Downloading BigQuery data to pandas using the BigQuery Storage API

The BigQuery Storage API provides fast access to data stored in BigQuery (/bigquery/docs). Use the BigQuery Storage API to download data stored in BigQuery for use in analytics tools such as the pandas library for Python (https://pandas.pydata.org/).

Objectives

In this tutorial you:

- Download query results to a pandas DataFrame by using the BigQuery Storage API from the IPython magics for BigQuery in a Jupyter notebook.
- Download query results to a pandas DataFrame by using the BigQuery client library for Python.
- Download BigQuery table data to a pandas DataFrame by using the BigQuery client library for Python.
- Download BigQuery table data to a pandas DataFrame by using the BigQuery Storage API client library for Python.

Costs

BigQuery is a paid product and you will incur BigQuery usage costs for the queries you run. The first 1 TB of query data processed per month is free. For more information, see the BigQuery Pricing (/bigquery/pricing) page.

BigQuery Storage API is a paid product and you will incur usage costs for the table data you scan when downloading a DataFrame. For more information, see the BigQuery Pricing (/bigquery/pricing) page.

Before you begin

Before you begin this tutorial, use the Google Cloud Console to create or select a project and enable billing.

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

Go to the project selector page (https://console.cloud.google.com/projectselector2/home/dashboar

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (/billing/docs/how-to/modify-project).

4. BigQuery is automatically enabled in new projects. To activate BigQuery in a preexisting project, Enable the BigQuery, BigQuery Storage API APIs.

   Enable the APIs (https://console.cloud.google.com/flows/enableapi?apiid=bigquery,bigquerystorage)

5. Set up a Python development environment.

   Setup Python (/python/setup)

6. Set up authentication for your development environment.

   Setup Authentication (/docs/authentication/getting-started)

You should also be familiar with the IPython magics for BigQuery (https://googleapis.github.io/google-cloud-python/latest/bigquery/magics.html), the BigQuery client library (/bigquery/docs/quickstarts/quickstart-client-libraries), and how to use the client library with pandas (https://googleapis.github.io/google-cloud-python/latest/bigquery/usage/pandas.html) before completing this tutorial.

Install the client libraries

Install the BigQuery Python client library version 1.9.0 (https://googleapis.github.io/google-cloud-python/latest/bigquery/changelog.html) or higher and the BigQuery Storage API Python client library.

PIP Conda (#conda)

Install the `google-cloud-bigquery` (https://pypi.org/project/google-cloud-bigquery/) and `google-cloud-bigquery-storage` (https://pypi.org/project/google-cloud-bigquery-storage/) packages.

```
pip install --upgrade google-cloud-bigquery[bqstorage,pandas]
```

Download query results using the IPython magics for BigQuery

Start the Jupyter notebook server and create a new Jupyter notebook. Load the IPython magics for BigQuery using the `%load_ext` magic (https://ipython.readthedocs.io/en/stable/config/extensions/index.html?highlight=load_ext#using-extensions)
.

```
%load_ext google.cloud.bigquery
```

Download large query results with the BigQuery Storage API by adding the `--use_bq_storage_api` argument to the `%%bigquery` magics.

```
%%bigquery tax_forms --use_bqstorage_api
SELECT * FROM `bigquery-public-data.irs_990.irs_990_2012`
```

When this argument is used with small query results, the magics use the BigQuery API to download the results.

```
%%bigquery stackoverflow --use_bqstorage_api
SELECT
  CONCAT(
    'https://stackoverflow.com/questions/',
    CAST(id as STRING)) as url,
  view_count
FROM `bigquery-public-data.stackoverflow.posts_questions`
WHERE tags like '%google-bigquery%'
ORDER BY view_count DESC
LIMIT 10
```

Set the `context.use_bqstorage_api` (https://googleapis.github.io/google-cloud-python/latest/bigquery/magics.html#google.cloud.bigquery.magics.Context.use_bqstorage_api) property to `True` to use the BigQuery Storage API by default.

```
import google.cloud.bigquery.magics

google.cloud.bigquery.magics.context.use_bqstorage_api = True
```

After you set the `context.use_bqstorage_api` property, run the `%%bigquery` magics without additional arguments to use the BigQuery Storage API to download large results.

```
%%bigquery tax_forms
SELECT * FROM `bigquery-public-data.irs_990.irs_990_2012`
```

Use the Python client libraries

Create Python clients

Use the following code to construct a BigQuery `Client` (https://googleapis.github.io/google-cloud-python/latest/bigquery/generated/google.cloud.bigquery.client.Client.html) object and a `BigQueryStorageClient`

(https://googleapis.github.io/google-cloud-python/latest/bigquery_storage/gapic/v1beta1/api.html#google.cloud.bigquery_storage_v1beta1.BigQueryStorageClient)

.

les/blob/5d43edba3890718e5643e63921e91ec665389aec/bigquery_storage/to_dataframe/main_test.py)

```
import google.auth
from google.cloud import bigquery
from google.cloud import bigquery_storage_v1beta1

# Explicitly create a credentials object. This allows you to use the same
# credentials for both the BigQuery and BigQuery Storage clients, avoiding
# unnecessary API calls to fetch duplicate authentication tokens.
credentials, your_project_id = google.auth.default(
    scopes=["https://www.googleapis.com/auth/cloud-platform"]
)

# Make clients.
bqclient = bigquery.Client(
    credentials=credentials,
    project=your_project_id,
)
bqstorageclient = bigquery_storage_v1beta1.BigQueryStorageClient(
    credentials=credentials
)
```

Use the google-auth Python library (https://google-auth.readthedocs.io/en/latest/) to create credentials that are sufficiently scoped for both APIs. Pass in a credentials object to each constructor to avoid authenticating twice.

Download query results using the BigQuery client library

Run a query by using the `query` (https://googleapis.github.io/google-cloud-python/latest/bigquery/generated/google.cloud.bigquery.client.Client.html#google.cloud.bigquery.client.Client.query) method. Call the `to_dataframe`

(https://googleapis.github.io/google-cloud-python/latest/bigquery/generated/google.cloud.bigquery.job.QueryJob.html#google.cloud.bigquery.job.QueryJob.to_dataframe)
method to wait for the query to finish and download the results by using the BigQuery Storage API.

les/blob/5d43edba3890718e5643e63921e91ec665389aec/bigquery_storage/to_dataframe/main_test.py)

```
# Download query results.
query_string = """
SELECT
CONCAT(
    'https://stackoverflow.com/questions/',
    CAST(id as STRING)) as url,
view_count
FROM `bigquery-public-data.stackoverflow.posts_questions`
WHERE tags like '%google-bigquery%'
ORDER BY view_count DESC
"""

dataframe = (
    bqclient.query(query_string)
    .result()
    .to_dataframe(bqstorage_client=bqstorageclient)
)
print(dataframe.head())
```

Download table data using the BigQuery client library

Download all rows in a table by using the `list_rows` (https://googleapis.github.io/google-cloud-python/latest/bigquery/generated/google.cloud.bigquery.client.Client.html#google.cloud.bigquery.client.Client.list_rows) method, which returns a `RowIterator` (https://googleapis.github.io/google-cloud-python/latest/bigquery/generated/google.cloud.bigquery.table.RowIterator.html) object. Download rows by using the BigQuery Storage API by calling the `to_dataframe`

(https://googleapis.github.io/google-cloud-python/latest/bigquery/generated/google.cloud.bigquery.table.RowIterator.html#google.cloud.bigquery.table.RowIterator.to_dataframe)
method with the `bqstorage_client` argument.

les/blob/5d43edba3890718e5643e63921e91ec665389aec/bigquery_storage/to_dataframe/main_test.py)

```
# Download a table.
table = bigquery.TableReference.from_string(
    "bigquery-public-data.utility_us.country_code_iso"
)
rows = bqclient.list_rows(
    table,
    selected_fields=[
        bigquery.SchemaField("country_name", "STRING"),
        bigquery.SchemaField("fips_code", "STRING"),
    ],
)
dataframe = rows.to_dataframe(bqstorage_client=bqstorageclient)
print(dataframe.head())
```

Download table data using the BigQuery Storage API client library

Use the BigQuery Storage API client library directly for fine-grained control over filters and parallelism. When only simple row filters
(https://googleapis.github.io/google-cloud-python/latest/bigquery_storage/gapic/v1beta1/types.html#google.cloud.bigquery_storage_v1beta1.types.TableReadOptions.row_restriction)
are needed, a BigQuery Storage API read session may be used in place of a query.

Create a TableReference
(https://googleapis.github.io/google-cloud-python/latest/bigquery_storage/gapic/v1beta1/types.html#google.cloud.bigquery_storage_v1beta1.types.TableReference)
object with the desired table to read. Create a TableReadOptions
(https://googleapis.github.io/google-cloud-python/latest/bigquery_storage/gapic/v1beta1/types.html#google.cloud.bigquery_storage_v1beta1.types.TableReadOptions)
object to select columns or filter rows. Create a read session using the `create_read_session`

(https://googleapis.github.io/google-cloud-python/latest/bigquery_storage/gapic/v1beta1/api.html#google.cloud.bigquery_storage_v1beta1.BigQueryStorageClient.create_read_session)
method.

If there are any streams on the session, begin reading rows from it by using the `read_rows` (https://googleapis.github.io/google-cloud-python/latest/bigquery_storage/gapic/v1beta1/api.html#google.cloud.bigquery_storage_v1beta1.BigQueryStorageClient.read_rows)
method. Call the `to_dataframe` (https://googleapis.github.io/google-cloud-python/latest/bigquery_storage/gapic/v1beta1/reader.html#google.cloud.bigquery_storage_v1beta1.reader.ReadRowsStream.to_dataframe)
method on the reader to write the entire stream to a pandas DataFrame. For better performance, read from multiple streams in parallel, but this code example reads from only a single stream for simplicity.

les/blob/5d43edba3890718e5643e63921e91ec665389aec/bigquery_storage/to_dataframe/main_test.py)

```
table = bigquery_storage_v1beta1.types.TableReference()
table.project_id = "bigquery-public-data"
table.dataset_id = "new_york_trees"
table.table_id = "tree_species"

# Select columns to read with read options. If no read options are
# specified, the whole table is read.
read_options = bigquery_storage_v1beta1.types.TableReadOptions()
read_options.selected_fields.append("species_common_name")
read_options.selected_fields.append("fall_color")

parent = "projects/{}".format(your_project_id)
session = bqstorageclient.create_read_session(
    table,
    parent,
    read_options=read_options,
    # This API can also deliver data serialized in Apache Avro format.
    # This example leverages Apache Arrow.
    format_=bigquery_storage_v1beta1.enums.DataFormat.ARROW,
    # We use a LIQUID strategy in this example because we only read from a
    # single stream. Consider BALANCED if you're consuming multiple streams
    # concurrently and want more consistent stream sizes.
    sharding_strategy=(
```

```
        bigquery_storage_v1beta1.enums.ShardingStrategy.LIQUID
    ),
)

# This example reads from only a single stream. Read from multiple streams
# to fetch data faster. Note that the session may not contain any streams
# if there are no rows to read.
stream = session.streams[0]
position = bigquery_storage_v1beta1.types.StreamPosition(stream=stream)
reader = bqstorageclient.read_rows(position)

# Parse all Avro blocks and create a dataframe. This call requires a
# session, because the session contains the schema for the row blocks.
dataframe = reader.to_dataframe(session)
print(dataframe.head())
```

Source code for all examples

▶ View the complete source code for all client library examples.

Cleaning up

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

Delete your project. You didn't create any BigQuery resources this tutorial, but deleting your project removes any other resources that you created.

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an **`appspot.com`** URL, delete selected resources inside the project instead of deleting the whole project.

If you plan to explore multiple tutorials and quickstarts, reusing projects can help you avoid exceeding project quota limits.

1. In the Cloud Console, go to the **Manage resources** page.

   Go to the Manage resources page (https://console.cloud.google.com/iam-admin/projects)

2. In the project list, select the project that you want to delete and then click **Delete** .
3. In the dialog, type the project ID and then click **Shut down** to delete the project.

What's next

- **Explore the Python client libraries reference** —
  - BigQuery client library for Python reference (https://googleapis.github.io/google-cloud-python/latest/bigquery/reference.html)
  - BigQuery Storage API client library for Python reference (https://googleapis.github.io/google-cloud-python/latest/bigquery_storage/index.html#api-reference)

Last updated 2020-06-25 UTC.