Disclaimer: These slides are copyrighted and strictly for personal use only

- This document is reserved for people enrolled into the Ultimate Big Data Masters Program (Cloud Focused) by Sumit Sir

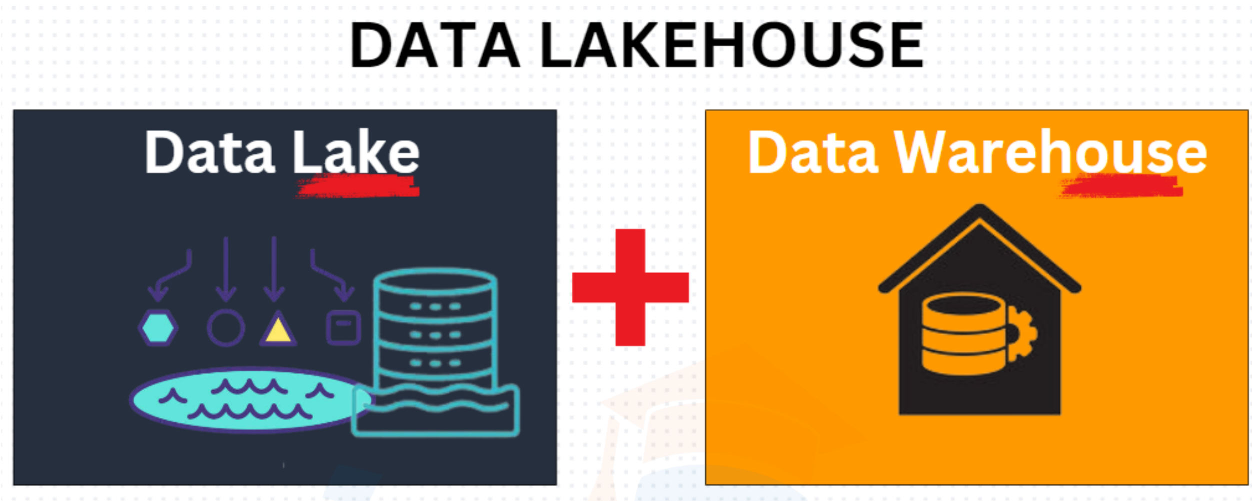
- **Please do not share this document,** it is intended for personal use and exam preparation only, thank you.

- If you've obtained these slides for free on a website that is not the course's website, please reach out to legal@trendytech.in . Thanks!

- All the Best and Happy Learning!

Lakehouse Architecture

Provides the best of both the worlds, Data Lake and Data Warehouse



Datalake Benefits

1. Inexpensive - Cost of storage is very less.
2. Scalable - Can store petabytes of Data.
3. Stores all kinds of Data - Can store Structured, Semi-Strucutured and Unstructured data in a raw / native form.
4. Supports Open File formats - Like Parquet which are not vendor specific formats.

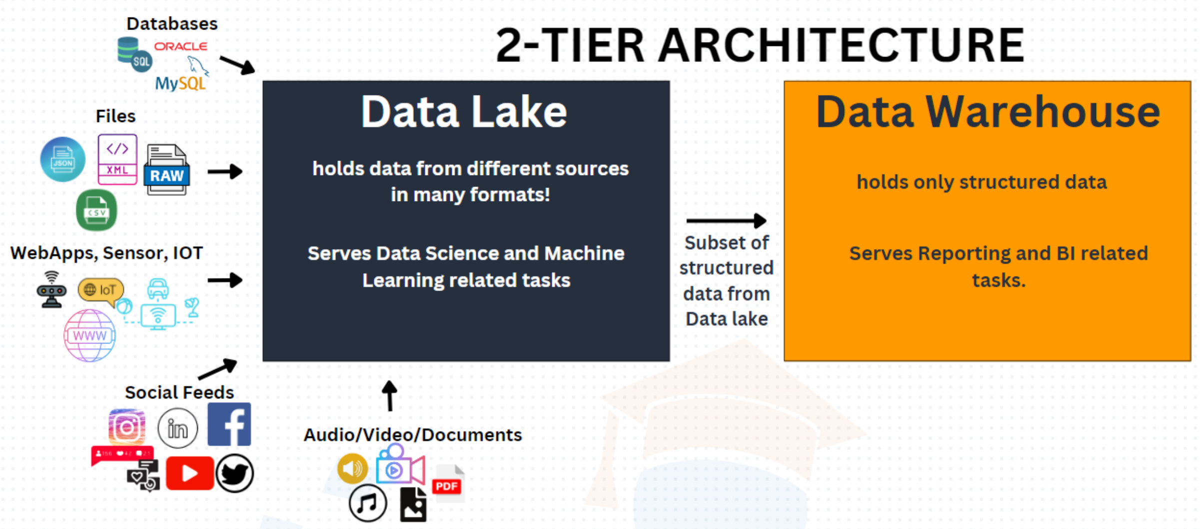
Datalake Challenges

1. ACID guarantees are not supported .
2. Not optimized for Reporting / BI workloads.

Note:

- Datalake alone couldn't support all the workloads in the data domain, like - Data Science, Machine Learning, Reporting, BI, Transactional Capabilities.
- **Modern two tier datawarehouse architecture -** In order to achieve this, companies used to make use of two systems to meet their requirement of supporting all the usecases, i.e, Datalake and Datawarehouse combined together.

- Initially all the data was pushed to Datalake in its raw form and a subset of this data was structured and sent to datawarehouse for serving Reporting and BI usecases.



Challenges of 2-tier architecture

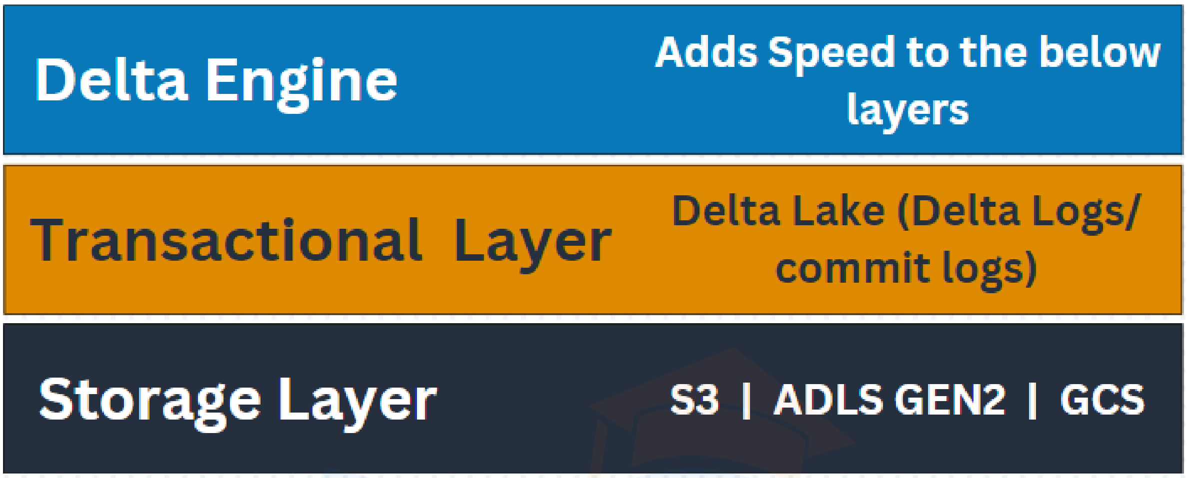
1. Requires two different systems
2. Data Duplication Issues
3. Increased cost
4. Additional ETL activity for moving the data from Datalake to Datawarehouse
5. Stale Data

Lakehouse Architecture

provides an effective solution for the above challenges with the following benefits :

1. Inexpensive Storage
2. Supports all kinds of Data Forms
3. Supports Open File Formats
4. Reduces Data Duplication
5. Reduces ETL operations
6. Supports are kinds of Workloads like Data Science, Machine Learning, BI, Reporting..

Databricks Lakehouse Architecture

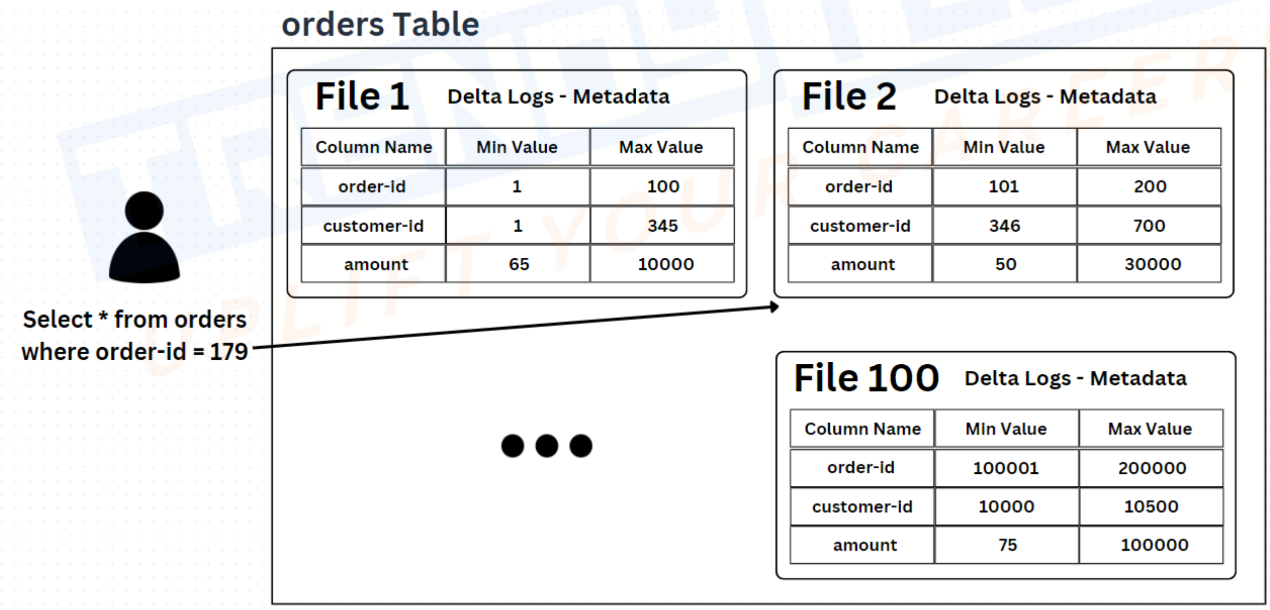


Optimization Techniques

1. **Data Skipping using Stats**

As part of delta logs, Each file will have a metadata associated which contains the min and max value of each column

Example Orders Table - consisting of 100 files



File 1

Column Name	Min Value	Max Value
order-id	1	100
customer-id	1	345
amount	65	10000

File 2

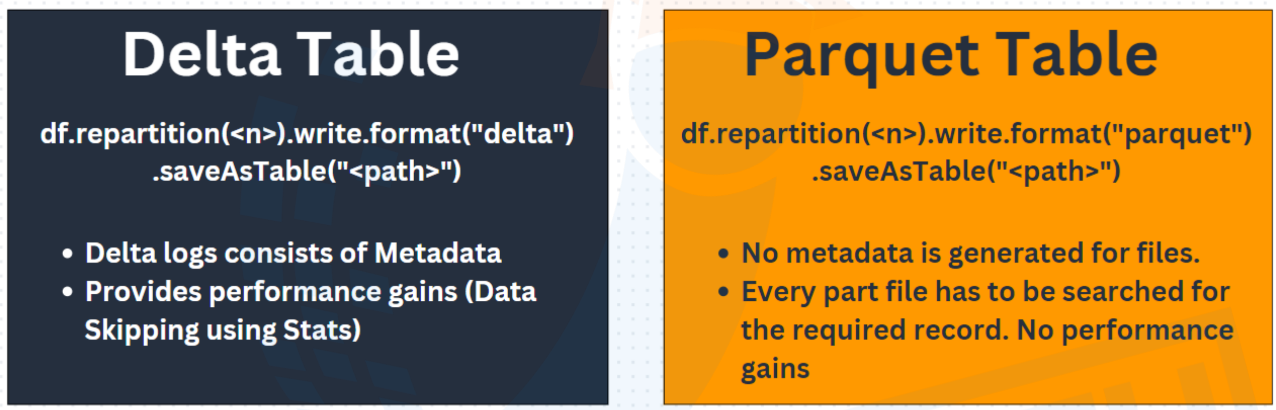
Column Name	Min Value	Max Value
order-id	101	200
customer-id	346	700
amount	50	30000

File 100

Column Name	Min Value	Max Value
order-id	100001	200000
customer-id	10000	10500
amount	75	100000

- Since we have statistics of each file, we need not search all the files for a given record. With the metadata available we can look for a record only in a specific file where the record falls in the file’s range
- The stats in the form of metadata will help in skipping the data and considering only a subset of data. If this optimization wasn’t done, then all the files had to be searched for the required record.
- Without even opening the file, we can get to know if the record is present in a particular file or not with just the metadata, leading to huge performance gain.

Example to differentiate the performance of Delta table and Parquet table



2. **Delta Cache**

Taking the data from the storage layer and cache it on the worker node’s local disk. Also, it is stored in a format which is very efficient and quick for fast retrievals.

- **Ways to enable delta cache**
  1. Use Specific Machines (like Delta Accelerated VMs)
  2. Set a property to enable delta cache in case of a normal cluster.

  By Default, the delta cache is disabled. Can check this using the command :

spark.conf.get(“spark.databricks.io.cache.enabled”) gives false as the result

To enable delta cache, set the property to true spark.conf.set(“spark.databricks.io.cache.enabled”, “true”)

3. Can be manually enabled using the cache keyword before executing the query

   [%sql

   Cache

   Select * from <Database-Name.Table-Name>

   ]

Small File Problem

Example

Table-A having 10000 small files with 10 records each

Table-B with 4 big files of 25000 records each

Key points :

- Table-B is more efficient, if say you are running a query that has to open and search through all the files. The overhead of opening all the files in case of Table-A would be high as it has a huge number of files.

Practicals to demonstrate Small File Problem

Step 1: Create a Database

Step 2: Load any sample csv file from DBFS file system with few GBs of data.

Step 3: Create a Delta Table with a large number of files (~500 files) for each partition using the repartition option.

On this table, executing even a simple filter query would take a considerable amount of time.

The Small File Problem can be overcome using a technique

“Compaction / Bin-packing” -

This technique involves taking multiple small files and merging them into large files.

In Databricks, OPTIMIZE command is used to compact delta files of up-to 1GB

- Optimize has to be performed periodically as lot of new small files get generated regularly.

- Optimize is a resource intensive operation, therefore has to be performed at non-peak hours.

Ex:

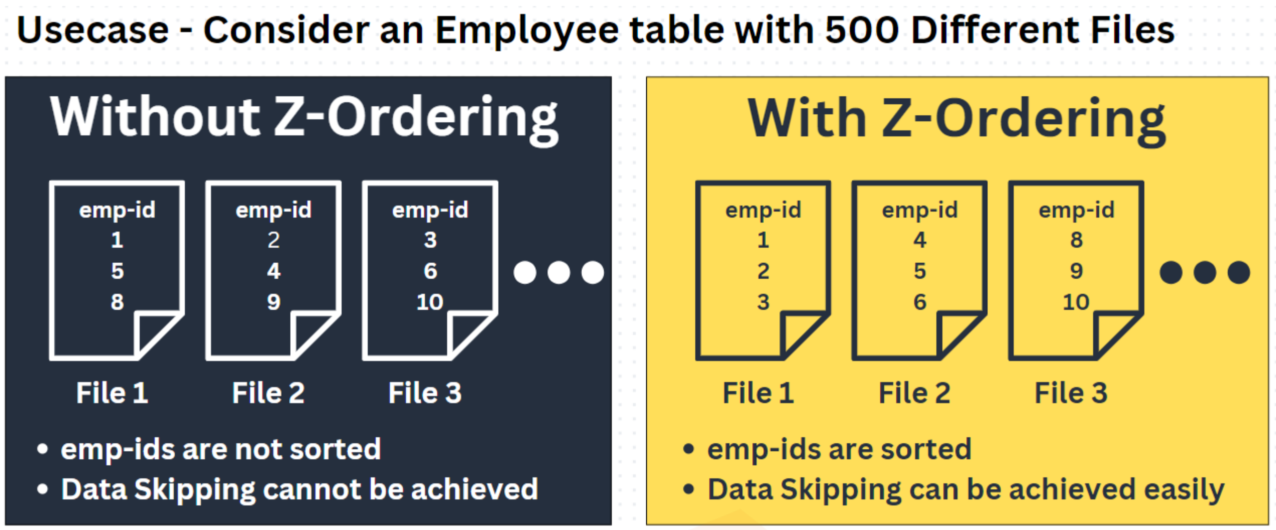
%sql

OPTIMIZE <Database-name.Table-name>

Z-ordering

Z-ordering is used along with Optimize to achieve Data Skipping.

- The column which is z-ordered will be sorted and the data is distributed in an effective way which allows for Data Skipping optimization.
- Equivalent to clustered index as in the case of Databases.
- Z-ordering is a technique to co-locate related information in the same set of files.
- Co-locality is used in databricks to achieve data skipping and provide performance benefits
- This will drastically reduce the amount of data to be scanned.
- The columns that would be used for filter, join, group by,... operations can be z-ordered for performance gains.
- Partitioning and Bucketing is by default designed in a manner to achieve Data Skipping and therefore achieve performance optimization. In the case of partitioning, data is sorted and partitioned into folders while in case of Bucketing, data is sorted into files.



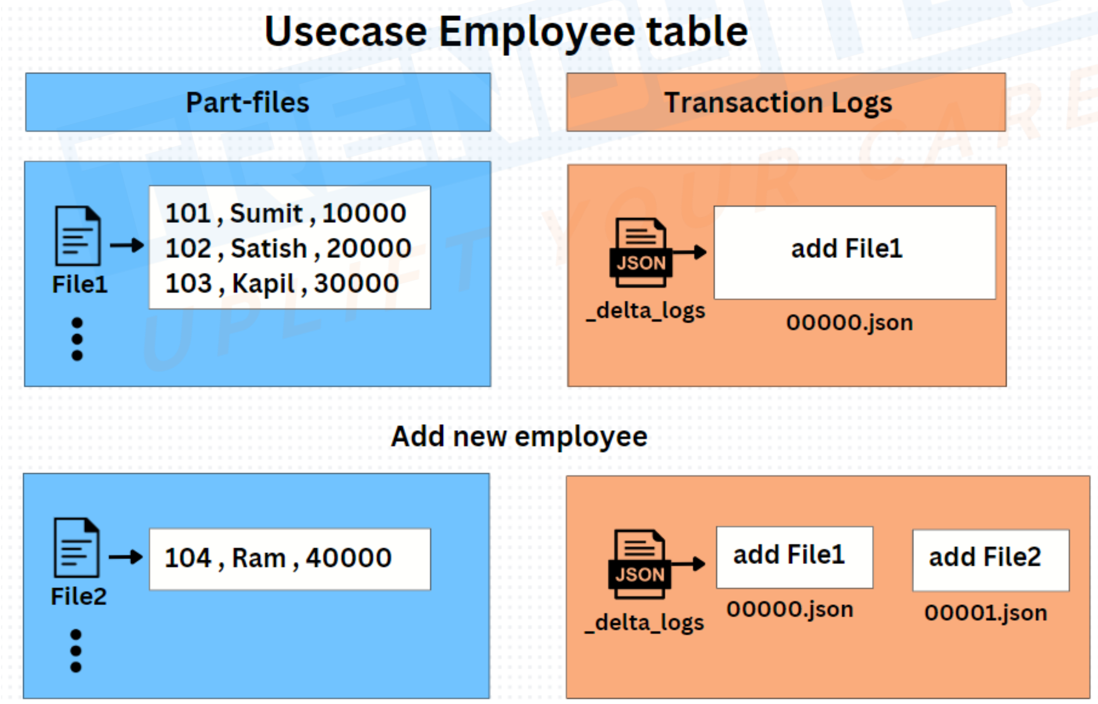
Ex:

%sql

OPTIMIZE <Database-name.Table-name> **zorder by** <Column-name>

VACUUM Command

It is necessary to delete the older files in the transaction logs that are not referenced anymore as it could pile up and lead to wastage of storage resources.



As shown in the diagram above, the transaction logs keep on getting added on every operation like add / update / delete. These log files that are older and not referenced need to be deleted to free the storage space.

VACCUM command is used for this purpose

1. It removes the files that are no longer referenced in the latest transaction logs
2. Removes the files older than a retention threshold (default - 7 days)
3. Vacuum command affects the Time Travel as the older versions are deleted and are not accessible.

Ex:

%sql

VACUUM <Database-name.Table-name> **RETAIN 1 HOURS DRY RUN**

(Since the retain hours is less than the default value of 7 days, we need to also set a databricks property value as below)

set spark.databricks.delta.retentionDurationCheck.enabled = false

Optimized writes

Performs auto-compaction of data leading to auto-optimization.

Ex : This can be achieved by setting the following property

TBLPROPERTIES (delta.autoOptimize.optimizeWrite = true)

- This ensures that if there are many small files getting generated, the respective tasks automatically combine them to form larger files with a slight write overhead.
- It creates bigger files before writing to the disk

Auto compact

TBLPROPERTIES (delta.autoOptimize.autoCompact = true)

- Once the files are written to the disk, the small files are compacted to form larger files.
- Auto compact works only when there are more than 50 small files.

Photon Query Engine

- Photon is a native vectorized engine developed in C++
- Since it uses low level languages, it helps in achieving hardware level optimizations and benefits.
- This engine dramatically improves the query performance.
- Since some of the parts of the spark engine are rewritten using C++ to achieve hardware benefits and thereby provide the best performance.

What kind of queries achieve benefits with Photon Query Engine?

1. Photon query engine is meant for Compute intensive queries.
2. Queries which are short will not gain much performance benefits with the Photon query engine.

Enabling the Photon Query Engine

Enable "Use Photo Acceleration" option while creating the cluster