* How ... over charts for ...

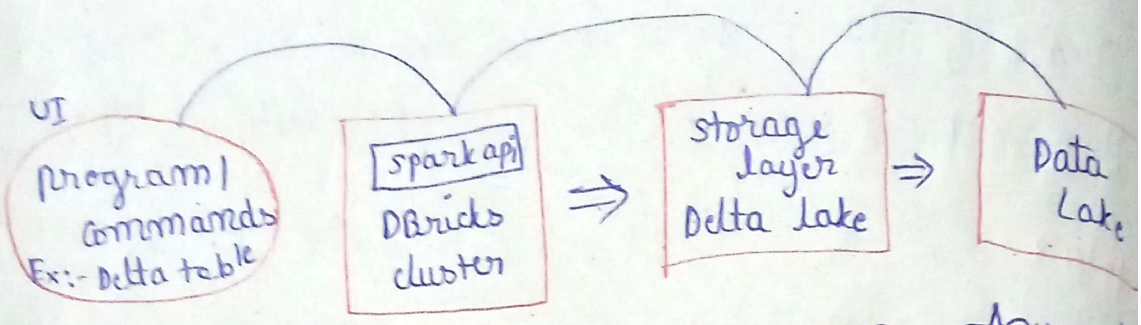
14/2/23

## Databricks Delta Lake

Create delta table with existing data in Databricks.

* Delta lake is an open-source storage layer that brings reliability to data lakes.
* Delta lake provides (i) ACID transactions
  (ii) scalable metadata handling
  (iii) unifies streaming
  (iv) batch data processing
* Delta lake runs on top of your existing data lake and is fully compatible with Apache Spark APIs
* It tells how to create an external table over stored data in specified location.
* Datalake :- place where the data is stored. (like a warehouse)

  Deltalake :- storage layer that provides foundation for storing data and tables in databricks.
* fully compatible with Spark APIs
* It is default storage format for all operations on Azure databricks




Ex :- Azure data lake

delta

## Create a table

### SQL

* To create a delta table, write a data frame out in the delta format.
* These operations create a new delta table using the schema that was inferred from Dataframe.

```
create table delta.`/tmp/delta-table1` using delta
as select col1 as id
from values 10,20,30,40,50;
```

### Python

```
import pyspark
from delta import *
builder = pyspark.sql.SparkSession.builder.
appName("MyApp")\
.config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension")\
.config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.catalog.DeltaCatalog
```

Scanned with OKEN Scanner

```
spark = configure_spark_with_
delta_pip(builder).getOrCreate()
# Creating table
data = spark.range(0,10)
data.write.format("delta").save
("/tmp/delta-table
ex")
```

## Reading data from delta table

Read the data by specifying path to files :-

"/tmp/delta-table"

```
select * from delta.
`/tmp/delta-table1`;
```

```
df = spark.read.format
("delta").load("/tmp/delta
-tablex")

df.show()
```

## update table data

1) overwrite

```
data = spark.range(55,60)
data.write.format
("delete").mode("overwrite")
.save("/tmp/delta-tablex")
```

```
insert overwrite delta.
`/tmp/delta-table1` select
col1 as id from values
5,6,7,8,9;
```

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

deltaTable = DeltaTable.forPath
(spark, "/tmp/delta-tablex")

deltaTable.update(
condition = expr("id%2==0"),
set = {"id": expr("id+100")})
```

2) Conditional update without overwriting

Update even value by adding 1000 to it

```
update delta.`/tmp/delta-
table1` set id = id+1000
where id%2 == 0;
```

Scanned with OKEN Scanner

Update every odd value by adding 200 to it.

```
update delta.`/tmp/delta-table1` set id = id + 200 where id%2!=0;
```

```
deltaTable.update(
condition = expr("id%2==
set = { "id": expr("id + 10
```

Delete

delete every even value

```
delete from delta.`/tmp/delta-table1` where id%2==0;
```

```
deltaTable.delete(condition
= expr("id%2==0"))
```

Upsert (merge) new data

```
create temp view newData
as select col1 as id
from values 1,3,5,7,9...19;
merge into delta.`/tmp/delta-table1` as oldData
using newData
on oldData.id = newData.id
when matched
  then update set id = newData.id
when not matched
  then insert (id) values
    (newData.id);
select * from delta.`/tmp/delta-table1`;
```

```
newData = spark.range(0,20
deltaTable.alias("oldData")\
.merge(
  newData.alias("newData"),
  "oldData.id = newData.id")
.whenMatched Update(set =
{ "id": col("newData.id")}
.whenNotMatched Updat Inse
(values = { "id": col("new
-ata.id") }) \
.execute()

deltaTable.toDF()
show(25)
```

Scanned with OKEN Scanner

Read older versions of the data which we have inserted in the starting is displayed.

```
select * from delta.`/tmp/delta-table1` version as of 0;
```

data using time travel.

```
df = spark.read.format("delta").option("versionAsOf", 0).load("/tmp/delta-table2")
```

---

Writing stream of data to table

```
streamingDF = spark.readStream.format("rate").load()
stream = streamingDF.selectExpr("value as id").writeStream.format("delta").option("checkpointLocation", "/tmp/checkpoint").start("/tmp/delta-table4")
```

Read stream of changes from table

```
stream2 = spark.readStream.format("delta").load("/tmp/delta-table4").writeStream.format("console").start()
```

---

Scanned with OKEN Scanner