Writing from Spark into Greenplum Database

添加到库

[RSS](https://docs.vmware.com/en/VMware-Tanzu-Greenplum-Connector-for-Apache-Spark/rss.xml" \t "_blank)

反馈

更新时间 2022年07月12日

选择的产品版本：**VMware Tanzu Greenplum Connector for Apache Spark 2.1**

Writing a Spark DataFrame into a Greenplum Database table loads each Row in the DataFrame into the table. You can use the Spark Scala API or the spark-shell interactive shell to write Spark data to a Greenplum Database table that you created with the CREATE TABLE SQL command.

The VMware Tanzu Greenplum Connector for Apache Spark Connector provides a Spark data source optimized for writing Spark data into Greenplum Database data. To write to a Greenplum Database table, you must identify the Connector data source name and provide write options for the export.

Connector Data Source

A Spark data source provides an access point to structured data. Spark provides several pre-defined data sources to support specific file types and databases. You specify a Spark data source using either its fully qualified name or its short name.

The Connector exposes a Spark data source named greenplum to transfer data between Spark and Greenplum Database. The Connector supports specifying the data source only with this short name.

Use the .format(datasource: String) Scala method to identify the data source. You must provide the Connector data source short name greenplum to the .format() method. For example:

dfToWrite.write.format("greenplum")

Connector Write Options

The greenplum data source supports the write options identified in the table below. An option is required unless otherwise specified.

| **Option Key** | **Value Description** |  |
| --- | --- | --- |
| url | The JDBC connection string URL; see [Constructing the Greenplum Database JDBC URL](https://docs.vmware.com/en/VMware-Tanzu-Greenplum-Connector-for-Apache-Spark/2.1/tanzu-greenplum-connector-spark/GUID-using_the_connector.html" \l "use_jdbcurl). |  |
| dbschema | The name of the Greenplum Database schema in which dbtable resides. Optional, the default schema is the schema named public. |  |
| dbtable | The name of the Greenplum Database table. The Connector creates this table in the schema named dbschema if it does not exist. |  |
| driver | The fully qualified class path of the custom JDBC driver. Optional, specify only when using a custom JDBC driver. |  |
| user | The Greenplum Database user/role name. |  |
| password | The Greenplum Database password for the user. You can omit the password if Greenplum Database is configured to not require a password for the specified user, or if you use kerberos authentication and provide the required authentication properties in the JDBC connection string URL. Optional. |  |
| truncate | The table overwrite mode. Governs the table creation actions of the Connector when you specify SaveMode.Overwrite and the target Greenplum Database table exists. The default value is false; the Connector drops and then re-creates the target table before it writes any data. When true, the Connector truncates the target table before writing data. Optional. | Write |
| distributedBy | The distribution column(s) of the Greenplum table. Governs the table creation action of the Connector when the target Greenplum Database table does not exist, or when you specify SaveMode.Overwrite on a write operation and truncate is false. The Connector (re)creates the table with random distribution by default. When you provide one or more distributedBy columns, the Connector (re)creates the table with a DISTRIBUTED BY clause that specifies these column names. Optional. | Write |

Writing to Greenplum Database

When you write a Spark DataFrame to a Greenplum Database table, you identify the Connector data source, provide the write options, and invoke the DataFrameWriter.save() method on the Spark DataFrame that you want to write. For example:

val gscWriteOptionMap = Map(

"url" -> "jdbc:postgresql://gpdb-master:5432/testdb",

"user" -> "bill",

"password" -> "changeme",

"dbschema" -> "myschema",

"dbtable" -> "table2",

)

dfToWrite.write.format("greenplum")

.options(gscWriteOptionMap)

.save()

You can specify the behaviour of the Connector when the Greenplum Database table already exists, or when the table already contains data. You provide this information with the DataFrameWriter.mode(SaveMode savemode) method.

The Connector supports the following Spark SaveMode settings:

| **SaveMode** | **Behaviour** |
| --- | --- |
| ErrorIfExists | The Connector returns an error if the Greenplum Database table already exists. This is the default value. |
| Append | The Connector appends the Spark data to data that may already exist in the Greenplum Database table. |
| Ignore | If the Greenplum Database table already exists, the Connector ignores the write request; it neither writes data to the table nor does it disturb the existing data. |
| Overwrite | If the Greenplum Database table already exists, the truncate option value governs whether the Connector drops and recreates the target table (default, truncate is false), or truncates the target table (truncate is true) before writing the data. **Note**: The Connector cannot truncate a Greenplum Database table if it includes any foreign key constraints. |

You must enter import org.apache.spark.sql.SaveMode in your Spark Scala application or the spark-shell when you specify a SaveMode. For example, to specify the Append mode on write in a Scala application:

import org.apache.spark.sql.SaveMode

dfToWrite.write.format("greenplum")

.options(gscWriteOptionMap)

.mode(SaveMode.Append)

.save()

To specify the Append mode on a write in a PySpark application, provide the mode text string:

.mode("Append")

When you call save() on a Spark DataFrame to write to a Greenplum Database table, the Connector either saves the entire DataFrame to the table, or aborts the write operation.

**Warning**: If the Connector encounters an error during .save(), the target Greenplum table may be left in an inconsistent state.

Table Creation Modes

When you write Spark data to a Greenplum Database table, the Connector creates the table for you if it does not already exist, or when you specify SaveMode.Overwrite and the truncate write option is set to false. Alternatively, you can choose to pre-create the target table.

*When the Connector creates the Greenplum Database table for you*, the Connector:

* Creates the Greenplum Database table with random distribution unless you specify one or more distribution columns with the distributedBy write option. You can specify a single distribution column name, or a comma-separated list of columns names.
* Creates the Greenplum table using the column names defined in the Spark DataFrame (for example, "flt\_month"). Note that the Spark column name reflects any transformations you may have performed on the Spark data. For example, if you used the avg() method to average the data in a column named depdelayminutes, the Spark column name is literally avg(depdelayminutes).

You can use the .as() method to provide a different name for the column. For example: .avg("depdelayminutes").as("depdelaymins").

* Specifies the column names in double-quotes. This renders the column names case-sensitive to Greenplum Database.
* Creates the Geenplum table column with the NOT NULL clause when the Spark DataFrame column is not nullable.
* Creates the table columns in the order that they are defined in the Spark DataFrame.
* Maps the data type of each Spark column to the equivalent Greenplum data type.

*If you choose to pre-create the target Greenplum Database table*, take the following into consideration:

* Use Greenplum table columns names that match the column names defined in the Spark DataFrame that you write. Keep in mind that the Connector treats column names as case-sensitive.
* Select a data type for each Greenplum table column that is equivalent to the data type of the related Spark DataFrame column. Refer to [Spark to Greenplum Data Type Mapping](https://docs.vmware.com/en/VMware-Tanzu-Greenplum-Connector-for-Apache-Spark/2.1/tanzu-greenplum-connector-spark/GUID-reference-datatype_mapping.html#dtm_write) for data type mapping information.
* If a Spark data column contains NULL values, ensure that you do not specify the NOT NULL clause when you create the associated Greenplum Database table column.
* You may create the Greenplum Database table with a subset of the columns in the Spark DataFrame, or with a different column ordering. The Greenplum table must not include columns that are not present in the Spark DataFrame.
* Ensure that you assign the Greenplum Database user/role the permissions required to write to the table. The user may also require permission to (re-)create the table. Refer to [Role Privileges](https://docs.vmware.com/en/VMware-Tanzu-Greenplum-Connector-for-Apache-Spark/2.1/tanzu-greenplum-connector-spark/GUID-install_cfg.html#cfggp_priv) for information about configuring Greenplum Database privileges.

Example - Transferring Data Between Greenplum Database and Spark (Python)

添加到库

[RSS](https://docs.vmware.com/en/VMware-Tanzu-Greenplum-Connector-for-Apache-Spark/rss.xml)

反馈

更新时间 2022年07月12日

选择的产品版本：**VMware Tanzu Greenplum Connector for Apache Spark 2.1**

This example utilizes the public *Airline On-Time Statistics and Delay Cause* data set. This data set records flights by date, airline, originating and destination airports, and many other flight details.

In this example, you:

* Follow [Greenplum Database tutorials](http://greenplum.org/gpdb-sandbox-tutorials/) to load the flight record data set into Greenplum Database.
* Use the pyspark shell and the VMware Tanzu Greenplum Connector for Apache Spark to read a fact table from Greenplum Database into Spark.
* Perform transformations and actions on the data within Spark.
* Write transformed Spark data into a new Greenplum Database table.

Prerequisites

Before starting this exercise, ensure that you are able to:

* Access your Greenplum Database and Spark clusters
* Identify your Greenplum Database master node hostname or IP address and port
* Identify your Greenplum Database user/role name and password
* Identify the absolute path to the Connector JAR file on your system

Procedure 1: Read from Greenplum Database

Perform the following procedure to load flight record data into Greenplum Database, read this data into Spark, and use Spark to transform and view the table data.

**Greenplum Database Operations**

1. Log in to your Greenplum Database master node and set up your environment. For example:
2. $ ssh gpadmin@<gpmaster>
3. gpadmin@gpmaster$ . /usr/local/greenplum-db/greenplum\_path.sh
4. Download and unpack the flight record example data set:
5. gpadmin@gpmaster$ git clone https://github.com/greenplum-db/gpdb-sandbox-tutorials.git
6. gpadmin@gpmaster$ cd gpdb-sandbox-tutorials
7. gpadmin@gpmaster$ tar zxf faa.tar.gz
8. Perform the following exercises in the Greenplum Database tutorial in the specified order:
   1. [Create Users and Roles](http://greenplum.org/gpdb-sandbox-tutorials/create-users-roles/)
   2. [Create and Prepare Database](http://greenplum.org/gpdb-sandbox-tutorials/create-prepare-database/)
   3. [Create Tables](http://greenplum.org/gpdb-sandbox-tutorials/create-tables/)
   4. [Data Loading](http://greenplum.org/gpdb-sandbox-tutorials/data-loading/)
9. When you complete Step 3, you will have created:
   1. A user named user2.
   2. A Greenplum Database database named tutorial.
   3. A schema named faa.
   4. Several Greenplum Database tables in the faa schema and loaded them with flight data.
10. Identify your Greenplum Database user/role name and password:

If you are performing this exercise in the Greenplum Sandbox VM, you will use the Connector as user2 (password pivotal). If you are performing this exercise in your own Greenplum Database instance, you may have a different user name.

1. Assign required privileges to the Greenplum Database user to access the tutorial database, faa schema, and relevant tables. For example, if your Greenplum Database user name is user2, the Greenplum Database administrator would execute the commands:
2. gpadmin@gpmaster$ psql -d tutorial
3. tutorial=# GRANT USAGE, CREATE ON SCHEMA faa TO user2;
4. tutorial=# GRANT SELECT ON faa.otp\_c TO user2;
5. tutorial=# ALTER USER user2 CREATEEXTTABLE(type = 'writable', protocol = 'gpfdist');
6. tutorial=# \q
7. Verify that the flight data loaded correctly:
   1. Connect to the tutorial database as user user2. Recall that the password for user2 is pivotal:
   2. gpadmin@gpmaster$ psql -d tutorial -U user2
   3. Password for user user2:
   4. List all of the tables in the tutorial database schema named faa:
   5. tutorial=> \dt faa.\*
   6. List of relations
   7. Schema | Name | Type | Owner | Storage
   8. --------+----------------------+-------+---------+----------------------
   9. faa | d\_airlines | table | user1 | heap
   10. faa | d\_airports | table | user1 | heap
   11. faa | d\_cancellation\_codes | table | user1 | heap
   12. faa | d\_delay\_groups | table | user1 | heap
   13. faa | d\_distance\_groups | table | user1 | heap
   14. faa | d\_wac | table | user1 | heap
   15. faa | faa\_load\_errors | table | user1 | heap
   16. faa | faa\_otp\_load | table | gpadmin | heap
   17. faa | otp\_c | table | gpadmin | append only columnar
   18. faa | otp\_c\_1\_prt\_mth\_1 | table | gpadmin | append only columnar
   19. faa | otp\_c\_1\_prt\_mth\_10 | table | gpadmin | append only columnar
   20. faa | otp\_c\_1\_prt\_mth\_11 | table | gpadmin | append only columnar
   21. ...
   22. (27 rows)

The \dt output lists 27 tables when the flight data was correctly loaded.

* 1. Examine the definition of the table named otp\_c:
  2. tutorial=> \d faa.otp\_c
  3. Append-Only Columnar Table "faa.otp\_c"
  4. Column | Type | Modifiers
  5. ----------------------+------------------+-----------
  6. flt\_year | smallint |
  7. flt\_quarter | smallint |
  8. flt\_month | smallint |
  9. flt\_dayofmonth | smallint |
  10. flt\_dayofweek | smallint |
  11. flightdate | date |
  12. uniquecarrier | text |
  13. airlineid | integer |
  14. carrier | text |
  15. flightnum | text |
  16. origin | text |
  17. origincityname | text |
  18. originstate | text |
  19. originstatename | text |
  20. dest | text |
  21. destcityname | text |
  22. deststate | text |
  23. deststatename | text |
  24. crsdeptime | text |
  25. deptime | integer |
  26. depdelay | double precision |
  27. depdelayminutes | double precision |
  28. departuredelaygroups | smallint |
  29. taxiout | smallint |
  30. wheelsoff | text |
  31. wheelson | text |
  32. taxiin | smallint |
  33. crsarrtime | text |
  34. arrtime | text |
  35. arrdelay | double precision |
  36. arrdelayminutes | double precision |
  37. arrivaldelaygroups | smallint |
  38. cancelled | smallint |
  39. cancellationcode | text |
  40. diverted | smallint |
  41. crselapsedtime | integer |
  42. actualelapsedtime | double precision |
  43. airtime | double precision |
  44. flights | smallint |
  45. distance | double precision |
  46. distancegroup | smallint |
  47. carrierdelay | smallint |
  48. weatherdelay | smallint |
  49. nasdelay | smallint |
  50. securitydelay | smallint |
  51. lateaircraftdelay | smallint |
  52. Checksum: t
  53. Number of child tables: 17 (Use \d+ to list them.)
  54. Distributed by: (uniquecarrier, flightnum)
  55. Partition by: (flightdate)

The table named otp\_c is a column-oriented, partitioned fact table.

**Spark Operations**

1. Open a terminal window and log in to your Spark client node:
2. $ ssh user@<spark-client>
3. user@spark-client$
4. Construct the JDBC connection string URL to access Greenplum Database. For example, this JDBC connection string accesses a database named tutorial using the Greenplum Database master host gpmaster.domain at the default connection port:
5. jdbc:postgresql://gpmaster.domain/tutorial

Save this URL string, you will use it in an upcoming step.

1. Start pyspark. Replace <gsc-jar> with the full path to your Connector JAR file:
2. user@spark-client$ pyspark --jars <gsc-jar>
3. < ... pyspark startup output messages ... >
4. >>>

You enter the pyspark interactive python shell.

1. Prepare to read the otp\_c table into Spark. In a text editor, construct a map of read options for the greenplum data source. You want to load the Greenplum Database table named otp\_c in the schema named faa, specifying airlineid as the partition column. For example, if you are the user user2 with password pivotal:
2. gscPythonOptions = {
3. "url": "jdbc:postgresql://gpmaster.domain/tutorial",
4. "user": "user2",
5. "password": "pivotal",
6. "dbschema": "faa",
7. "dbtable": "otp\_c",
8. "partitionColumn": "airlineid"
9. }
10. Copy/paste the options map to your pyspark shell terminal window, and enter return to submit the command. For example:
11. >>> gscPythonOptions = {
12. ... "url": "jdbc:postgresql://gpmaster.domain/tutorial",
13. ... "user": "user2",
14. ... "password": "pivotal",
15. ... "dbschema": "faa",
16. ... "dbtable": "otp\_c",
17. ... "partitionColumn": "airlineid"
18. ... } <return>
19. >>>
20. Load the data from the Greenplum Database table otp\_c into a Spark DataFrame, providing the options that you constructed. For example:
21. >>> gpdf = spark.read.format("greenplum").options(\*\*gscPythonOptions).load()

The Greenplum Database table is not actually loaded until you perform an action on the returned DataFrame.

1. Print the schema of the Greenplum Database table:
2. >>> gpdf.printSchema()
3. root
4. |-- flt\_year: short (nullable = true)
5. |-- flt\_quarter: short (nullable = true)
6. |-- flt\_month: short (nullable = true)
7. |-- flt\_dayofmonth: short (nullable = true)
8. |-- flt\_dayofweek: short (nullable = true)
9. |-- flightdate: date (nullable = true)
10. |-- uniquecarrier: string (nullable = true)
11. |-- airlineid: integer (nullable = true)
12. |-- carrier: string (nullable = true)
13. |-- flightnum: string (nullable = true)
14. |-- origin: string (nullable = true)
15. |-- origincityname: string (nullable = true)
16. |-- originstate: string (nullable = true)
17. |-- originstatename: string (nullable = true)
18. |-- dest: string (nullable = true)
19. |-- destcityname: string (nullable = true)
20. |-- deststate: string (nullable = true)
21. |-- deststatename: string (nullable = true)
22. |-- crsdeptime: string (nullable = true)
23. |-- deptime: integer (nullable = true)
24. |-- depdelay: double (nullable = true)
25. |-- depdelayminutes: double (nullable = true)
26. |-- departuredelaygroups: short (nullable = true)
27. |-- taxiout: short (nullable = true)
28. |-- wheelsoff: string (nullable = true)
29. |-- wheelson: string (nullable = true)
30. |-- taxiin: short (nullable = true)
31. |-- crsarrtime: string (nullable = true)
32. |-- arrtime: string (nullable = true)
33. |-- arrdelay: double (nullable = true)
34. |-- arrdelayminutes: double (nullable = true)
35. |-- arrivaldelaygroups: short (nullable = true)
36. |-- cancelled: short (nullable = true)
37. |-- cancellationcode: string (nullable = true)
38. |-- diverted: short (nullable = true)
39. |-- crselapsedtime: integer (nullable = true)
40. |-- actualelapsedtime: double (nullable = true)
41. |-- airtime: double (nullable = true)
42. |-- flights: short (nullable = true)
43. |-- distance: double (nullable = true)
44. |-- distancegroup: short (nullable = true)
45. |-- carrierdelay: short (nullable = true)
46. |-- weatherdelay: short (nullable = true)
47. |-- nasdelay: short (nullable = true)
48. |-- securitydelay: short (nullable = true)
49. |-- lateaircraftdelay: short (nullable = true)

Compare this Spark output with that of the Greenplum Database \d faa.otp\_c command that you invoked earlier. Note that the Greenplum Database data type names differ from those of Spark. For example, the distancegroup column is of Greenplum Database type smallint, while the Spark data type is short. For detailed information about how the Connector maps data types between Greenplum Database and Spark, refer to the [Greenplum Database → Spark Data Type Mapping](https://docs.vmware.com/en/VMware-Tanzu-Greenplum-Connector-for-Apache-Spark/2.1/tanzu-greenplum-connector-spark/GUID-reference-datatype_mapping.html) documentation.

1. Use the .count() method to count the number of rows loaded:
2. >>> gpdf.count()
3. 1024552
4. Use the .select() and .filter() methods to show the origin city, month, and carrier of all flights cancelled in the month of December. Order the results by airline ID and origin city.

You must cast the filter constants to smallint to enable predicate pushdown. This is required because the Greenplum table cancelled and flt\_month columns were created with type smallint.

Copy/paste the command:

>>> gpdf.select("origincityname", "flt\_month", "airlineid", "carrier") \

.filter("cancelled = CAST(1 as SMALLINT)") \

.filter("flt\_month = CAST(12 as SMALLINT)") \

.orderBy("airlineid", "origincityname") \

.show()

+---------------+---------+---------+-------+

| origincityname|flt\_month|airlineid|carrier|

+---------------+---------+---------+-------+

| Detroit, MI| 12| 19386| NW|

| Detroit, MI| 12| 19386| NW|

| Milwaukee, WI| 12| 19386| NW|

|Minneapolis, MN| 12| 19386| NW|

| Phoenix, AZ| 12| 19386| NW|

| Houston, TX| 12| 19393| WN|

| Houston, TX| 12| 19393| WN|

| Las Vegas, NV| 12| 19393| WN|

| Las Vegas, NV| 12| 19393| WN|

| Manchester, NH| 12| 19393| WN|

| Omaha, NE| 12| 19393| WN|

| Phoenix, AZ| 12| 19393| WN|

| San Jose, CA| 12| 19393| WN|

| Tampa, FL| 12| 19393| WN|

| Washington, DC| 12| 19393| WN|

| Anchorage, AK| 12| 19704| CO|

| Anchorage, AK| 12| 19704| CO|

| Austin, TX| 12| 19704| CO|

| Houston, TX| 12| 19704| CO|

| Houston, TX| 12| 19704| CO|

+--------------------+---------+---------+-------+

only showing top 20 rows

1. Use the .groupBy(), .agg(), and avg() methods to identify the average departure delay for each day of the week, sorting by the day of the week with the .sort() method. The avg() method is located in the pyspark.sql.functions module; you must first import this module:
2. >>> from pyspark.sql.functions import avg
3. >>> gpdf.groupBy("flt\_dayofweek").agg(avg("depdelayminutes")).sort("flt\_dayofweek").show()
4. +-------------+--------------------+
5. |flt\_dayofweek|avg(depdelayminutes)|
6. +-------------+--------------------+
7. | 1| 14.738491569779914|
8. | 2| 11.237272024020244|
9. | 3| 11.198198256252295|
10. | 4| 12.056892575385985|
11. | 5| 12.455024249521957|
12. | 6| 12.69586361271813|
13. | 7| 14.818271192603715|
14. +-------------+--------------------+
15. Use the like() method to display the cancelled flights for the month of December for all origin cities whose name starts with the letters Mi. Copy/paste the command:
16. >>> gpdf.select("origincityname", "destcityname", "flightnum", "carrier", "airlineid", "flt\_month") \
17. .filter("cancelled = CAST(1 as SMALLINT)") \
18. .filter("flt\_month = CAST(12 as SMALLINT)") \
19. .filter(gpdf.origincityname.like("Mi%")) \
20. .orderBy("origincityname", "destcityname") \
21. .show()
22. +--------------------+--------------------+---------+-------+---------+---------+
23. | origincityname| destcityname|flightnum|carrier|airlineid|flt\_month|
24. +--------------------+--------------------+---------+-------+---------+---------+
25. | Miami, FL| Chicago, IL| 846| AA| 19805| 12|
26. | Miami, FL|Greensboro/High P...| 4197| MQ| 20398| 12|
27. | Miami, FL| Washington, DC| 1068| AA| 19805| 12|
28. | Milwaukee, WI| Baltimore, MD| 817| FL| 20437| 12|
29. | Milwaukee, WI| Denver, CO| 5838| OO| 20304| 12|
30. | Milwaukee, WI| Memphis, TN| 3799| 9E| 20363| 12|
31. | Milwaukee, WI| Minneapolis, MN| 7177| NW| 19386| 12|
32. | Milwaukee, WI| Newark, NJ| 2504| OO| 20304| 12|
33. | Minneapolis, MN| Atlanta, GA| 1073| DL| 19790| 12|
34. | Minneapolis, MN| Grand Forks, ND| 4003| 9E| 20363| 12|
35. | Minneapolis, MN| Houston, TX| 2399| XE| 20374| 12|
36. | Minneapolis, MN| Lincoln, NE| 3798| 9E| 20363| 12|
37. | Minneapolis, MN| Miami, FL| 925| AA| 19805| 12|
38. | Minneapolis, MN| Milwaukee, WI| 7165| NW| 19386| 12|
39. | Minneapolis, MN| Rapid City, SD| 3931| 9E| 20363| 12|
40. |Mission/Mcallen/E...| Memphis, TN| 4066| 9E| 20363| 12|
41. | Missoula, MT| Salt Lake City, UT| 4461| OO| 20304| 12|
42. +--------------------+--------------------+---------+-------+---------+---------+
43. Exit the pyspark shell:
44. >>> quit()

Procedure 2: Write from Spark to Greenplum Database

Perform the following procedure to write Spark data that you transformed in the previous procedure into a new Greenplum Database table.

**Note**: This procedure assumes that you have completed Procedure 1 of this example and have retained the example runtime environment.

**Greenplum Database Operations**

1. Locate your Greenplum Database terminal window.
2. Assign the Greenplum privileges required to write to a Greenplum Database table. For example, if your Greenplum Database user name is user2, the Greenplum Database administrator would execute the commands:
3. gpadmin@gpmaster$ psql -d tutorial
4. tutorial=# ALTER USER user2 CREATEEXTTABLE(type = 'readable', protocol = 'gpfdist');
5. You will return to the Greenplum terminal window at the end of this procedure.

**Spark Operations**

1. Identify the average departure delay for each day of the week with the statement you specified in Procedure 1, this time saving, rather than displaying, the DataFrame. Assign the data to a variable named delaydf:
2. >>> delaydf = gpdf.groupBy("flt\_dayofweek").agg(avg("depdelayminutes")).sort("flt\_dayofweek")
3. Copy/paste the following options map setting to your pyspark shell terminal window, and enter return to submit the command. For example:
4. gscPythonWriteOptions = {
5. "url": "jdbc:postgresql://gpmaster.domain/tutorial",
6. "user": "user2",
7. "password": "pivotal",
8. "dbschema": "faa",
9. "dbtable": "avgdelay"
10. }
11. Write delaydf to the Greenplum Database table named avgdelay. If the table does not exist, the Connector will create the table for you before loading the data. Specify the Append SaveMode to instruct the Connector to append the data to the table if it already exists.
12. >>> delaydf.write.format("greenplum").options(\*\*gscPythonWriteOptions).mode("Append").save()
13. Exit the pyspark shell:
14. >>> quit()

**Greenplum Database Operations**

1. Run psql and connect to Greenplum Database as user user2:
2. gpadmin@gpmaster$ psql -d tutorial -U user2
3. Examine the schema of Greenplum Database table avgdelay:
4. tutorial=> \d+ faa.avgdelay
5. Table "faa.avgdelay"
6. Column | Type | Modifiers | Storage | Description
7. ----------------------+------------------+-----------+---------+-------------
8. flt\_dayofweek | smallint | | plain |
9. avg(depdelayminutes) | double precision | | plain |
10. Has OIDs: no
11. Distributed by: (flt\_dayofweek)
12. Examine the table contents:
13. tutorial=> SELECT \* FROM faa.avgdelay ORDER BY flt\_dayofweek;
14. flt\_dayofweek | avg(depdelayminutes)
15. ---------------+----------------------
16. 1 | 14.7384915697799
17. 2 | 11.2372720240202
18. 3 | 11.1981982562523
19. 4 | 12.056892575386
20. 5 | 12.455024249522
21. 6 | 12.6958636127181
22. 7 | 14.8182711926037
23. (7 rows)

The table contents are slightly different than that displayed for the DataFrame in the pyspark shell. The Greenplum Database double precision data type holds 15 digits, while Spark utilizes 17 digits.