Delta Lake Introduction with Examples [using Pyspark]

Ansab Iqbal · Follow

6 min read · Aug 20, 2023

Listen Share More

What is Delta lake

In the yesteryears of data management, data warehouses reigned supreme with their structured storage and optimized querying. However, these warehouses struggled when confronted with the deluge of unstructured and semi-structured data, revealing their limitations. This void led to the emergence of data lakes, offering a promising solution by accommodating diverse data types without immediate structure. Yet, the initial euphoria surrounding data lakes gave way to challenges of maintaining data integrity, ensuring consistency, and handling updates and deletions effectively.

Enter Delta Lake, a technological evolution that seeks to address the shortcomings of traditional data warehouses and data lakes alike. By seamlessly combining ACID transactions and versioned data, Delta Lake acts as a bridge between these two paradigms. In this short write up, we will go through the foundational operations of the Delta format using Python and PySpark.

If you are still unconvinced about why you should take a serious look at Lakehouse architecture and Delta Lake, Databricks has an amazing introductory blog that might change your mind.

What Is a Lakehouse?

Learn more about the new data management paradigm data lakehouses -- its evolution, adoption, common use cases, and its...

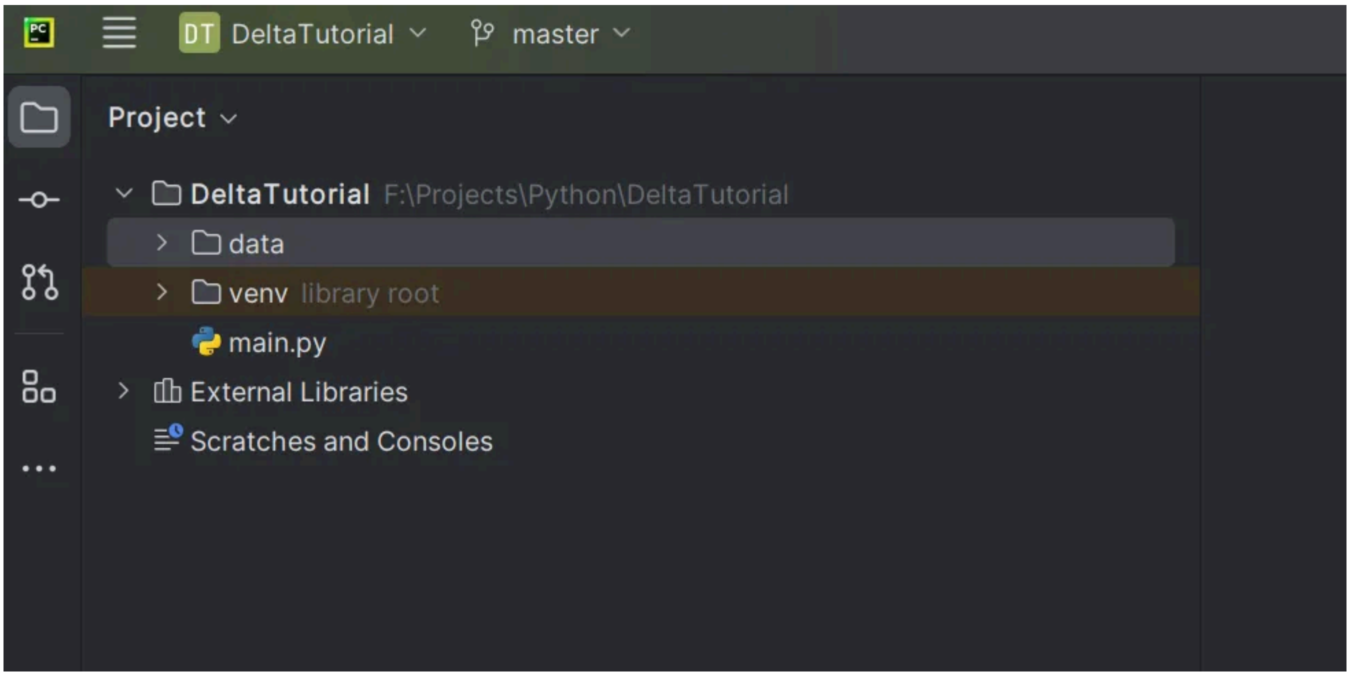
www.databricks.com

Environment setup

Project creation

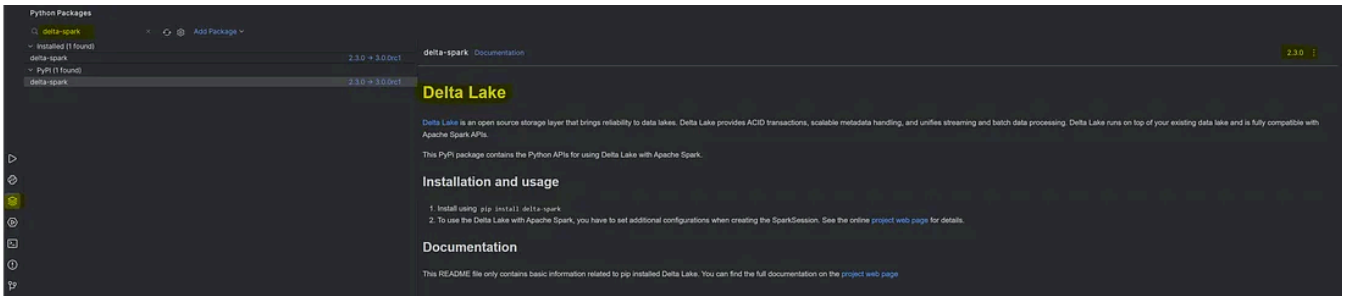
As a prerequisite to following this tutorial you should have PySpark and PyCharm configured on your machine. After doing that Create a simple Python project in the IDE/Code editor of your choice, for the examples I am going to use PyCharm.

After the project is created, create a folder **data** at the root, this will contain our delta format file for the following examples.



Installing Delta Python package

Go to python packages in the PyCharm menu and search for delta-spark package. Look for the compatible delta-spark package version and install. You can look for the compatible delta package version against your spark from this link.


Python Packages
delta-spark
Add Package
Installed (1 found)
delta-spark 2.3.0 → 3.0.0rc1
PyPI (1 found)
delta-spark 2.3.0 → 3.0.0rc1
delta-spark Documentation
2.3.0

Delta Lake

Delta Lake is an open source storage layer that brings reliability to data lakes. Delta Lake provides ACID transactions, scalable metadata handling, and unifies streaming and batch data processing. Delta Lake runs on top of your existing data lake and is fully compatible with Apache Spark APIs.

This PyPi package contains the Python APIs for using Delta Lake with Apache Spark.

Installation and usage

1. Install using `pip install delta-spark`
2. To use the Delta Lake with Apache Spark, you have to set additional configurations when creating the SparkSession. See the online project web page for details.

Documentation

This README file only contains basic information related to pip installed Delta Lake. You can find the full documentation on the project web page

Spark Imports

After that we need to import delta and specify in the spark session

```
import pyspark
from delta import *
from pyspark.sql.types import *
from delta.tables import *
from pyspark.sql.functions import *

#  Create a spark session with Delta
builder = pyspark.sql.SparkSession.builder.appName("DeltaTutorial") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension")
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.cata

# Create spark context
spark = configure_spark_with_delta_pip(builder).getOrCreate()
spark.sparkContext.setLogLevel("ERROR")
```

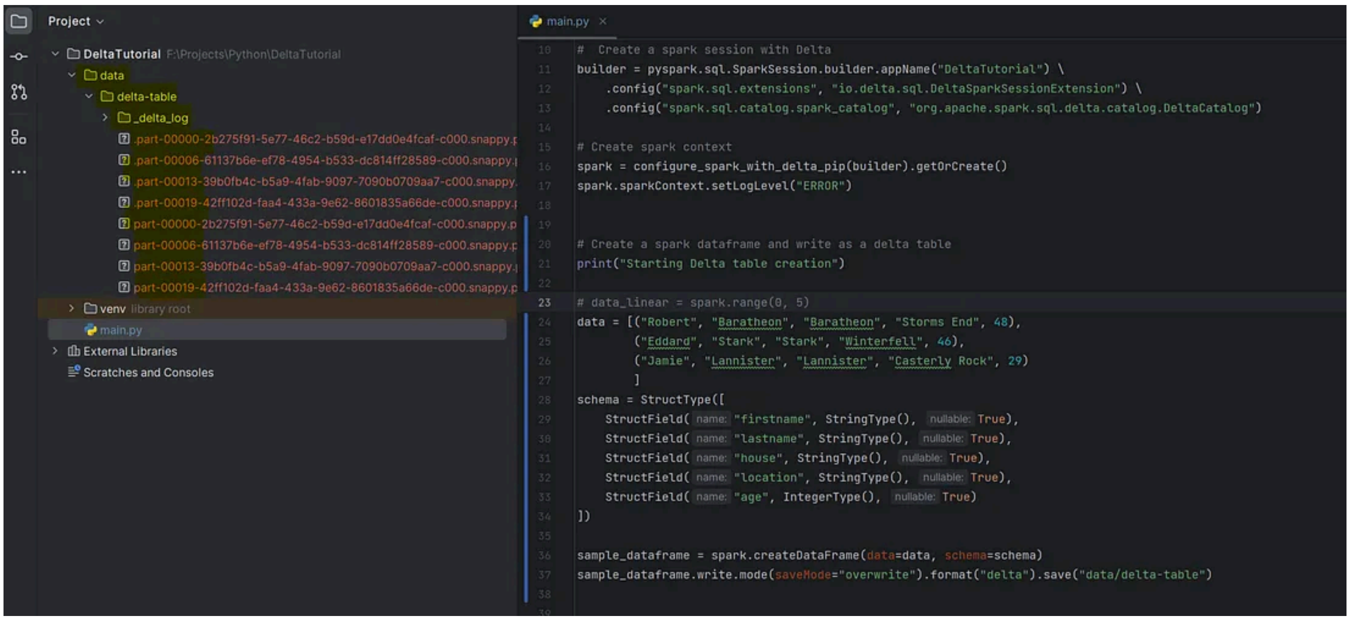
Delta Features

Create a delta table

```
 spark dataframe and write as a delta table
rting Delta table creation")

Robert", "Baratheon", "Baratheon", "Storms End", 48),
Eddard", "Stark", "Stark", "Winterfell", 46),
Jamie", "Lannister", "Lannister", "Casterly Rock", 29)

tructType([
Field("firstname", StringType(), True),
Field("lastname", StringType(), True),
Field("house", StringType(), True),
Field("location", StringType(), True),
Field("age", IntegerType(), True)


aframe = spark.createDataFrame(data=data, schema=schema)
aframe.write.mode(saveMode="overwrite").format("delta").save("data/delta-table")
```

First we first define a spark data frame. In this case we have the schema and data for our favourite characters from Game of thrones. The magic line for storing that dataframe in delta format is the **.format("delta")** and then we specify the location to be the data folder that we created earlier.

As soon as we run the program, we can see the delta log and the snappy parquet files created.



```
DeltaTutorial F:\Projects\Python\DeltaTutorial
  data
    delta-table
      _delta_log
      .part-00000-2b275f91-5e77-46c2-b59d-e17dd0e4fcaf-c000.snappy.p
      .part-00006-61137b6e-ef78-4954-b533-dc814ff28589-c000.snappy.p
      .part-00013-39b0fb4c-b5a9-4fab-9097-7090b0709aa7-c000.snappy.
      .part-00019-42ff102d-faa4-433a-9e62-8601835a66de-c000.snappy.p
      part-00000-2b275f91-5e77-46c2-b59d-e17dd0e4fcaf-c000.snappy.p
      part-00006-61137b6e-ef78-4954-b533-dc814ff28589-c000.snappy.p
      part-00013-39b0fb4c-b5a9-4fab-9097-7090b0709aa7-c000.snappy.
      part-00019-42ff102d-faa4-433a-9e62-8601835a66de-c000.snappy.p
  venv library root
  main.py
External Libraries
Scratches and Consoles
```

```
#  Create a spark session with Delta
builder = pyspark.sql.SparkSession.builder.appName("DeltaTutorial") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog")

# Create spark context
spark = configure_spark_with_delta_pip(builder).getOrCreate()
spark.sparkContext.setLogLevel("ERROR")


# Create a spark dataframe and write as a delta table
print("Starting Delta table creation")

# data_linear = spark.range(0, 5)
data = [("Robert", "Baratheon", "Baratheon", "Storms End", 48),
        ("Eddard", "Stark", "Stark", "Winterfell", 46),
        ("Jamie", "Lannister", "Lannister", "Casterly Rock", 29)
        ]
schema = StructType([
    StructField( name: "firstname", StringType(),  nullable: True),
    StructField( name: "lastname", StringType(),  nullable: True),
    StructField( name: "house", StringType(),  nullable: True),
    StructField( name: "location", StringType(),  nullable: True),
    StructField( name: "age", IntegerType(),  nullable: True)
])

sample_dataframe = spark.createDataFrame(data=data, schema=schema)
sample_dataframe.write.mode(saveMode="overwrite").format("delta").save("data/delta-table")
```

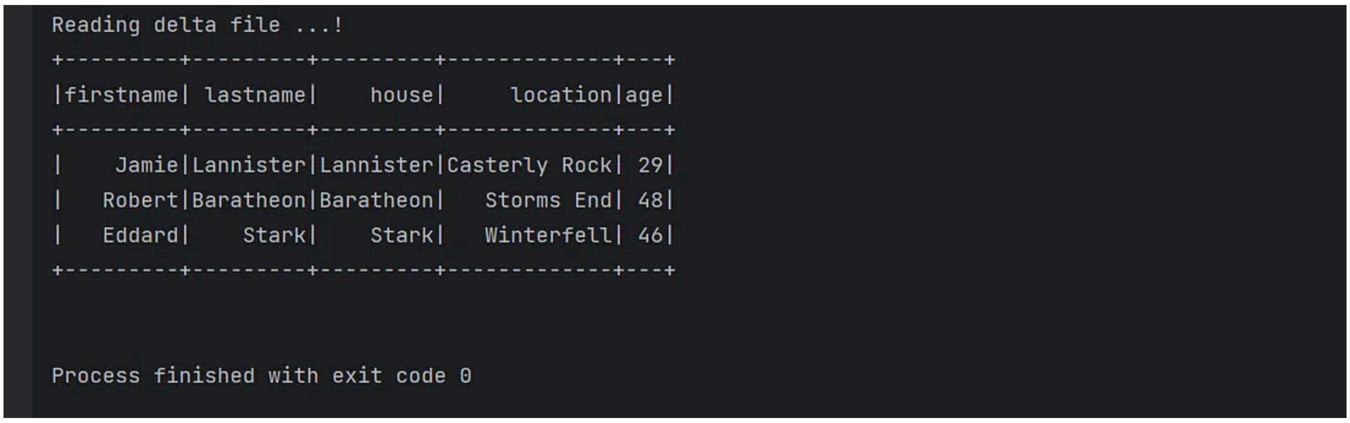
Read a delta table

Reading is as easy as again just specifing the the **.format("delta")** in the spark read api

```
# Read Data
print("Reading delta file ...!")
```

```
got_df = spark.read.format("delta").load("data/delta-table")
got_df.show()
```

and in the console you can see your desired data.



```
Reading delta file ...!
+---------+---------+---------+-------------+---+
|firstname| lastname|    house|     location|age|
+---------+---------+---------+-------------+---+
|    Jamie|Lannister|Lannister|Casterly Rock| 29|
|   Robert|Baratheon|Baratheon|   Storms End| 48|
|   Eddard|    Stark|    Stark|   Winterfell| 46|
+---------+---------+---------+-------------+---+


Process finished with exit code 0
```

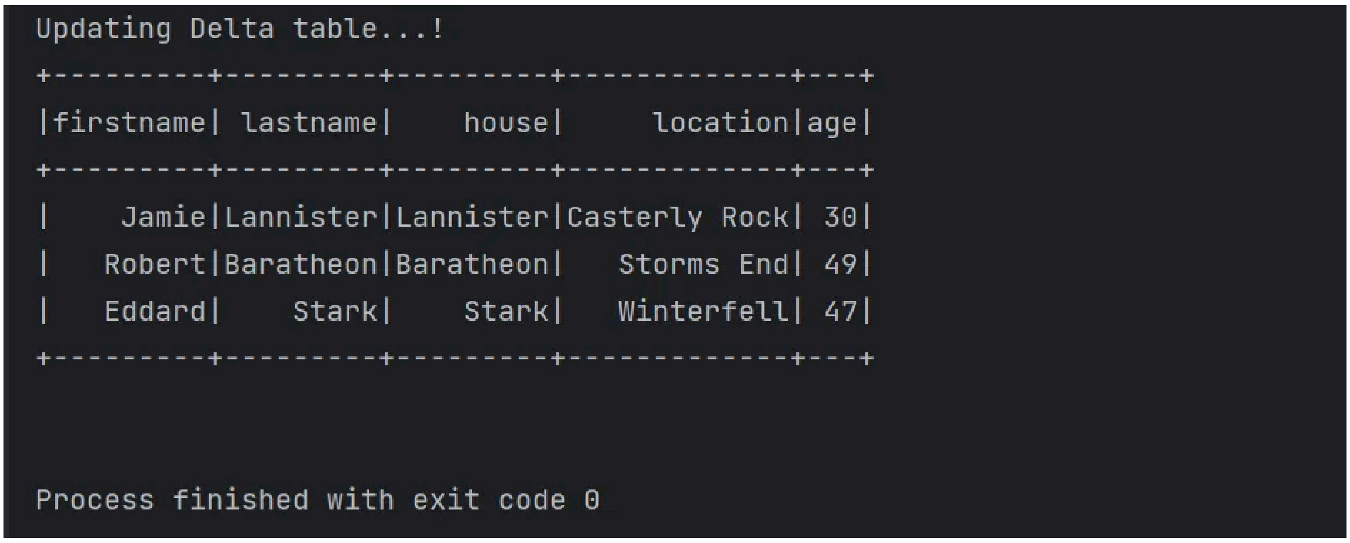
Update a delta table

Overwrite whole table

In case you want to simply overwrite the delta table you can simple provide the **.mode(saveMode="overwrite")** command

```
# Update data
print("Updating Delta table...!")
data = [("Robert", "Baratheon", "Baratheon", "Storms End", 49),
        ("Eddard", "Stark", "Stark", "Winterfell", 47),
        ("Jamie", "Lannister", "Lannister", "Casterly Rock", 30)
        ]
schema = StructType([
    StructField("firstname", StringType(), True),
    StructField("lastname", StringType(), True),
    StructField("house", StringType(), True),
    StructField("location", StringType(), True),
    StructField("age", IntegerType(), True)
])
sample_dataframe = spark.createDataFrame(data=data, schema=schema)
sample_dataframe.write.mode(saveMode="overwrite").format("delta").save("data/de
```

In case we defined a new dataframe and it has different age values for all the people and it reflects when we read the data again.



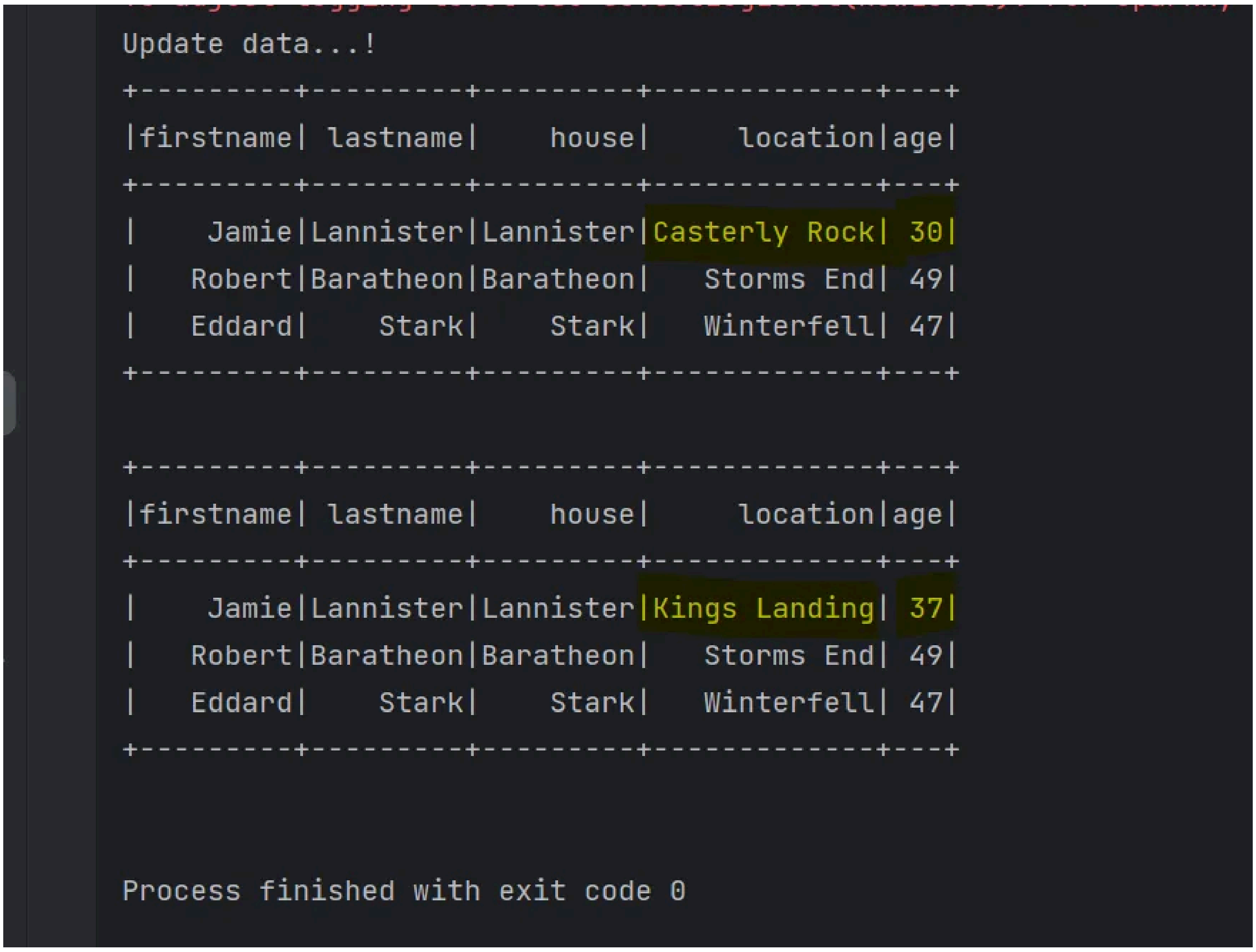
```
Updating Delta table...!
+---------+---------+---------+-------------+---+
|firstname| lastname|    house|     location|age|
+---------+---------+---------+-------------+---+
|    Jamie|Lannister|Lannister|Casterly Rock| 30|
|   Robert|Baratheon|Baratheon|   Storms End| 49|
|   Eddard|    Stark|    Stark|   Winterfell| 47|
+---------+---------+---------+-------------+---+



Process finished with exit code 0
```

Conditional Update

If we want to update a record or few records according to a condition we can simple use the .update method like this

```
# Update data in Delta
print("Update data...!")

# delta table path
deltaTable = DeltaTable.forPath(spark, "data/delta-table")
deltaTable.toDF().show()

deltaTable.update(
    condition=expr("firstname == 'Jamie'"),
    set={"firstname": lit("Jamie"), "lastname": lit("Lannister"), "house": lit(
         "location": lit("Kings Landing"), "age": lit(37)})

deltaTable.toDF().show()
```

In this case we updated the location and age of the records whose firstname was jamie. and we can see the result with before and after of the dataframe console output.



```
Update data...!
+---------+---------+---------+-------------+---+
|firstname| lastname|    house|     location|age|
+---------+---------+---------+-------------+---+
|    Jamie|Lannister|Lannister|Casterly Rock| 30|
|   Robert|Baratheon|Baratheon|   Storms End| 49|
|   Eddard|    Stark|    Stark|   Winterfell| 47|
+---------+---------+---------+-------------+---+


+---------+---------+---------+-------------+---+
|firstname| lastname|    house|     location|age|
+---------+---------+---------+-------------+---+
|    Jamie|Lannister|Lannister|Kings Landing| 37|
|   Robert|Baratheon|Baratheon|   Storms End| 49|
|   Eddard|    Stark|    Stark|   Winterfell| 47|
+---------+---------+---------+-------------+---+


Process finished with exit code 0
```

Upsert a delta table

Upsert is simple a combination of two operations (update and insert hence very intuitively called upsert). In order to upsert records we do something like

```
# Upsert Data
print("Upserting Data...!")
# delta table path
deltaTable = DeltaTable.forPath(spark, "data/delta-table")
deltaTable.toDF().show()

# define new data
data = [("Gendry", "Baratheon", "Baratheon", "Kings Landing", 19),
        ("Jon", "Snow", "Stark", "Winterfell", 21),
        ("Jamie", "Lannister", "Lannister", "Casterly Rock", 36)
        ]
schema = StructType([
    StructField("firstname", StringType(), True),
    StructField("lastname", StringType(), True),
    StructField("house", StringType(), True),
    StructField("location", StringType(), True),
    StructField("age", IntegerType(), True)
])
```

```
newData = spark.createDataFrame(data=data, schema=schema)

deltaTable.alias("oldData") \
    .merge(
    newData.alias("newData"),
    "oldData.firstname = newData.firstname") \
    .whenMatchedUpdate(
    set={"firstname": col("newData.firstname"), "lastname": col("newData.lastna
         "location": col("newData.location"), "age": col("newData.age")}) \
    .whenNotMatchedInsert(
    values={"firstname": col("newData.firstname"), "lastname": col("newData.las
            "location": col("newData.location"), "age": col("newData.age")}) \
    .execute()

deltaTable.toDF().show()
```

First we define a new data frame which has updates to jamie again with his age and then we have two new records for Jon Snow and Gendry Baratheon.

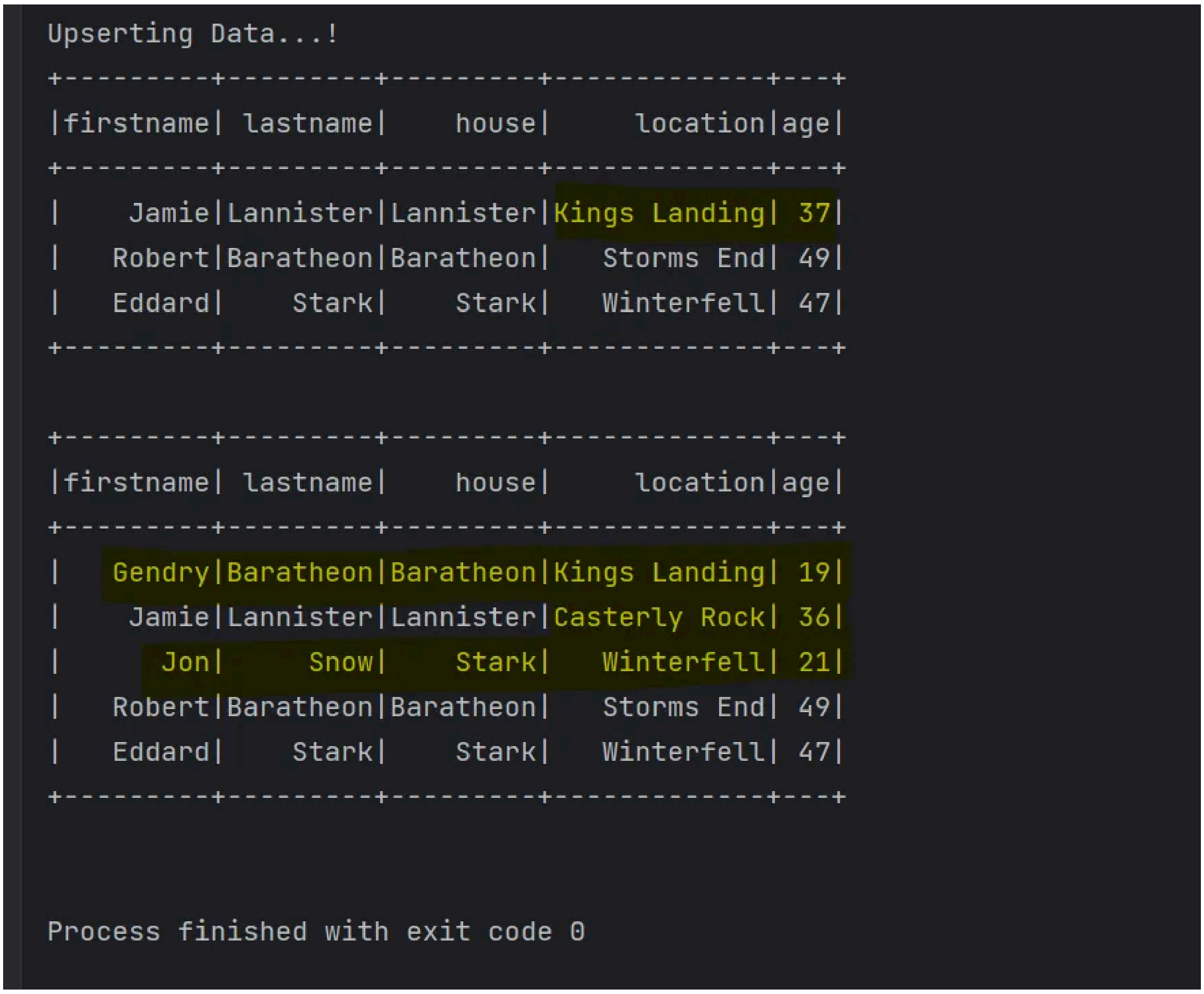
The magic function that we use for upsert is merge. In this case we assign alias to the old and new dataframes and set the rules of what to do if a record mathes with the existing data record. the condition we are looking for is **"oldData.firstname = newData.firstname"**. And if it matches we update everything to the new values

```
.whenMatchedUpdate(
    set={"firstname": col("newData.firstname"), "lastname": col("newData.lastna
         "location": col("newData.location"), "age": col("newData.age")})
```

If it doesn't we insert and execute

```
.whenNotMatchedInsert(
    values={"firstname": col("newData.firstname"), "lastname": col("newData.las
            "location": col("newData.location"), "age": col("newData.age")}) \
    .execute()
```

if we take a look at before and after of our operation on the dataframe, we can clearly see that the records have been upserted correctly.



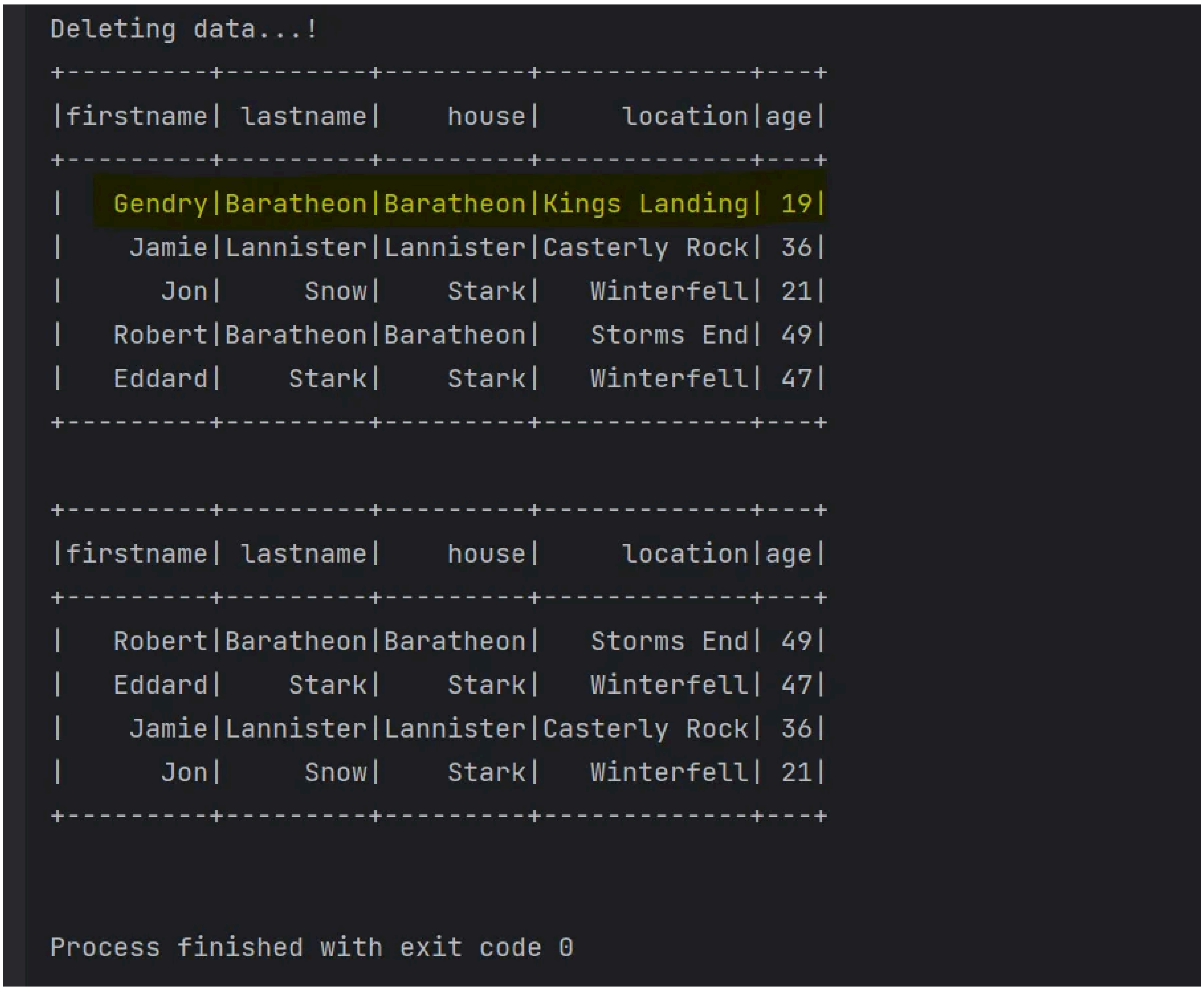
```
Upserting Data...!
+---------+---------+---------+-------------+---+
|firstname| lastname|    house|     location|age|
+---------+---------+---------+-------------+---+
|    Jamie|Lannister|Lannister|Kings Landing| 37|
|   Robert|Baratheon|Baratheon|   Storms End| 49|
|   Eddard|    Stark|    Stark|   Winterfell| 47|
+---------+---------+---------+-------------+---+


+---------+---------+---------+-------------+---+
|firstname| lastname|    house|     location|age|
+---------+---------+---------+-------------+---+
|   Gendry|Baratheon|Baratheon|Kings Landing| 19|
|    Jamie|Lannister|Lannister|Casterly Rock| 36|
|      Jon|     Snow|    Stark|   Winterfell| 21|
|   Robert|Baratheon|Baratheon|   Storms End| 49|
|   Eddard|    Stark|    Stark|   Winterfell| 47|
+---------+---------+---------+-------------+---+



Process finished with exit code 0
```

Delete a delta table

We can also delete a particular record based on filter just like we did for update

```
# Delete Data
print("Deleting data...!")

# delta table path
deltaTable = DeltaTable.forPath(spark, "data/delta-table")
deltaTable.toDF().show()

deltaTable.delete(condition=expr("firstname == 'Gendry'"))

deltaTable.toDF().show()
```

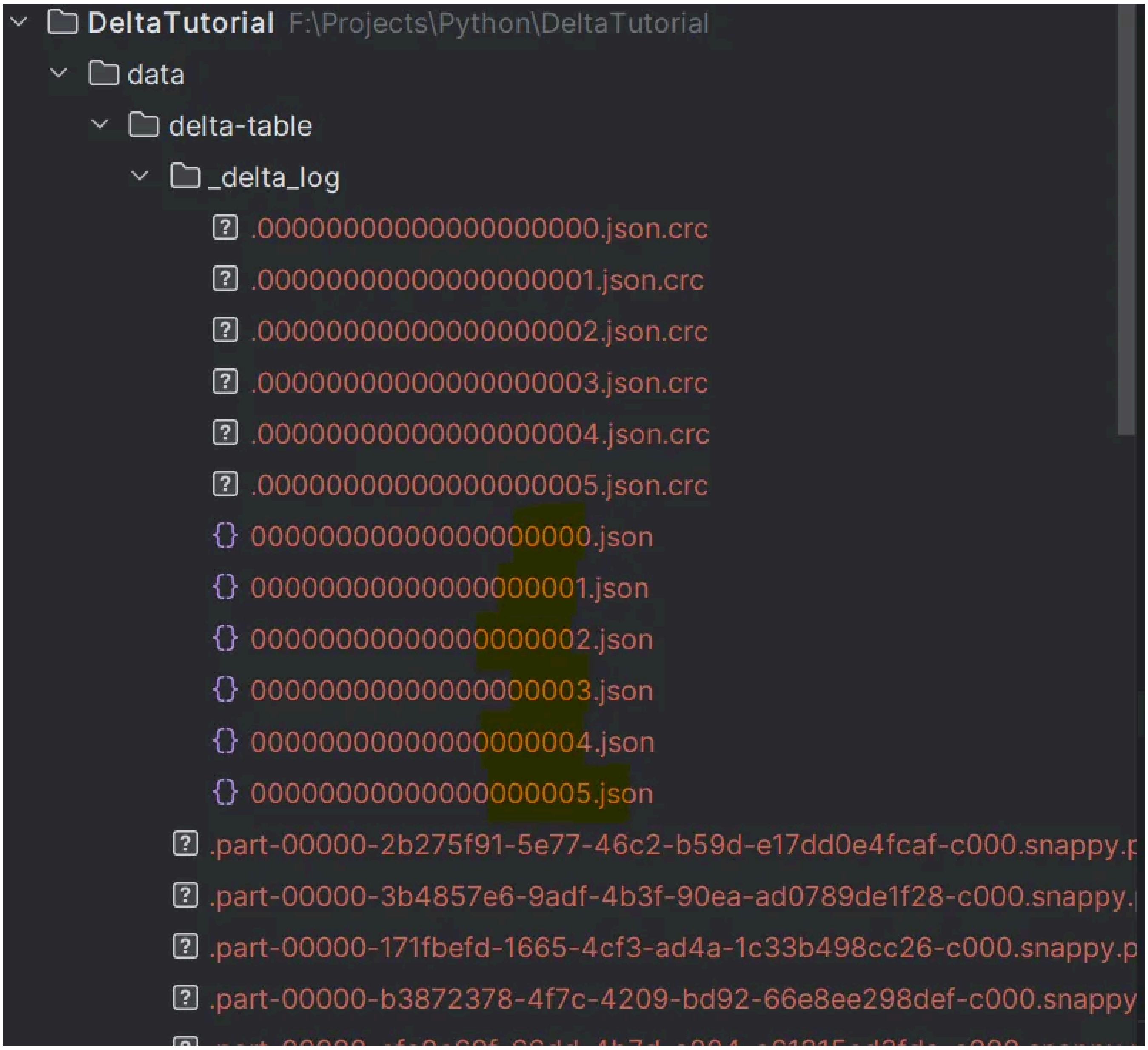
In this case we deleted the record for Gendry and it is reflected in the data frame states before and after.



```
Deleting data...!
+---------+---------+---------+-------------+---+
|firstname| lastname|    house|     location|age|
+---------+---------+---------+-------------+---+
|   Gendry|Baratheon|Baratheon|Kings Landing| 19|
|    Jamie|Lannister|Lannister|Casterly Rock| 36|
|      Jon|     Snow|    Stark|   Winterfell| 21|
|   Robert|Baratheon|Baratheon|   Storms End| 49|
|   Eddard|    Stark|    Stark|   Winterfell| 47|
+---------+---------+---------+-------------+---+


+---------+---------+---------+-------------+---+
|firstname| lastname|    house|     location|age|
+---------+---------+---------+-------------+---+
|   Robert|Baratheon|Baratheon|   Storms End| 49|
|   Eddard|    Stark|    Stark|   Winterfell| 47|
|    Jamie|Lannister|Lannister|Casterly Rock| 36|
|      Jon|     Snow|    Stark|   Winterfell| 21|
+---------+---------+---------+-------------+---+



Process finished with exit code 0
```

Read Historic data for Delta Table

Delta lake also allows you to read differnt historic versions of the data. the version history is stored in the _delta_log folder. we can inspect it to exactly know the kind of operation that happened on that point in time

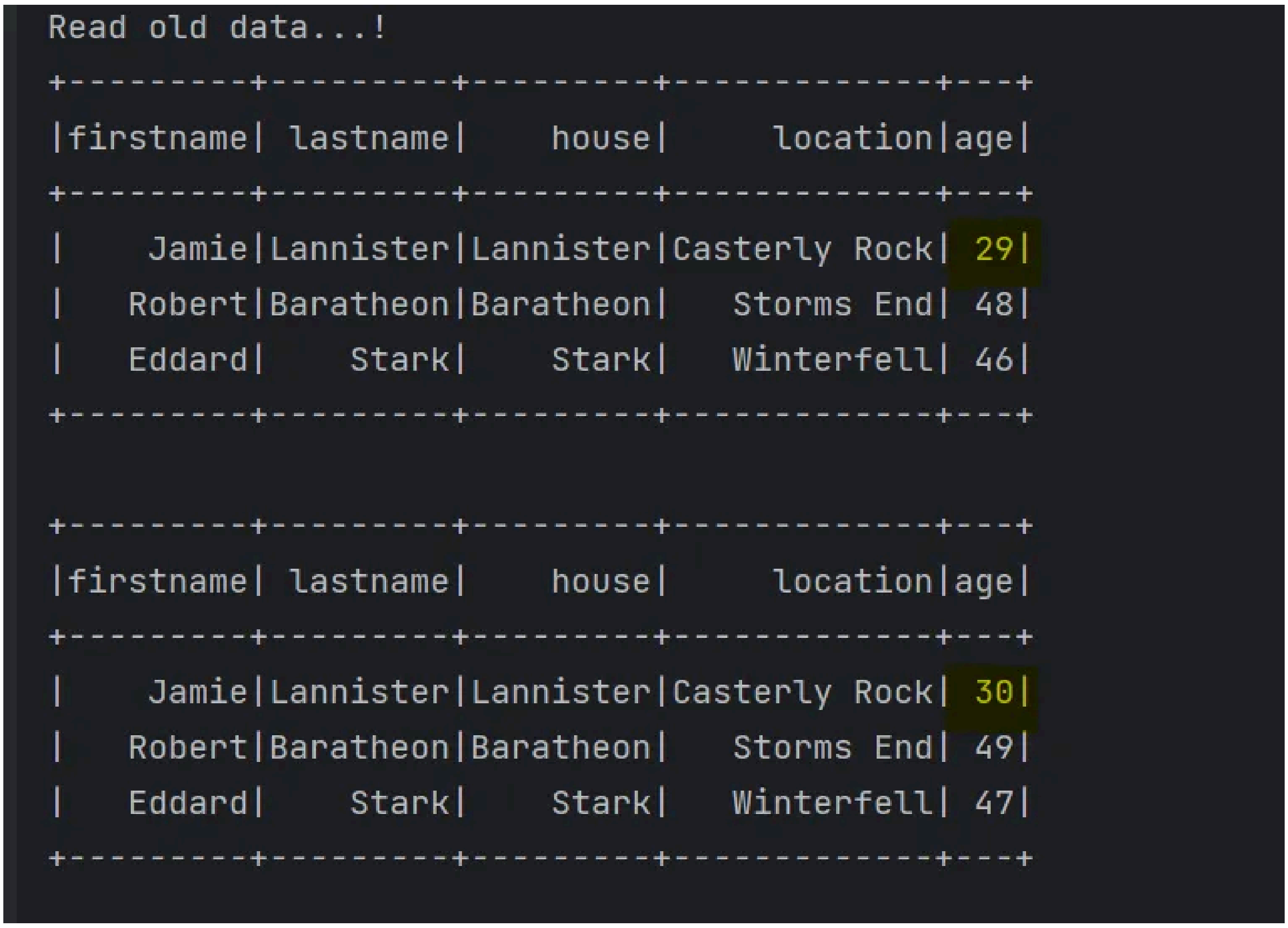


```
DeltaTutorial F:\Projects\Python\DeltaTutorial
  data
    delta-table
      _delta_log
        .00000000000000000000.json.crc
        .00000000000000000001.json.crc
        .00000000000000000002.json.crc
        .00000000000000000003.json.crc
        .00000000000000000004.json.crc
        .00000000000000000005.json.crc
        00000000000000000000.json
        00000000000000000001.json
        00000000000000000002.json
        00000000000000000003.json
        00000000000000000004.json
        00000000000000000005.json
      .part-00000-2b275f91-5e77-46c2-b59d-e17dd0e4fcaf-c000.snappy.p
      .part-00000-3b4857e6-9adf-4b3f-90ea-ad0789de1f28-c000.snappy.
      .part-00000-171fbefd-1665-4cf3-ad4a-1c33b498cc26-c000.snappy.p
      .part-00000-b3872378-4f7c-4209-bd92-66e8ee298def-c000.snappy
```

In order to read the data we can specify versions and read like a normal dataframe.

```
# Reading Older version of Data
print("Read old data...!")

df_versionzero = spark.read.format("delta").option("versionAsOf", 0).load("data
df_versionzero.show()

df_versionzone = spark.read.format("delta").option("versionAsOf", 1).load("data
df_versionzone.show()
```

Since here we specified the first and the second version.



```
Read old data...!
+---------+---------+---------+--------------+---+
|firstname| lastname|    house|      location|age|
+---------+---------+---------+--------------+---+
|    Jamie|Lannister|Lannister|Casterly Rock| 29|
|   Robert|Baratheon|Baratheon|   Storms End| 48|
|   Eddard|    Stark|    Stark|   Winterfell| 46|
+---------+---------+---------+--------------+---+


+---------+---------+---------+--------------+---+
|firstname| lastname|    house|      location|age|
+---------+---------+---------+--------------+---+
|    Jamie|Lannister|Lannister|Casterly Rock| 30|
|   Robert|Baratheon|Baratheon|   Storms End| 49|
|   Eddard|    Stark|    Stark|   Winterfell| 47|
+---------+---------+---------+--------------+---+
```

We can see the data updated in the console.

There is a lot more that Delta lake and Lakehouse offers than we covered here. Please check out the official documentation which has a lot of easy to grasp examples.

https://docs.delta.io/latest/delta-intro.html

Happy Hacking !

References

What Is a Lakehouse?

Learn more about the new data management paradigm data lakehouses -- its evolution, adoption, common use cases, and its…

www.databricks.com

Open in app ↗

Search

Z

Data Engineering Data Science Software Development Data Data Lake

Follow

Written by Ansab Iqbal

33 Followers

Software/Data Engineer, passionate about Data and ML solutions, Write about anything that might make a difference. All opinions my own.

More from Ansab Iqbal

Ansab Iqbal

Setting Up Apache Spark/PySpark on Windows 11 machine

Apache Spark

5 min read · Feb 18, 2023

45 5

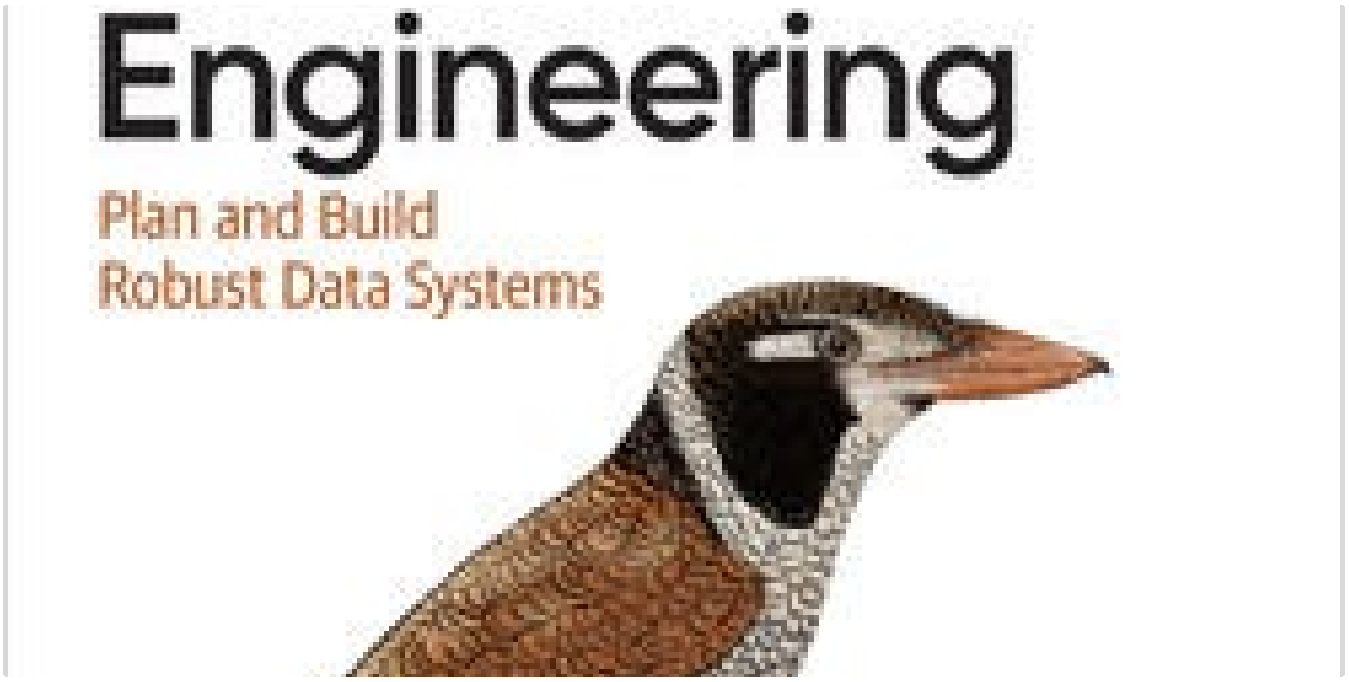
Ansab Iqbal

Introduction to NATs [Part 1] : A lightweight messaging platform

NATS is a highly performant, open-source messaging system designed for building distributed systems and microservices architectures. It...

3 min read · Aug 5, 2023

7



Ansab Iqbal

Best Data Engineering books

Data Engineering is a rapidly growing field and with new technologies, frameworks and libraries coming at an alarming rate specially in the...

2 min read · Jan 19, 2023

2

Ansab Iqbal

Running PySpark in IntelliJ PyCharm | Windows 11

In order to use PySpark in PyCharm IDE, you need to follow the following steps.

2 min read · Feb 19, 2023

1

See all from Ansab Iqbal

Recommended from Medium

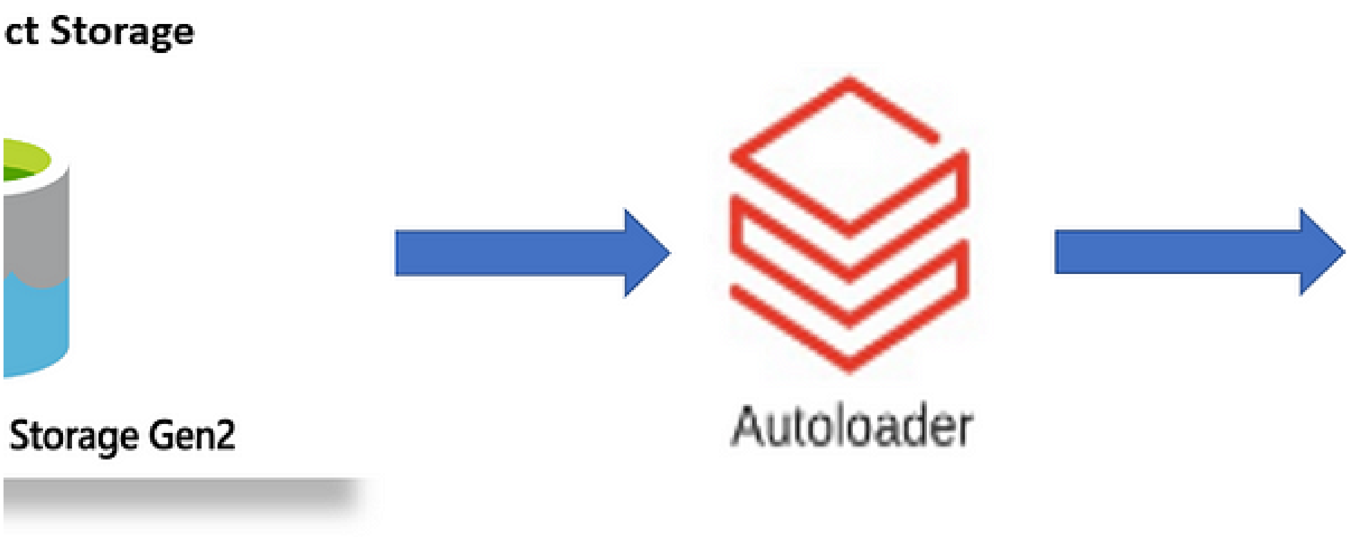
Muqtada Hussain Mohammed

Efficient Change Data Capture (CDC) on Databricks Delta Tables with Spark

In today's data-driven applications, organizations face a critical challenge: ensuring near-real-time data aggregation and accuracy for...

5 min read · Oct 20, 2023

72



Shubhodaya Hampiholi

Streaming Data Ingestion with Databricks Auto Loader

Use case: As part of out Data Ingestion framework we wanted to adapt to a robust, scalable and reusable ingestion mechanism which can cater...

5 min read · Nov 27, 2023

1

Lists

Predictive Modeling w/ Python

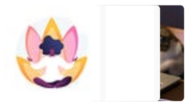
20 stories · 1060 saves

Coding & Development

11 stories · 538 saves

General Coding Knowledge

20 stories · 1076 saves



Stories to Help You Grow as a Software Developer

19 stories · 950 saves

ISHAN PRADHAN

How to read a .snappy.parquet file in databricks

In Databricks, learn how to read .snappy.parquet files of your delta tables.

5 min read · Oct 11, 2023

6

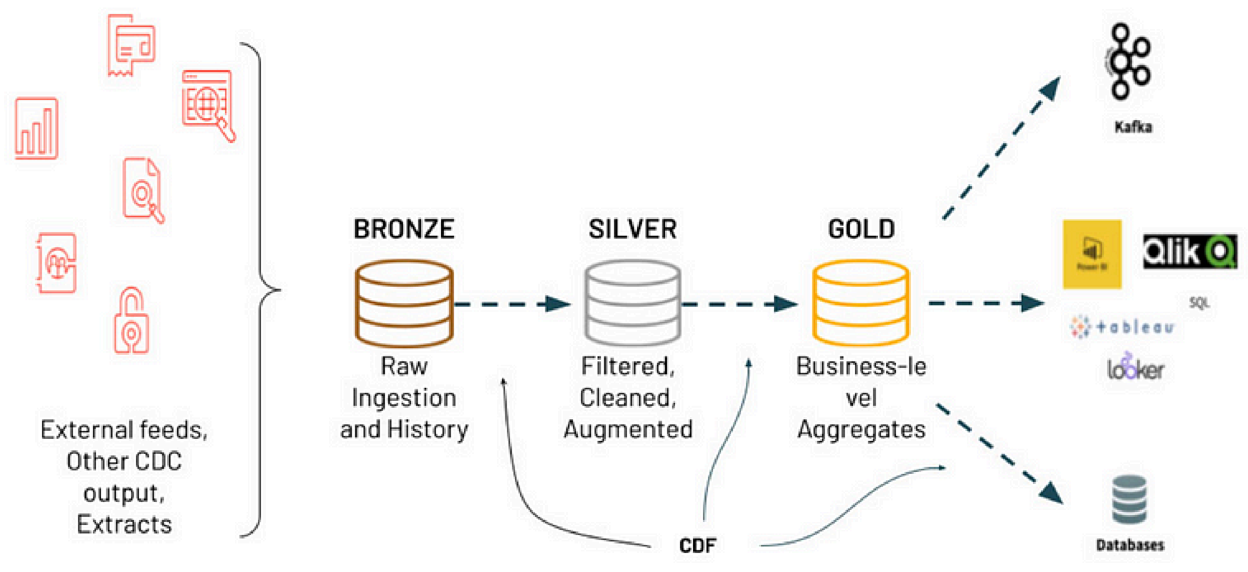
Nick Hass

How to Connect Local PySpark to AWS S3 and Read a Delta Table

While you could use AWS EMR and automatically have access to the S3 file system, you can also connect Spark to your S3 file system on your…

2 min read · Oct 4, 2023

5


External feeds, Other CDC output, Extracts
BRONZE
Raw Ingestion and History
SILVER
Filtered, Cleaned, Augmented
GOLD
Business-level Aggregates
CDF
Kafka
SQL
tableau
looker
Databases

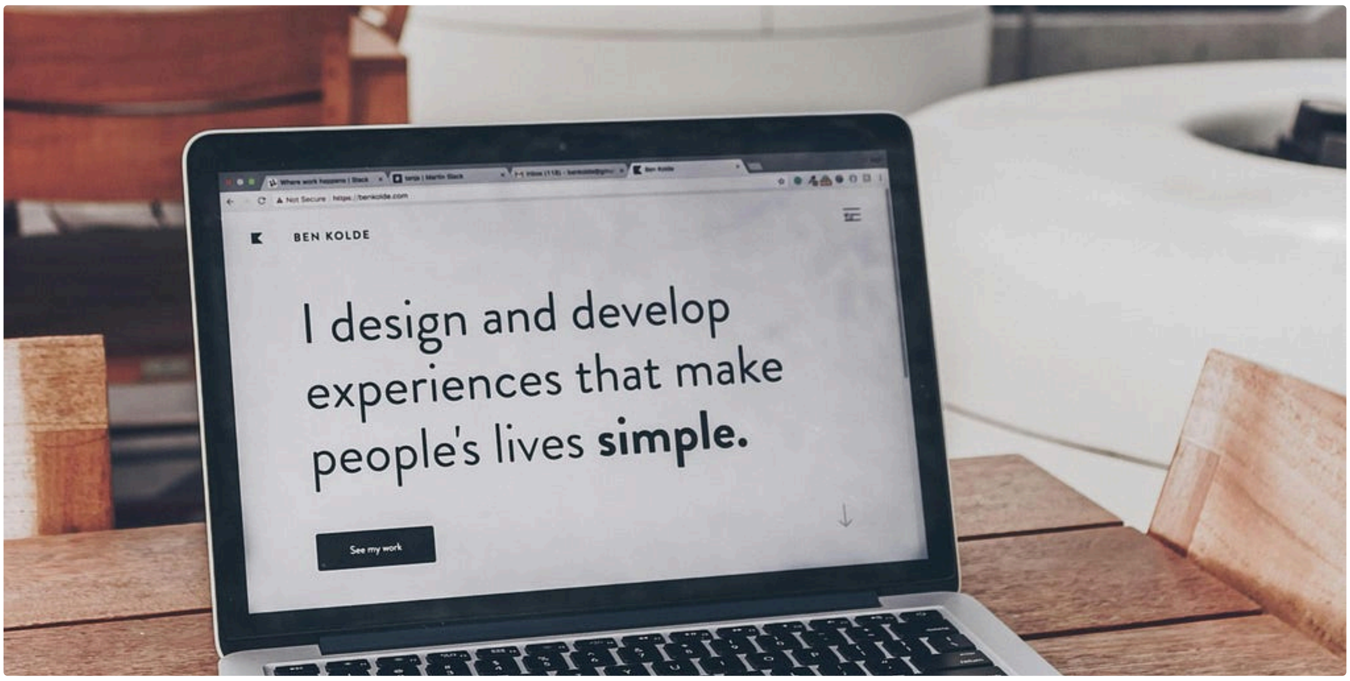
Matthew Salminen

Optimize your Delta Tables & ETLs with Change Data Feed (CDF) in Databricks

After explaining what Delta Live Tables are and then going in depth on how we can record data source changes of those tables with Change…

· 7 min read · Oct 8, 2023

112


BEN KOLDE
I design and develop experiences that make people's lives simple.
See my work

Rahul Madhani in Data Engineer Things

4 Proven Methods to Download Files from Databricks Locally

A Detailed Guide to Fast and Effective File Downloads from Databricks to Your Local Machine with examples and code

4 min read · Feb 20, 2024

52

See more recommendations