Transfer data between SQL and Spark pool in Azure Synapse Analytics

Note:

You are not required to complete the processes, tasks, activities, or steps presented in this example. The various samples provided are for illustrative purposes only and it's likely that if you try this out you will encounter issues in your system.
In this section, you will learn about using Azure Active Directory to transfer data to and from an Apache Spark pool and a dedicated SQL pool attached within the workspace you have created for your Azure Synapse Analytics account.

If you're using the notebook experience from the Azure Synapse Studio environment linked to your workspace resource, you don't have to use import statements. Import statements are only required when you don't go through the integrated notebook experience.

It is important that the Constants and the SqlAnalyticsConnector are set up as shown below:

```
#scala
 import com.microsoft.spark.sqlanalytics.utils.Constants
 import org.apache.spark.sql.SqlAnalyticsConnector._
```

To read data from a dedicated SQL pool, you should use the Read API. The Read API works for Internal tables (Managed Tables) and External Tables in the dedicated SQL pool.

The Read API using Azure AD looks as follows:

```
#scala
val df = spark.read.sqlanalytics("<DBName>.<Schema>.<TableName>")
```

The parameters it takes in are:

- **DBName**: the name of the database.
- **Schema**: the schema definition such as dbo.
- **TableName**: the name of the table you want to read data from.

To write data to a dedicated SQL Pool, you should use the Write API. The Write API creates a table in the dedicated SQL pool. Then, it invokes Polybase to load the data into the table that was created. One thing to keep in mind is that the table can't already exist in the dedicated SQL pool. If that happens, you'll receive an error stating: "There is already an object named..."

The Write API using Azure Active Directory (Azure AD) looks as follows:

```
1
df.write.sqlanalytics("<DBName>.<Schema>.<TableName>", <TableType>)
```

The parameters it takes in are:

- **DBName**: the name of the database.
- **Schema**: the schema definition such as dbo.
- **TableName**: the name of the table you want to read data from.
- **TableType**: specification of the type of table, which can have two values: Constants.INTERNAL - Managed table in dedicated SQL pool Constants.EXTERNAL - External table in dedicated SQL pool

The TableType parameter in the Write API has some extra parameters to consider as mentioned above.

An example of a SQL pool-managed table looks as follows:

```
1
df.write.sqlanalytics("<DBName>.<Schema>.<TableName>", <TableType>)
```

To use a SQL pool external table, you need to have an EXTERNAL DATA SOURCE and an EXTERNAL FILE FORMAT that exists on the pool using the following examples:

```
1
2
3
4
5
6
7
8
9
10
11
12
--For an external table, you need to pre-create the data source and file format
in dedicated SQL pool using SQL queries:
CREATE EXTERNAL DATA SOURCE <DataSourceName>
WITH
  ( LOCATION = 'abfss://...' ,
    TYPE = HADOOP
  ) ;

CREATE EXTERNAL FILE FORMAT <FileFormatName>
WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
```

```
);
```

It is not necessary to create an EXTERNAL CREDENTIAL object if you are using Azure AD pass-through authentication from the storage account. The only thing you need to keep in mind is that you need to be a member of the "Storage Blob Data Contributor" role on the storage account.

The next step is to use the df.write command within Scala with DATA_SOURCE, FILE_FORMAT, and the sqlanalytics command in a similar way to writing data to an internal table.

The example is shown below:

```
df.write.
    option(Constants.DATA_SOURCE, <DataSourceName>).
    option(Constants.FILE_FORMAT, <FileFormatName>).
    sqlanalytics("<DBName>.<Schema>.<TableName>", Constants.EXTERNAL)
```