BigQuery I/O

ng: Dataflow SDK 1.x for Java is unsupported as of October 16, 2018. After August 12, 2020, Dataflow will no
sing Dataflow 1.x and below. See Migrating from Dataflow SDK 1.x for Java
aflow/docs/guides/migrate-java-1-to-2) for migration guidance.

ocumentation on this page applies only to the Dataflow SDK 1.x for Java.

ataflow SDK 2.x for Java and the Dataflow SDK for Python are based on Apache Beam. See the Beam BigQuer
nentation (https://beam.apache.org/documentation/io/built-in/google-bigquery/) for SDK 2.x.

The Dataflow SDKs have built-in **Read** and **Write** transforms that can read data from, and write data to, a Google BigQuery (/bigquery) table. You can read an entire table that you specify by name, or you can read partial data by using a query string.

Specifying a BigQuery Table Name

To read or write from a BigQuery table, you must provide a fully-qualified BigQuery table name. A fully-qualified BigQuery table name consists of three parts:

- A **Project ID**: The ID for your Google Cloud Project. The default value comes from your pipeline options object.
- A **Dataset ID**: The BigQuery dataset ID, which is unique within a given Cloud Project.
- A **Table ID**: A table ID, which is unique within a given dataset.

Java

Note that you can use `BigQueryIO` without supplying a project name; if omitted, `BigQueryIO` uses the default project from your `PipelineOptions` object.

The BigQuery Java Client API (/bigquery/client-libraries) takes an object of type **`TableReference`** to identify the target BigQuery table. The **`BigQueryIO`** package in the Dataflow SDK for Java contains a helper method, **`BigQueryIO.parseTableSpec`**, that you can use to construct a **`TableReference`** from a **`String`** containing the three parts of your BigQuery table name.

Most of the time, you won't need to use a **`TableReference`** object explicitly; the static factory methods for a **`BigQueryIO`** transform take the table name as a **`String`**; they then use **`parseTableSpec`** internally to construct a **`TableReference`** object from the provided **`String`**.

Table Name String Formats

You can specify the target BigQuery table using a string containing one of the following formats:

```
[project_id]:[dataset_id].[table_id]
Example: "clouddataflow-readonly:samples.weather_stations"
```

Java

You can also omit `project_id`. If you omit `project_id`, Cloud Dataflow will use the default project ID from your pipeline options object. In Java, the ID can be accessed with `PipelineOptions.getProject`.

```
[dataset_id].[table_id]
Example: "samples.weather_stations"
```

BigQuery Table Rows and Schemas

Java

`BigQueryIO` read and write transforms produce and consume data as `PCollection`s of BigQuery `TableRow` objects. `TableRow` is part of the BigQuery Java Client API, in the package `com.google.api.services.bigquery.model.TableRow`.

In addition, when writing to BigQuery, you'll need to supply a `TableSchema` object for the fields you want to write to the target table. You'll need to make use of both the BigQuery `TableSchema` and

`TableFieldSchema` classes. These classes are defined in the packages `com.google.api.services.bigquery.model.TableSchema` and `com.google.api.services.bigquery.model.TableFieldSchema`, respectively.

Reading from BigQuery

Java

To read from a BigQuery table, you `apply` a `BigQueryIO.Read` transform. `BigQueryIO.Read` returns a `PCollection` of BigQuery `TableRow` objects, where each element in the `PCollection` represents a single row in the table.

You can read an entire BigQuery table by supplying the BigQuery table name to `BigQueryIO.Read` by using the `.from` operation. The following example code shows how to apply the `BigQueryIO.Read` transform to read an entire BigQuery table:

```
PipelineOptions options = PipelineOptionsFactory.create();
Pipeline p = Pipeline.create(options);

PCollection<TableRow> weatherData = p.apply(
  BigQueryIO.Read
       .named("ReadWeatherStations")
       .from("clouddataflow-readonly:samples.weather_stations"));
```

If you don't want to read the entire table, you can supply a query string to `BigQueryIO.Read` by using the `.fromQuery` operation. The following example code shows how to read specific fields from a BigQuery table by using a query string:

```
PipelineOptions options = PipelineOptionsFactory.create();
Pipeline p = Pipeline.create(options);

PCollection<TableRow> weatherData = p.apply(
  BigQueryIO.Read
       .named("ReadYearAndTemp")
       .fromQuery("SELECT year, mean_temp FROM [samples.weather_stations]");
```

Alternatively, you can use BigQuery's standard SQL dialect (/dataflow/java-sdk/JavaDoc/com/google/cloud/dataflow/sdk/io/BigQueryIO.Read), as shown in the following example:

```
PCollection<TableRow> weatherData = p.apply(
BigQueryIO.Read
    .named("ReadYearAndTemp")
    .fromQuery("SELECT year, mean_temp FROM `samples.weather_stations`")
    .usingStandardSql();
```

Note that when reading from **`BigQuery`**, the integer values in the **`TableRow`** objects are encoded as **`String`**s to match BigQuery's exported JSON format.

When you apply a **`BigQueryIO.Read`** transform in batch mode, Dataflow invokes a BigQuery export request (/bigquery/docs/exporting-data). Note that Dataflow's use of this API is subject to BigQuery's Quota (https://cloud.google.com/bigquery/quota-policy#export) and Pricing (https://cloud.google.com/bigquery/pricing) policies.

Writing to BigQuery

Java

To write to a BigQuery table, you `apply` a `BigQueryIO.Write` transform. You'll need to `apply` the transform to a `PCollection<TableRow>`.

In general, you'll need to use another transform, such as ParDo (/dataflow/model/par-do), to format your output data into a collection of BigQuery **`TableRow`** objects.

When you construct a `BigQueryIO.Write` transform, you'll need to provide some additional information based on the target table. In addition to the table name, you'll need to provide the following:

- The target table's `CreateDisposition`. `CreateDisposition` specifies whether the target table must exist or can be created by the write operation.

- The target table's `WriteDisposition`. `WriteDisposition` specifies whether the data you write will *replace* an existing table, *append* rows to an existing table, or write only to an *empty* table.

In addition, if your write operation creates a new BigQuery table, you must supply schema information about the target table. In this case, you will need to include a `TableSchema` object with your write operation.

CreateDisposition

The `CreateDisposition` controls whether or not your BigQuery write operation should create a table if the target table does not exist. You specify the `CreateDisposition` when constructing your `BigQueryIO.Write` transform by invoking the method `.withCreateDisposition`.

`CreateDisposition` is an `enum` with the following valid values:

- `BigQueryIO.Write.CreateDisposition.CREATE_NEVER`: Specifies that a table should never be created. If the target table does not exist, the write operation fails.
- `BigQueryIO.Write.CreateDisposition.CREATE_IF_NEEDED`: Specifies that the write operation should create a new table if one does not exist. If you use this value, you'll need to also supply a table schema using the `.withSchema` operation. `CREATE_IF_NEEDED` is the default behavior.

Note that if you specify **`CREATE_IF_NEEDED`** as the **`CreateDisposition`** and you don't supply a **`TableSchema`**, the transform may fail at runtime with a **`java.lang.IllegalArgumentException`** if the target table does not exist.

WriteDisposition

The `WriteDisposition` controls how your BigQuery write operation applies to an existing table. You specify the `WriteDisposition` when constructing your `BigQueryIO.Write` transform by invoking the method `.withWriteDisposition`.

`WriteDisposition` is an `enum` with the following valid values:

- `BigQueryIO.Write.WriteDisposition.WRITE_TRUNCATE`: Specifies that the write operation should *replace* an existing table. Any existing rows in the target table are removed, and the new rows are added to the table.
- `BigQueryIO.Write.WriteDisposition.WRITE_APPEND`: Specifies that the write operation should *append* the rows to the end of the existing table.

- `BigQueryIO.Write.WriteDisposition.WRITE_EMPTY`: Specifies that the write operation should fail at runtime if the target table is not empty. `WRITE_EMPTY` is the default behavior.

When using **`WRITE_EMPTY`** for the **`WriteDisposition`**, note that the check for whether or not the target table is empty may occur far in advance of the actual write operation. In addition, such a check doesn't guarantee that your pipeline will have exclusive access to the table. If two programs, running concurrently, attempt to write to the same output table with a **`WriteDisposition`** of **`WRITE_EMPTY`**, both may succeed.

Creating a TableSchema for Writing to a New Table

If your BigQuery write operation creates a new table, you'll need to provide schema information. You provide the schema information by creating a `TableSchema` object. You pass the `TableSchema` using the `.withSchema` operation when you construct your `BigQueryIO.Write` transform.

A `TableSchema` object contains information about each field in the table, using objects of type `TableFieldSchema`. You construct a `TableSchema` by first building a `List` of the fields in the table. Then you pass the list using the `.setFields` operation when you construct the `TableSchema`.

The following example code shows how to construct a `TableSchema` for a table with two fields of type `String`:

```
List<TableFieldSchema> fields = new ArrayList<>();
fields.add(new TableFieldSchema().setName("source").setType("STRING"));
fields.add(new TableFieldSchema().setName("quote").setType("STRING"));
TableSchema schema = new TableSchema().setFields(fields);
```

Applying a BigQueryIO.Write Transform

The following example code shows how to `apply` a `BigQueryIO.Write` transform to write a `PCollection<TableRow>` to a BigQuery table. The write operation creates a table if needed; if the table already exists, it will be replaced.

```
PCollection<TableRow> quotes = ...;

quotes.apply(BigQueryIO.Write
    .named("Write")
    .to("my-project:output.output_table")
    .withSchema(schema)
```

```
      .withWriteDisposition(BigQueryIO.Write.WriteDisposition.WRITE_TRUNCATE)
      .withCreateDisposition(BigQueryIO.Write.CreateDisposition.CREATE_IF_NEEDED));
```

`BigQueryIO.Write` uses APIs that are subject to BigQuery's Quota (https://cloud.google.com/bigquery/quota-policy) and Pricing (https://cloud.google.com/bigquery/pricing) policies.

When you **`apply`** a **`BigQueryIO.Write`** transform in batch mode to write to a single table, Dataflow invokes a BigQuery load job (/bigquery/loading-data). When you **`apply`** a **`BigQueryIO.Write`** transform in streaming mode or in batch mode using a function to specify the destination table, Dataflow uses BigQuery's streaming inserts (/bigquery/streaming-data-into-bigquery).