



# Data Engineering

## DELTA

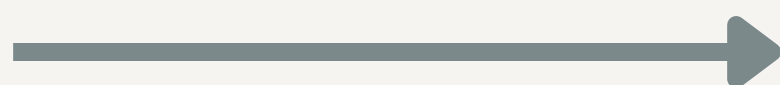
ALL CONCEPTS TO GET STARTED



# DELTA



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Lakehouse Architecture

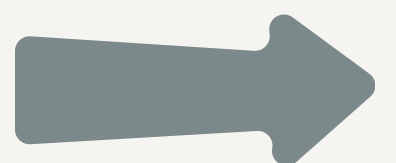
The Lakehouse combines the best aspects of data lakes and data warehouses, providing flexibility, cost-efficiency, and scale with ACID transactions and schema enforcement.

*Example*

```
spark.sql("CREATE TABLE delta./path/to/table (id INT, name  
STRING, age INT) USING DELTA")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

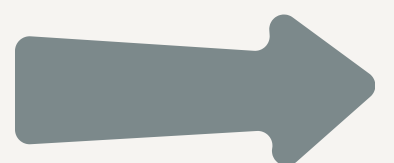


# Medallion Architecture

A design pattern for organizing data in a lakehouse into bronze, silver, and gold layers for progressively refined datasets.

*Example*

```
bronzeDF = spark.read.format("delta").load("/path/to/bronze")  
silverDF = bronzeDF.filter("value IS NOT NULL")  
silverDF.write.format("delta").mode("overwrite").save("/path/to/silver")
```

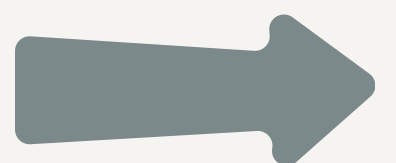


# Transaction Support

Delta Lake supports ACID transactions to ensure data reliability and consistency.

*Example*

```
from delta.tables import * deltaTable =  
DeltaTable.forPath(spark, "/path/to/delta-table")  
deltaTable.update("id = 1", { "name": "Updated Name" })
```



# Schema Enforcement

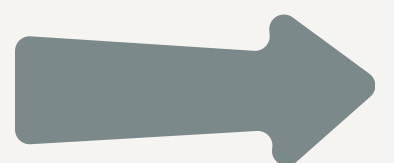
Delta Lake ensures data integrity with schema enforcement, which prevents the insertion of data that does not match the table schema.

*Example*

```
spark.sql("ALTER TABLE delta.ypath/to/table ADD COLUMNS  
(new_col STRING)")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

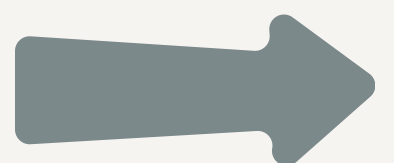


# Schema Evolution

Allows for the modification of schema as data changes over time, supporting additions and updates to schema.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
df.write.format("delta")  
.option("mergeSchema", "true")  
.mode("overwrite").save("/path/to/delta-table")
```

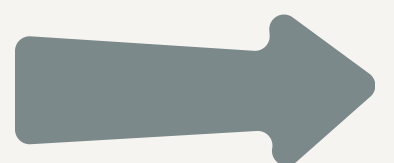


# Batch Processing

Delta Lake handles batch processing with high reliability and scalability.

*Example*

```
df = spark.read.format("csv").option("header",  
"true").load("/path/to/csv")  
df.write.format("delta").mode("append")  
.save("/path/to/delta-table")
```



# Stream Processing

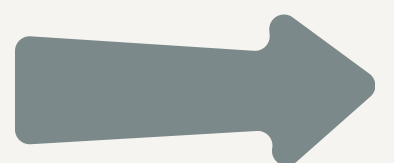
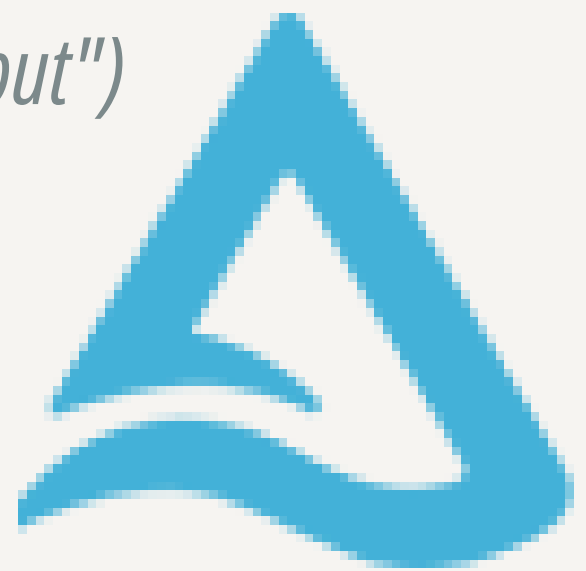
Delta Lake supports stream processing, allowing continuous data ingestion and processing.

*Example*

*streamingDF =*

*spark.readStream.format("delta").load("/path/to/streaming")*

*streamingDF.writeStream.format("delta").option("checkpointLocation", "/path/to/checkpoint").start("/path/to/output")*





# Time Travel

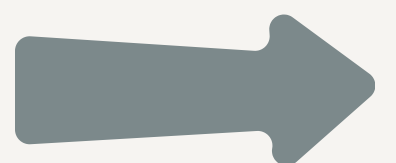
Delta Lake allows querying previous versions of data using time travel.

*Example*

```
df = spark.read.format("delta")  
.option("versionAsOf", 1).load("/path/to/delta-table") df.show()
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

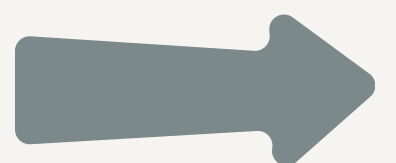


# Optimized Writes

Delta Lake optimizes writes to ensure efficient storage and performance.

*Example*

```
df.write.format("delta") \  
.option("dataChange", "false") \  
.mode("append").save("/path/to/delta-table")
```



# Z-Ordering

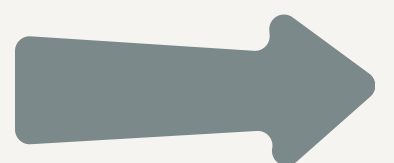
A technique to colocate related information in the same set of files to improve query performance.

*Example*

```
deltaTable.optimize().executeZOrderBy("column_name")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

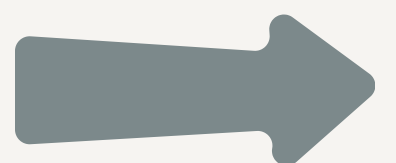


# Partitioning

Delta Lake supports partitioning, which can improve performance by limiting the amount of data read during queries.

*Example*

```
df.write.format("delta").partitionBy("column_name") \  
.save("/path/to/delta-table")
```

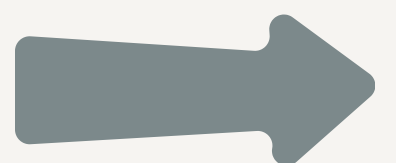


# Delta Log

A transaction log that tracks all changes made to data, providing a full audit trail.

*Example*

```
logData = spark.read.format("delta").json("/path/to/delta-table/_delta_log/00000000000000000000000010.json") logData.show()
```



# Vacuum

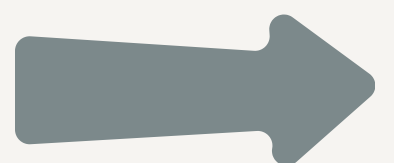
Delta Lake's vacuum operation removes old data files and frees up storage.

*Example*

*`deltaTable.vacuum(168)`*



Shwetank Singh  
GritSetGrow - GSGLearn.com

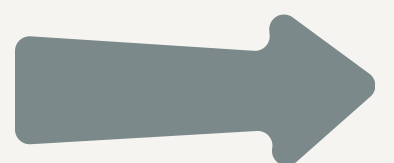


# Concurrency Control

Delta Lake supports optimistic concurrency control, ensuring multiple users can read and write data simultaneously without conflict.



Shwetank Singh  
GritSetGrow - GSGLearn.com

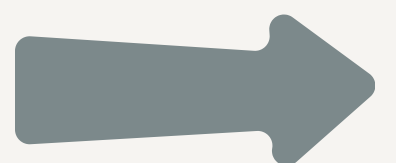


# Change Data Feed

Delta Lake allows capturing changes in data for audit and compliance purposes.

*Example*

```
df = spark.read.format("delta").option("readChangeFeed",  
"true").table("my_table") df.show()
```



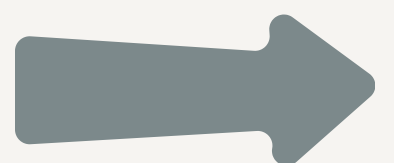


# ACID Transactions

Ensures reliable transactions that are Atomic, Consistent, Isolated, and Durable.



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Lakehouse Security

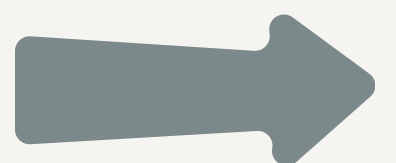
Implements security features like row-level security and data masking.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
df.createOrReplaceTempView("table_view") spark.sql("SELECT *  
FROM table_view WHERE role = 'admin'")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

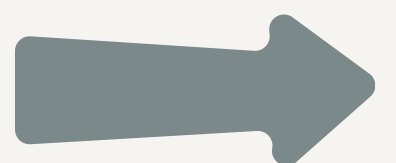


# Data Quality

Ensures high data quality through features like data validation and cleansing.

*Example*

```
from pyspark.sql.functions import *  
df = spark.read.format("delta").load("/path/to/delta-table")  
cleanDF = df.filter(col("value").isNotNull())  
cleanDF.write.format("delta").save("/path/to/cleaned-delta-table")
```



# Data Governance

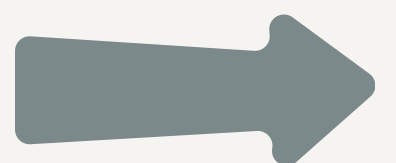
Delta Lake supports robust data governance to manage data privacy and compliance.

*Example*

```
spark.sql("ALTER TABLE delta.ypath/to/table ADD CONSTRAINT  
value_check CHECK (value > 0)")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Data Sharing

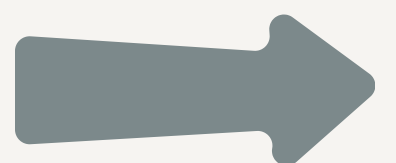
Delta Lake allows secure data sharing across organizational boundaries.

*Example*

```
df.write.format("delta").save("/path/to/shared-delta-table")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Delta Cache

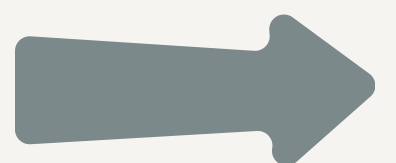
Improves read performance by caching data.

*Example*

```
spark.conf.set("spark.databricks.io.cache.enabled", "true")  
df = spark.read.format("delta").load("/path/to/delta-table")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

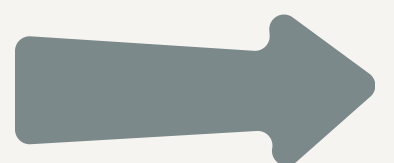


# Compaction

Reduces the number of small files to improve performance.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
df.repartition(1).write.format("delta")  
.mode("overwrite").save("/path/to/delta-table")
```

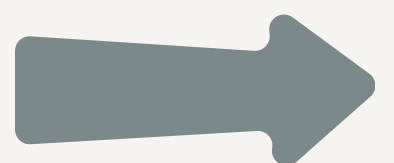


# Data Skipping

Data skipping is a technique that reduces the amount of data read by skipping over files that do not match the query criteria.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
df.createOrReplaceTempView("table_view") spark.sql("SELECT *  
FROM table_view WHERE date > '2023-01-01'")
```



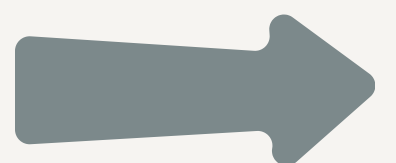


# Bloom Filter Index

A Bloom filter index helps in reducing the amount of data scanned by filtering out non-matching data quickly.

*Example*

```
spark.sql("CREATE BLOOMFILTER INDEX idx_bloom ON  
delta./path/to/delta-table (email)")
```

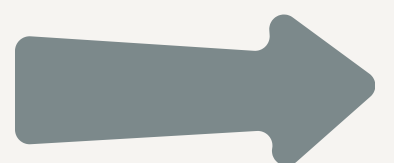


# Optimize Command

The optimize command helps to compact small files into larger ones, improving read performance.

*Example*

```
from delta.tables import * deltaTable =  
DeltaTable.forPath(spark, "/path/to/delta-table")  
deltaTable.optimize().executeCompaction()
```

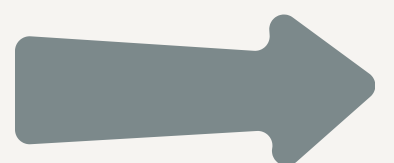


# File Statistics

File statistics provide metadata that helps in data skipping and improves query performance.

*Example*

```
deltaTable = DeltaTable.forPath(spark, "/path/to/delta-table")  
stats = deltaTable.history().select("operationMetrics")  
.where("version = 0") stats.show()
```

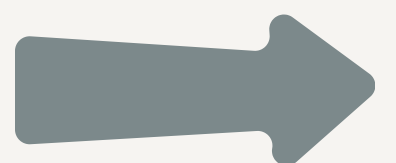


# Cluster By

Cluster by is a technique to organize data based on certain columns to optimize query performance.

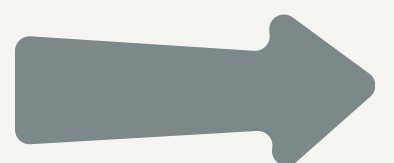
*Example*

```
df.write.format("delta")  
.option("clusterBy", "column_name").save("/path/to/delta-  
table")
```



# Partition Pruning

Partition pruning helps in skipping irrelevant partitions, thereby reducing the amount of data read during queries.

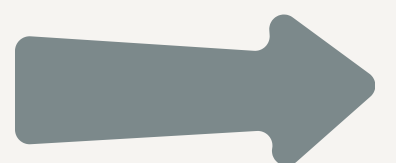


# Schema Validation

Schema validation ensures that the data being written to the Delta Lake matches the expected schema.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
df.write.format("delta").option("mergeSchema",  
"true").save("/path/to/delta-table")
```

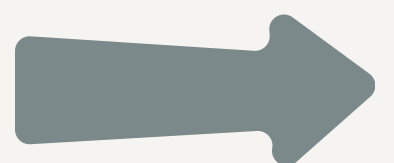


# Delta Log Analysis

Analyzing the Delta log helps in understanding the changes made to the data over time.

*Example*

```
logData = spark.read.format("delta").json("/path/to/delta-table/_delta_log/00000000000000000000000010.json") logData.show()
```



# Incremental Data Processing

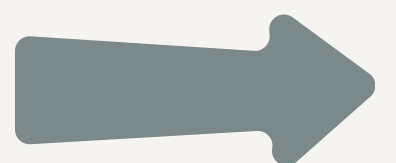
Incremental data processing allows processing only new or changed data, improving efficiency.

*Example*

```
df = spark.read.format("delta")  
.option("startingVersion", "1").load("/path/to/delta-table")  
df.show()
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



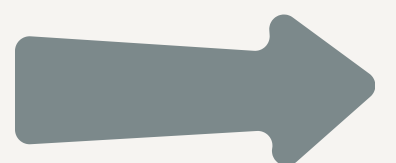


# Delta Cache

Delta cache improves read performance by caching data in memory.

*Example*

```
spark.conf.set("spark.databricks.io.cache.enabled", "true")  
df = spark.read.format("delta").load("/path/to/delta-table")
```

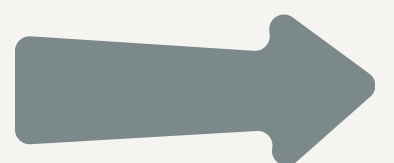


# Delta Sharing Protocol

A protocol for secure data sharing across organizations, enabling direct access to Delta tables without the need for data export.



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Support for Transactional Streaming

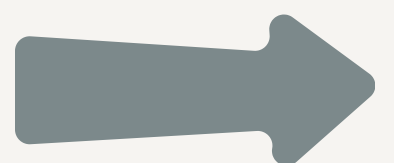
Delta Lake allows seamless switching between batch and streaming operations, providing consistent and reliable data access.

*Example*

*streamingDF =*

*spark.readStream.format("delta").load("/path/to/delta-table")*

*streamingDF.writeStream.format("delta").outputMode("append").start("/path/to/output-table")*

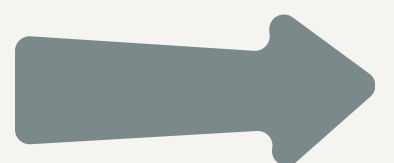


# Data Quality Management

Delta Lake ensures high data quality through schema enforcement, validation, and data lineage tracking.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
cleanDF = df.filter("value IS NOT NULL")  
cleanDF.write.format("delta").mode("overwrite").save("/path/to/  
/clean-delta-table")
```



# Streaming Delta as Source

Using Delta Lake as a source for streaming data, enabling real-time data processing and analysis.

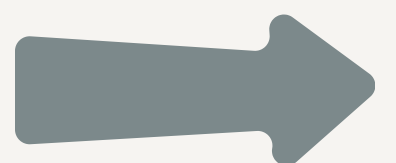
*Example*

*streamingDF =*

*spark.readStream.format("delta").load("/path/to/delta-table")*



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Streaming Delta as Sink

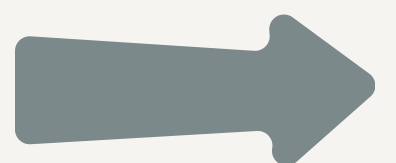
Using Delta Lake as a sink for streaming data, supporting high-volume data ingestion and processing.

*Example*

```
streamingDF.writeStream.format("delta").outputMode("append")  
    .start("/path/to/delta-table")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

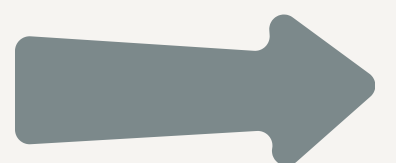


# Compaction and Optimize

Techniques to merge small files into larger ones to improve read performance in Delta Lake.

*Example*

```
deltaTable = DeltaTable.forPath(spark, "/path/to/delta-table")  
deltaTable.optimize().executeCompaction()
```

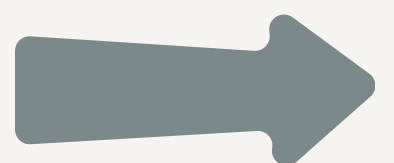


# Z-Ordering

A technique to colocate related data in the same set of files, improving query performance through data skipping.

*Example*

```
deltaTable.optimize().executeZOrderBy("column_name")
```





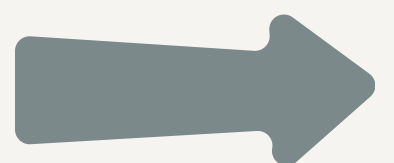
# Data Lineage

Tracks the data journey through various transformations in Delta Lake, ensuring traceability and auditability.

*Example*

*lineageDF =*

*spark.read.format("delta").option("readChangeData",  
"true").table("my\_table") lineageDF.show()*

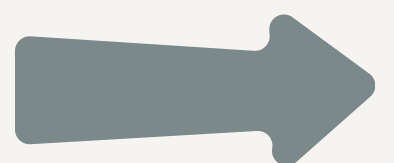


# Data Encryption

Delta Lake supports data encryption to ensure data security and compliance with regulations.

*Example*

```
spark.conf.set("spark.sql.files.encryption.key",  
"your_encryption_key")  
df.write.format("delta").save("/path/to/encrypted-delta-table")
```

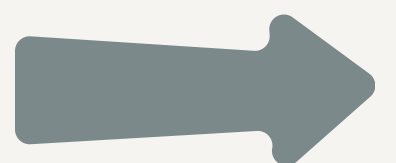


# File Compaction

Reduces the number of small files in Delta Lake, improving read and write performance.

*Example*

```
deltaTable = DeltaTable.forPath(spark, "/path/to/delta-table")  
deltaTable.optimize().executeCompaction()
```



# Automatic Optimization

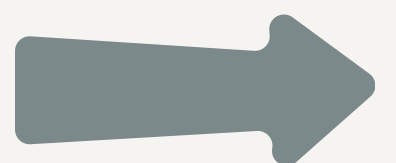
Delta Lake can automatically optimize data layout for better performance.

*Example*

```
spark.conf.set("spark.databricks.delta.optimizeWrites.enabled",  
"true")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Delta Log Maintenance

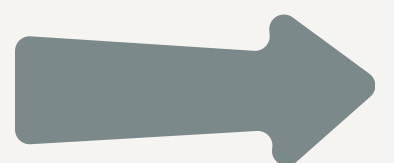
Regular maintenance of the Delta log ensures optimal performance and prevents issues.

*Example*

*`deltaTable.vacuum(168)`*



Shwetank Singh  
GritSetGrow - GSGLearn.com

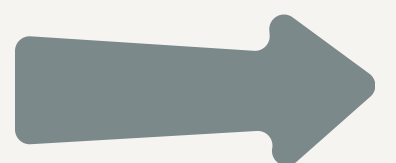


# Metadata Management

Efficient metadata management in Delta Lake ensures quick access to table properties and statistics.

*Example*

```
deltaTable = DeltaTable.forPath(spark, "/path/to/delta-table")  
metadata = deltaTable.history() metadata.show()
```

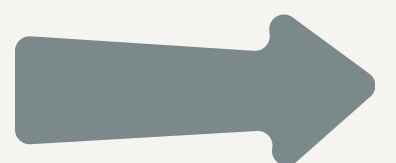


# Checkpointing

Checkpointing helps in recovering from failures by saving the state of streaming applications.

*Example*

```
streamingDF.writeStream.format("delta")  
.option("checkpointLocation", "/path/to/checkpoint")  
.start("/path/to/delta-table")
```



# Dynamic Partition Pruning

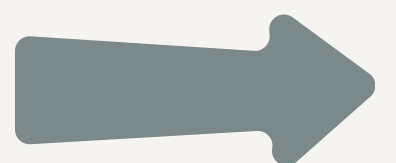
Enhances query performance by dynamically pruning partitions that are not needed for the query.

*Example*

```
spark.conf.set("spark.sql.optimizer.dynamicPartitionPruning.enabled", "true")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com





# Query Acceleration

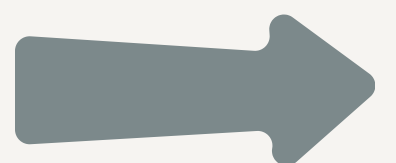
Delta Lake supports query acceleration features to speed up data retrieval.

*Example*

```
spark.conf.set("spark.databricks.delta.optimizeWrites", "true")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Adaptive Query Execution (AQE)

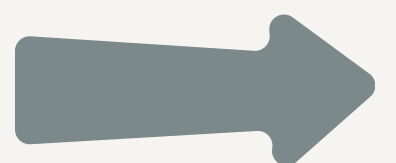
Automatically optimizes query plans at runtime based on runtime statistics.

*Example*

```
spark.conf.set("spark.sql.adaptive.enabled", "true")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Scalable Metadata Handling

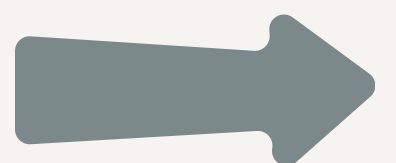
Delta Lake efficiently handles metadata for large tables to maintain performance.

*Example*

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



Shwetank Singh  
GritSetGrow - GSGLearn.com

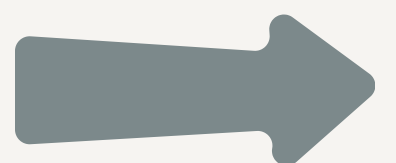


# Custom Metrics

Delta Lake allows custom metrics to track performance and diagnose issues.

*Example*

```
streamingDF.writeStream.format("delta")  
.option("checkpointLocation", "/path/to/checkpoint")  
.foreachBatch(writeToDeltaLake).start("/path/to/delta-table")
```



# Delta Lake on Kubernetes

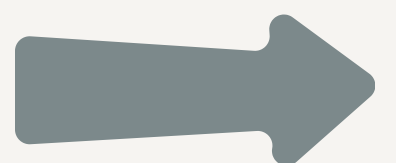
Delta Lake can be deployed on Kubernetes for scalable and resilient data processing.

*Example*

```
spark.conf.set("spark.kubernetes.container.image", "delta-lake-image")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

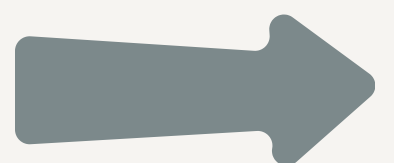


# Cross-region Replication

Supports replicating Delta tables across different regions for disaster recovery and data locality.

*Example*

```
deltaTable = DeltaTable.forPath(spark, "/path/to/delta-table")  
deltaTable.clone("/path/to/replica-table")
```



# Write Serialization

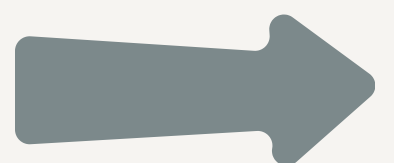
Delta Lake ensures that write operations are serialized to avoid conflicts and ensure consistency.

*Example*

```
spark.conf.set("spark.databricks.delta.writeSerialization.enabled", "true")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Streaming Joins

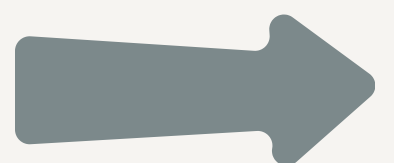
Delta Lake supports joining streaming data with static or other streaming data efficiently.

*Example*

*streamingDF1 =*

*spark.readStream.format("delta").load("/path/to/delta-table1") streamingDF2 =*

*spark.readStream.format("delta").load("/path/to/delta-table2") joinedDF = streamingDF1.join(streamingDF2, "id")*



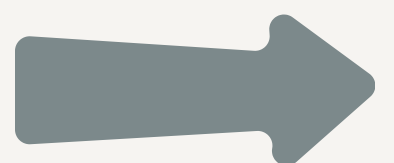


# Delta Lake Time Travel

Allows users to query previous versions of a Delta table using a version number or timestamp.

*Example*

```
df = spark.read.format("delta").option("versionAsOf",  
1).load("/path/to/delta-table") df.show()
```

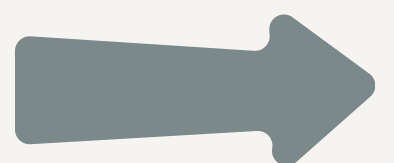


# Streaming Aggregations

Delta Lake supports aggregating data in streaming applications for real-time analytics.

*Example*

```
streamingDF = spark.readStream.format("delta")  
.load("/path/to/delta-table")  
aggregatedDF = streamingDF.groupBy("category").count()  
aggregatedDF.writeStream.format("delta")  
.outputMode("complete").start("/path/to/output-table")
```



# Delta Sharing

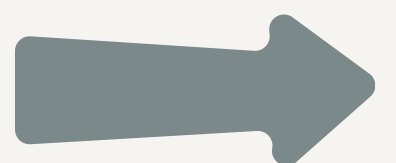
Securely shares data across organizations without the need for data duplication.

*Example*

```
spark.sql("CREATE SHARE my_share AS SELECT * FROM  
delta./path/to/delta-table")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Delta Lake on Databricks

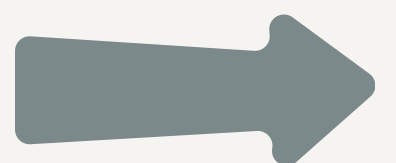
Delta Lake on Databricks provides advanced features and optimizations for performance and scalability.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
df.write.format("delta").save("/path/to/delta-table")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Delta Lake on AWS

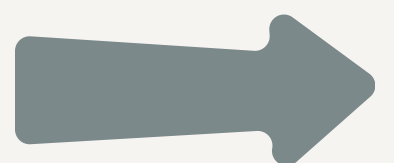
Delta Lake can be deployed on AWS for scalable and cost-effective data processing.

*Example*

```
spark.conf.set("spark.delta.logStore.class",  
"org.apache.spark.sql.delta.storage.S3SingleDriverLogStore")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

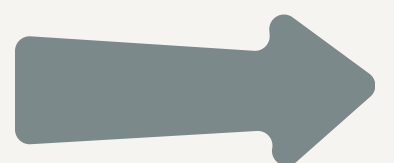


# Unified Data Management

Delta Lake provides a unified data management platform for batch and streaming data.



Shwetank Singh  
GritSetGrow - GSGLearn.com

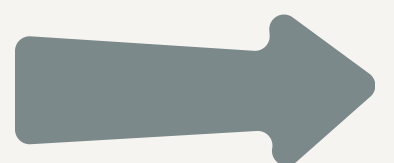


# Data Masking

Delta Lake supports data masking to protect sensitive data in queries and reports.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
maskedDF = df.withColumn("masked_col", lit("*****"))  
maskedDF.write.format("delta").save("/path/to/masked-delta-table")
```

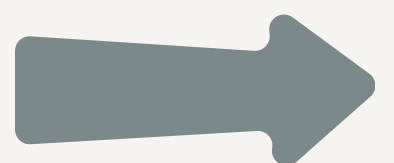


# Delta Lake Schema Management

Ensures that schema changes are handled gracefully without breaking existing data pipelines.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
df.write.format("delta").option("mergeSchema",  
"true").save("/path/to/delta-table")
```





# Delta Lake Auditing

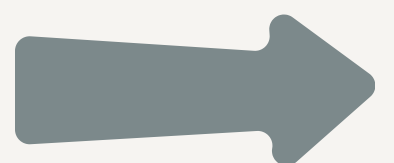
Provides detailed auditing capabilities to track data changes and access patterns.

*Example*

```
auditDF = spark.read.format("delta").option("readChangeData",  
"true").table("audit_table") auditDF.show()
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

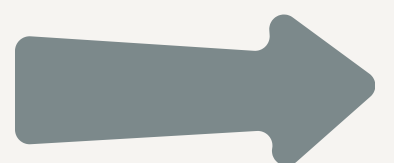


# Streaming Data Enrichment

Delta Lake supports enriching streaming data with additional context or reference data.

*Example*

```
streamingDF = spark.readStream.format("delta")  
.load("/path/to/streaming-table")  
referenceDF = spark.read.format("delta")  
.load("/path/to/reference-table")  
enrichedDF = streamingDF.join(referenceDF, "id")  
enrichedDF.writeStream.format("delta")  
.start("/path/to/output-table")
```



# Delta Lake Data Lineage

Tracks the lineage of data transformations to ensure traceability and compliance.

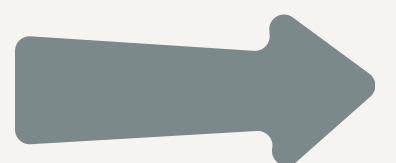
*Example*

*lineageDF =*

```
spark.read.format("delta").option("readChangeData",  
"true").table("lineage_table") lineageDF.show()
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



# Real-time Data Processing

Delta Lake supports real-time data processing for immediate insights and actions.

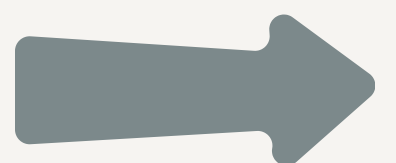
*Example*

*streamingDF =*

*spark.readStream.format("delta").load("/path/to/delta-table")*



Shwetank Singh  
GritSetGrow - GSGLearn.com

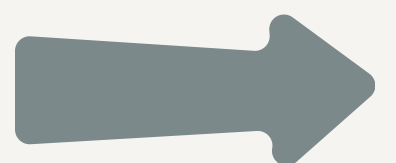


# Data Validation

Ensures data quality by validating data against predefined rules and constraints.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
validDF = df.filter("value IS NOT NULL")  
validDF.write.format("delta").save("/path/to/validated-delta-table")
```

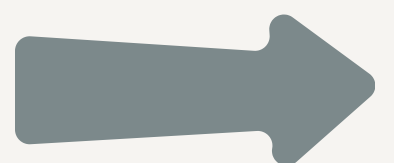


# Delta Lake Caching

Improves read performance by caching data in memory.

*Example*

```
spark.conf.set("spark.databricks.io.cache.enabled", "true")  
df = spark.read.format("delta").load("/path/to/delta-table")
```



# Unified Batch and Streaming

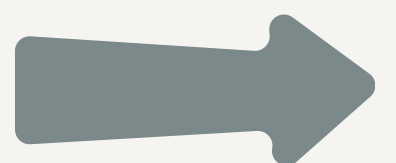
Delta Lake provides a unified platform for processing both batch and streaming data.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")
```

```
streamingDF =
```

```
spark.readStream.format("delta").load("/path/to/delta-table")
```



# Dynamic File Pruning

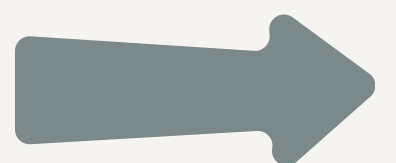
Delta Lake dynamically prunes files to optimize query performance.

*Example*

```
spark.conf.set("spark.databricks.delta.dynamicFilePruning.enabled", "true")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com



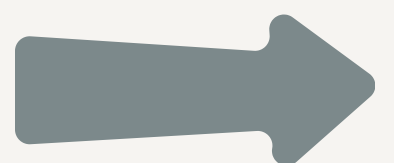


# Data Lakehouse

Combines the best of data lakes and data warehouses, providing flexible and scalable data management.



Shwetank Singh  
GritSetGrow - GSGLearn.com

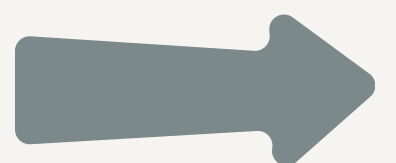


# Delta Lake Table Cloning

Delta Lake supports table cloning for creating quick, efficient copies of data.

*Example*

```
deltaTable = DeltaTable.forPath(spark, "/path/to/delta-table")  
deltaTable.clone("/path/to/cloned-table")
```



# Data Retention Policies

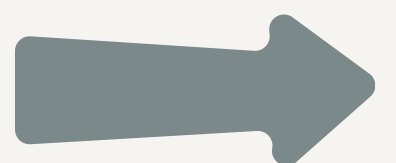
Delta Lake allows setting data retention policies to manage data lifecycle and compliance.

*Example*

```
spark.conf.set("spark.databricks.delta.retentionDurationCheck.enabled", "true")
```



Shwetank Singh  
GritSetGrow - GSGLearn.com

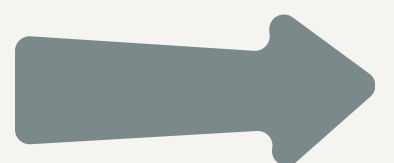


# Streaming Upserts

Delta Lake supports upserting data in streaming applications for real-time data integration.

*Example*

```
streamingDF = spark.readStream.format("delta")  
.load("/path/to/streaming-table")  
deltaTable = DeltaTable.forPath(spark, "/path/to/delta-table")  
deltaTable.alias("t")  
.merge(streamingDF.alias("s"), "t.id = s.id")  
.whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()
```

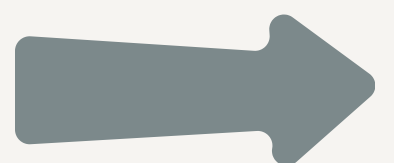


# Advanced Data Masking

Delta Lake supports advanced data masking techniques to protect sensitive information.

*Example*

```
df = spark.read.format("delta").load("/path/to/delta-table")  
maskedDF = df.withColumn("masked_col", lit("*****"))  
maskedDF.write.format("delta").save("/path/to/masked-delta-table")
```



THANK YOU