upsolver

Technical Whitepaper:

Databricks Delta Lake vs Data Lake ETL: Overview and Comparison

Executive Summary

Earlier this year, Databricks released Delta Lake to open source. Described as 'a transactional storage layer' that runs on top of cloud or on-premise object storage, Delta Lake promises to add a layer or reliability to organizational data lakes by enabling ACID transactions, data versioning and rollback.

In this article we'll take a closer look at Delta Lake and compare it to a data lake ETL approach, in which data transformations are performed in the lake rather than by a separate storage layer. Obviously, we have a horse in this race since Upsolver is a data lake ETL platform; readers are welcomed to scrutinize our claims and test them against their own real-world scenarios.

The Challenge of Storing Transactional Data in a Data Lake

With all the talk surrounding data lakes, it can be easy to forget that what we're essentially talking about is files stored in a folder (e.g. on Amazon S3). As we've previously explained, in a data lake approach you store all your raw data on inexpensive, decoupled object storage, and then employ a variety of analytics and data management tools to transform, analyze and drive value from the data.

Since the storage layer is composed of files partitioned by time rather than tables with primary and foreign keys, data lakes are traditionally seen as append-only. The lack of indices means that in order to delete or update a specific record, a query engine will need to scan every single record in the lake which isn't a reasonable solution.

Because data lakes are so difficult to update, they are often

seen as less desirable for transactional use cases - for example:

- Data that needs to be frequently updated - such as sensitive customer information that might need to be deleted due to GDPR requests.
- Data that must be absolutely reliable, such as financial transactions that could be cancelled due to charge-backs and fraud.
- Reflecting changes in operational databases via change data capture (CDC).

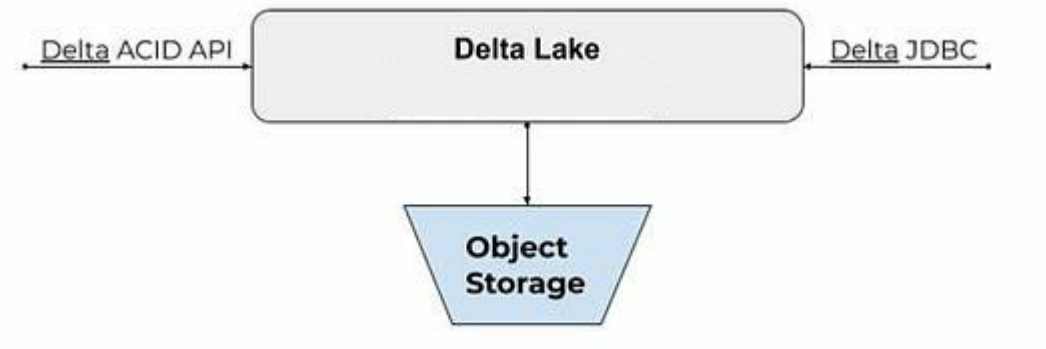
Delta Lake purports to address these and similar scenarios - let's talk about how.

The Delta Lake Solution

Going off the materials Databricks has published online, as well as the coverage in various media outlets, we can get a

pretty good impression of how Delta Lake works.

Basically, Delta Lake is a file system that stores batch and streaming data on object storage, along with Delta metadata for table structure and schema enforcement. Getting data into the lake is done with Delta ACID API and getting data out of the lake is done with Delta JDBC connector. The data is Delta is not queryable by other SQL query engines like AWS Athena, Redshift Spectrum, Apache Presto and vanilla SparkSQL.



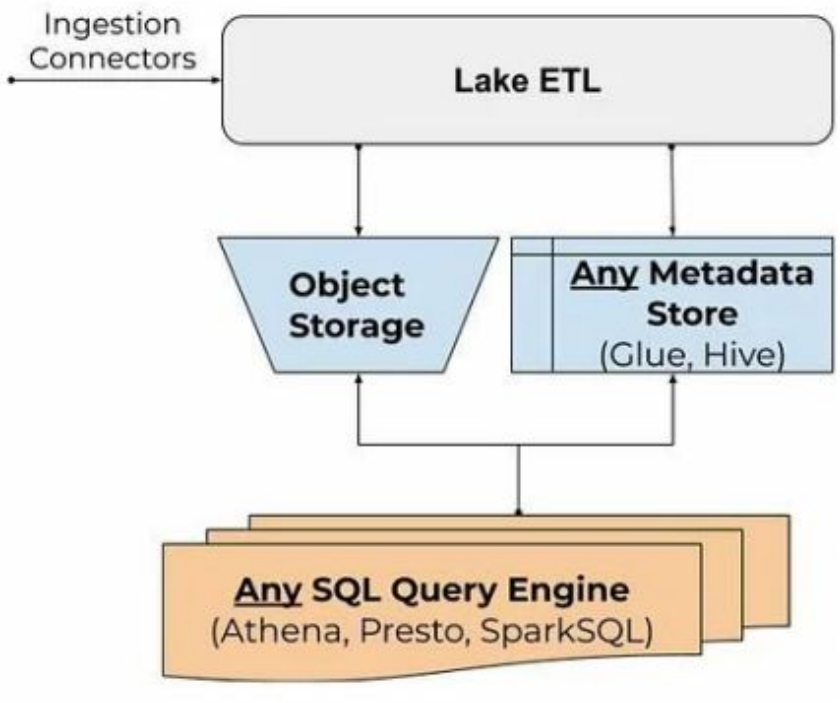
If this sounds a lot like a database built on decoupled architecture, that's probably not a coincidence. While the word "database" is notably absent from the documentation and marketing materials related to Delta Lake, it's safe to say that the software behaves very similarly to decoupled databases such as Snowflake and BigQuery: a separate transactional layer on object storage that uses an ACID API and JDBC connector.

To understand how this differs from a data lake ETL approach, let's look at what the latter entails:

The Data Lake ETL Solution

A data lake ETL solution needs to plug into an existing stack and not introduce new proprietary APIs. Hence, ingestion is performed using connectors and queries are performed by any query engine like AWS Athena, Redshift Spectrum,

Apache Presto and SparkSQL. Metadata stores like Hive Metastore or AWS Glue are used to expose table schema for data on object storage.

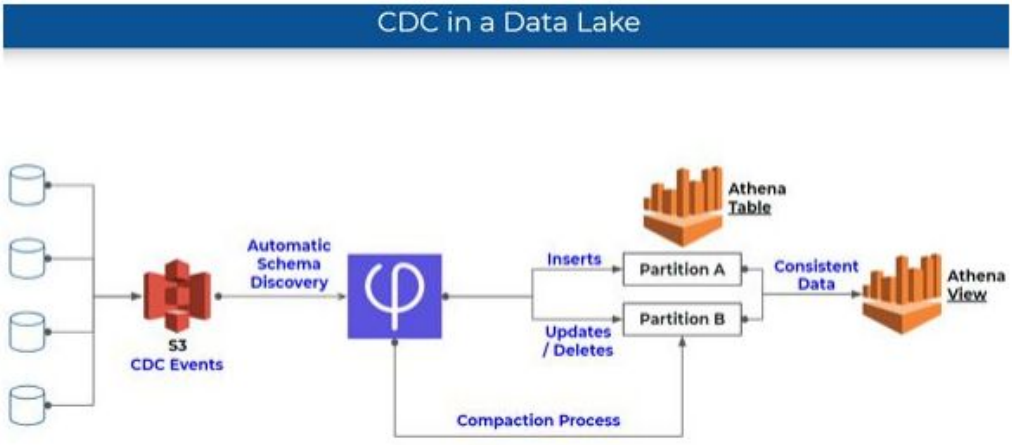


The challenge for data lake ETLs is how to keep the table-data consistent in real-time for queries while maintaining good performance. This is how Upsolver does it (using Athena as an example of a query engine):

1. A user in Upsolver creates an ETL job, with the purpose of transforming raw data to a table in Athena with a primary key.
2. Metadata - Upsolver's engine creates a table and a view in the AWS Glue metadata store. The table has 2 types of partitions: 1 for inserts (new keys) and 1 for updates/deletes.
3. Data - Upsolver's engine writes the table-data to object storage using the standard data lake append-only model. By keeping an index for the table primary key, it's possible to route each row to the right partition (insert or update or delete).
4. Query - a user in Athena will see the new table and view in the Athena console since Athena is integrated with the AWS Glue Data Catalog. Queries will run against

the view (and not the table) that joins insert, update and delete rows from different partitions and returns exactly 1 row per key.

5. Performance - this part is the trickiest to execute. As we recall, data lakes don't include indexing and join of 2 large tables is not recommended. Therefore the view's performance will deteriorate as the number of updates/deletes will grow in the partition. That's why Upsolver rewrites the data every minute, (in a process called compaction. Read more about compaction in dealing with small files), merging the updates/deletes into the original data. This process keeps the number of updates/deletes on the low side so the view queries run fast.



Key Differences

Lock-in to one query engine

Delta Lake tables are a combination of Parquet based storage, a Delta transaction log and Delta indexes which can only be written/read by a Delta cluster. This goes against the basic logic of a data lake which is meant to allow users to work with data their way, using a wide variety of services per use case.

To use Delta Lake, it's necessary to change ingestion to use Delta ACID API and run queries using the Delta JDBC. Delta cannot ensure data consistency across partitions, while updates to partitions need to be done manually in order to ensure users see the most up-to-date version of a table.

Data lake ETLs only make changes to the underlying table data so there is no lock-in and any SQL engine can be used to query the data. Replacing a vendor for Lake ETLs doesn't

require changing ingestion or query APIs.

Ingestion Performance

Lake ETLs will out-perform Delta by an order of magnitude since they perform append-only writes to object storage while Delta needs to update indexes for each ACID transaction.

Ease-of-use

The Delta Lake approach offers a simple insert/update/delete API but requires a project to change existing ingestion and query interfaces to Delta.

While data lake ETL requires a significant ETL effort as explained above- especially when it comes to dealing with small files - platforms like Upsolver can be used to reduce this effort to nothing.

Summary:

	Delta Lake	Lake ETLs
Lock-in	**High** Need to change ingestion and query interfaces to Delta, no support for reliable concurrent writes.	**Low**. No change in interfaces and no proprietary metadata Lake ETL vendor can be replaced with home-grown ETL.
Ingestion Performance	**Low**. ACID transactions and indexes.	**High**. Append-only writes.
Entry barrier	**High.** Requires non-trivial expertise in Spark coding	**Low** Visual interface and SQL
Ease-of-use	**Medium**. ACID operations replace ETLs but all ingestion and query interfaces need to be migrated to Delta. Delta requires a DBA for operations like Vacuum and Optimize	**Depends** on ETL platform Upsolver offers a turn-key solution, automating ETLs.

Next Steps:

- Watch our webinar with AWS and ironSource to learn how ironSource operationalize a petabyte-scale data lake in the cloud: https://www.upsolver.com/webinar/aws-frictionless-data-lake
- Schedule a free demo of Upsolver today:

  https://www.upsolver.com/schedule-demo