Querying and Aggregating Individual Tables with Google BigQuery

As we work through these examples we will be using many tables from the MIMIC-III demo dataset. You can find data dictionaries that explain what is in each tables in more detail on the MIMIC-III website.

Selecting All Data

If you want to select all columns and rows from a table you need to use the "*" in your select statement. Note that in Google BigQuery (and in most systems that have multiple datasets/databases available) you will need to include the dataset name before you call the table name. In this case the MIMIC-III demo data is stored in the dataset named "mimic3_demo". To query the PATIENTS table you can copy/paste the following code:

```
SELECT * FROM mimic3_demo.PATIENTS
```

You may get a warning about being charged for querying all the data in this data set. This is ok, remember that all of the charges for your queries of data in this project are being supported by our Industry Partner Google Cloud. (Note that if you were to query non course data sets your Google account would be billed for those queries.)

Selecting Variables (Columns) and Records (Rows)

If we only want to select the patient identifier and gender from the PATIENTS table we can run the following query:

```
SELECT SUBJECT_ID, GENDER FROM mimic3_demo.PATIENTS
```

Notice that this query returned a result for all 100 patients. If we run the same query but use "where" to only return patients who are women:

```
SELECT SUBJECT_ID, GENDER FROM mimic3_demo.PATIENTS where GENDER="F"
```

You should find that only 55 records are returned. You may be wondering, where am I finding these numbers? Google BigQuery returns results across multiple pages, you can navigate through them at the bottom of the table.

Techniques for Filtering Records

Up to now we have assumed you know what data is in a column and you know exactly what you want to select. Far more commonly in data science we know which variable we want to use to filter, but we're not sure the format of the data in the column. In addition to looking at the whole table by hand with the Preview option, there are two tools you can use:

Identify All Unique Variable Values

If the variable has some level of structure or format, then getting a list of all unique values of the variable makes it much easier to review. In traditional SQL we would use the "distinct" argument. So a generic query would be something like:

```
SELECT distinct gender FROM Patients
```

Unfortunately Google BigQuery only allows the distinct argument when applying functions (we'll see this later). To get a list of unique items, instead you will want to use the "group by" syntax.

Let's look at ADMISSIONS table and the ADMISSION_LOCATION variable. If we wanted to see all possible places the patients were admitted from, we can use the following query:

```
SELECT ADMISSION_LOCATION FROM mimic3_demo.ADMISSIONS

  GROUP BY ADMISSION_LOCATION
```

When you run this query you can see that there are only five unique places for admission location. You can then use the values of interest in your select query in the future!

Try it out for yourself: Using the ICUSTAYS table, how many types of FIRST_CAREUNIT are there?

Search the Variable Using "Like"

Although the previous technique is powerful, in some cases the variable will actually contain unstructured data or have too many options to make manually reviewing all distinct values unfeasible. In this case you can look for keywords or fuzzy matches with the syntax "like". This works just like a "where" clause, but instead of an equals sign, you use the "like" command to get more complicated matches.

Let's look at the D_ICD_DIAGNOSES table and apply the unique value approach:

```
SELECT LONG_TITLE FROM mimic3_demo.D_ICD_DIAGNOSES

  GROUP BY LONG_TITLE
```

You will find that there are more than 14,000 unique values for LONG_TITLE. If our goal was to find all unique entries that had something to do with diabetes, reading through all 14,562 options would take forever. Instead let's try the following query:

```
SELECT LONG_TITLE FROM mimic3_demo.D_ICD_DIAGNOSES

 WHERE lower(LONG_TITLE) like "%diabetes%"

 GROUP BY LONG_TITLE
```

Let's look a bit closer at this query. I have added a "where" statement on the second line. In this statement I use the lower() function to make data in LONG_TITLE all lower case - this is to make sure that capitalization doesn't matter. Then I use the "like" clause to say what the variable should match. In quotes I put my term "diabetes" and surround it with two percent (%) signs. In BigQuery the % symbol means that it can match any character. So this would catch any statement that has something before or after the word diabetes. When you run this query you'll see that it matched things where diabetes was the first word, somewhere in the middle, or at the end. Here are just a couple of illustrative examples of matches found:

Diabetes with other specified manifestations, type II or unspecified type, uncontrolled
Secondary diabetes mellitus with neurological manifestations, uncontrolled
Polyneuropathy in diabetes

Try it out for yourself: Using the D_ITEMS table, how many values in LABEL have the word "weight" in them?

Aggregating Data

Although much of our Specialization uses R to aggregate and analyze data, especially when you are working with large data it is more efficient to do some of the data aggregation in SQL.

SQL Functions (e.g., count(), min(), max(), etc.)

SQL functions are tools you can apply to your column select statement to get transformed variables. The most simple example is count(). This function counts the number of records or rows that are returned. For example if we wanted to know how many entries were in the ADMISSIONS table we can

```
SELECT count(*) FROM mimic3_demo.ADMISSIONS
```

You can also count any particular variable. For instance if we want to query the number of HADM_IDs we can query:

```
SELECT count(HADM_ID) FROM mimic3_demo.ADMISSIONS
```

You'll find that this returns the number 129 - the same as the number of rows in the ADMISSIONS table. This is because each admission gets a unique HADM_ID. You may also notice that the column name is "f0_". This is the default way that Google BigQuery returns function columns. If you had a second function - let's say find the minimum (or earliest) time of admission, you would get a second column, "f1_".

```
SELECT count(HADM_ID), min(ADMITTIME) FROM mimic3_demo.ADMISSIONS
```

Because these names are not very descriptive, it's useful to rename your columns when you apply functions.

```
SELECT count(HADM_ID) as NUMBER_OF_ADMISSIONS, min(ADMITTIME) as
EARLIEST_ADMISSION

  FROM mimic3_demo.ADMISSION
```

You can apply filters to these queries and it will only return the counts of rows that match the criteria. For example, if we want to see the number of admissions where the ADMISSION_LOCATION was the emergency room, we can run the following query:

```
SELECT count(HADM_ID) FROM mimic3_demo.ADMISSIONS

  WHERE ADMISSION_LOCATION = "EMERGENCY ROOM ADMIT"
```

This should return a count of 81.

Try it out for yourself:

1. Using the ICUSTAYS table, count the number of number of ICU Stays in the database.
2. What is the earliest INTIME from the ICUSTAYS table?
3. What is the latest OUTTIME from the ICUSTAYS table?
4. How many ICU stays started (FIRST_CAREUNIT) in the MICU?

Distinct and Grouped Values

As described above, count() returns a count of all rows that match the criteria. Sometimes it's useful to know that total number of unique values in a particular column. In this case you can use the term "distinct" to limit the count to only unique values.

Let's see how many patients are actually included in the ADMISSIONS table. We can count the number of unique or *distinct* SUBJECT_IDs.

```
SELECT count(distinct SUBJECT_ID) FROM mimic3_demo.ADMISSIONS
```

We can see that 100 patients have had the 129 admissions in the database. This means that some patients have had more than one admission. If we want to see how many admissions each patient has had, we can apply the "group by" syntax.

```
SELECT SUBJECT_ID, count(distinct HADM_ID) as ADMISSION_COUNT FROM
mimic3_demo.ADMISSIONS

 GROUP BY SUBJECT_ID
```

In this query we take each SUBJECT_ID and count the number of unique HADM_IDs that have the same SUBJECT_ID. If you have a lot of data it may be helpful to sort this information.

```
SELECT SUBJECT_ID, count(distinct HADM_ID) as ADMISSION_COUNT FROM mimic3_demo.ADMISSIONS

 GROUP BY SUBJECT_ID

 ORDER BY ADMISSION_COUNT
```

This will list the rows by the number of admissions from smallest to largest. If you want to do the reverse order, simply apply the "desc" argument.

```
SELECT SUBJECT_ID, count(distinct HADM_ID) as ADMISSION_COUNT FROM mimic3_demo.ADMISSIONS

 GROUP BY SUBJECT_ID

 ORDER BY ADMISSION_COUNT desc
```