Hello Delta Lake !

System Setup

Following is my system setup while exploring and deploying Delta Lake - Exploring Notebook is on **Zeppelin**, will share the zeppelin json

Apache Spark **2.4.5-SNAPSHOT, Custom Build**
Hadoop Version **2.9**
DeltaLake **0.5.0**
Storage **AWS S3**
Deployment on **Kubernetes**
Language **Scala**

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.

Basically Delta Lake provides ACID transactions on top of Apache Spark by internally following parquet Format. As my sample application was parquet heavy, moving to Delta Lake was pretty easy

Why to use Delta Lake

ACID transactions and building an easier Data Lake.

- Saving data as parquet using Apache Spark is easy but while writing, spark doesn't prevent for any schema related damages. Its extremely expensive to realize that any schema change has corrupted some data pipeline, but Delta lake solves that.
- Delta Lake versioning builds data history automatically, so auditing is much easier.
- Everything in Delta Lake is a highly optimized Apache Spark job, so every step register events on Spark UI, easy to debug & explore.

How easy is it to create a Delta Table

Its very much Spark write only.

```
df.write.format("delta").save(“/path/to/write/in(s3 or local)”)
```

What happens when there is a Schema Error while writing Delta Table

- There is no damage over the existing data
- A very precise & detailed error pops up

- Even with overwrite mode, it stops changing schema unless mergeSchema is set.

```
org.apache.spark.sql.AnalysisException: A schema mismatch detected when writing to the Delta table.
To enable schema migration, please set:
'.option("mergeSchema", "true")'.

Table schema:
root
-- bookingid: string (nullable = true)
-- eventtime: string (nullable = true)
-- coordinatelatitude: long (nullable = true)
-- coordinatelongitude: long (nullable = true)


Data schema:
root
-- bookingid: string (nullable = true)
-- eventtime: string (nullable = true)
-- coordinatelatitude: long (nullable = true)
-- coordinatelongitude: long (nullable = true)
-- category: string (nullable = true)
```

How does Delta Lake History help

Delta table maintains entire History of changes over a delta table. Histories are technically preserved on top of JSON meta files and Delta lake provides a neat api to read the history & audit it.
The History returns Spark Dataframe, implicitly bringing all the benefits of spark !

```
import io.delta.tables._
val dataTable = DeltaTable.forPath(spark, PATH_TO_WRITE)
dataTable.history.show()
```

```
+-------+-------------------+------+--------+---------+--------------------------------------------+----+--------+---------+-----------+--------------+-------------+
|version|timestamp          |userId|userName|operation|operationParameters                         |job |notebook|clusterId|readVersion|isolationLevel|isBlindAppend|
+-------+-------------------+------+--------+---------+--------------------------------------------+----+--------+---------+-----------+--------------+-------------+
|2      |2020-01-12 15:18:56|null  |null    |WRITE    |[mode -> Append, partitionBy -> []]         |null|null    |null     |1          |null          |true         |
|1      |2020-01-12 10:05:46|null  |null    |WRITE    |[mode -> Append, partitionBy -> []]         |null|null    |null     |0          |null          |true         |
|0      |2020-01-12 09:58:16|null  |null    |WRITE    |[mode -> ErrorIfExists, partitionBy -> []]|null|null    |null     |null       |null          |true         |
+-------+-------------------+------+--------+---------+--------------------------------------------+----+--------+---------+-----------+--------------+-------------+
```

Another Way based on Version Number

```
spark.read.format("delta").option("versionAsOf", 0).load(PATH_TO_WRITE).show
```

File Compaction in Delta Lake as well provides the history.

Purging Data is Easier

With Delta Lake, purging older data is just a single command known as vacuum. As there are multiple versions of data available, a straightforward approach is provided to simply call vacuum function.
Data can be deleted based on version or threshold.

More details.

```
# Config spark.databricks.delta.retentionDurationCheck.enabled
val dataTable = DeltaTable.forPath(spark, "/path/to/read")
dataTable.vacuum(0.000001)
```

The default retention period is 7 days .

How does Data Update/Delete/Merge Works ?

By generating an entire new version of parquet file internally, so that may lead to a performance dent. But as it brings Parquet, so developer can design the query to use partition pruning and other performance related approaches.

Note: Merging/Update is purely based on Columnar layout and in merge only the files that matches the update conditions will be re-written. So Partition has a key role here.

One very nice example provided here.
Currently with merge, there is a high chance of generating large number of unnecessary files. Read more about it here.

What is dataChange flag ?

During any on-going Delta Lake transaction, `dataChange = False` indicates that the operation doesn't change the data, so any concurrent operations over the same data is minimally impacted.

```
spark.read
  .format("delta")
  .load(path).repartition(numFiles).write
  .option("dataChange", "false")
  .format("delta")
  .mode("overwrite")
  .save(path)
```

Why vacuum on a large table seems slow ?

Because vacuum is by-design Single Threaded & with S3 specific case, I found it too slow sometimes.

There is another way to make vacuum faster by using spark conf

```
spark.sql.sources.parallelPartitionDiscovery.parallelism
```

But above won't solve the single thread issue but for slower cluster, if default 10000 value of parallelism is too large, it can be tuned.

If there is an issue of read got blocked while running vacuum, there is a high chance of parallelism issue, one should reduce it .

Is Delta Lake married to Apache Spark Only ?

Nope by Design. Delta Lake is currently using apache spark to process logs but there is a new connector project, named ***connectors*** launched which is designed to support other JVM based engine like Presto, Hive, etc.

The project can be found here .

Why Delta specific to parquet only ?

Parquet columnar nature & data skipping abilities which give Delta Lake ability for higher performance.
But by design, *Delta Lake is not tied up to a particular Storage Format*. Currently community is working on supporting other formats as well.

```
trait DeltaFileFormat {
  /** Return the underlying Spark `FileFormat` of the Delta table. */
  def fileFormat: FileFormat = new ParquetFileFormat()
}
```

Follow here.

Does Delta Lake allow to create Table without mentioning the Path ?

As of Now, NO.

Delta Lake allows registering an existing table from metastore.

```
%sql
create table sample using delta location 's3a://somepath/delta/bookings'
```

But it doesn't allow to create table without mentioning the path

```
%sql
CREATE TABLE IF NOT EXISTS delta_test (name string, money bigint) USING
delta

%sql
CREATE TABLE IF NOT EXISTS delta_test (name string, money bigint) USING
delta LOCATION 's3a://aws-acc-001-1053-r42-offlinemapmatch/delta/deltatest'
```

```
Error
delta does not allow user-specified schemas.;
```

Currently work is going on to add this feature as well. *With Spark 3.0 release (DataSource V2)*, many features around Table/Hive Metastore/External Datastore should be available with Delta lake.

Is SchemaEvolution is currently supported

Nope, Schema Evolution is currently not supported by Delta Lake.
So if you have a table with evolved schema, currently it won't change anything in the output.

```
Schema 1
root
 |-- bookingid: string (nullable = true)
 |-- eventtime: string (nullable = true)
 |-- number: long (nullable = false)

Schema 2
root
 |-- bookingid: string (nullable = true)
 |-- eventtime: string (nullable = true)
 |-- number: integer (nullable = false)
 |-- extra_1: integer (nullable = false)
 |-- extra_2: string (nullable = true)
```

```
Merge Query
DeltaTable.forPath(spark,"s3a://path/for/first")
.as("fir")
  .merge(
    second.as("sec"),
    "fir.bookingid = sec.bookingid")
  .whenMatched
  .updateAll()
  .execute()
```

Schema of above remains same like first.

But this an upcoming feature & work is already on-going.

There are many upcoming notables changes are on the way.
Kindly follow the project - https://github.com/delta-io/delta
Issues - https://github.com/delta-io/delta/issues