

Azure Assignment-4

Name: Priyeshwar

Mail: priyesh2664@gmail.com

Databases vs. Data Warehouses vs. Data Lakes

1. Database

A database is a structured collection of data designed to support Online Transaction Processing (OLTP) — fast inserts, updates, and queries for operational systems.

Characteristics

- Stores structured and sometimes semi-structured data.
- Supports ACID transactions for data integrity.
- Optimized for small, frequent read/write operations.
- Can be relational (SQL) or non-relational (NoSQL).

Common Uses

- Application backends (e.g., e-commerce carts, banking systems).
- Storing operational records (e.g., customers, orders, inventory).

Examples

- Relational: MySQL, PostgreSQL, Oracle, SQL Server.
 - NoSQL: MongoDB, Cassandra, Redis.
-

2. Data Warehouse

A data warehouse is a centralized repository optimized for Online Analytical Processing (OLAP) — querying and analyzing large volumes of structured, historical data from multiple sources.

Characteristics

- Stores structured and some semi-structured data.
- Requires a predefined schema (*schema-on-write*).
- Data loaded via ETL (Extract-Transform-Load) or ELT processes.
- Optimized for analytical queries, aggregations, and reporting.
- Data may be updated periodically (not real-time).

Common Uses

- Business Intelligence (BI) dashboards.
- Historical trend analysis.
- KPI reporting.

Examples

- Snowflake, Amazon Redshift, Google BigQuery, Azure Synapse.
-

3. Data Lake

A data lake is a large-scale storage repository for raw data in its original format, supporting a wide range of analytics and machine learning use cases.

Characteristics

- Stores structured, semi-structured, and unstructured data (e.g., JSON, logs, images, videos).
- No schema required at ingestion (*schema-on-read*).
- Highly scalable and cost-effective.
- Can integrate with multiple processing frameworks (e.g., Spark, Presto, Hive).
- Suitable for both batch and streaming data.

Common Uses

- Data science and machine learning workloads.
- Exploratory analytics.
- Long-term, low-cost data archiving.

Examples

- Storage: AWS S3, Azure Data Lake Storage, Google Cloud Storage.
 - Query Engines: Databricks, AWS Athena, Presto.
-

Comparison Table

Feature	Database	Data Warehouse	Data Lake
Main Workload	Operational (OLTP)	Analytical (OLAP)	Analytical / ML
Data Type	Structured / Semi-structured	Structured / Semi-structured	All types (structured, semi, unstructured)
Schema	Fixed or flexible	Fixed, predefined (<i>schema-on-write</i>)	None until read (<i>schema-on-read</i>)
Data Freshness	Real-time	Batch updates	Variable (real-time to batch)
Primary Users	Developers	BI analysts, data scientists	Data scientists, engineers
Strengths	Fast transactions	High-performance analytics	Flexible, cheap, scalable

Feature	Database	Data Warehouse	Data Lake
Limitations	Limited analytics	Rigid schema, ETL required	Requires prep before analysis

Create a table

```

Python 03:44 PM (15s) 1
%python
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, TimestampType

schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("firstName", StringType(), True),
    StructField("middleName", StringType(), True),
    StructField("lastName", StringType(), True),
    StructField("gender", StringType(), True),
    StructField("birthDate", TimestampType(), True),
    StructField("ssn", StringType(), True),
    StructField("salary", IntegerType(), True)
])

df = spark.read.format("csv").option("header", True).schema(schema).load("dbfs:/mnt/hexafile/export.csv")

# Save the DataFrame as a table with overwrite mode
df.write.mode("overwrite").saveAsTable("people_10m")

(1) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

```

Create a Delta table

```

Python Just now (2s) 2
from delta.tables import DeltaTable

DeltaTable.createIfNotExists(spark).tableName("people_10m").addColumn("id", "INT").addColumn(
    "firstName", "STRING").addColumn("middleName", "STRING").addColumn("lastName", "STRING", comment =
    "surname").addColumn("gender", "STRING").addColumn("birthDate", "TIMESTAMP").addColumn("ssn",
    "STRING").addColumn("salary", "INT").execute()

<delta.tables.DeltaTable at 0x7fc1806a5d30>

```

Merge & Upsert to a table

To merge a set of updates and insertions into an existing Delta table, you use the `DeltaTable.merge` method for Python and Scala, and the `MERGE INTO` statement for SQL. For example, the following example takes data from the source table and merges it into the target Delta table. When there is a matching row in both tables, Delta Lake updates the data column using the given expression. When there is no matching row, Delta Lake adds a new row. This operation is known as an upsert.

```
%%python
from pyspark.sql.types import StructType, StructField, StringType, IntegerType, DateType
from datetime import date

schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("firstName", StringType(), True),
    StructField("middleName", StringType(), True),
    StructField("lastName", StringType(), True),
    StructField("gender", StringType(), True),
    StructField("birthDate", DateType(), True),
    StructField("ssn", StringType(), True),
    StructField("salary", IntegerType(), True)
])

data = [
    (99999998, 'Billy', 'Tommie', 'Luppitt', 'M', date.fromisoformat('1992-09-17'), '953-38-9452', 55250),
    (99999999, 'Elias', 'Cyril', 'Leadbetter', 'M', date.fromisoformat('1984-05-22'), '906-51-2137', 48500),
    (10000000, 'Joshua', 'Chas', 'Broggio', 'M', date.fromisoformat('1968-07-22'), '988-61-6247', 90000),
    (20000001, 'John', '', 'Doe', 'M', date.fromisoformat('1978-01-14'), '345-67-8901', 55500),
    (20000002, 'Mary', '', 'Smith', 'F', date.fromisoformat('1982-10-29'), '456-78-9012', 98250),
    (20000003, 'Jane', '', 'Doe', 'F', date.fromisoformat('1981-06-25'), '567-89-0123', 89900)
]

people_10m_updates = spark.createDataFrame(data, schema)
people_10m_updates.createOrReplaceTempView("people_10m_updates")

# ...
```

```
from delta.tables import DeltaTable

deltaTable = DeltaTable.forName(spark, 'people_10m')

(deltaTable.alias("people_10m")
 .merge(
     people_10m_updates.alias("people_10m_updates"),
     "people_10m.id = people_10m_updates.id")
 .whenMatchedUpdateAll()
 .whenNotMatchedInsertAll()
 .execute()
)
```

▶

▼

02:37 PM (1s)

3

```
df = spark.read.table("people_10m")
df_filtered = df.filter(df["id"] >= 9999998)
display(df_filtered)
```

(1) Spark Jobs

▶

df: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

▶

df_filtered: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

Read a table

▶

▼

02:37 PM (1s)

3

Python

⚡

🔍

⋮

```
df = spark.read.table("people_10m")
df_filtered = df.filter(df["id"] >= 9999998)
display(df_filtered)
```

(1) Spark Jobs

▶

df: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

▶

df_filtered: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

Table ▼

+

🔍

🔍

🔍

🔍

	id	firstName	middleName	lastName	gender	birthDate	ssn	salary
1	9999999	Elias	Cyril	Leadbetter	M	1984-05-22T00:00:00.000+00:00	906-51-2137	48500
2	10000000	Joshua	Chas	Broggio	M	1968-07-22T00:00:00.000+00:00	988-61-6247	90000
3	20000002	Mary		Smith	F	1982-10-29T00:00:00.000+00:00	456-78-9012	98250
4	20000003	Jane		Doe	F	1981-06-25T00:00:00.000+00:00	567-89-0123	89900
5	9999998	Billy	Tommie	Luppitt	M	1992-09-17T00:00:00.000+00:00	953-38-9452	55250
6	20000001	John		Doe	M	1978-01-14T00:00:00.000+00:00	345-67-8901	55500

6 rows | 0.71s runtime

Refreshed 3 hours ago

▶

▼

02:38 PM (1s)

4

Python

⚡

🔍

⋮

```
people_df = spark.read.table("people_10m")
display(people_df)
```

(2) Spark Jobs

▶

people_df: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

Table ▼

+

🔍

🔍

🔍

🔍

Write to a table

```
▶ ✓ 02:39 PM (1s) 5

df.write.mode("append").saveAsTable("people_10m")

▶ (1) Spark Jobs
```

```
▶ ✓ 02:40 PM (2s)

df.write.mode("overwrite").saveAsTable("people_10m")

▶ (1) Spark Jobs
```

Update a table

```
▶ ✓ 02:41 PM (6s) 7

from delta.tables import *
from pyspark.sql.functions import *

deltaTable = DeltaTable.forName(spark, "people_10m")

# Declare the predicate by using a SQL-formatted string.
deltaTable.update(
    condition = "gender = 'F'",
    set = { "gender": "'Female'" }
)

# Declare the predicate by using Spark SQL functions.
deltaTable.update(
    condition = col('gender') == 'M',
    set = { 'gender': lit('Male') }
)

▶ (14) Spark Jobs
```

Display table history

02:42 PM (1s) 8

```
from delta.tables import *

deltaTable = DeltaTable.forName(spark, "people_10m")
display(deltaTable.history())
```

(1) Spark Jobs

	version	timestamp	userId	userName	operation	operationParameters
2	7	2025-08-11T09:11:27.000+00:00	141397643554594	azuser4022_mml.local@techademy.co...	UPDATE	> {"predicate":"(gender
3	6	2025-08-11T09:11:23.000+00:00	141397643554594	azuser4022_mml.local@techademy.co...	UPDATE	> {"predicate":"(gender
4	5	2025-08-11T09:10:30.000+00:00	141397643554594	azuser4022_mml.local@techademy.co...	CREATE OR REPLACE TABLE AS SELECT	> {"partitionBy":[""],"clust
5	4	2025-08-11T09:09:45.000+00:00	141397643554594	azuser4022_mml.local@techademy.co...	WRITE	> {"mode":"Append","stat
6	3	2025-08-11T09:07:07.000+00:00	141397643554594	azuser4022_mml.local@techademy.co...	MERGE	> {"predicate":"(id#257
7	2	2025-08-11T09:05:22.000+00:00	141397643554594	azuser4022_mml.local@techademy.co...	CREATE OR REPLACE TABLE AS SELECT	> {"partitionBy":[""],"clust
8	1	2025-08-11T08:56:04.000+00:00	141397643554594	azuser4022_mml.local@techademy.co...	MERGE	> {"predicate":"(id#123
9	0	2025-08-11T07:18:35.000+00:00	141397643554594	azuser4022_mml.local@techademy.co...	CREATE TABLE AS SELECT	> {"partitionBy":[""],"clust

9 rows | 1.40s runtime Refreshed 3 hours ago

Query an earlier version of the table (time travel)

02:43 PM (2s) 9

```
from delta.tables import *

deltaTable = DeltaTable.forName(spark, "people_10m")
deltaHistory = deltaTable.history()

display(deltaHistory.where("version == 0"))
# Or:
display(deltaHistory.where("timestamp == '2024-05-15T22:43:15.000+00:00'"))
```

(3) Spark Jobs

deltaHistory: pyspark.sql.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]

	version	timestamp	userId	userName	operation	operationParameters
1	0	2025-08-11T07:18:35.000+00:00	141397643554594	azuser4022_mml.local@techademy.co...	CREATE TABLE AS SELECT	> {"partitionBy":[""],"clusterBy":[""],"descri

1 row | 1.53s runtime Refreshed 3 hours ago

02:44 PM (1s) 10

```
df = spark.read.option('versionAsOf', 0).table("people_10m")
# Or:
df = spark.read.option('timestampAsOf', '2025-08-11T07:18:35.000+00:00').table("people_10m")

display(df)
```

(1) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [id: integer, firstName: string ... 6 more fields]

	id	firstName	middleName	lastName	gender	birthDate	ssn	salary
1	8807768	Darren	Minh	Pilsworth	M	1967-03-07T05:00:00.000+00:...	908-34-1027	45983
2	8807769	Winston	Hiram	Ahlin	M	1957-07-10T04:00:00.000+00:...	946-32-4802	45867
3	8807770	Arnold	Jason	Durrant	M	1987-10-18T04:00:00.000+00:...	978-34-4842	58623
4	8807771	Garth	Dino	Eddins	M	1998-07-16T04:00:00.000+00:...	992-39-5689	33952
5	8807772	Sebastian	Levi	Evans	M	1958-01-23T05:00:00.000+00:...	963-59-2493	31453
6	8807773	Fritz	Elisha	Grisley	M	1957-09-02T04:00:00.000+00:...	980-27-9156	99236
7	8807774	Aurelio	Porter	Tetford	M	1981-05-27T04:00:00.000+00:...	905-72-1685	73241
8	8807775	Salvador	Timothy	Meneo	M	1990-08-07T04:00:00.000+00:...	927-35-8548	56896
9	8807776	Brett	Felix	Caunce	M	1984-04-21T05:00:00.000+00:...	945-33-2714	88297
10	8807777	Steve	Edwardo	Dobbing	M	1954-07-03T04:00:00.000+00:...	666-46-9615	65176
11	8807778	Mariano	Angelo	Van der Beek	M	1971-05-26T04:00:00.000+00:...	930-11-2586	73224
12	8807779	Jordan	Mohammad	Brownhill	M	1963-04-24T05:00:00.000+00:...	942-72-5098	84630

Optimize a table

After you have performed multiple changes to a table, you might have a lot of small files. To improve the speed of read queries, you can use the optimize operation to collapse small files into larger ones:

02:45 PM (8s) 11

```
from delta.tables import *

deltaTable = DeltaTable.forName(spark, "people_10m")
deltaTable.optimize().executeCompaction()
```

(6) Spark Jobs

```
mergedFiles:struct<num:bigint,size:bigint>,numOutputCubes:bigint,mergedNumCubes:bigint>,clusteringStats:struct<inputZCubeFiles:struct<numFiles:bigint,size:bigint>,inputOtherFiles:struct<numFiles:bigint,size:bigint>,inputNumZCubes:bigint,mergedFiles:struct<numFiles:bigint,size:bigint>,numOutputZCubes:bigint>,numBins:bigint,numBatches:bigint,totalConsideredFiles:bigint,totalFilesSkipped:bigint,preserveInsertionOrder:boolean,numFilesSkippedToReduceWriteAmplification:bigint,numBytesSkippedToReduceWriteAmplification:bigint,startTimeMs:bigint,endTimeMs:bigint,totalClusterParallelism:bigint,totalScheduledTasks:bigint,autoCompactParallelismStats:struct<maxClusterActiveParallelism:bigint,minClusterActiveParallelism:bigint,maxSessionActiveParallelism:bigint,minSessionActiveParallelism:bigint>,deletionVectorStats:struct<numDeletionVectorsRemoved:bigint,numDeletionVectorRowsRemoved:bigint>,recompressionCodec:string,numTableColumns:bigint,numTableColumnsWithStats:bigint,totalTaskExecutionTimeMs:bigint,skippedArchivedFiles:bigint>,clusteringMetrics:struct<sizeOfTableInBytesBeforeLazyClustering:bigint,isNewMetadataCreated:boolean,isPOTtriggered:boolean,isFull:boolean,approxClusteringQuality:double,approxClusteringQualityPerColumn:array<double>,approxClusteringCoverage:double,numFilesSkippedWithoutStats:bigint,numFilesClassifiedToIntermediateNodes:bigint,sizeOfFilesClassifiedToIntermediateNodesInBytes:bigint,logicalSizeOfFilesClassifiedToIntermediateNodesInBytes:bigint,numFilesClassifiedToLeafNodes:bigint,sizeOfFilesClassifiedToLeafNodesInBytes:bigint,logicalSizeOfFilesClassifiedToLeafNodesInBytes:bigint,numThreadsForClassifier:int,clusterThresholdStrategy:string,minFileSize:bigint,maxFileSize:bigint,nodeMinNumFilesToCompact:bigint,numIdealFiles:bigint,numIdealFilesWithTrimmedStringMaxValue:bigint,numAddedFilesWithSameMinMaxOnClusteringColumns:array<bigint>,numClusteringTasksPlanned:int,numClusteringTasksNotPlannedDueToPO:int,numCompactionTasksPlanned:int,numCompactionTasksPlannedUndoneDueToPO:int,numOptimizeBatchesPlanned:int,numLeafNodesExpanded:bigint,numLeafNodesClustered:bigint,numGetFilesForNodeCalls:bigint,numSamplingJobs:bigint,numLeafNodesCompacted:bigint,numLeafNodesCompactedUndoneDueToPO:bigint,numIntermediateNodesCompacted:bigint,numIntermediateNodesCompactedUndoneDueToPO:bigint,totalSizeOfDataToCompactInBytes:bigint,totalSizeOfDataToCompactInBytesUndoneDueToPO:bigint,totalLogicalSizeOfDataToCompactInBytes:bigint,totalLogicalSizeOfDataToCompactInBytesUndoneDueToPO:bigint,numIntermediateNodesClustered:bigint,numFilesSkippedAfterExpansion:bigint,totalSizeOfFilesSkippedAfterExpansionInBytes:bigint,totalLogicalSizeOfFilesSkippedAfterExpansionInBytes:bigint,totalSizeOfDataToRewriteInBytes:bigint,totalLogicalSizeOfDataToRewriteInBytes:bigint,timeMetrics:struct<classifierTimeMs:bigint,optimizerTimeMs:bigint,metadataLoadTimeMs:bigint,totalGetFilesForNodeCallsTimeMs:bigint,totalSamplingTimeMs:bigint,metadataCreationTimeMs:bigint>,maxOptimizeBatchesInParallel:bigint,currentIteration:int,maxIterations:int,clusteringStrategy:string>>]
```

Z-order by columns

To improve read performance further, you can collocate related information in the same set of files by z-ordering. Delta Lake data-skipping algorithms use this collocation to dramatically reduce the

amount of data that needs to be read. To z-order data, you specify the columns to order on in the z-order by operation. For example, to collocate by gender, run:

```
02:45 PM (2s) 12

from delta.tables import *

deltaTable = DeltaTable.forName(spark, "people_10m")
deltaTable.optimize().executeZOrderBy("gender")

(4) Spark Jobs
mergedFiles:struct<num:bigint,size:bigint>,numOutputCubes:bigint,mergedNumCubes:bigint>,clusteringStats:struct<inputZCubeFiles:struct<numFiles:bigint,size:bigint>,inputOtherFiles:struct<numFiles:bigint,size:bigint>,inputNumZCubes:bigint,mergedFiles:struct<numFiles:bigint,size:bigint>,numOutputZCubes:bigint>,numBins:bigint,numBatches:bigint,totalConsideredFiles:bigint,totalFilesSkipped:bigint,preserveInsertionOrder:boolean,numFilesSkippedToReduceWriteAmplification:bigint,numBytesSkippedToReduceWriteAmplification:bigint,startTimeMs:bigint,endTimeMs:bigint,totalClusterParallelism:bigint,totalScheduledTasks:bigint,autoCompactParallelismStats:struct<maxClusterActiveParallelism:bigint,minClusterActiveParallelism:bigint,maxSessionActiveParallelism:bigint,minSessionActiveParallelism:bigint>,deletionVectorStats:struct<numDeletionVectorsRemoved:bigint,numDeletionVectorRowsRemoved:bigint>,recompressionCodec:string,numTableColumns:bigint,numTableColumnsWithStats:bigint,totalTaskExecutionTimeMs:bigint,skippedArchivedFiles:bigint,clusteringMetrics:struct<sizeOfTableInBytesBeforeLazyClustering:bigint,isNewMetadataCreated:boolean,isPOTriggered:boolean,isFull:boolean,approxClusteringQuality:double,approxClusteringQualityPerColumn:array<double>,approxClusteringCoverage:double,numFilesSkippedWithoutStats:bigint,numFilesClassifiedToIntermediateNodes:bigint,sizeOfFilesClassifiedToIntermediateNodesInBytes:bigint,logicalSizeOfFilesClassifiedToIntermediateNodesInBytes:bigint,numFilesClassifiedToLeafNodes:bigint,sizeOfFilesClassifiedToLeafNodesInBytes:bigint,logicalSizeOfFilesClassifiedToLeafNodesInBytes:bigint,numThreadsForClassifier:int,clusterThresholdStrategy:string,minFileSize:bigint,maxFileSize:bigint,nodeMinNumFilesToCompact:bigint,numIdealFiles:bigint,numIdealFilesWithTrimmedStringMaxValue:bigint,numAddedFilesWithSameMinMaxOnClusteringColumns:array<bigint>,numClusteringTasksPlanned:int,numClusteringTasksNotPlannedDueToPO:int,numCompactionTasksPlanned:int,numCompactionTasksPlannedUndoneDueToPO:int,numOptimizeBatchesPlanned:int,numLeafNodesExpanded:bigint,numLeafNodesClustered:bigint,numGetFilesForNodeCalls:bigint,numSamplingJobs:bigint,numLeafNodesCompacted:bigint,numLeafNodesCompactedUndoneDueToPO:bigint,numIntermediateNodesCompacted:bigint,numIntermediateNodesCompactedUndoneDueToPO:bigint,totalSizeOfDataToCompactInBytes:bigint,totalSizeOfDataToCompactInBytesUndoneDueToPO:bigint,totalLogicalSizeOfDataToCompactInBytes:bigint,totalLogicalSizeOfDataToCompactInBytesUndoneDueToPO:bigint,numIntermediateNodesClustered:bigint,numFilesSkippedAfterExpansion:bigint,totalSizeOfFilesSkippedAfterExpansionInBytes:bigint,totalLogicalSizeOfFilesSkippedAfterExpansionInBytes:bigint,totalSizeOfDataToRewriteInBytes:bigint,totalLogicalSizeOfDataToRewriteInBytes:bigint,timeMetrics:struct<classifierTimeMs:bigint,optimizerTimeMs:bigint,metadataLoadTimeMs:bigint,totalGetFilesForNodeCallsTimeMs:bigint,totalSamplingTimeMs:bigint,metadataCreationTimeMs:bigint>,maxOptimizeBatchesInParallel:bigint,currentIteration:int,maxIterations:int,clusteringStrategy:string>>]
```

Clean up snapshots with VACUUM

```
02:46 PM (43s)

from delta.tables import *

deltaTable = DeltaTable.forName(spark, "people_10m")
deltaTable.vacuum()

(18) Spark Jobs

DataFrame[]
```