1. 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.

1. How to read the table?

df1 = spark.read.format("delta").option("timestampAsOf", timestamp\_string).load("/mnt/delta/events")

df2 = spark.read.format("delta").option("versionAsOf", version).load("/mnt/delta/events")

1. Steps for creating the logic apps and Event grids
2. Create blank logic app
3. Add an event grid trigger (we have provide the tenant name)
4. Add condition
5. Send the notifications
6. Querying the older snapshot of a table

df1 = spark.read.format("delta").option("timestampAsOf", timestamp\_string).load("/mnt/delta/events")

df2 = spark.read.format("delta").option("versionAsOf", version).load("/mnt/delta/events")

1. Write to table

df.write.format("delta").mode("append").save("/mnt/delta/events")

df.write.format("delta").mode("append").saveAsTable("events")

1. Overwrite the table

df.write \

.format("delta") \

.mode("overwrite") \

.option("replaceWhere", "date >= '2017-01-01' AND date <= '2017-01-31'") \

.save("/mnt/delta/events")

1. Delete from a table

from delta.tables import \*

from pyspark.sql.functions import \*

deltaTable = DeltaTable.forPath(spark, "/data/events/")

deltaTable.delete("date < '2017-01-01'") # predicate using SQL formatted string

deltaTable.delete(col("date") < "2017-01-01") # predicate using Spark SQL functions

1. Update the table:

from delta.tables import \*

from pyspark.sql.functions import \*

deltaTable = DeltaTable.forPath(spark, "/data/events/")

deltaTable.update("eventType = 'clck'", { "eventType": "'click'" } ) # predicate using SQL formatted string

deltaTable.update(col("eventType") == "clck", { "eventType": lit("click") } ) # predicate using Spark SQL functions

1. Difference between upsert and update:

Sometime **update** old records and insert new records. Now **Upsert** is come in to the picture to full fill the both insert and **update** operations. Existing records will be **updated** and new records will be inserted

1. Upsert into table using merge:

from delta.tables import \*

deltaTable = DeltaTable.forPath(spark, "/data/events/")

deltaTable.alias("events").merge(

updatesDF.alias("updates"),

"events.eventId = updates.eventId") \

.whenMatchedUpdate(set = { "data" : "updates.data" } ) \

.whenNotMatchedInsert(values =

{

"date": "updates.date",

"eventId": "updates.eventId",

"data": "updates.data"

}

) \

.execute()