Introduction to SQL: 1st lesson – Getting Started With SQL & BigQuery

Structured Query Language (SQL):

* The programming language used with databases (domain-specific language), and it is an important skill for any data scientist for managing data contained in a Relational Database Management System (RDBMS) or for stream processing in a Relational Data Stream Management System (RDSMS), particularly useful in handling structured data like data-incorporating relations among entities and variables.
* SQL was initially developed at IBM by Donald D. Chamberlin and Raymond F. Boyce during post-lecture about the relational model given by Edgar F. Codd in the early 1970s, known as SEQUEL (an acronym for **S**tructured **E**nglish **QUE**ry **L**anguage), which designed to manipulate and retrieve data stored in its original quasirelational database management system, IBM System R, which formed as group of IBM San Jose Research Laboratory.

BigQuery commands:

* To use BigQuery, we'll import the Python package below:

from google.cloud import bigquery

The first step in the workflow is to create a Client object. As you'll soon see, this Client object will play a central role in retrieving information from BigQuery datasets.

# Create a "Client" object

client = bigquery.Client()

Using Kaggle's public dataset BigQuery integration.

* We'll work with a dataset of posts on Hacker News, a website focusing on computer science and cyber-security news. In BigQuery, each dataset is contained in a corresponding project. In this case, our hacker\_news dataset is contained in the bigquery-public-data project. To access the dataset:
* Start the process by constructing a reference to the dataset with the dataset() method.
* Next step, use the get\_dataset() method, along with the reference that has been constructed, to fetch the dataset.

# Construct a reference to the "hacker\_news" dataset

dataset\_ref = client.dataset("hacker\_news", project="bigquery-public-data")

# API request - fetch the dataset

dataset = client.get\_dataset(dataset\_ref)

Every dataset is just a collection of tables. Users can think about dataset as a spreadsheet file containing multiple tables, the entire composition of rows and columns. Use the list\_tables() method to the list the tables in the dataset:

# List all the tables in the "hacker\_news" dataset

tables = list(client.list\_tables(dataset))

# Print names of all tables in the dataset (there are four!)

for table in tables:

print(table.table\_id)

comments

full

full\_201510

stories

* Similar to how the dataset is fetched, the user can fetch a table. In the code cell below, fetch the full table in the hacker\_news dataset.

# Construct a reference to the "full" table

table\_ref = dataset\_ref.table("full")

# API request - fetch the table

table = client.get\_table(table\_ref)

Table schema:

* The structure of a table is called its schema. Users need to understand a table's schema to effectively pull out the data that users want. In this example, investigate the full table that has been fetched above.

# Print information on all the columns in the "full" table in the "hacker\_news" dataset

table.schema

[SchemaField('title', 'STRING', 'NULLABLE', 'Story title', (), None),

SchemaField('url', 'STRING', 'NULLABLE', 'Story url', (), None),

SchemaField('text', 'STRING', 'NULLABLE', 'Story or comment text', (), None),

SchemaField('dead', 'BOOLEAN', 'NULLABLE', 'Is dead?', (), None),

SchemaField('by', 'STRING', 'NULLABLE', "The username of the item's author.", (), None),

SchemaField('score', 'INTEGER', 'NULLABLE', 'Story score', (), None),

SchemaField('time', 'INTEGER', 'NULLABLE', 'Unix time', (), None),

SchemaField('timestamp', 'TIMESTAMP', 'NULLABLE', 'Timestamp for the unix time', (), None),

SchemaField('type', 'STRING', 'NULLABLE', 'Type of details (comment, comment\_ranking, poll, story, job, pollopt)', (), None),

SchemaField('id', 'INTEGER', 'NULLABLE', "The item's unique id.", (), None),

SchemaField('parent', 'INTEGER', 'NULLABLE', 'Parent comment ID', (), None), SchemaField('descendants', 'INTEGER', 'NULLABLE', 'Number of story or poll descendants', (), None),

SchemaField('ranking', 'INTEGER', 'NULLABLE', 'Comment ranking', (), None),

SchemaField('deleted', 'BOOLEAN', 'NULLABLE', 'Is deleted?', (), None)]

Each SchemaField describes about a specific column (which also referred to as a field). In order, the information is:

* The name of the column
* The field type, or data type in the column
* The mode of the column {‘NULLABLE’ means that a column allows NULL values, and is the default}
* A description of the data in that column
* The first field has the SchemaField:

SchemaField('by', 'string', 'NULLABLE', "The username of the item's author.",())

This tells the user:

* The field, or the column, is called by
* The data in this field is strings
* Null values are allowed
* Contains the usernames corresponding to each item’s author