be called a *primary key*. Read about primary keys here. In our case, our id column is a primary key. Let's update things:

```
CREATE TABLE persons (
    id INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER,
    cash REAL,
    resume BLOB
);
```

Excellent! Running this query in sqlite will create a brand new table that we can insert data into! Very cool! Let's say we had an R dataframe full of data in this format called `our_df`:

```
our_df <- data.frame("id"=1:2, "age"=c(14, 27), "cash"=c(16.47, 1000.50),
resume=c(NA, NA))

```

If we wanted to do that using R and `RSQLite`, we could do the following:

```
library(RSQLite)
ct <- "CREATE TABLE persons (
    id INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER,
    cash REAL,
    resume BLOB
);"
# create a connection to a new database that we are calling "out_new.db"
connection <- dbConnect(RSQLite::SQLite(), "our_new.db")

# execute the CREATE TABLE statement, ct, creating the table inside our
database
```

```r
dbExecute(connection, ct)

# load the our_df dataframe data into our new persons table
dbWriteTable(connection, "persons", our_df, overwrite=T)
```

So you can imagine, if we had a csv, we could load the csv into R, and use similar code to create an sqlite database, cool!

Okay, only one last topic to discuss. Let's say we have another table:

personId	id	name	type	mpg
1	1	Geneve	Honda Civic	35.4
1	2	Roger	Toyota Truck	17.3
2	3	Big Data	VW Van	13.0

Here, the types are clear (from left to right): `INTEGER`, `INTEGER`, `TEXT`, `TEXT`, `REAL`. The id column, once again, looks like a prime candidate to be a primary key. Each value should be unique, and we can identify a vehicle using it. How about the personId column?

In this example, the personId column contains integers that *definitely* match a person from our persons table. In this case, John owns the first two vehicles and Anna owns the last. This column is called a *foreign key*. Here is how we would add this table (and data) to our database, `our_new.db`.

```r
# previous code...

vehicles <- data.frame("personId"=c(1, 1, 2), "id"=c(1, 2, 3), "name"=c("Geneve", "Roger", "Big Data"), "type"=c("Honda Civic", "Toyota Truck", "VW Van"), "mpg"=c(35.4, 17.3, 13.0))

vt <- "CREATE TABLE vehicles (
    personId INTEGER,
    id INTEGER PRIMARY KEY,
    name TEXT,
    type TEXT,
    mpg REAL,
    FOREIGN KEY(personId) REFERENCES persons(id)
);"

# execute the CREATE TABLE statement, ct, creating the table inside our database
dbExecute(connection, vt)

# load the our_df dataframe data into our new persons table
dbWriteTable(connection, "vehicles", vehicles, overwrite=T)
```

As you can see, when we define the foreign key, we should which column the foreign key is in `FOREIGN KEY(personId)` and we explicitly define from what table and what column this foreign key is referencing `REFERENCES persons(id)` (where persons is the name of the table, and id is the name of the column in the table).

Very cool! Now you have some exposure to creating tables in sqlite!