**A diagram of a database

Description automatically generated**

**A white text with black text

Description automatically generated**

**A white background with black text

Description automatically generated**

**A white background with black text

Description automatically generated**

**Before going to talk about lake house architecture , Let see evaluation of lake house arch,**

**Data warehouse only work for structured data only. So its called Its first generation system.**

**Structed data in the form of Rows and columns. Once transformation has been done then we are moving to DW. From they were generate the reports using power bi**

**Adv**

**Support the ACID**

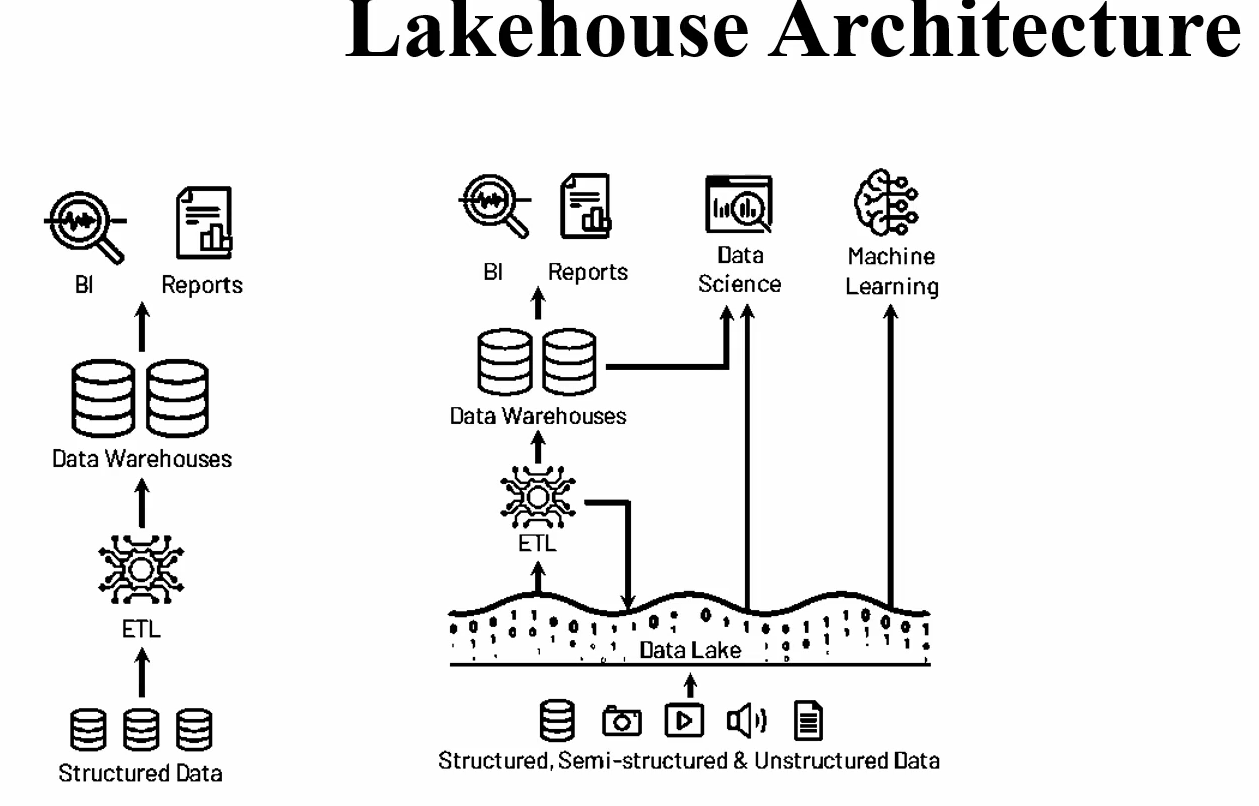
**Performs CRUD**

**Data governs: restrictions**

**Draw back:**

**Only support structed data,**

**ML**

****

While data lakes and [data warehouses](https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-a-data-warehouse/) are similar in that they both store and process data, each have their own specialties, and therefore their own use cases.

That's why it's common for an enterprise-level organization to include a data lake and a data warehouse in their analytics ecosystem.

Both repositories work together to form a secure, end-to-end system for storage, processing, and faster time to insight.

**A data lake captures both relational and non-relational data from a variety of sources**—business applications, mobile apps, IoT devices, social media, or streaming—without having to define the structure or schema of the data until it is read.

Schema-on-read ensures that any type of data can be stored in its raw form.

As a result, data lakes can hold a wide variety of data types, from structured to semi-structured to unstructured, at any scale.

Their flexible and scalable nature make them essential for performing complex forms of data analysis using different types of compute processing tools like [Apache Spark](https://go.microsoft.com/fwlink/?linkid=2240158) or [Azure Machine Learning](https://azure.microsoft.com/en-us/products/machine-learning/).

1. Object storage

2. Cheaper option for storage

3. BI

4. its called 2 tier architecture

5. we can read any kind of data (structured or unstru)

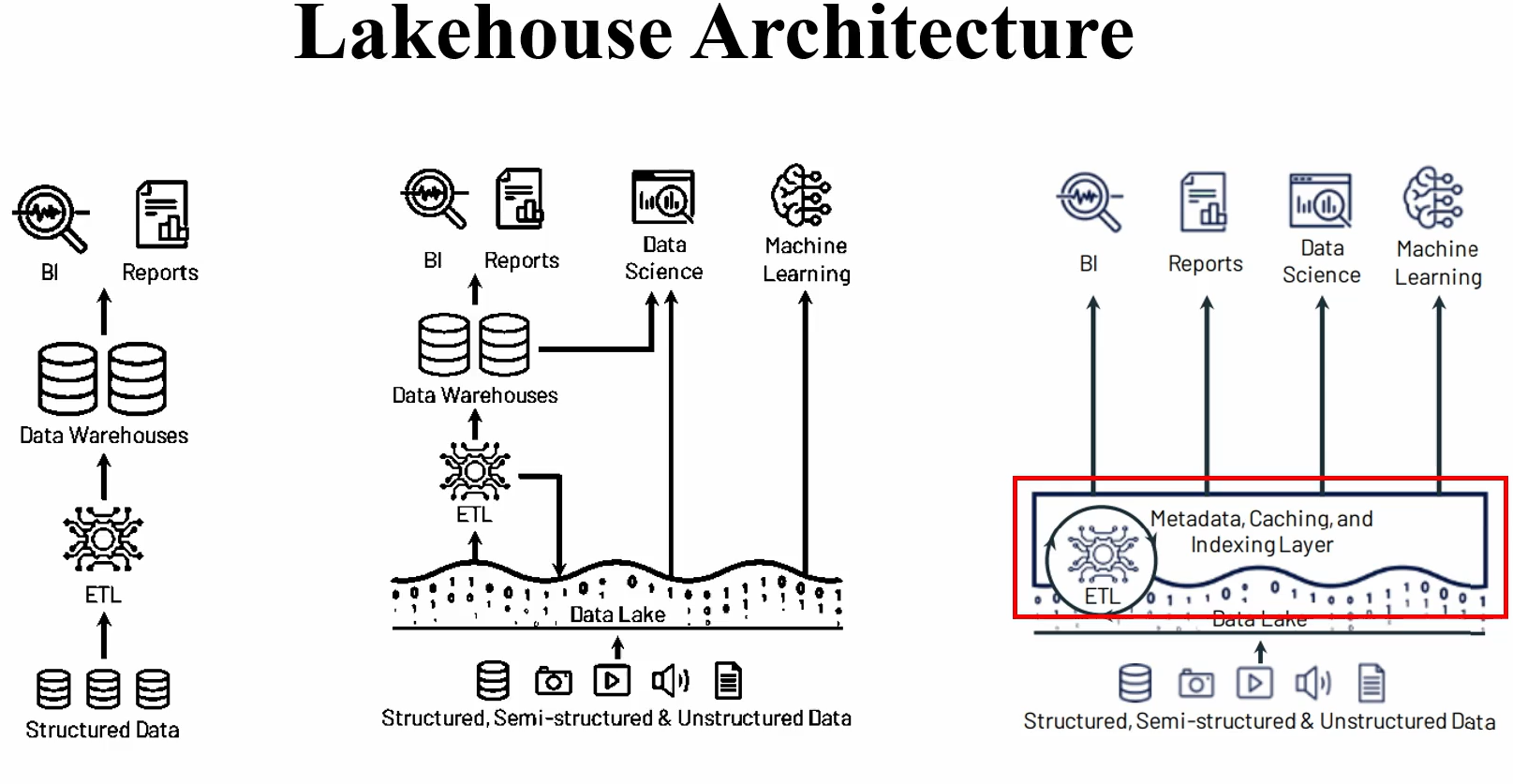
6. INgestinon of dfata is very faster

7. scalable

**Drawback:**

1. No ACID
2. Schema enforcement
3. DL like ml kind of report
4. For Report Level it can’t support – still it will depends DW
5. We need to pay for additional warehouses – specially reporting purpose

To Solve the above mentioned



1. DL topof it uses metadata , Transaction logs

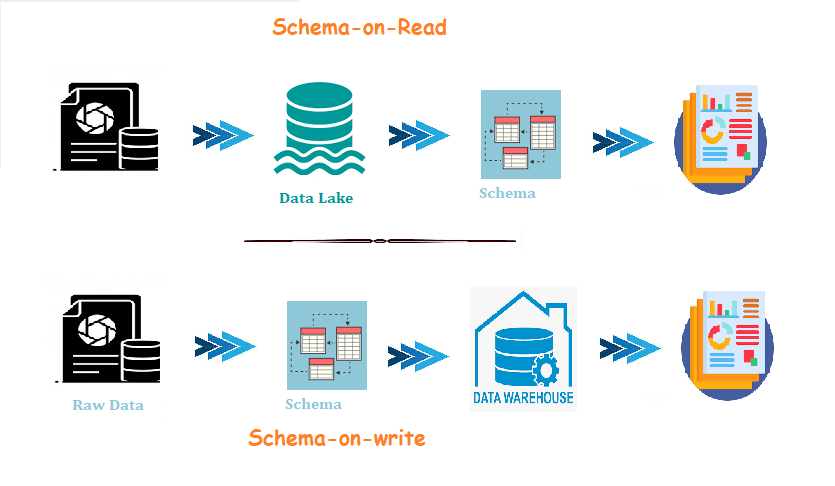
A diagram of a lakehouse architecture

Description automatically generated

A close-up of a company logo

Description automatically generated

https://www.udemy.com/course/azure-databricks-end-to-end-project-with-unity-catalog-cicd/learn/lecture/41599950#content



n our traditional approach, we create a relational database with tables by specifiying the schema. Once we have configured the schemas, created the tables, we can begin to load the data. We may be loading data in bulk or as small batches, however we will be ingesting data based on the schema configured. And, once the data is loaded into the table, we can begin to execute analytical queries on our tables. This is the **schema-on-write**

Now comes a problem, we have data and we don't know the schema, to create table and to analyze the data we need to know the schema ahead, so practically the schema-on-write fails in this situation.

What if we load the files/data and then analyze its structure and characteristics, now this quantifies for **schema-on-read**

**Schema-on-read** provides scalability and flexibility to apply any analytical operations,

**schema-on-write** helps to model a structured data model, thereby enabling faster reads based on a predefined schema.

**A white paper with black text

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**Why Delta Lake?** In modern-day big data projects, there are many cloud object data lake storages such as Amazon S3 and Azure Data Lake are some of the largest and most cost-effective storage systems. Unfortunately, their implementation as key-value stores makes it difficult to achieve ACID transactions, high performance, and cross-key consistency, and Metadata operations such as schema enforcements, and listing objects are expensive and consistency guarantees are limited. To take over these limitations delta lake is introduced.

**1. What is Delta Lake?**

Delta Lake is an open-source storage layer that enables building a data lakehouse on top of existing storage systems over cloud objects with additional features like ACID properties, schema enforcement, and time travel features enabled.

Underlying data is stored in snappy parquet format along with delta logs.

It supports both Batch and Streaming sources under a single platform in Databricks.

A close-up of a data format

Description automatically generated

A diagram of a computer program

Description automatically generated

A diagram of a computer

Description automatically generated

A diagram of a computer program

Description automatically generated with medium confidence

**2. Features of Delta Lake**

2.1. Added ACID Properties:

ACID properties are the ones that define a transaction (Atomicity, Consistency, Isolation, Durability). They help us in guaranteeing the below.

* **Atomicity**: Makes sure each transaction(to read, write, update or delete data) was completed all at once or doesn’t happen at all. This helps in preventing data loss and data quality issues.
* **Consistency** – ensures the data objects are consistent before and after transaction.
* **Isolation** – enables transactions to ensure that inter collation is not happening in-between transactions and provides ordering among transactions. It makes possible to read and append when there is an update going on.
* **Durability** – ensures that transactions on your data made successfully will be saved, even in the event of system failure.

**Note**: ACID properties ensure that your data never falls into a bad data quality state because of a transaction that was only partially completed or failed and provide data reliability and integrity.

2.2. Schema enforcement

When we write data into Delta Lake, it also stores the schema of your data in JSON format inside the transaction log. Delta Lake uses these JSON files for schema validation in the subsequent writes. By default, It compares your existing schema with the incoming schema and if any mismatches are found then it makes the entire transaction fail. This is one of the features that provides schema evaluation and data quality

In case of schema changes between existing and incoming data, we can handle the failure due to mismatch by merging the schema changes. To achieve this we have to enable auto-merge in the spark configuration or add mergeSchema option as true while writing the dataFrame.

//Enabling autoMerge in spark configuration

spark.conf.set("spark.databricks.delta.schema.autoMerge.enabled","true")

OR

//mergeSchema to true while writing dataFrame

dataFrame.write.format("delta")

.option("mergeSchema", "true")

.mode("append")

.save(DELTALAKE\_PATH)

2.3. Time Travel

All the changes on a table in Delta Lake were tracked, stored in delta logs, and resulted in a newer version of that dataset. Whenever we query the table it showcases the latest version of it. It also uses this versioning concept to track and revert back to previous versions for Audits and rollbacks in Databricks.

To deep dive more into time travel, please have a look at this article [Time Travel with Delta Tables in Databricks?](https://sparkbyexamples.com/spark/time-travel-in-delta-tables-on-databricks/)

2.4. UPSERT Operations

Delta Lake supports upsert (Insert or Update) operations on the existing datasets by comparing changes in existing data with the incoming data and ensuring no duplicates were inserted based on the primary keys used.

MERGE INTO table1 a

USING table2 b

ON a.column\_x = b.column\_y

WHEN MATCHED THEN

UPDATE SET

column\_a = b.column\_p,

column\_b = c.column\_q,

........

........

WHEN NOT MATCHED

THEN INSERT (

column\_a,

column\_b,

......,

......,

......

)

VALUES (

b.column\_p,

b.column\_q,

b.column\_r,

..........,

..........

)

**3. Problems solved by Delta Lake**

* Complexity in appending data to existing dataset reduced.
* Difficulty in modification of existing data was reduced.
* Ensure data consistency and quality even after jobs fail midway.
* Real-time operations were easy.
* Decrease in Cost to keep historical data versions.

**4. Delta Lake is Not**

* Proprietary technology
* storage format
* storage medium
* Database service or data warehouse

**5. Delta Lake is**

* Open-source.
* Builds upon standard data formats: It is powered primarily by parquet format.
* Optimized for cloud object storage.
* Built for scalable metadata handling.

**6. Conclusion**

The primary objective of delta lake is resolving the time taken for quickly returnable queries and providing data consistency even on Upsert operations. Delta Lake decouples storage and computing costs and provides optimized performance on data.

**What is a medallion architecture?**

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

Databricks recommends taking a multi-layered approach to building a single source of truth for enterprise data products.

This architecture guarantees atomicity, consistency, isolation, and durability as data passes through multiple layers of validations and transformations before being stored in a layout optimized for efficient analytics.

The terms **Bronze**(raw),**Silver**(filtered, cleaned, augmented), and **Gold**(business-level aggregates) describe the quality of the data in each of these layers.

A screenshot of a computer

Description automatically generated