databricks

Guide

Snowflake to Databricks Migration Guide

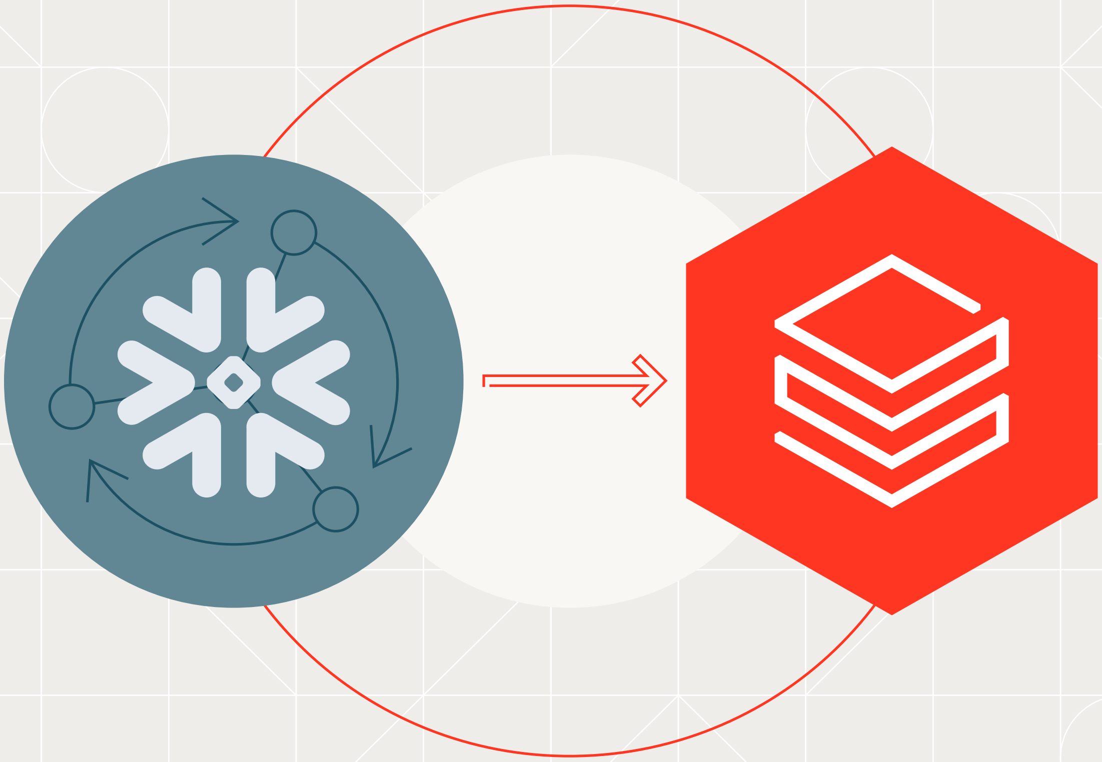
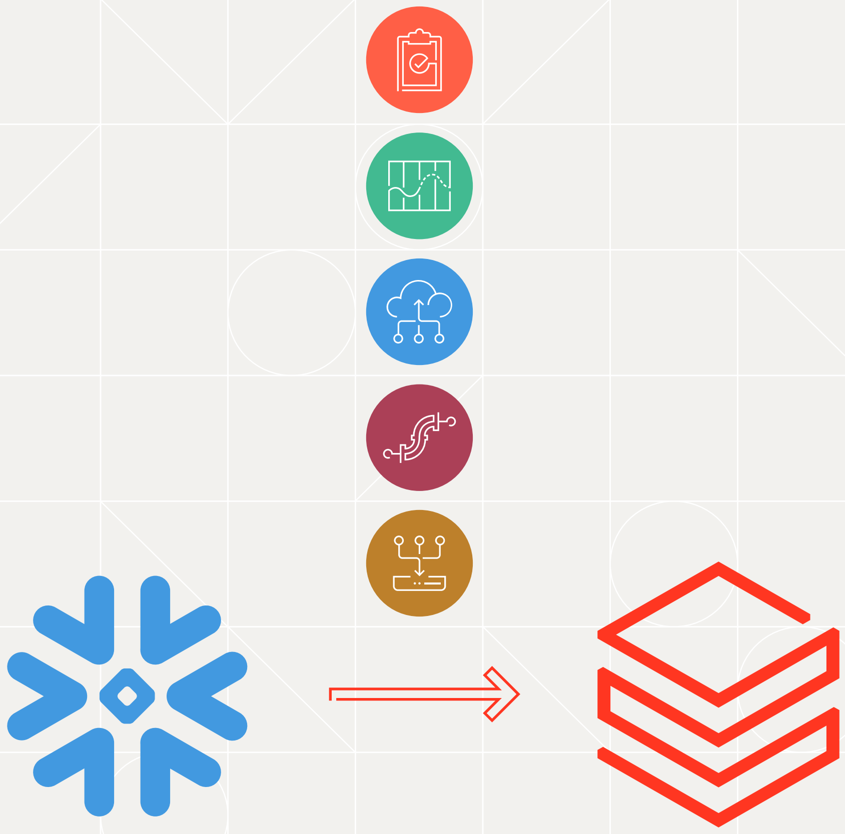


Table of Contents

Preface

The purpose of this document is to provide an overview of the process of migrating workloads from Snowflake to Databricks. The goal is to lay out foundational differences, common patterns in migrating data/code, best practices, tooling options, and more from Databricks' collective experience.



Migration Strategy

When migrating from Snowflake to Databricks, it is crucial to plan and execute the process carefully to ensure a successful outcome. By adopting a structured approach, it is possible to minimize risks and increase the chances of success. The migration process can take different routes depending on various factors such as the current architecture state, workload types and migration goals. The strategy employed in the migration process is influenced by several key factors, including the urgency and timelines, workload dependency (integrated vs. isolated pipelines), shared vs. isolated warehouses, current architectural limitations, road map backlog, new business requirements, access to migration tools, and migration effort.

Based on these factors, one can choose to undertake a bulk migration or a phased migration. A phased migration involves executing the migration in stages such as use case, schema, data mart or data pipeline. It is recommended to adopt a phased migration approach to mitigate risks and show progress early in the process. From a high-level strategy perspective, there are two popular approaches to migration.

1. Lead with migrating data ingestion and transformation workloads by landing all data in the cloud storage (Amazon S3, Azure Data Lake Storage Gen2, or Google Cloud Storage), taking advantage of the commoditized cloud storage, in Delta Lake format. Perform ETL — on both batch and streaming data — using Databricks and move cleaned, aggregated, ready-to-serve data to consumers.

   On top of this there are plenty of features in Databricks Lakehouse to support end-to-end ETL orchestration using Delta Live Tables and Databricks Workflows, unified governance and data lineage using Unity Catalog, and cross-organization data collaboration using Delta Sharing.

   This approach allows Snowflake use, in the interim, for serving downstream applications and BI dashboards. This way you could minimize the disruption to end users and slowly migrate downstream applications to the data consumption (Gold) layer on Databricks eventually.

2 | Lead with modernizing the reporting layer by replicating the data warehouse "Gold/presentation" layer tables from Snowflake in Databricks. This quickly unlocks the value by breaking data silos and enabling cross-functional reporting and cross-functional data science projects. Then work on reconfiguring the ETL in Databricks and cut off the ingestion and transformation in Snowflake.

In the next few sections, we will dive into the migration process, focusing on the approach and leading with the first approach described above: migrating ETL workloads first and then migrating BI workloads.

Overview of the Migration Process

Typically, data and ETL migrations from legacy on-prem technologies to cloud are complex and lengthy engagements. Whereas migrations from Snowflake are relatively easy because of SQL support in Apache Spark™ and both platforms being cloud based managed applications, a lot of concepts are similar. The migration process typically consists of the following steps, but can vary depending on customer situation and needs.

- Migration Discovery and Assessment
- Architecture and Feature Mapping Workshop
- Data Migration
- Data Pipeline Migration
- Downstream Tools Integration

In addition to migrating technical artifacts, a common activity that spans the entire migration process is change management, which involves user enablement and adoption. This will start with creating a few champions at the beginning and scale out to more developers and consumers. Databricks Academy is a good place to get started with some self-learning.

In general, migrating from one platform to another platform can be complex, which is why it is recommended to consider implementing using experts, at least for the initial pipelines. The Databricks Professional Services team has experience, skills, automation and access to expert partners in helping customers reduce risks and successfully migrate from Snowflake to Databricks. The Databricks Brickbuilder Solution for migrations has preferred partners who have demonstrated a unique ability in migrating Snowflake workloads to lakehouse successfully.

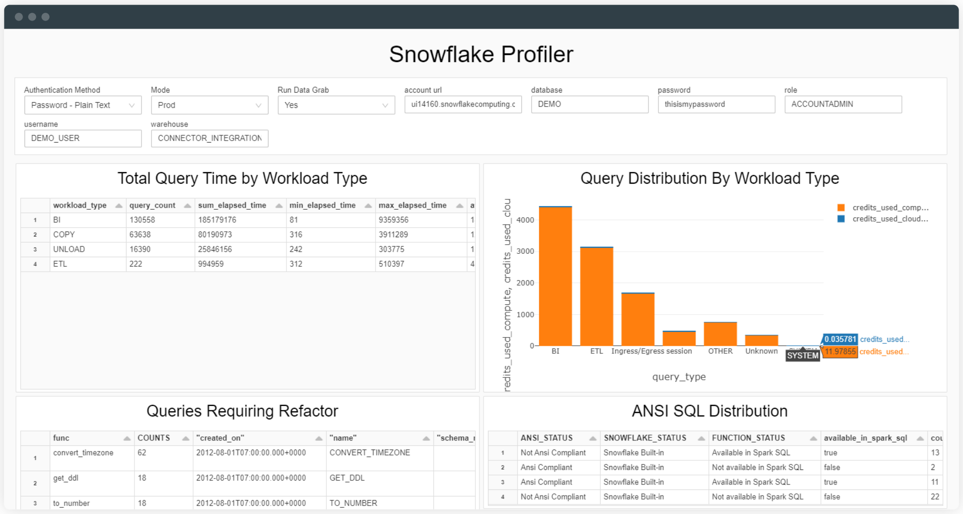
Phase 1: Migration Discovery and Assessment

Before migrating any data or workloads, one or more migration assessments should be conducted in order to:

- Understand the types of workloads (ETL, BI, ingress/egress, etc.) and their size by warehouses and databases
- Understand the scope of data and queries/workloads to be migrated
- Understand the upstream and downstream technologies and applications involved in the architecture
- Understand the current security setup and protocols
- Understand the level of effort required to complete the migration
- Collect information relevant to developing estimates for infrastructure costs and person-hours

Databricks strongly recommends using automation tools (**Snowflake Profiler** and/or the **BladeBridge Code Analyzer**) to expedite the process of gathering migration-related information.

The Snowflake Profiler reads system tables in Snowflake (e.g., snowflake.account_usage.query_history) and other warehouse tables, and returns insights such as workload types, long running ETL queries and surface background optimizations costs. This analysis provides guidance on identifying warehouses/databases/DAGs contributing to high costs and supports prioritization. The profiler classifies queries into T-shirt sizes for complexity, evaluates function compatibility, and extracts information for data migration (clustering keys, table size).



Snowflake Profiler

Authentication Method	Password - Plain Text
Mode	Prod
Run Data Grab	Yes
account url	ui14160.snowflakecomputing.c
database	DEMO
password	thisismypassword
role	ACCOUNTADMIN
username	DEMO_USER
warehouse	CONNECTOR_INTEGRATION

Total Query Time by Workload Type

	workload_type	query_count	sum_elapsed_time	min_elapsed_time	max_elapsed_time	a
1	BI	130558	185179176	81	9359356	1
2	COPY	63638	80190973	316	3911289	1
3	UNLOAD	16390	25846156	242	303775	1
4	ETL	222	994959	312	510397	4

Queries Requiring Refactor

	func	COUNTS	"created_on"	"name"	"schema_
1	convert_timezone	62	2012-08-01T07:00:00.000+0000	CONVERT_TIMEZONE	
2	get_ddl	18	2012-08-01T07:00:00.000+0000	GET_DDL	
3	to_number	18	2012-08-01T07:00:00.000+0000	TO_NUMBER	

ANSI SQL Distribution

	ANSI_STATUS	SNOWFLAKE_STATUS	FUNCTION_STATUS	available_in_spark_sql	cou
1	Not Ansi Compliant	Snowflake Built-in	Available in Spark SQL	true	13
2	Ansi Compliant	Snowflake Built-in	Not available in Spark SQL	false	2
3	Ansi Compliant	Snowflake Built-in	Available in Spark SQL	true	11
4	Not Ansi Compliant	Snowflake Built-in	Not available in Spark SQL	false	22

Figure 1: Sample output from example profiler result

Another option for gaining deeper insights on data types, DDL (data definition language), DML (data manipulation language) code complexity and data dependencies is by running the BladeBridge Analyzer tool. The tool parses Snowflake SQL code and generates:

- An inventory of the code base: Table DDLs, Views, Stored Procedures, Functions, Tasks, Sequences, etc.
- Complexity of each script categorized from low to very complex based on number of statements
- List of data types and functions
- List of code and table cross-references that can provide support in understanding a table popularity

The complexity counts are used to size the software costs for using the BladeBridge Code Converter and help you estimate the number of hours you should forecast for the migration project. This tool is available free of cost and a Databricks Architect can assist you in running the tool. Figure 2 shows the summary output from the Analyzer results of sample Snowflake code base.

CODE BASE DETAILS	
Total SQL Scripts	906
Total FILE Scripts	906
Total DDLs	341
Total CTAS Scripts	22
Total Tables (in scripts)	363
Total Views	4
Total Materialized Views	0
Total Packages	0
Total Procedures	65
Total Functions	20
Total Tasks	0
Total Loops & Cursors	3
Total Conditionals	2
Total Lines of Code	414637
Total Duplicated SQL Items	1

JOB COMPLEXITY CATEGORIZATION	
LOW	519
MEDIUM	330
COMPLEX	31
VERY_COMPLEX	25

Figure 2: Summary of output from BladeBridge Analyzer

The outcome of this phase is a migration scope covering information across queries/ workloads, data/tables, and usage of any Snowflake proprietary features/tools.

Phase 2: Architecture and Feature Mapping Workshop

When planning your Snowflake migration, it is important to correctly map Snowflake capabilities to Databricks Lakehouse capabilities. The second phase of the migration journey is understanding current-state architecture and Snowflake features in order to map them to future-state architecture and Databricks features. In this phase, we design the ideal lakehouse architecture that serves all the data and AI use cases, and design how each component of the current architecture needs to be modernized to map to the target architecture.

We then will compare/contrast current- and future-state architecture diagrams and align on the intermediate phases of the architecture as the migration progresses through each phase. As an example, the intermediate architecture will require running ETL pipelines in both Snowflake and Databricks in parallel for some period of time, so we will want to architect an optimal solution to ensure data stays in sync and external systems/tooling can continue to function without impact to the rest of the organization.

Following architectural alignment, we will conduct a deep dive into all features currently used in Snowflake. Phase 1 will help surface this information, but Phase 2 will dive deeper into how they are used to ensure each existing Snowflake feature/functionality is mapped to the appropriate Databricks feature/functionality.

SNOWFLAKE VS. DATABRICKS FEATURE MAPPING EXAMPLE

FEATURES	SNOWFLAKE	DATABRICKS
Compute	One type of compute for all workloads called **Snowflake Virtual Warehouse**	Databricks Managed Clusters optimized for workload types with a runtime: • All-purpose for interactive/developer use • Jobs for scheduled pipelines • SQL warehouse for BI workloads
Storage	Snowflake's own storage layer	Cloud storage (Amazon S3, Azure Blob Storage, Azure Data Lake Storage Gen2, Google Cloud Storage)
Format	Snowflake FDN format (proprietary compressed format)	Open source Delta format (Parquet)
Architecture Layers	Cloud services layer Query processing Database storage	Control plane Data plane

FEATURES	SNOWFLAKE	DATABRICKS
Stages/Tables	External stage Internal stage External tables Internal tables	External tables Managed tables
Interface	Snowpark Snowsight SnowSQL CLI	Databricks collaborative notebooks Databricks SQL workspace Databricks CLI
Database Objects	Tables, temporary tables, transient tables Views, materialized views Stored procedures UDFs	Tables, temporary views Views, materialized views Databricks notebooks UDFs
Metadata Catalog	No native cataloging feature Third-party tools such as Collibra, Alation	Unity Catalog
Data Ingestion	COPY INTO Snowpipe	COPY INTO CONVERT TO DELTA Auto Loader DataFrame Reader Integrations via Partner Connect Add data UI
Data Types	Data Types in Snowflake	Data Types in Databricks
Workload Management	Load monitoring chart, custom query tags	Cluster configuration (policies), ganglia metrics
Security	System defined roles and custom roles Hierarchy of roles Table-/column-/row-level security	System roles: account admins, workspace admins, metastore admin, account users and workspace users Users, groups and service principals Object based controls using access control lists (ACLs) Table-/column-/row-level security
Data Clustering	Clustering	Z-ordering
Programming Language	SQL, Java, JavaScript, Python, Scala	SQL, Python, R, Scala, Java
Data Integration	External tools (dbt, Matillion, Talend, Pentaho, Informatica, etc.)	Delta Live Tables Databricks Jobs External tools (dbt, Matillion, Prophecy, Informatica, Talend, etc.)
Orchestration	Snowflake Tasks External third-party tools (e.g., Airflow)	Databricks Workflows External third-party tools (e.g., Airflow)

FEATURES	SNOWFLAKE	DATABRICKS
Machine Learning	Python tools on Snowpark, external third-party tools (e.g., Alteryx, DataRobot)	Databricks ML (Runtime with OSS ML packages, MLflow, Feature Store, AutoML)
Change Data Capture	Snowflake Streams	Delta Change Data Feed
Time Travel	Snowflake Time Travel	Delta Time Travel
Data Sharing	Snowflake Secure Data Sharing Snowflake Marketplace	Delta Sharing Delta Sharing Marketplace
Pricing Unit	Snowflake credits Snowflake storage	Databricks units (DBUs)

The table above is an example of mapping key features between Snowflake and Databricks. A similar exercise comparing all relevant features for your environment should be performed in this step.

Typically, by the end of this phase we have a good handle on the scope and complexity of the migration, and can come up with a more accurate migration plan and migration cost estimate.

Phase 3: Data Migration

Before you start the migration process, one or more Databricks workspaces will need to be provisioned if not available. The decision to create one or more workspaces typically revolves around the following considerations:

- **Separation of environments:** Requiring different workspaces for development, staging, production and other environments
- **Separation of business units:** Requiring different workspaces for marketing, finance, risk management and other departments
- **Implementation of modern data architectures:** Requiring different workspaces to support modern data architectures, such as Data Mesh architecture, to decentralize data ownership for different domains

Once the workspace is set up, the first step of the migration is migrating the data. Databricks is optimized for cloud object storage: Amazon S3, ADLS2 and Google Cloud Storage. In addition to cloud storage, Databricks can read/write from, write to other storage endpoints, including relational databases (Oracle, SQL Server, Teradata, etc.), HDFS, Apache Hive, NoSQL (HBase, Cassandra, Neo4j, MongoDB), in-memory cache (Redis, RocksDB), message bus (Kafka), files (delimited text files, JSON, Parquet, ORC, Avro), JDBC/ODBC sources/sinks, and many more.

When migrating data out of Snowflake, there are key decisions that need to be made. Some of these could include:

- What is the target design for the tables being migrated?
- Should the destination retain the same hierarchy of catalogs, databases, schemas, tables?
- Should there be any cleanup or organization of the existing data footprint in Snowflake?

Once these are decided, the data migration can proceed. Generally speaking, we do not recommend introducing many changes in the schema structure during migration. Given the Schema Evolution capability in Delta, it is a common practice to evolve the schema after it is put in Delta. This approach also allows us to easily compare the data in Snowflake and Databricks during parallel runs.

RECOMMENDED APPROACH

It is important to note that not every data migration will follow the same pattern, but Databricks recommends the following flow:

1. Migrate EDW (enterprise data warehouse) and staging tables (optional) into Delta Lake medallion data architecture
2. Migrate/build data pipelines that incrementally populate Bronze/Silver/Gold in Delta Lake
3. Backfill Bronze/Silver/Gold tables as needed

Typically, data in a Snowflake ELT pipeline moves through a staging or landing zone (Bronze), then an optional integration layer (Silver), and ends up in a data model in an EDW/reporting layer (Gold). Whatever data model (dimensional model, data vault, etc.) is implemented in Snowflake can be implemented in the Databricks Lakehouse in a more performant manner using Delta Lake. Data architecture in Databricks Lakehouse follows Delta Lake — Bronze, Silver, Gold paradigm — and the data model belongs in the Gold layer.

SCHEMA MIGRATION

Before the tables are offloaded to Databricks, the schema of the tables must be created in Databricks. If you have DDL scripts, you could leverage that with some tweaks to data types used in the DDLs. DDLs can also be extracted from Snowflake using the get_ddl function. Once the scripts are extracted, automation tools (e.g., BladeBridge code converter) can handle the converting Snowflake DDLs to Databricks DDLs. Refer to guidance around matching data types in both Databricks and Snowflake in Appendix 1.

DATA MIGRATION

In terms of implementation, several approaches have been validated by Databricks for migrating the data and are already in production use by various customers. For the initial Gold table migration, options include:

1. Leveraging Snowflake's COPY INTO command to push data out of Snowflake and into cloud storage in Parquet format, then using one of these options to load into Databricks to write to Delta Lake format tables using one of the following options:
   - Auto Loader
   - Databricks COPY INTO command
   - Spark batch/streaming APIs
2. Leveraging the Snowflake Connector for Spark to read data from Snowflake and write to Delta Lake format tables
   - *This bypasses the write to Parquet, but requires running Snowflake virtual warehouses and Databricks clusters in parallel, which can be costly*
   - *This method is commonly used for small tables up to 15GB in size and a manageable number of tables, but for larger tables, the first approach above is recommended*
3. Leverage data ingestion partners such as Arcion from Databricks Partner Connect for quick data migration using automation with built-in schema management, high availability (HA), and auto-scaling

After the initial Gold table offload, recurring jobs should be set up to continuously sync data from Snowflake to Databricks for those Gold tables until data pipelines are migrated to Databricks and are feeding data to those Gold tables. For continuous replication and real-time sync, options include:

1 | Leveraging the Snowflake Stream Reader library to ingest data from Snowflake in a CDC (change data capture) fashion into cloud storage and load into Delta Lake tables using Databricks Auto Loader

2 | Leveraging real-time change data capture tools from Databricks Partner Connect. Read this blog for an example of how to implement real-time sync using Arcion, a data ingestion partner of Databricks. Databricks and any partners involved will work with your team to align on the best approach(es) for each team's requirements.

As you go through the migration, the current architecture slowly changes as you start offloading data and workloads in a phased manner. Figure 3 shows the architecture during the data migration phase.

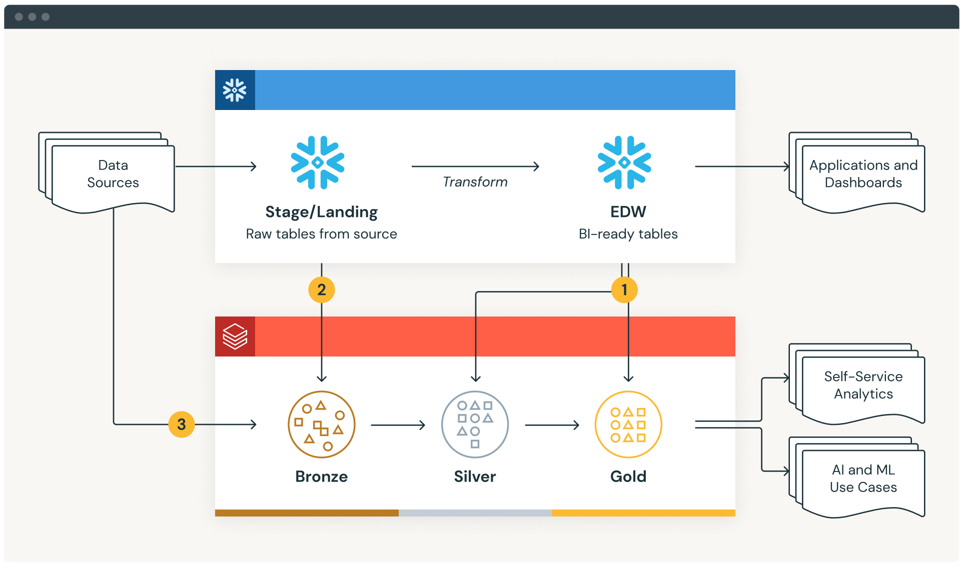


Figure 3: Transient state architecture: post-data migration

1 | One-time offload of EDW tables into Silver and Gold layers via:

- COPY INTO External Stage as Parquet and use Auto Loader to Delta or
- Use Spark Snowflake Connector

Continuous jobs to offload change data from Snowflake EDW until data pipelines are cutover

2 One-time offload of Stage/Landing tables via:

- COPY INTO External Stage as Parquet and use Auto Loader to Delta or
- Use Spark Snowflake Connector

If stage is a cloud storage, convert data into Delta

3 For continuous data source jobs, use Auto Loader to monitor S3/ADLS/GCS buckets for change data and MERGE into Bronze Delta Table

After validation and testing, the Bronze and Gold layer data is available for immediate use in Databricks by end users for ad hoc analysis and machine learning while data pipelines are being offloaded to Databricks in parallel. This is the advantage of the lakehouse architecture, to make the data available for different use cases instantly without having to move data around.

KEY CONSIDERATIONS: SCHEMA MIGRATION AND DATA MIGRATION

- **Data Modeling:** As part of the migration, there might be a need to refactor the data model or reproduce a similar data model in an automated and scalable manner. Visual data modeling tools such as a Quest ERWIN or SqlDBM, from Databricks Partner Connect, can accelerate this development and deployment of the data model in a few clicks. Both of these tools can reverse engineer a Snowflake data model (table structures) and implement them in Databricks easily.

- When migrating DDLs of a table, it is important to check the schema of the source data. For example, let's say one of the data sources is in Parquet format. Some numeric column types in DDL generated from the Snowflake table will be different from the type in the source Parquet files. If we don't use the source column types during DDL creation, we will be forced to have unnecessary casting on our ingestion pipeline after the data is migrated from Snowflake.

- In Snowflake and Databricks, keeping schemas in sync during the transient stage can become critical if there are changes introduced to table schemas during migration. Approaches such as making a change in a central MDM (master data model) first — leveraging tools like SqlDBM — and then implementing it in both Snowflake and Databricks are gaining traction.

Phase 4: Data Pipeline Migration

An end-to-end view of the pipelines from data sources to the consumption layer considering the governance aspects must be thoroughly understood to effectively migrate the workloads. Data pipeline migrations from Snowflake to Databricks consist of several key components: orchestration, source/sink migration, query migration and refactoring.

ORCHESTRATION MIGRATION

An ETL orchestration can refer to orchestrating and scheduling end-to-end pipelines covering data ingestion, data integration, result generation or orchestrating DAGs of a specific workload type like data integration. In Snowflake the orchestration is typically done using external tools such as Airflow, Matillion or by using Snowflake Tasks. There are generally two options when migrating these workflows.

1. Use Databricks Workflows to orchestrate the migrated pipelines. In addition, Delta Live Tables can be used for building reliable and efficient data processing pipelines. Using Delta Live Tables provides a standard framework for building both batch and streaming use cases along with critical data engineering features such as automatic data testing, deep pipeline monitoring and recovery. Tasks in Snowflake get created as Databricks Workflows. It also has out-of-the-box functionality to SCD Type 1 and Type 2 tables.
2. It is also possible to use the external tools for orchestration and repoint these tools from Snowflake compute to Databricks compute. You would be just translating Snowflake SQL queries to Spark SQL queries in the orchestration job while retaining most of the orchestration elements. However, it is recommended to use Databricks Workflows for better integration, simplicity and lineage.

SOURCE/SINK MIGRATION

Similar to orchestration, in most cases an external ETL tool is seen in Snowflake architecture to extract data from source systems, transform (optional) and load it into the Snowflake staging layer.

1. Source data connections:
   - Ingestion pipelines using tools such as Fivetran and Qlik Replicate can be duplicated and configured to point to Databricks Delta Lake instead of Snowflake staging. Delta is an open source format and widely supported as the target data format for popular data ingestion tools.
   - Ingestion pipelines using Snowpipe are replaced with Databricks Auto Loader or Spark DataFrame APIs. Delta Live Tables supports Auto Loader and Spark DataFrame APIs.
   - Native Spark integrations (e.g., Kafka) are leveraged to refactor the streams
2. Sink data connections:
   - Ingestion tools and framework will now generate data in Delta Lake format instead of Snowflake format

QUERY MIGRATION AND REFACTORING

Queries here refer to any DML query that transforms the data or to ad hoc data analysis queries run by the user for data analysis. The interface from where queries are initiated could be directly in Snowflake or coming from an external ETL tool such as dbt or Matillion.

In situations where SQL queries are triggered from an external ETL platform such as Matillion, the refactoring is straightforward, especially for user-written SQL queries. Any Snowflake-specific integration features should be refactored, which in most cases is equivalent to tweaking underlying Snowflake SQL query.

Migrate from Snowflake SQL to Spark SQL, identifying and replacing any incompatible/ proprietary Snowflake SQL functions or syntax. A few options for tackling this are:

1. (Recommended) Use BladeBridge Converter to automate lift-and-shift portion of query migration
2. (Recommended) Use SQLGlot to convert Snowflake SQL to Spark SQL
3. (Not recommended) Develop custom script in-house to convert Snowflake SQL to Spark SQL
4. (Not recommended) Manually convert Snowflake SQL to Spark SQL

Given that both Snowflake and Databricks support ANSI SQL standards, a large portion of the Snowflake SQL query can be automatically converted to Databricks syntax to accelerate the migration. The BladeBridge conversion tool supports schema conversion (tables and views), SQL queries (select statements, expressions, functions, user-defined functions, etc.), stored procedures and data loading utilities such as COPY INTO. The conversion configuration is externalized, meaning conversion rules can be extended by users during migration projects to handle new code pattern sets to achieve a greater percentage of automation. Check out this short demonstration of the conversion tool.

Refer to Appendix 3 for some examples of SQL translation differences between Snowflake and Databricks. Check out this handy cheat sheet packed with essential tips and tricks to help you get started on Databricks using SQL programming in no time!

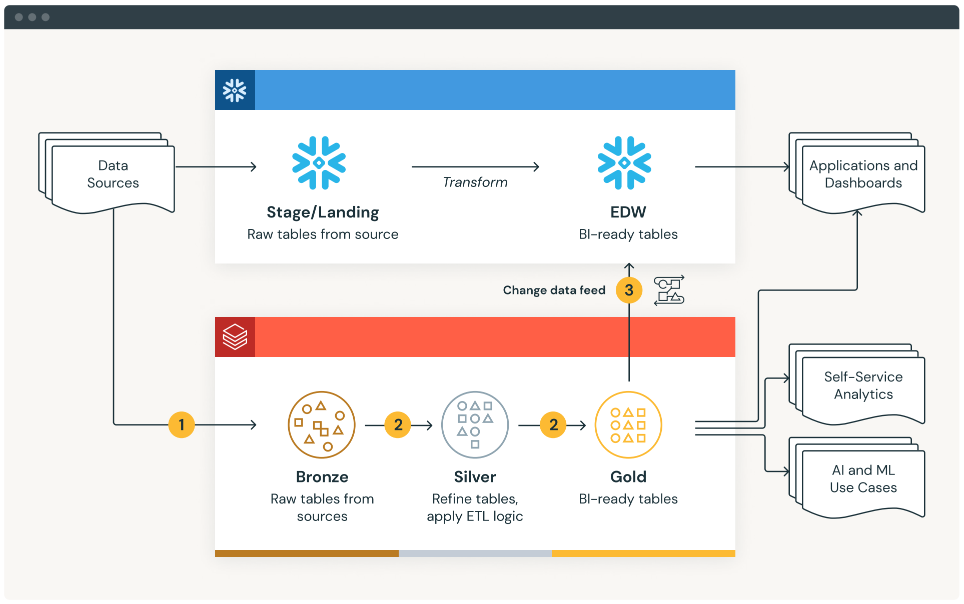


Figure 4: Transient state architecture during pipeline migration

1 | Source data pipelines are duplicated and refactored to save to Delta

For continuous data source jobs, use Auto Loader to monitor S3/ADLS/GCS buckets for change data and MERGE into Bronze Delta Table

2 | Implement Medallion data flow architecture and convert transformation pipelines from Snowflake SQL to Spark SQL

3 | Extract change data feed from Gold tables into Parquet or flat file format and load into Snowflake EDW tables via Snowpipe

Data quality checks should be performed at the end of each change data offloading job until data pipelines are cutover

MIGRATION VALIDATION

Validation is mostly done around the data in both the platforms. Establish a unit testing for sink-to-sink comparisons. As there might be thousands of tables migrated, it is manually impossible to compare the data values in Snowflake and Databricks. Generally a testing framework with a script to compare values automatically in both the platforms is used. Some example data points to compare include:

- Check if the table exists
- Check the counts of rows and columns across the tables
- Calculate the sum of numeric columns and compare
- Calculate the distinct count of values in string columns and compare

Run the pipelines in parallel for a week or two and review the comparison results to ensure the data is flowing correctly. For more advanced table data and schema comparison, tools like Datacompy can be used.

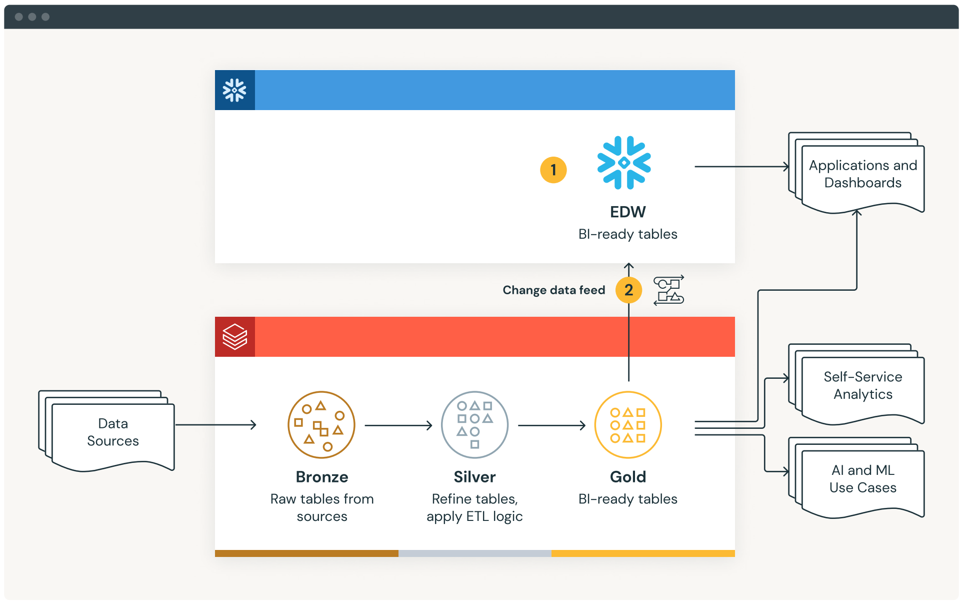


Figure 5: Transient state architecture — post-data and ETL pipeline migration

1 Data ingestion and transformation in Snowflake are retired and the EDW tables are hydrated from Databricks

2 Extract change data feed from Gold tables into Parquet or flat file format and load into Snowflake EDW tables via Snowpipe

Propagate schema changes to Snowflake tables

KEY CONSIDERATIONS: DATA PIPELINE MIGRATION

DATA PIPELINE REFACTORING AND OPTIMIZATION

- During the migration, there is a high probability for some portion of data pipeline queries to be refactored and/or optimized. This includes but is not limited to:

  1 | Rewriting queries to avoid nonperformant join strategies

  2 | Changing query filters due to changes in clustering keys

  3 | Adjusting queries to account for function syntax changes

  Here are some strategies to overcome inefficiencies and improve query performance:

  A | Join strategies

  a. Avoid Cartesian products if at all possible (these are heavy in any query engine)

  b. Avoid self, exploding joins that result in Cartesian products (self-join with sliding window calculations as keys); instead, shift logic into a CTE or view so that the sliding window calculations are materialized pre-join

  c. Preferred join strategies (in descending order):

  i. Broadcast Hash Join
  ii. Shuffle Hash Join
  iii. Sort Merge Join
  iv. Shuffle Nested Loop Join (Cartesian Product)

  B | Clustering keys

  a. Select high cardinality columns for Z-ordering

  b. Z-order is effective for up to 3-5 columns

  c. Clustering keys in Snowflake can generally be used as Z-order columns with Delta Lake; the exception is for tables > 1TB — we recommend partitioning by low cardinality columns and Z-ordering by high cardinality columns

  C | See Delta Lake & Performance Optimization section for other recommendations

- Consider reengineering some ETL pipelines to leverage new capabilities in Delta Live Tables such as SCD Type 2 which are not straightforward to implement in Snowflake using Snowflake Streams and Tasks. One popular use case where reengineering is almost always considered is modernizing CDC ingestion and streaming workloads using the power of Spark Structured Streaming and Delta Live Tables. Although this results in additional migration effort, this is critical for long-term cost reduction and any value-add your team would like to realize.

- Consider creating a Git repository of the queries being migrated and refresh this repository frequently if the queries/pipelines are allowed to change during the migration so that there are fewer conflicts to resolve during the final migration. It is recommended to impose code freeze during migration if possible.

DATA PIPELINE CUTOVER

During data pipeline migration, there will be a period over which data pipelines will be running in Databricks and Snowflake concurrently. This is expected, but in order to minimize the costs associated with this, we recommend defining the following:

1. Cutover schedule
2. Criteria for approving/disapproving production readiness in Databricks
3. Criteria for approving/disapproving data pipeline deprecation in Snowflake
4. Upstream/downstream integration validation
5. Communication strategy for all applicable stakeholders

The approach described until now ensures the ETL pipelines are fully migrated and running in Databricks and the Gold layer data in Snowflake is kept in sync with the Gold layer of Databricks. While it is possible to incur data movement costs between the platforms, the significant savings gained from ETL costs would easily offset these costs. In the next phase of migration, the architecture is evolved to support business intelligence and other serving use cases.

Phase 5: Downstream Tools Integration

To further consolidate data platform infrastructure and maintain a single source of truth of data, organizations have adopted Databricks SQL, a data warehousing product in Databricks Lakehouse, to meet their data warehousing needs and support downstream applications and business intelligence dashboards.

Databricks SQL offers world-class price/performance for analytics workloads as well as support for high-concurrency use cases with auto-scaling SQL warehouses. Databricks SQL includes Photon, which is a query engine built from scratch in C++ and is vectorized to exploit both data-level and instruction-level parallelism.

Once data and transformation pipelines are migrated to the Databricks Lakehouse, it is critical to ensure business continuity of downstream applications and data consumers. Databricks Lakehouse has validated large-scale BI integrations with many popular BI tools in the market such as Tableau, Power BI, Qlik, ThoughtSpot, Sigma, Looker, and more. The expectation for a given set of dashboards or reports to work is to ensure all the upstream tables and views are migrated along with the associated pipelines and dependencies.

As described in the blog (see section 3.5 Repointing BI workloads), one of the common ways to repoint BI workloads after a data migration is by testing sample reports and working by renaming the data source/tables names of existing tables and pointing to the new ones.

Typically, if the schema of the tables and views post-migration hasn't changed, the repointing is a straightforward exercise of how you handle switching databases on the BI dashboard tool. If the schema of the tables has changed, you will need to modify the tables/views in the lakehouse to match to the expected schema of the report/dashboard and publish it as a new data source for the reports.

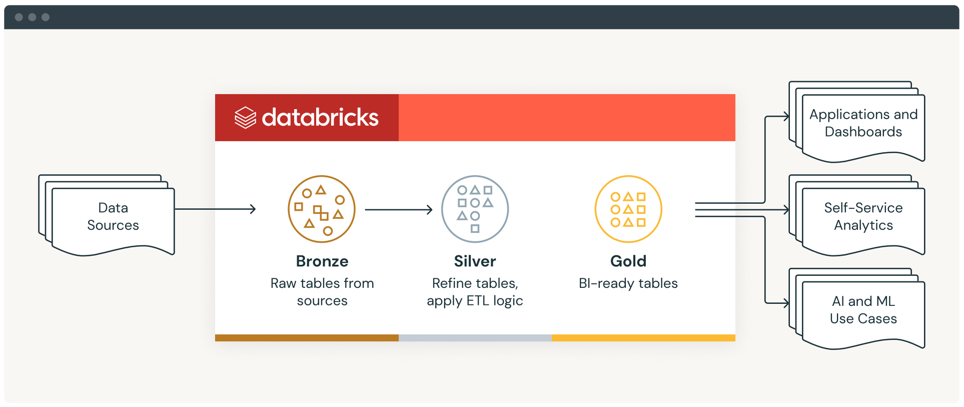


Figure 6: Future-state architecture

We recommend testing the approach with a small set of dashboards or reports and iterating through the remainder of the reporting layer throughout the migration.

During the reports migration, a potential situation you may run into is the need to expand the permission of BI tool access to cloud storage buckets. This is because Databricks uses "Cloud Fetch" to support high bandwidth data exchange. With this architecture, for a given BI query, the BI tool gets back pre-signed URLs, so that the BI tool downloads data in parallel directly from cloud storage. This might require enabling new access permissions if not already configured.

Best Practices

The Databricks Lakehouse Platform provides a number of ways to evolve and optimize the platform to achieve best performance at the lowest cost when using and administering Databricks. Here are some best practices to implement in Databricks pertaining to different areas of ETL workloads.

DATABRICKS PLATFORM

It's important to understand some basic concepts used in Databricks before you get started.

- Databricks Interface →
- Cluster Configuration →
- Cluster Policies →
- Data Governance →
- GDPR & CCPA Compliance →
- Delta Lake →
- Structured Streaming →
- CI/CD →

DELTA LAKE AND PERFORMANCE OPTIMIZATION

Optimize the performance of the migrated workload by tweaking the configuration of the Databricks environment and the workload itself. This includes identifying and eliminating any bottlenecks and improving the overall performance. Below are a few best practices to consider during performance tuning.

File Sizing

- Databricks Runtime automatically tunes file sizes based on table size and also based on workload — for example, to accelerate write-intensive operations
- File sizes can be manually adjusted by setting delta.targetFileSize as a table property or Spark configuration

Partitioning

- Avoid partitioning tables < 1TB
- Ideal size of partitions is > 1GB
- Use generated columns to avoid over-partitioning
- Partition on lower cardinality columns

Data Skipping

- Statistics will be automatically computed for you to facilitate data skipping
- Tracks file-level statistics like min, max, etc.
- Helps avoid scanning irrelevant files/data
- By default, Databricks Delta collects statistics on the first 32 columns defined in the table schema. This default value can be updated using the table property, delta. dataSkippingNumIndexedCols
- A best practice to keep in mind is to move numerical columns and high cardinality query predicates to the left of the 32nd ordinal position, and move strings and complex data types after the 32nd ordinal position of the table

Z-Ordering (Clustering)

- Effective on up to 3-5 columns
- Z-order on higher cardinality columns, columns for Z-ordering must be in the first 32 columns

Merge/Upsert

- Ensure you are using DBR 10.4+ to take advantage of Low Shuffle Merge
  - Avoids write amplification due to merge's use of fullOuterJoin
- With Low Shuffle Merge, fullOuterJoin is broken into an inner and leftOuterJoin followed by read > filter > write using file + rowId map
  - This helps optimize merge performance significantly

Generated Columns

- Special column type that gets defined based on a user-specified function over other columns in a Delta table
- Values for generated columns are computed at runtime
- Generated columns allow users to avoid over-/under-partitioning

Join Strategies

- Broadcast Hash Join / Broadcast Nested Loop Join
  - Requires one side of the join to be small
  - No shuffle, no sort, very fast
- Shuffle Hash Join by read > filter > write using file + rowId map
  - Needs to shuffle data, but avoids sort
  - Handles large tables, but will result in an out-of-memory error if data is skewed
- Sort Merge Join
  - Handles any data size
  - Requires shuffle and sort
  - Slower in most cases when table size is small due to excessive shuffle
- Shuffle Nested Loop Join/Cartesian Product
  - Does not require join keys
  - Extremely heavy operation; avoid at all costs if possible

Query Profile

- In the case of data warehouse usage, the SQL warehouse query profile is a powerful tool located inside the Databricks SQL workspace. Its objective is to troubleshoot slow-running queries, optimize query execution plans, and analyze granular metrics to see where compute resources are being spent.
- The query profile provides value in these three capability areas:
  - Detailed information about the three main components of query execution, which are time spent in tasks, number of rows processed and memory consumption
  - Two types of graphical representations. A tree view to easily spot slow operations at a glance, and a graph view that breaks down how data is transformed across tasks.
  - Ability to understand mistakes and performance bottlenecks in queries
- Three common performance bottleneck problems surfaced by query profile are listed below:
  - Inefficient file pruning
  - Full table scans
  - Exploding joins (Cartesian product)

Analyze Table

- The ANALYZE TABLE command collects statistics on tables in Databricks and ensures that the query optimizer finds the most optimal query execution plan. SQL syntax is as follows:

```
ANALYZE TABLE my_table COMPUTE STATISTICS for COLUMNS col1, col2, col3
```

- One important point to remember is that you will want to prioritize statistics for columns that are frequently used in joins and other query predicates
- Best practice is to run ANALYZE TABLE as a separately scheduled job on a regular cadence (e.g., weekly or monthly)

GOVERNANCE AND SECURITY

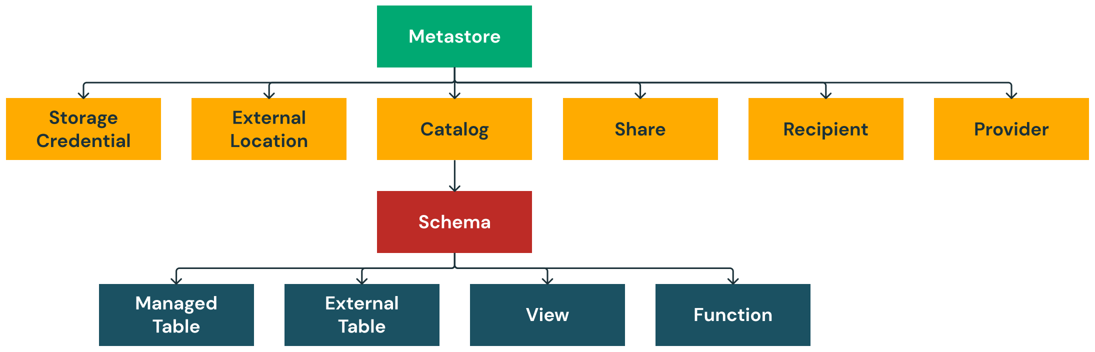
Reference Materials:

- Data Governance Guide →
- Unity Catalog →

Identity Management

- Identities exist at the Databricks account level. Identity federation allows for these account-level identities to be federated downward to workspaces
- Single sign-on (SSO) can be set up to manage account-level identities
- Identity Types
  - Users
  - Groups
  - Service Principals

Privileges and Securable Objects



- Securable Objects →
- Inheritance Model →
- Privileges Types →

Compute

- Clusters & SQL Warehouses with Unity Catalog →

Need Help Migrating?

Regardless of size and complexity, the Databricks Professional Services team, along with an ecosystem of services partners and ISV partners, offers different levels of support (advisory, staff augmentation, scoped implementation) to accelerate your migration and ensure successful implementation. Aside from steps outlined in this migration guide, the services offered can include architecture design workshops, Databricks foundation setup, change management, cutover operations, and more.

Working with BladeBridge, Databricks has developed automated tooling for code complexity assessment and code migration (DDLs, DMLs) that produces outcomes tuned to best practices on Databricks Lakehouse. The conversion tool is available for use either with your preferred services vendor or a services vendor recommended by Databricks. Additionally, Databricks partners have developed several other automation tools to accelerate your migration.

Contact your Databricks representative or reach out to us using this form for more information. Rest assured that we can work with you and make your migration successful.

Appendix

APPENDIX 1: DELTA VS. SNOWFLAKE – STORAGE FORMAT COMPARISON

	DELTA	SNOWFLAKE
Default Format Type	Columnar (OSS Parquet)	Columnar (proprietary FDN format)
IO Unit	File	Micro-partition
Size of IO Unit	16MB – 1GB (depending on table size, also configurable)	16MB
Sort Order Within IO Unit	None	None
Sort Order Between IO Units	Ingestion Time Clustering + Z-order	Clustering (default on ingest, optionally based on keys)
Column Statistics collected on...	Default on first 32 columns, configurable to more (unlimited)	All columns
Stats updated by...	Write operations	Write operations
Caching	FIFO data and result sets on local memory/SSD	FIFO data and result sets on local memory/SSD
Tricks to Reduce IO	Pruning via stats, partitioning, bloom filters, compression	Pruning via stats, compression

APPENDIX 2: DATA TYPES

SQL data type conversions are relevant primarily within the DDL conversion. But DML statements can also have explicit data type conversions (using CAST or short format like colName::datatype). Data types supported in Snowflake can be easily mapped to the data types supported in Databricks. Data types not directly supported, such as VARIANT type, can be easily implemented using STRING types and processed using JSON processing function.

In certain instances, the process of type promotion may happen, which pertains to the process of casting a type into another type within the same type family which contains all possible values of the original type. As a concrete illustration, **TINYINT** has a range from -128 to 127, and all its possible values can be safely promoted to the **INTEGER** type. Refer to the link here and Databricks' official release notes for the full list of supported SQL data types.

Below is the summary of data types conversion rules.

DATA TYPE CATEGORY	SNOWFLAKE DATA TYPE	CONVERTED DATA TYPE	NOTES
Character	1 VARCHAR	1 VARCHAR → STRING	
	2 CHAR	2 CHAR → STRING	
	3 CHARACTER	3 CHARACTER → STRING	
	4 STRING	4 STRING → STRING	
	5 TEXT	5 TEXT → STRING	
	6 BINARY	6 BINARY → STRING	
	7 VARBINARY	7 VARBINARY → STRING	
Numeric	8 NUMBER	8 NUMBER → NUMERIC	* Important Note about differences in supported values ranges for Integer type columns. In Snowflake all Integer types are Synonymous with NUMBER, and defaults to NUMBER(38, 0). This means different types like TINYINT and BIGINT columns can have the same value range. Where as on Databricks, supported column value range depends on the actual integer types
	9 DECIMAL	9 DECIMAL → DECIMAL	* Note that there is a difference in default value of precision for DECIMAL and NUMERIC. Snowflake SQL defaults to 38 where as Databricks SQL default to 10
	10 NUMERIC	10 NUMERIC → STRING*	** Note on float data types from Snowflake docs *"The names FLOAT, FLOAT4, and FLOAT8 are for compatibility with other systems; Snowflake treats all three as 64-bit floating-point numbers"*
	11 INT	11 INT → STRING	
	12 INTEGER	12 INTEGER → INTEGER	
	13 BIGINT	13 BIGINT → BIGINT	
	14 SMALLINT	14 SMALLINT → SMALLINT	
	15 TINYINT	15 TINYINT → TINYINT	
	16 BYTEINT	16 BYTEINT → BYTE	
	17 FLOAT	17 FLOAT → DOUBLE**	
	18 FLOAT4	18 FLOAT4 → DOUBLE**	
	19 FLOAT8	19 FLOAT8 → DOUBLE**	
	20 DOUBLE	20 DOUBLE → DOUBLE	
	21 DOUBLE PRECISION	21 DOUBLE PRECISION → DOUBLE	
	22 REAL	22 REAL → DOUBLE	
Boolean	23 BOOLEAN	23 BOOLEAN → DOUBLE	
DateTime	24 DATE	24 DATE → DATE	* DATETIME is an alias for TIMESTAMP_NTZ in Snowflake
	25 DATETIME	25 DATETIME*	** The usual practice is to create a new column to store an indicator for Timezone while the Timestamp is used for storing the actual value.
	26 TIME	26 TIME → NOT SUPPORTED (use STRING or TIMESTAMP)	
	27 TIMESTAMP	27 TIMESTAMP → TIMESTAMP	
	28 TIMESTAMP_LTZ	28 TIMESTAMP_LTZ**	
	29 TIMESTAMP_NTZ	29 TIMESTAMP_NTZ**	
	30 TIMESTAMP_TZ	30 TIMESTAMP_TZ**	

DATA TYPE CATEGORY	SNOWFLAKE DATA TYPE	CONVERTED DATA TYPE	NOTES
Semi-structured Data Types	VARIANT OBJECT ARRAY	VARIANT → STRING* OBJECT → STRUCT ARRAY → ARRAY**	* Can be mapped to STRING and values processed using built-in JSON parsing functions ** Note that Array type in Snowflake can support values with heterogeneous data types by using variant type whereas Array type values in Databricks need to be homogenous
Geospatial	GEOGRAPHY GEOMETRY	GEOGRAPHY* GEOMETRY*	* While there are no built-in geospatial data types in DeltaLake tables, there are a plethora of options to process geospatial data at scale with Databricks platform by leveraging the open source community developed libraries. Refer to the following blogs: → Processing Geospatial Data at Scale With Databricks → Building a Geospatial Lakehouse, Part 1 → Building a Geospatial Lakehouse, Part 2

APPENDIX 3: EXAMPLE SQL DIFFERENCES

Databricks SQL supports ANSI standard SQL dialect by default. This supports seamless migration on hundreds or thousands of queries from data warehouses. Most of the SQL you have in Snowflake can be dropped in and will just work on Databricks SQL. To make this process simpler for customers, we continue to add SQL features that remove the need to rewrite queries. There are, however, certain query patterns that need to be adjusted. Here are some SQL differences:

	SNOWFLAKE	DATABRICKS
Convention	Naming convention used to refer to tables: ***<schema>.<database>.<table>***	Naming convention used to refer to tables: ***<catalog>.<database>.<table>***
Unquoted Identifiers	Snowflake SQL unquoted identifiers start with a letter or an underscore and can contain letters, digits and dollar sign	Mostly compatible with identifiers on Databricks except they cannot contain a dollar sign
Quoted Identifiers	Snowflake SQL quoted identifiers are enclosed with double quotes(")	Databrick SQL quoted identifiers are enclosed using backtick characters (`)
SQL Variables	Session variables are defined and accessed using SET, UNSET, and SHOW VARIABLES statements	For Batch workloads Spark session parameters can be set and variable substitution in SQL is enabled by default using syntax ${varName} Additionally, Widgets in Notebooks and Query Parameters in Databricks SQL can be used for passing arguments

	SNOWFLAKE	DATABRICKS
Time Travel	AT or BEFORE clauses are used for time travel in Snowflake Example: `SELECT * FROM table_x AT(TIMESTAMP => '2019-01-29 00:37:58'::timestamp)`	Databricks supports time travel using temporal specification along with Delta Lake table name. Databricks Syntax (with timestamp) maps to the AT clause in Snowflake Syntax as it is inclusive. Example: `SELECT * FROM table_x TIMESTAMP AS OF '2019-01-29 00:37:58'`
Change Tracking	`CHANGE_TRACKING = TRUE | FALSE` table attribute is used in Snowflake to enable change tracking	In Databricks the change tracking can be enabled using the table property: `TBLPROPERTIES (delta.enableChangeDataFeed = true)`

Delete Records	1) `DELETE FROM TABLE_A;`
	
	If using a USING clause in DELETE
	
	2) `DELETE FROM TABLE_A`
	`USING (SELECT X FROM TABLE_B)`
	`as TABLE_B`
	`WHERE TABLE_A.X = TABLE_B.X;`
		1) `DELETE FROM TABLE_A;`
		
		Use the MERGE operation:
		
		2) `MERGE INTO TABLE_A`
		`USING (SELECT X FROM TABLE_B)`
		`as TABLE_B`
		`ON TABLE_A.X = TABLE_B.X`
		`WHEN MATCHED THEN DELETE;`

Update Records	1) `UPDATE TABLE_A`
	`SET COL_A='A'`
	`WHERE COL_B='B';`
	
	If using a FROM clause in UPDATE
	
	2) `UPDATE TABLE_A`
	`SET COL_A= TABLE_B.COL_A`
	`FROM TABLE_B`
	`WHERE TABLE_A.X = TABLE_B.X;`
		1) `UPDATE TABLE_A`
		`SET COL_A='A'`
		`WHERE COL_B='B';`
		
		Use the MERGE operation:
		
		2) `MERGE INTO TABLE_A`
		`USING (SELECT COL_A FROM`
		`TABLE_B) as TABLE_B`
		`ON TABLE_A.X = TABLE_B.X`
		`WHEN MATCHED THEN UPDATE;`

Loading Data to Tables	`COPY INTO` command is used to load files in Stage to an existing table	The `COPY INTO` command is comparable in functionality
Unloading Data from Tables	`COPY INTO` command is used to unload from a table to Stage location	Databricks doesn't use COPY INTO to unload. Instead use a. INSERT OVERWRITE DIRECTORY (only on Databricks Runtime) b. Use EXTERNAL TABLE definition pointing to required relocation c. Use one of the Spark DataFrame API