SQL Guide: Getting started

This reading covers some of the best practices for formatting queries so they are easy to read and understand. You will be introduced to conventions that help the purpose and function of a query to stand out. This involves how you work with commands in queries as well as how you work with multiple fields. The reading closes with an introduction to how you add comments to your queries to help explain your thinking and the expected outcome of a query.

Capitalization and indentation

Shown below is a recommended format for writing queries. Capitalize SELECT, FROM, and WHERE. Make sure to add a new line and indent when adding the fields.

```
SELECT
            Columns you want to look at
FROM
            Table the data lives in
WHERE
            Certain condition is met
```

Here's an example of what this could look like in BigQuery.

```
SELECT
      first_name
FROM
      customer_data.customer_name
WHERE
      first_name = 'Tony'
```

The query uses three commands to locate customers with the first name Tony.

1. SELECT the column named 'first_name'
2. FROM a table named 'customer_name' (in a database named 'customer_data')
3. but only the data WHERE 'first_name' is 'Tony'

Tip: This is like filling in a template. Always start queries by writing the SELECT, FROM, and WHERE statements in this format. Enter your table name after the FROM command, enter the field(s) you want after the SELECT command, and then finally, enter the conditions you want to place on your query after the WHERE command. You'll find this makes it easier to write SQL queries and it might be more naturally intuitive to proceed from large to small by entering the large details (table) before the small details (fields and conditions).

Multiple fields

Using the indentation previously described also makes it easier for you to group multiple fields together that are affected by the same command.

If you are requesting data from a table that has multiple fields, you need to include these fields in your SELECT command:

```
SELECT
        ColumnA,
        ColumnB,
        ColumnC
FROM
        Table where the data lives
WHERE
        Certain condition is met
```

Here is an example of what this could look like in BigQuery:

```
SELECT
        customer_id,
        first_name,
        last_name
FROM
        customer_data.customer_name
WHERE
        first_name = 'Tony'
```

The query uses three commands to locate customers with the first name Tony.

1. SELECT the columns named 'customer_id', 'first_name', and 'last_name'
2. FROM a table named 'customer_name' (in a database named 'customer_data')
3. but only the data WHERE 'first_name' is 'Tony'

This query is no different than the previous query except that more data columns were selected. In general, it's a better use of resources to select columns that you'll make use of in your query.

It makes sense to select more columns if you use the additional fields in your WHERE statement. If you have multiple conditions in your WHERE statement they may be written like this:

```
SELECT
            ColumnA,
            ColumnB,
            ColumnC
FROM
            Table where the data lives
WHERE
            Condition 1
            AND Condition 2
            AND Condition 3
```

Notice that unlike the SELECT command that uses a comma to separate fields/variables/parameters, the WHERE command uses the AND statement to connect conditions. This is important to remember because as you become a more advanced writer of queries you will make use of other connectors/operators such as OR and NOT to connect your conditions.

Here is an example of what this could look like in BigQuery:

```
SELECT
       customer_id,
       first_name,
       last_name
FROM
       customer_data.customer_name
WHERE
       customer_id > 0
       AND first_name = 'Tony'
       AND last_name = 'Magnolia'
```

The query uses three commands to locate customers with a valid (greater than 0) customer ID whose first name is Tony and last name is Magnolia.

1. SELECT the columns named ‘customer_id’, ‘‘first_name’, and ‘last_name’
2. FROM a table named ‘customer_name’ (in a database named ‘customer_data’)
3. but only the data WHERE ‘customer_id’ is greater than zero, ‘first_name’ is ‘Tony’, and ‘last_name’ is ‘Magnolia’.

Note that one of the conditions is a logical condition where we are checking to see if customer_id is greater than zero.

Comments as reminders

The more comfortable you get with SQL, the easier it will be to read and understand queries at a glance. Still, it never hurts to have comments in a query to remind yourself of what you’re trying to do. This also makes it easier for others to understand your query if your query is shared. As your queries become more and more complex, this practice will save you a lot of time and energy to understand complex queries you wrote months or years ago.

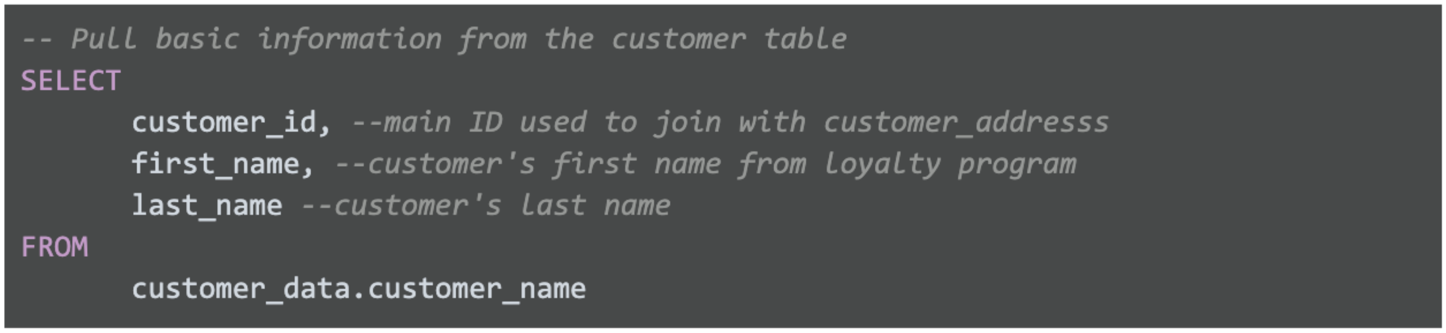
Over time, your style of query will change and comments will help you remember what your thinking was at the time. Use the “--” symbols to make comments in your query. It tells SQL to ignore whatever comes after the “--” within the same line. For example:

```
-- This is an important query used later to join with the accounts table
      SELECT
            rowkey,  -- key used to join with account_id
            Info.date,  -- date is in string format YYYY-MM-DD HH:MM:SS
            Info.code  -- e.g., 'pub-###'

      FROM  Publishers
```

Notice that comments can be added outside of a statement as well as within a statement. You can use this flexibility to provide an overall description of what you are going to do, step-by-step notes about how you achieve it, and why you set different parameters/conditions.

Here is an example of how comments could be written in BigQuery:



```
-- Pull basic information from the customer table
SELECT
      customer_id, --main ID used to join with customer_addresss
      first_name, --customer's first name from loyalty program
      last_name --customer's last name
FROM
      customer_data.customer_name
```

In the example , we provide a comment next to each of the column names and give a description of the column and its uses. Two dashes (--) are generally supported. So it’s best to use -- and be consistent with it. You can use # in place of -- in the above query but # is not recognized in all SQL versions; for example, MySQL doesn’t recognize #. You can also use /* before and */ after a comment if the database you’re using supports it.

As you develop your skills professionally, depending on the SQL database you use, you can pick the appropriate comment delimiting symbols you prefer and stick with those as a consistent style.