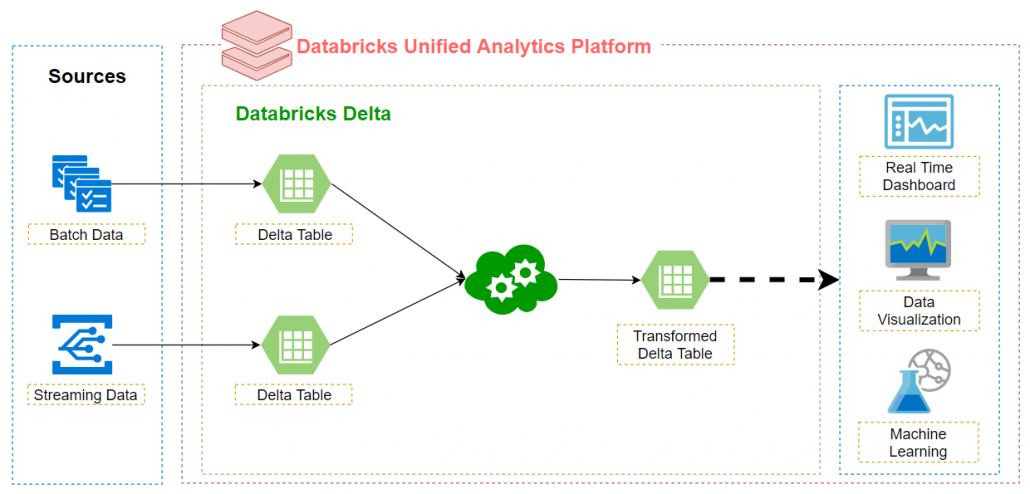
**Databricks Delta Lake**

In this blog, we are going to talk about the Azure Databricks Delta Lake.

**What is 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.



**Benefits of Delta Lake**

* ACID Transactions: Delta guarantees that all readers and writers are working with consistent data, even in highly transactional environments.
* Schema Enforcement: Automatically handles schema variations to prevent insertion of bad records during ingestion.
* Upserts and Deletes: Supports merge, update and delete operations to enable complex use cases like change-data-capture, slowly-changing-dimension (SCD) operations, streaming upserts, and so on.
* Scalable Metadata Handling: Delta improves on one of the most common issues with Data Lakes by managing metadata for billions of records at scale.
* Time Travel: Delta can store historical data and automatically query relevant data for any requested time period.
* Snapshot Isolation – Ensures that multiple writers can write to a dataset simultaneously without interfering with jobs that are reading the dataset.

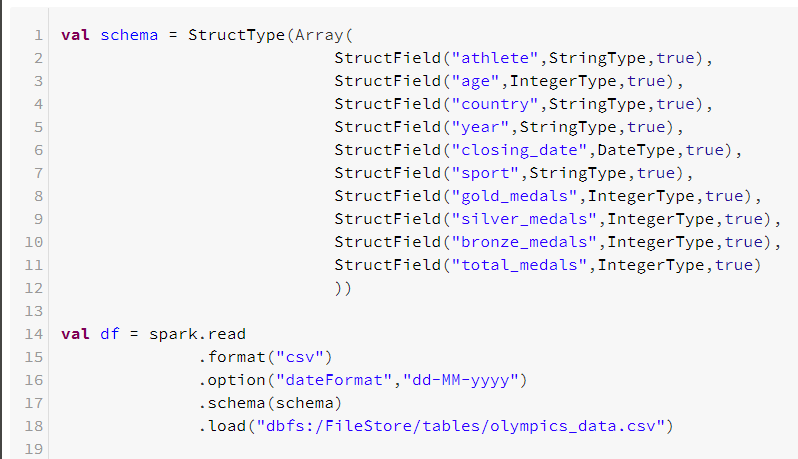
**Delta Lake with Spark**

* As we all knows that Apache Spark is a just processing engine. It doesn't have its own storage system and metadata store.
* For all these requirements it relies on some other systems. For storage we've multiple options such as HDFS, AWS S3, Azure Blob Storage etc...
* We can run spark on yarn, mesos and Kubernetis also.
* Spark allows us to create database objects such as tables and views. These things requires a meta store and spark relies on a hive meta store for this purpose.
* This is one of the main reason why spark never provided some of the most essential features of a reliable data processing system such as atomic api(s) and acid transactions.
* If we want to provide acid properties, we need to place an intermediary service between apache spark and storage layer. This is what the Delta Lake is doing.
* Delta Lake plays an intermediary service between apache spark and storage layer.
* Instead of directly interacting with the storage layer our programs talk to the Delta Lake for reading and writing data.
* Now the responsibility of all ACID transactions are taken care by Delta Lake.
* The underlying storage system could be anything like HDFS, AWS S3 or Azure Blob Storage.
* All we need to make sure that we have the correct version of Delta Lake that supports our underlying storage system.

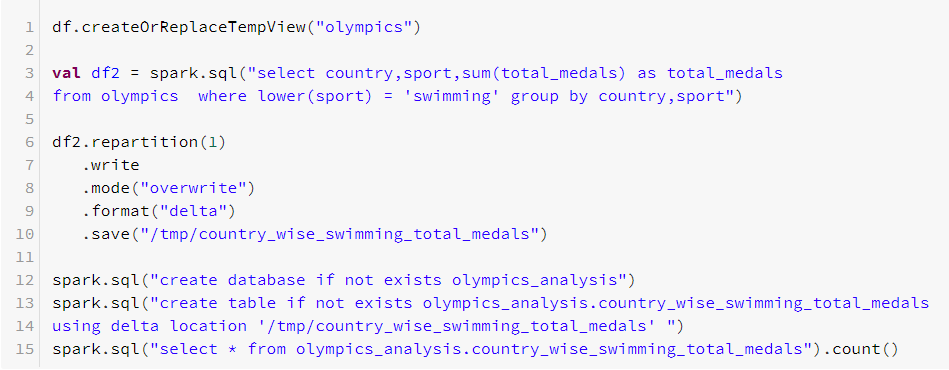
**How to create and load data into Delta Lake Table**

Let’s create delta table and load some data into it and see what happened internally. For this we’ll be working with Scala and Spark.

Creating a DataFrame by loading a csv data



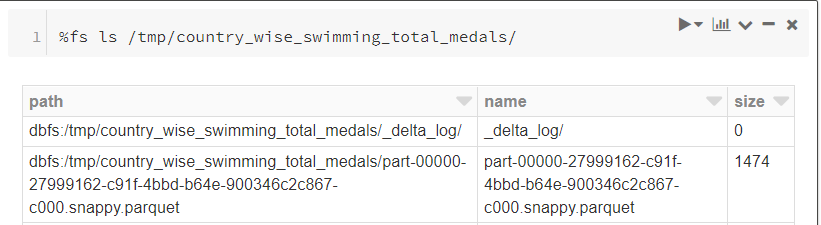
Saving the data in Delta format is as simple as replacing the .format("parquet") function with .format("delta"). However, we there is a major difference when we look at the table creation. When creating a table using Delta, we don't have to specify the schema, because the schema is already strongly defined when we save the data.



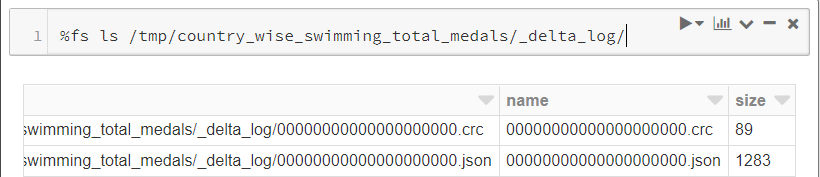
After saving the data into /tmp/country\_wise\_swimming\_total\_medals we can see two things under /tmp/country\_wise\_swimming\_total\_medals directory

1. Data file (Actual data is being stored into this file)
2. \_delta\_log directory (Commit file)

Delta Tables by default uses parquet format and snappy compression technique to store the data



In \_delta\_log directory we’ve a commit file (json file). This commit file is a json file and it contains a lot of information.



When we read the Delta table using spark read api spark DataFrameReader api will first read the commit file and it contains the latest data file.



So, we have seen how to create Delta Tables and how to load data into Delta Tables and what happens internally.