towardsdatascience.com

3 Ways To Create Tables With Apache Spark - Towards Data Science

Antonello Benedetto

10–13 minutes

Introduction

Apache Spark is a distributed data processing engine that allows you to create two main types of tables:

1. **Managed (or Internal) Tables:** for these tables, Spark manages both the data and the metadata. In particular, data is usually saved in the Spark SQL warehouse directory - that is the *default* for managed tables - whereas metadata is saved in a meta-store of relational entities (including *databases, tables, temporary views*) and can be accessed through an interface known as the "catalog".
2. **Unmanaged (or External) Tables:** for these tables, Spark only manages the metadata, but requires you to specify the exact location where you wish to save the table or, alternatively, the source directory from which data will be pulled to create a table.

   Moreover, because of their different purpose:

- **if you delete a managed table**, Spark will delete both the table data in the warehouse and the metadata in the meta-store,

meaning that you will neither be able to query the table directly or to retrieve data into it.

- **if you delete an unmanaged table**, Spark will just delete the metadata, meaning that you won't be able to query the table anymore, as your query won't be resolved against the catalog in the execution analysis phase; but you will still find the tables you created in the external location.

In this tutorial, I will share three methods to create managed and unmanaged tables and explain the cases when it make sense to use one or the other.

Initial Dataset Manipulation

If you wish to follow along, but are relatively new to Spark and don't have a better option, I would strongly suggest to use Databrick's community edition as it gives you access to a cluster with 15GB of memory and 2 Cores to execute Spark code.

The `sales_redords` dataset I am going to use, is quite big (600MB) as it includes 5 million rows and 14 columns - you can download it here. I chose a sizable dataset to - at least partially - replicate the volume of data you will have to deal with in the real world.

Because the dataset comes in a semi-structured CSV format, in order to create a DataFrame in your SparkSession, make sure to upload the original file in the `/FileStore/` directory in the DataBricks File System (DBFS) first, then run the following code:

To simulate the cleaning process that raw data would undergo as part of a daily ETL pipeline, let's suppose that you wished to:

- Convert the original column names to lowercase and replace blanks " " with and underscore "_";

- Convert the original *"Order Date"* from `STRING` to `DATE` and the original *"Units Sold", "Unit Price*" and *"Total Revenue"* from `STRING` to `FLOAT` ;
- Drop the following columns as they are not required by your stakeholders: [*"Region", "Country", "Order Priority", "Ship Date", "Total Profit", "Total Cost", "Unit Cost"*];
- Remove duplicates in the original "*Order ID*" field.

  This can be achieved executing the code below, that creates a new DataFrame named `df_final` that only includes 7 columns and 2M rows.(*first 5 rows are also displayed*):

```
+---------+----------+---------------+-------------+----------+----------+-------------+
| order_id|order_date|      item_type|sales_channel|units_sold|unit_price|total_revenue|
+---------+----------+---------------+-------------+----------+----------+-------------+
|954955518|2014-01-19|         Snacks|       Online|    1414.0|    152.58|    215748.12|
|970755660|2019-04-26|         Cereal|      Offline|    7027.0|     205.7|    1445453.9|
|309317338|2012-03-03|Office Supplies|       Online|    2729.0|    651.21|    1777152.1|
|380507028|2010-04-18|      Beverages|       Online|    9340.0|     47.45|     443183.0|
|504055583|2015-01-08|         Cereal|       Online|     103.0|     205.7|      21187.1|
+---------+----------+---------------+-------------+----------+----------+-------------+
only showing top 5 rows
```

  The `df_final` has been manipulated to obtain the preferred result and it's now ready to be used as a source to create tables.

  However, in order to show you all 3 different methods, I also had to make `df_final` available as a **temporary view** and as **parquet file** (split in 2 partitions) in the DBFS :

Create Managed Tables

As mentioned, when you create a *managed* table, Spark will manage both the table data and the metadata (*information about the table itself*). In particular data is written to the default Hive warehouse, that is set in the `/user/hive/warehouse` location. You can change this behavior, using the `spark.sql.warehouse.dir` configuration while generating a

`SparkSession`.

METHOD #1

The most straightforward way to create a managed table is to write the `df_final` through the Structured API `saveAsTable()` method, without specifying any paths:

You can check that the command successfully created a permanent table named `salesTable_manag1` with `tableType = 'MANAGED'` by running:

spark.catalog.listTables()Out[1]:
[Table(name='salestable_manag1', database='default', description=None, tableType='MANAGED', isTemporary=False)

You should prefer this method in most cases, as its syntax is very compact and readable and avoids you the additional step of creating a temp view in memory.

METHOD #2

An alternative way to create a managed table is to run a SQL command that queries all the records in the temp `df_final_View`:

In this case I used the `%sql` magic in Databricks to run a SQL command directly, without wrapping it into `spark.sql()`. However, you can achieve the exact same result with the syntax:

spark.sql("CREATE TABLE IF NOT EXISTS salesTable_manag2 AS SELECT * FROM df_final_View")

If you have a SQL background, this method is probably the most familiar, as you don't need to bother with the "standard" Structured API syntax and can even perform additional

manipulation on the fly. However, while working with big data, you should take into account the extra space required to create a temp view on your cluster.

METHOD #3

The last method you can use, is similar to the previous one, but it involves two steps as you first create a the table `salesTable_manag3` and then insert data into it by querying the temp view:

You should opt for this method when you wish to change the column types, or if you already created a table and want to replace or append data into it instead of deleting it and start from scratch.

Create Unmanaged Tables

Unmanaged tables provide much more flexibility, as the table data can be stored in a location of your choice, or the table can be built directly on top of data available in an external directory. In turn, this means that in Spark, a location is *mandatory* for external tables. Metadata is again saved in the meta-store and accessible through the catalog.

> *Unmanaged* tables provide much more flexibility, as the table data can be stored in a location of your choice, or the table can be built directly on top of data available in an external directory.

In the example below, I am going to use *Databricks File System* to to simulate an external location with respect to the default Spark SQL warehouse, but of course, it is possible to save unmanaged tables (or create them on top of) every file system compatible with Spark, including ***cloud data warehouses***.

METHOD #1

To create an unmanaged (external) table you can simply specify a path before the `saveAsTable()` method:

When you run this code, Spark will:

- Shuffle data in the `df_final` DataFrame to create 2 partitions and write these to the `/FileStore/tables/salesTable_unmanag1` directory.
- Create an external table named `salesTable_unmanag1` using the partitions stored at that location and save relevant information in the meta-store.

METHOD #2

As similar result can be obtained by specifying the location as part of a SQL query. In this case your will need to use the temp view as a data source:

Also remember to use the `CREATE EXTERNAL TABLE` syntax instead of `CREATE TABLE` .

METHOD #3

Finally, if the data you intend to use to create the table, is already available in an external location, you can simply build the table on the top of it by pointing to the location through the `USING format OPTIONS (path 'path to location')` syntax:

▸ (1) Spark Jobs

	order_id	order_date	item_type	sales_channel	units_sold	unit_price	total_revenue
1	834403312	2017-02-28	Cosmetics	Online	7337	437.2	3207736.5
2	928538978	2013-10-08	Snacks	Offline	4173	152.58	636716.3
3	629185569	2015-06-11	Meat	Online	8847	421.89	3732460.8
4	847138869	2018-01-19	Cereal	Online	6072	205.7	1249010.4
5	206882274	2020-05-07	Vegetables	Offline	9917	154.06	1527813
6	782844817	2018-09-17	Office Supplies	Online	33	651.21	21489.93

7	115908181	2016-08-09	Snacks	Offline	8661	152.58	1321495.4

Showing all 10 rows.

In this case, the table is created on the top of the `df_final.parquet` file that I saved in the `FileStore` after manipulating the original dataset. Note that if you use this method, you will need to specify the type of each column.

If you try to list the tables in the catalog again:

spark.catalog.listTables()

You can see that now the output includes 6 tables of which 3 managed and 3 unmanaged, together with the temp view created at the beginning:

```
Out[9]: [Table(name='salestable_manag1', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='salestable_manag2', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='salestable_manag3', database='default', description=None, tableType='MANAGED', isTemporary=False),
 Table(name='salestable_unmanag1', database='default', description=None, tableType='EXTERNAL', isTemporary=False),
 Table(name='salestable_unmanag2', database='default', description=None, tableType='EXTERNAL', isTemporary=False),
 Table(name='salestable_unmanag3', database='default', description=None, tableType='EXTERNAL', isTemporary=False),
 Table(name='df_final_view', database=None, description=None, tableType='TEMPORARY', isTemporary=True)]
```

Executing the following code will delete metadata from both type of tables and data from managed tables only, whereas data will be preserved in the external locations you specified:

However, after erasing metadata you won't be able to run queries against any of the tables.

When To Use Managed or External Tables?

By now, you should have a solid grasp of the differences between these two types of table and should be ready to apply the code above to your specific use cases. However, perhaps you are still a bit confused when it comes to choose between one of the two, while working on a real project.

In general, you should store data in a **MANAGED table**:

1. **When you wish to use Spark as a database to perform ad hoc or interactive queries to explore and visualize data sets** → *for instance, you could devise an ETL pipeline in Spark that*

eventually stores data in a managed table and then use a JDBC-ODBC connector to query this table via Looker, Tableau, Power BI and other BI Tools.

2. **When you are working on a project and wish to temporarily save data in Spark for additional manipulation or testing, before writing it to the final location** → *for instance, managed tables could be handy while building, training and evaluating machine learning models in Spark, as they remove the need for an external storage to save partial iterations.*
3. **When you wish for Spark to take care of the complete lifecycle of the table data including its deletion or are concerned about security in the external file system** → *if data does not need to be shared with other clients immediately or there are security concerns, then saving data in the Spark warehouse could be a valid temporary solution.*
4. **When you are not worried about data reproducibility** → *if data can easily be retrieved from other sources or the computational effort that takes to transform it is not too extensive, then the risk of erroneously drop a managed table is more contained.*

You should instead store data in an **EXTERNAL** table:

1. **When you cannot create a table based on an existing DataFrame or view already available in your SparkSession, that comes with an inferred schema** → *in this case you must provide a preferred location and specify the correct field types.*
2. **When Spark is mainly employed to process large data sets in parallel, by distributing them across a cluster, or to implement end-to-end data pipelines through batches or streams**→ *in this case Spark is generally used for the*

computational heavy lifting, but data is eventually stored into an external data-lake or written to the final cloud data warehouse.

3. **When data needs to remain in the specified external location even after deleting the unmanaged table in Spark**→ *this is often the case when there are multiple tables or schemas that are built on the top of the same data and you don't wish to jeopardize their integrity.*
4. **When Spark should not own data lifecycle, like controlling settings, directories and schemas** → *for example you might already have another solution in place to cover these tasks.*