Introduction to Lake House

**Data Warehouse**

* Structured data store used for analytical processing and reporting (BI)
* It usually holds transformed data fed in from disparate sources via ETL pipelines

|  |  |
| --- | --- |
| Diagram  Description automatically generated | * Insights from structured data * ETL pipelines to connect up silos * SQL queries on data * Hybrid on-cloud and on-premises data * Phased migration of legacy data * Access to historic data * Processed and reliable data |

**Data Lake:**

Data lake has four key capabilities

1. Ingest: Data Collection
2. Store: Data Storage and management
3. Process: Data Processing and Transformation
4. Consume: Data Access and Retrieval

|  |  |
| --- | --- |
| Diagram  Description automatically generated | * Insights from structured, semistructured, unstructured, data * Eliminate silos entirely * SQL queries and data science * One-off migration of legacy data * Historic and real-time data * Raw data in unprocessed form * Store ML Models artifacts, realtime data and analytical outputs. |
| **Challenges with Data Lake:**   * Lake of Schema Enforcement * Lake of Consistency * Data corruption on job failure * No ACID Compliance | |

**Lake House:**

* A data Lakehouse is a new, open data management architecture that combines the flexibility, cost-efficiency, and scale of data lakes

|  |  |
| --- | --- |
| Timeline  Description automatically generated | Data Warehouse + Data Lake   * One platform for all use cases * High performance query engine * Data structures and data management offers structured transactional layer * Low-cost, centralized data lake storage of all your data |
| Metadata layers on top of data lakes  - Rich management features  - ACID properties  - Streaming data  - Table versioning  - Schema enforcement  - Data validation  -Offers high-performance access on low-cost storage | |

|  |  |  |
| --- | --- | --- |
| First Generation Platform | Current Two-tier Architecture | Lakehouse platform |

Introduction To Delta Lake

* Delta Lake is an open-source storage layer for Spark that enables relational database capabilities for batch and streaming data
* Delta lake Implements the Data Lakehouse in Spark to support SQL based data manipulation semantics with support for transactions and schema enforcement.
* It offers analytical store with many of the advantages of a relational database system with the flexibility of data file storage in a data lake.
* Delta Lake is supported in Azure Datbricks, Azure Synapse Analytics Spark pools for PySpark, Scala, and .NET code.

Delta Lake offers:

* ACID transactions so (Update/delete/merge) is implemented.
* scalable metadata handling,
* Unifies streaming and batch data processing on top of existing data lakes. Delta Lake tables can be used as both sinks (destinations) and sources for streaming data also.
* Schema enforcement
* Time Travel/Versioning (If data is corrupted you can go back and get previous data.)
* Data as Parquet and maintains transactional log in additional folder called in **delta\_log**

Delta Lake runs on top of your existing data lake and is fully compatible with Apache Spark APIs

* Think of Delta Lake as Intermediate service between Apache spark and storage.
* Program can talk to Delta Lake for read and write data. ACID Compliance is taken care by Delta Lake.

Graphical user interface, application, table, website

Description automatically generated

**Delta Lake:** Parquet +Transactional Logs

Delta lake adds transactional awareness to parquet with transactional log, maintained in additional folder called in **delta\_log.**

A screenshot of a computer

Description automatically generated with low confidence

**Databricks Lakehouse:**

The Databricks Lakehouse architecture combines data stored with the Delta Lake protocol in cloud object storage with metadata registered to a [metastore](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/data-objects" \l "metastore).

The benefits of using Delta Lake in a Synapse Analytics Spark pool

* **Relational tables that support querying and data modification**. With Delta Lake, you can store data in tables that support *CRUD* (create, read, update, and delete) operations. In other words, you can *select*, *insert*, *update*, and *delete* rows of data in the same way you would in a relational database system.
* **Support for *ACID* transactions**. Relational databases are designed to support transactional data modifications that provide *atomicity* (transactions complete as a single unit of work), *consistency* (transactions leave the database in a consistent state), *isolation* (in-process transactions can't interfere with one another), and *durability* (when a transaction completes, the changes it made are persisted). Delta Lake brings this same transactional support to Spark by implementing a transaction log and enforcing serializable isolation for concurrent operations.
* **Data versioning and *time travel***. Because all transactions are logged in the transaction log, you can track multiple versions of each table row and even use the *time travel* feature to retrieve a previous version of a row in a query.
* **Support for batch and streaming data**. While most relational databases include tables that store static data, Spark includes native support for streaming data through the Spark Structured Streaming API. Delta Lake tables can be used as both *sinks* (destinations) and *sources* for streaming data.
* **Standard formats and interoperability**. The underlying data for Delta Lake tables is stored in Parquet format, which is commonly used in data lake ingestion pipelines. Additionally, you can use the serverless SQL pool in Azure Synapse Analytics to query Delta Lake tables in SQL.

**Delta Tables**

Delta Lake Tables can be created by saving dataframe in delta format.

Example:

# Load a file into a dataframe

df = spark.read.load('/data/product.csv', format='csv', header=True)

# Save the dataframe as a delta table

delta\_table\_path = "/delta/products-delta"

df.write.format("delta").save(delta\_table\_path)

Use Serverless pool to query Delta Table

SELECT

TOP 100 \*

FROM

    OPENROWSET(

        BULK 'https://mtbanklake1234.dfs.core.windows.net/synapase-files/delta/products-delta/',

        FORMAT = 'DELTA'

    ) AS [result]

Tables in Databricks use the Delta Lake protocol by default.

When you create a new Delta table:

* Data and table metadata are saved to a directory in cloud object storage.
* Metadata used to reference the table is added to the metastore in the declared schema or database.

All Delta tables have:

* A directory containing table data in the Parquet file format.
* A sub-directory /\_delta\_log that contains metadata about table versions in JSON and Parquet format.

**Data Objects in Databricks Lakehouse**

|  |  |
| --- | --- |
| Unity Catalog object model diagram | [**Catalog**](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/data-objects#catalog): a grouping of databases.  [**Database**](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/data-objects#database) or schema: a grouping of objects in a catalog. Databases contain tables, views, and functions.  [**Table**](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/data-objects#table): a collection of rows and columns stored as data files in object storage.  [**View**](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/data-objects#view): a saved query typically against one or more tables or data sources.  [**Function**](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/data-objects#function): saved logic that returns a scalar value or set of rows. |

1. **Metastore:** The metastore contains all of the metadata that defines data objects in the lakehouse.
2. **Catlog:**Every database will be associated with a catalog. Catalogs exist as objects within a metastore.

The built-in Hive metastore only supports a single catalog, hive\_metastore.

1. **Database/Schema:** a grouping of objects in a catalog. Databases contain tables, views, and functions.

Databases will always be associated with a location on cloud object storage.

You can optionally specify a LOCATION when registering a database.

Location specified will be always Managed Location and determine the default location for data of all tables registered to that database.

1. **Table:** a collection of rows and columns stored as data files in object storage.

A Delta table stores data as a directory of files on cloud object storage and registers table metadata to the metastore within a catalog and schema.

There are two types of Tables:  [managed](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/data-objects#managed-table) and [unmanaged](https://learn.microsoft.com/en-us/azure/databricks/lakehouse/data-objects#unmanaged-table)

**Manged Table:**

* Azure Databricks manages both the metadata and the data for a managed table;
* The data for a managed table resides in the LOCATION of the database it is registered to
* Data analysts and other users that mostly work in SQL may prefer Managed table as dropping table deleted the underlying data too.
* in order to move a managed table to a new database, you must rewrite all data to the new location.

Example:

|  |
| --- |
| CREATE TABLE table\_name AS SELECT \* FROM another\_table |
| CREATE TABLE table\_name (field\_name1 INT, field\_name2 STRING) |
| df.write.saveAsTable("table\_name") |

**UnManaged Table:**

* Azure Databricks only manages the metadata for unmanaged (external) tables;
* Because data and metadata are managed independently, you can rename a table or register it to a new database without needing to move any data.
* Data engineers often prefer unmanaged tables and the flexibility they provide for production data.

**Example:**

|  |
| --- |
| CREATE TABLE table\_name  USING DELTA  LOCATION '/path/to/existing/data' |
| CREATE TABLE table\_name  (field\_name1 INT, field\_name2 STRING)  LOCATION '/path/to/empty/directory' |
| df.write.option("path", "/path/to/empty/directory").saveAsTable("table\_name") |

**Medallion Lake House Architecture**

Medallion architecture describes a series of data layers that denote the quality of data stored in the lakehouse.

This architecture guarantees **ACID** as data passes through multiple layers of validations and transformations before being stored in a layout optimized for efficient analytics.

Organize our data into layers or folders as defined as bronze, silver, and gold as follows:

1. **Bronze(raw)**: tables have **raw data** ingested from various sources (RDBMS data, JSON files, IoT data, etc.).Is appended incrementally and grows over time.
2. **Silver** (Validated): tables contain **curated(Validated)** data. We apply required business logic on top of Bronze, clean it . We can join fields from various bronze tables to improve streaming records or update account statuses based on recent activity. It will give a more refined view of our data.
3. **Gold(enriched)**: tables give business-level aggregates often used for dashboarding and reporting. Aggregations applied on top of silver tables. It has data that powers analytics, machine learning, and production applications. gold tables represent data that has been transformed into knowledge, rather than just information.

This would include aggregations such as weekly sales per store, daily active website users, or gross revenue per quarter by the department.

The end outputs are actionable insights, dashboards, and reports of business metrics. Delta tables can be accessed for BI and reporting along with data science and ML.

Chart, box and whisker chart

Description automatically generated

***Note:*** *medallion architecture does not replace other dimensional modeling techniques. Schemas and tables within each layer can take on a variety of forms and degrees of normalization depending on the frequency and nature of data updates and the downstream use cases for the data.*

**Lambda Architecture:**



**QuickStart Ref:**

https://docs.microsoft.com/en-us/azure/databricks/\_static/notebooks/delta/quickstart-python.html