



TERADATA TO SNOWFLAKE MIGRATION REFERENCE MANUAL



TABLE OF CONTENTS

- 2** Introduction
- 3** Preparing for the migration
- 8** Executing the migration
- 13** Migration success factors
- 14** Need help migrating?
- 15** Appendix A—Teradata databases to exclude when migrating to Snowflake
- 16** Appendix B—Teradata data types conversion to Snowflake data types
- 18** Appendix C—Query to evaluate Teradata data type usage
- 19** Appendix D—Other known migration issues from Teradata to Snowflake
- 20** Appendix E—Comparing data from Teradata with Snowflake
- 21** About Snowflake

INTRODUCTION

The purpose of this document is to provide the high level methodology needed to prepare for and execute the migration of an existing Teradata system to Snowflake. The appendices at the end of this document provide guidance relating to the differences between Teradata and Snowflake that you should consider as part of the migration.

The intended audiences of this document are solution architects, program managers and Snowflake solution partners that need a clearly defined approach on how to migrate an existing Teradata system to Snowflake.

PREPARING FOR THE MIGRATION

Jumping right into a migration from Teradata to Snowflake without proper preparation is not a wise move. There are many components to consider for such an endeavor. This section gives you a checklist of information to gather and decisions to make before you start the actual migration.

DOCUMENT THE EXISTING SOLUTION

Key outcomes:

- List of Teradata databases to migrate
- List of Teradata database objects to migrate
- List of processes and tools that populate and pull data from Teradata
- List of security roles, users and permissions
- List of Snowflake accounts that exist or need creating
- Frequency of security provisioning processes
- Documentation of the existing Teradata solution into an as-is architecture diagram

Begin preparing for your migration from Teradata to Snowflake by determining which Teradata databases within the Teradata system need migrating and which ones don't. Identify and document the Teradata

database objects within the Teradata databases that need migrating, including the size of the data, to establish the scope of the migration project. Plan not to migrate Teradata specific databases, such as DBC, TDWM, SYSLIB, TDStats, etc., since they aren't needed in Snowflake. A full list of Teradata databases to exclude from the migration are listed in Appendix A.

When you aren't sure which databases and database objects to migrate, and if DBQL has been enabled in Teradata, you can query the data within the DBQL tables to determine which databases and database objects are being used. Avoid moving unused objects, unless you need them for audit or historical purposes.

Once you've identified the Teradata databases and database objects that need migrating, the next step is to evaluate each data source that populates them to determine whether they come from on-premise and/or a cloud-based sources. This will help determine the methods available for loading the data into Snowflake. Specifically, will you need to load terabytes or even petabytes of on-premise data into Snowflake? If so, you may require capabilities such as AWS Snowball, Azure Data Box, or Google Transfer Appliance to move the data as efficiently as possible.

In addition to evaluating the data sources that populate Teradata, identify the processes and tools that move data in and out of Teradata and document them. Here are some examples:

- ETL/ELT tools
- Scripting languages
- Reporting/visualization tools
- Data science processes
- Machine learning processes

Use this list to evaluate the level of Snowflake support for the tools you currently use, and to provide guidance on what migration approach would best fit your needs.

Document the roles and users that currently exist within the Teradata system, and the granted permissions, to prepare for the security implementation in Snowflake. Pay special attention to sensitive data sets and how they're secured within Teradata. Also, determine how frequently security provisioning processes run in order to create similar security within Snowflake. In addition, capture the Snowflake accounts already set up and any Snowflake accounts needed for the migration, since they will have an impact on the security implementation.

If you do not have this information readily available, Snowflake Professional Services and/or a Snowflake solution partner can help capture this information.

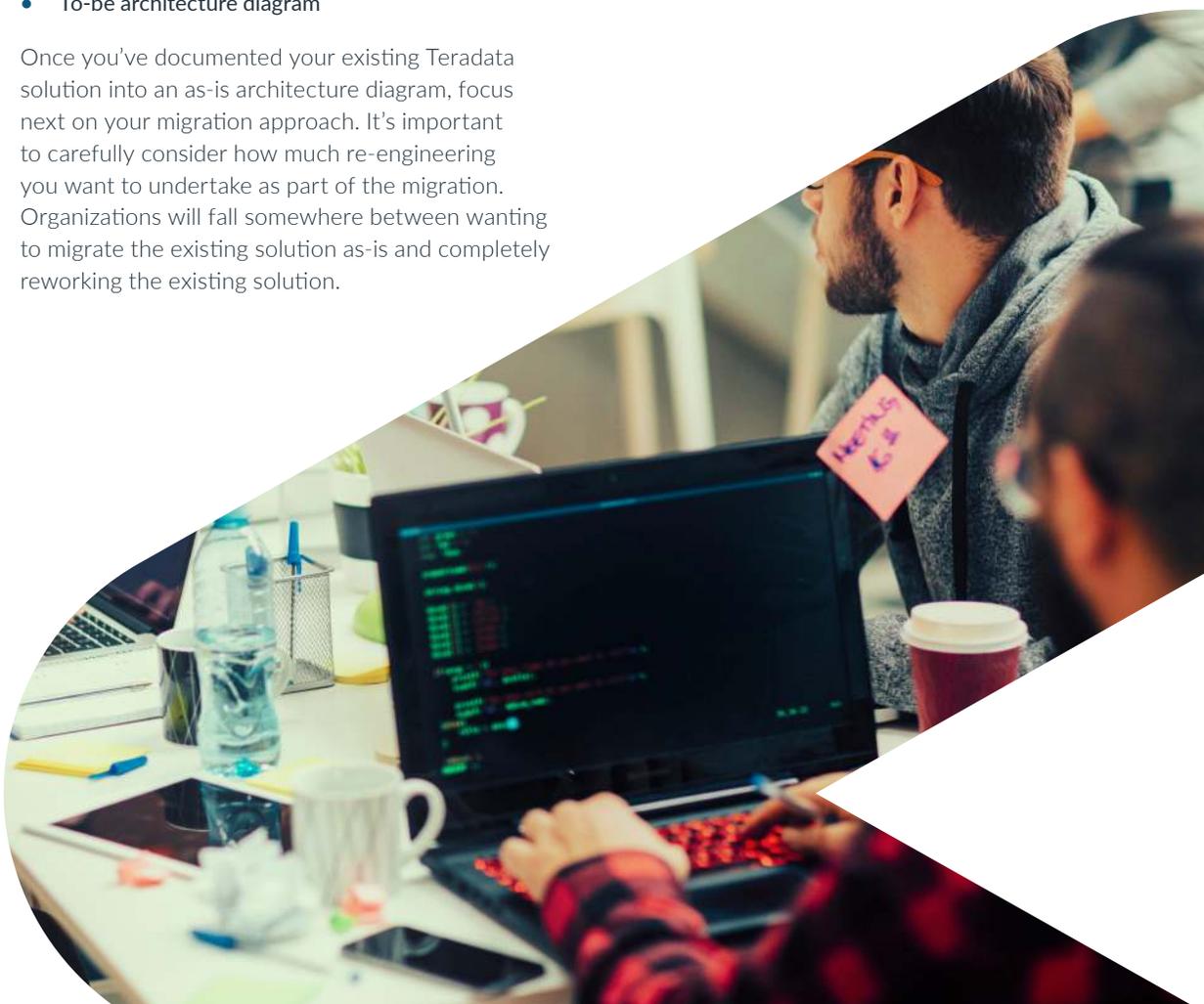
ESTABLISH A MIGRATION APPROACH

Key outcomes:

- List of processes to migrate as-is
- List of processes that need reengineering
- List of processes that need fixing
- Draft of migration deliverables
- To-be architecture diagram

Once you've documented your existing Teradata solution into an as-is architecture diagram, focus next on your migration approach. It's important to carefully consider how much re-engineering you want to undertake as part of the migration. Organizations will fall somewhere between wanting to migrate the existing solution as-is and completely reworking the existing solution.

Snowflake generally recommends minimal re-engineering for the first iteration unless your current system is truly broken. When you decide what you will re-engineer, remember that changes to the existing data structures will have an impact on downstream reporting/visualization tools. Also, more re-engineering requires more development and testing, which extends the length of a migration project.



There may also be issues with your existing implementation that must be resolved as part of the migration, so you should include these in your migration plan.

To avoid a single, big-bang, migration approach, break the migration into incremental deliverables that enable your organization to start making the transition to Snowflake more quickly. This will also provide value to your business sooner.

Use the as-is architecture diagram to create a to-be architecture diagram for communicating the migration approach and ensuring the approach meets the requirements of the business.

CAPTURE THE DEVELOPMENT AND DEPLOYMENT PROCESSES

Key outcomes:

- List of tools introduced with the migration
- List of tools deprecated after the migration
- List of development environments needed for the migration
- List of deployment processes used for the migration

Depending on your migration approach, you may introduce new tools and/or deprecate old tools as part of the migration. Since you documented your existing tools and processes in an earlier step, this is when you should document plans to introduce new tools or deprecate old tools.

Your organization may want to change your development or deployment processes as part of the migration. Whether these processes change or not, capture the development environments used for the migration (e.g. Pre-Prod/Prod, Dev/QA/Prod), and the deployment processes (e.g. source

control repository, method for deploying changes from one environment to another) used for the migration. This information is critical to how you will implement development and deployment.

PRIORITIZE DATA SETS FOR MIGRATION

Key outcomes:

- List of data sets to migrate first
- Method for identifying process dependencies for data sets
- Documentation of process dependencies for data sets

To deliver value as soon as possible, this step identifies which data sets you should migrate first. The ideal candidates for starting the migration provide value to the business with minimal migration effort. Rather than starting with the most complex data sets, begin with a simpler data set that provides a quick win and establishes a foundation of development and deployment processes from which to build the rest of the migration.

When you have a prioritized list of data sets, leverage it with the above principles in mind. If you don't have a prioritized list, identify those data sets and engage Snowflake Professional Services and/or a Snowflake solution partner, if necessary, to help capture this information.

To prioritize data sets for migration, pay careful attention to understanding the process dependencies of the data sets. Document those dependencies. By identifying dependencies through a solid process before beginning the migration work, you will experience fewer blockers during the migration.

```

mirror_ob.select=1
elif_operation
mirror_mod.use_x
mirror_mod.use_y = 1
mirror_mod.use_z = True

#selection at the end -add back
mirror_ob.select= 1
modifier_ob.select=1
bpy.context.scene.objects.active = modifier_ob
print("Selected" + str(modifier_ob)) # modifier ob
mirror_ob.select = 0

```

Ideally, you can create this dependency documentation using an automated process that iterates through the existing job schedules and captures the data within Snowflake. This eliminates having to depend on manual investigation. Using an automated process pays dividends throughout the migration project by updating dependency documentation as changes take place. This is important since the underlying systems are unlikely to be static during the migration.

IDENTIFY THE MIGRATION TEAM

Key outcomes:

- List of migration team members and roles
- Contact information for all team members

To complete the migration, document the people involved in the migration and the roles they will play. The documentation should include each team member's name, contact information and role. Team members may come from your team, Snowflake staff and/or a Snowflake solution partner.

Some of the obvious roles required for a migration are developer, quality assurance engineer, business owner, project manager, program manager, scrum master and communication.

When you engage Snowflake Professional Services and/or a Snowflake solution partner for a migration, they can fulfill multiple needs including solution design, requirements gathering, documentation, development, testing, delivery, and training. The

entire team works together to successfully complete the migration and communicate the progress of the migration to stakeholders.

DEFINE THE MIGRATION DEADLINES AND BUDGET

Key outcomes:

- List of business expectations for the migration deadline
- Documented migration plan and budget required for the migration project
- Template of estimated costs to run Snowflake

The expectations for when you complete your migration are an important planning input. In addition, consider other information such as the budget, availability of resources and amount of re-engineering required. By gathering all of this information, you can establish and communicate achievable deadlines, even if the deadlines are different than what was originally expected.

It's also critically important to create a migration plan, in order to understand the budget required to complete the migration. Snowflake Professional Services and/or a Snowflake solution partner can help estimate the migration costs and migration timeline for an end-to-end migration plan, as well as provide code conversion services to help speed up the migration and reduce the overall migration costs. Compare the amount of migration work and the costs associated with the migration to the available budget to ensure you have sufficient resources to complete the migration work.



A key input into the budget considerations are the Snowflake virtual warehouses (compute clusters) required during and after the migration. A Snowflake representative can provide a template, and work with you to capture the virtual warehouses needed to do the work (e.g. ETL/ELT, reporting/visualization, etc.). The template calculates the number of minutes a warehouse is expected to run each day, and the number of days a warehouse is expected to run each week. Once you complete the template, you will get an estimated, annual cost.

DETERMINE THE MIGRATION OUTCOMES

Key outcomes:

- List of high-level outcomes once the migration is completed
- Documented plan for communicating the migration project wins to stakeholders

The final step of preparing for your migration is to capture the high-level outcomes the migration achieved and the benefits provided to the business. For example, turning off a Teradata system could be one of your desired outcomes.

The documentation can also include benchmarks that compare process execution on Teradata and Snowflake. Once compiled, use this information to communicate the wins of the migration project to stakeholders.



EXECUTING THE MIGRATION

After gathering the information and decisions needed to prepare for the migration, it's time to execute the migration. This section guides you through the steps required to complete the migration.

If you need assistance with any part of executing the migration from Teradata to Snowflake, check with your Snowflake representative for recommended Snowflake Professional Services and/or Snowflake solution partners.

ESTABLISH SECURITY

When first setting up a Snowflake account, you can manually create users and roles to get up and running. From there, make it a priority to move to an automated process that creates users and assigns them to roles and removes them when they are no longer applicable. Depending on your security auditing requirements, create processes to capture role and user creation/deletion, and the granting/revoking of users to/from roles.

Your existing Teradata system security can be a good starting point for setting up security within Snowflake. However, determine if there are Teradata roles and users that are no longer needed or should be implemented differently as part of your migration to Snowflake.

Start by creating roles for at least the first data sets you will migrate. Then create users to assign to those roles, and assign the users to the roles based on their

migration work. With this setup complete, users will be able to log into Snowflake and see the roles they belong to in preparation for the next step of creating databases and warehouses.

You can establish common roles for developer access for non-production databases, read-only access, read and write access, and administrative access. You may require additional roles for restricting access to sensitive data.

DEVELOP A TEST PLAN

Determine and execute the appropriate level and scope of testing for each environment (e.g. schedules are executed only in QA and Prod, and data comparisons between Teradata and Snowflake occur only for Prod, etc.). Automate testing as much as possible, so it's repeatable and provides results for identifying any issues. Define, document and get agreement on acceptance criteria for the tests.

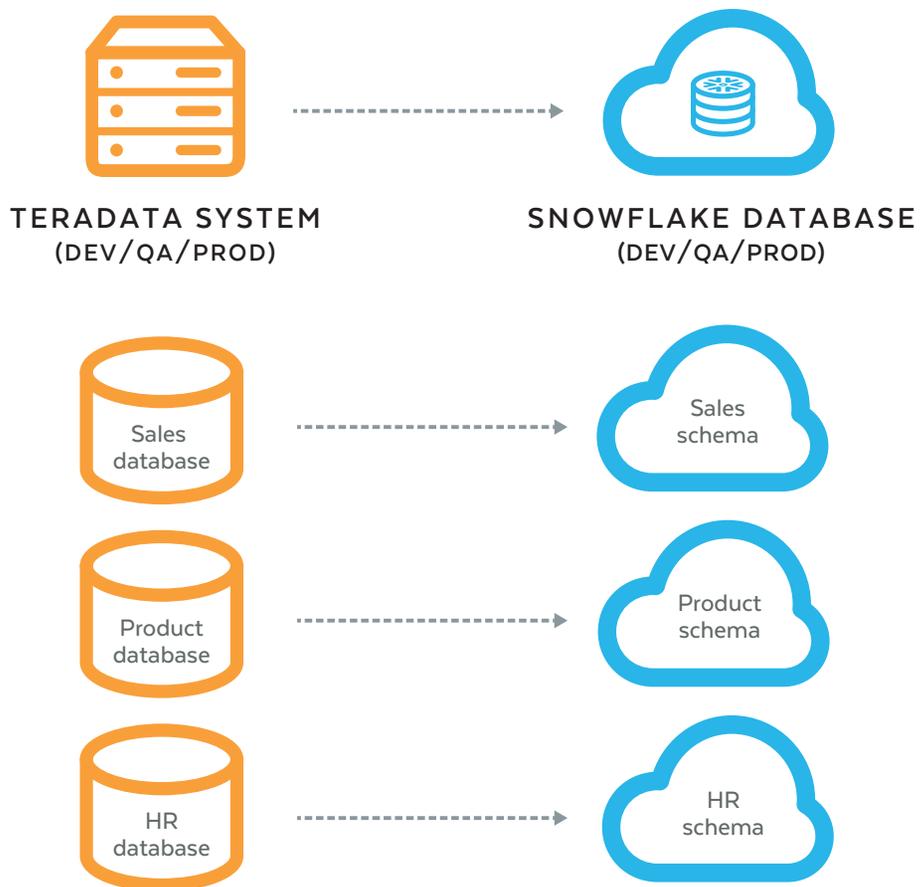
PREPARE SNOWFLAKE FOR LOADING

There are a couple of options for setting up your Snowflake implementation, depending on the number of Snowflake accounts you have. When you have one Snowflake account, create a Snowflake database for each Teradata environment that you need to migrate (for example, Dev/QA/Prod), instead of creating a Snowflake database for each Teradata database. In addition, create schemas for each of the Teradata databases you intend to migrate to Snowflake. The diagram provides examples of migrating a Teradata environment to a Snowflake database and Teradata databases to Snowflake schema using a single Snowflake account.

When you have multiple Snowflake accounts, you have the option of creating the Snowflake databases to match the Teradata databases or creating the Snowflake schemas to match the Teradata databases.

This approach clearly identifies the environment and uses schemas to contain the tables and views to more easily point tools from Teradata to Snowflake. Please be aware that since the Snowflake database contains the environment in its name, you will need to update any views that reference a database, as the view is deployed from one environment to another (e.g. deploying from QA to Prod).

After you create the databases and schemas in Snowflake, you can execute the DDL for creating the database objects in Snowflake.



Create the virtual warehouses based on the information captured during the migration preparation. There should be separate virtual warehouses for each function that the virtual warehouses will support (for example, ETL/ELT, reporting/visualization, etc.). The diagram contains a reference architecture for using virtual warehouses for different workloads.

You may require an AWS Snowball, Azure Data Box, or Google Transfer Appliance if your Teradata system is on-premises and you need to move terabytes or petabytes of data.

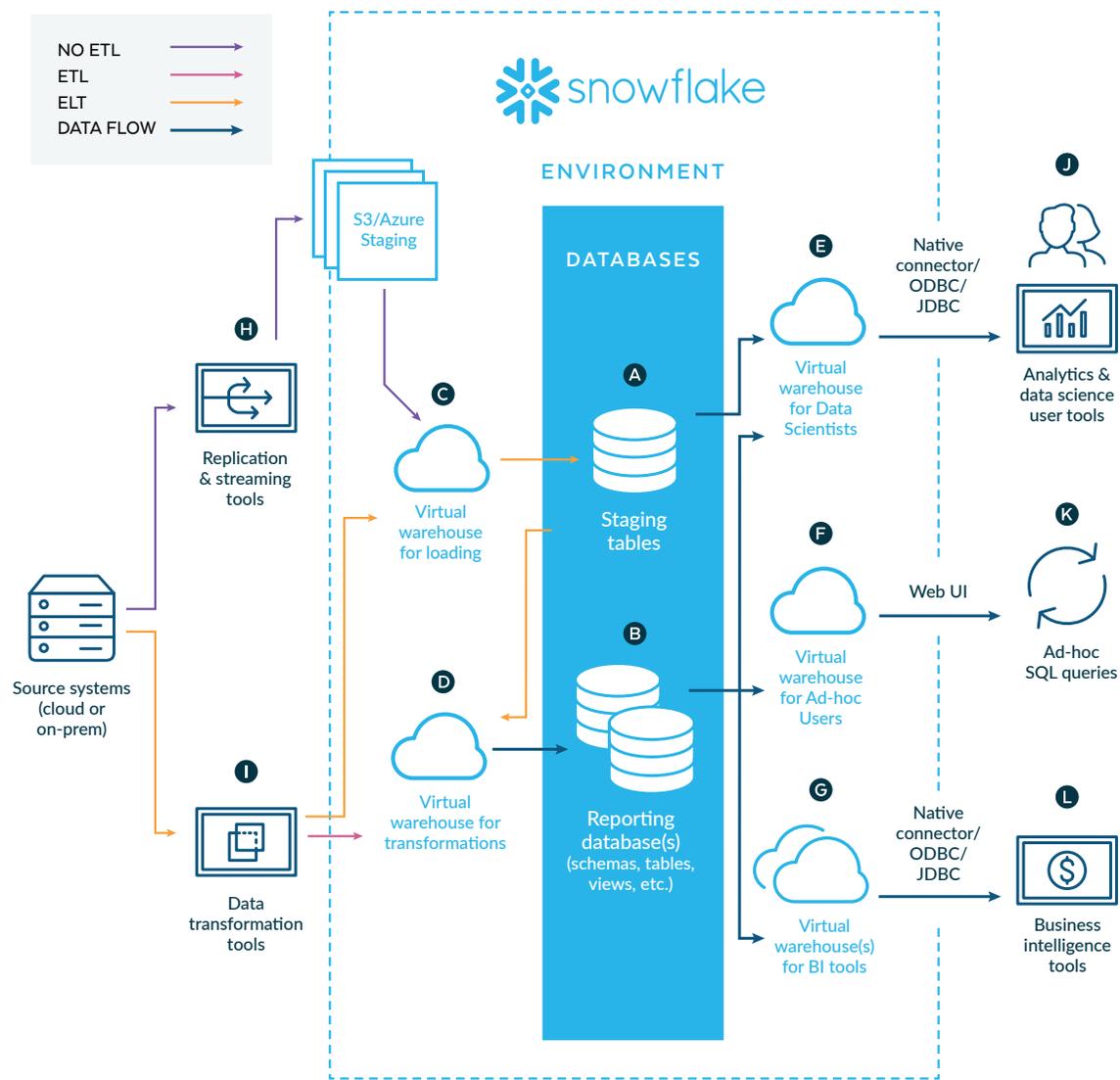
You should base the initial sizing of the virtual warehouses on the estimates done while preparing for the migration. Then, adjust as needed throughout the migration. Also, set up resource monitors to track usage and take appropriate action when limits are reached.

As you create the databases, database objects and virtual warehouses, assign the appropriate security roles.

LOAD INITIAL DATA SETS

You may require an AWS Snowball or Azure Data Box if your Teradata system is on-premise and you need to move terabytes or petabytes of data. Add an appropriate amount of time to the migration schedule to provision these boxes, load them with data, transport them to the cloud data center and offload the data into the cloud servers.

You can load data into Snowflake after you've



extracted the data from Teradata and moved the data to the cloud. Use this data loading to test the configuration of the databases and database objects, and virtual warehouses, as well as the security you have implemented.

There may be an opportunity to use cloning to move data within Snowflake from one database to another, depending on which Teradata environment the data came from and which Snowflake database you populate. Cloning in Snowflake doesn't require additional storage. Therefore, you avoid the headache and cost of loading the same data multiple times into different Snowflake databases.

Plan to extract data from Teradata and load it into Snowflake more than once. Also, begin with a subset of the data from Teradata, rather than trying to load the entire contents of the Teradata system at the beginning of the migration.

KEEP DATA UP TO DATE

You can implement the processes to keep the data current after you load the historical data sets from Teradata into Snowflake.

Set up the appropriate data loading schedules to reflect the existing Teradata loading processes or new processes to load into Snowflake. This is another opportunity to evaluate whether changes to the schedule would be beneficial and should be part of the migration.

To ensure you populate data in the correct order, create the appropriate schedules based on a clear understanding of the process dependencies captured

as part of preparing for the migration.

Along with scheduling the processes to run, monitor those processes so you can clearly understand and communicate the state of the data (e.g. loading is in progress, loading completed successfully, loading failures occurred). Use the monitoring to verify SLAs are being met within Snowflake, or to identify process issues that need resolving.

IMPLEMENT THE TEST PLAN

Begin the Snowflake implementation with the initial data sets loaded and processes running in order to keep the data current. Once this happens, start testing your Snowflake implementation. Be sure to engage the team members identified during the

preparation for the migration and additional groups to test their data sets and applications against Snowflake. Engage these additional groups after you complete initial testing and validate the data is ready for further scrutiny.

Compare data between the Teradata and Snowflake environments throughout the migration to provide a high level of confidence it went as planned. If there



are data differences, investigate to determine the cause and how to resolve the issue(s).

Part of your migration may include fixing processes known to be incorrect in Teradata. Knowing this, the test results may not match Snowflake, so use methods to validate the data is correct in Snowflake. Document and share any reasons why data won't match between Teradata and Snowflake with the groups performing the testing, so they don't spend time researching previously identified issues.

Also, compare the performance of the processes that load and consume data to ensure Snowflake is performing as expected. Share these comparisons with stakeholders to highlight the benefits of migrating from Teradata to Snowflake.

RUN TERADATA AND SNOWFLAKE IN PARALLEL

During the migration, you'll have to run the Teradata and Snowflake systems in parallel. We recommend keeping this to a minimum but long enough to validate you've completed the migration successfully before shutting down Teradata.

While running them in parallel, consider how to best run Teradata and Snowflake to compare data and performance. For example, you may need to create hashes as you extract data from Teradata in order to compare data at the row level between Teradata and Snowflake (This approach is explained further in Appendix E). Perform these comparisons in Snowflake to keep from negatively impacting your Teradata system.

REDIRECT TOOLS TO SNOWFLAKE

Use the list of tools you gathered while preparing for the migration, and the information on the level of support each tool has for Snowflake. Then, update the tool connections to redirect to Snowflake after you've migrated a sufficient amount of data to Snowflake for each tool.

Redirecting tools to Snowflake usually involves creating copies of the existing solution that connects to Teradata within the tool and updating the solution to connect to Snowflake instead. Compare the output of the tools to ensure the results are the same between Teradata and Snowflake. In addition, evaluate the performance of the tool to verify it's performing as expected in Snowflake.

CUT OVER TO SNOWFLAKE

The cut over from Teradata to Snowflake can only occur once you've migrated the initial data, enabled processes to keep the data current, completed

testing that verifies you've successfully migrated the data, and redirected the tools from Teradata to Snowflake.

Make sure you've planned and communicated the cut-over date in advance to your Teradata users. They should have the ability to log into Snowflake and run the tools they depend on which you've redirected to Snowflake.

To complete the cut-over, turn off data processes that populate Teradata. In addition, revoke access to Teradata so users and tools no longer have access.



MIGRATION SUCCESS FACTORS

Paying attention to certain success factors will reduce your risks in order to successfully complete the migration. This section provides insight into how to increase the speed of delivery of a successful migration from Teradata to Snowflake.

IDENTIFY AND MITIGATE DIFFERENCES BETWEEN TERADATA AND SNOWFLAKE

Use Appendix D at the end of this reference manual to identify issues early in the migration process.

Present these issues to the business along with available mitigation strategies. Then, confirm that your proposed approach will meet their requirements.

RESOLVE MIGRATION ISSUES

There will inevitably be issues that occur during and after the migration. Establish processes to document and escalate migration issues so you can resolve them as quickly as possible.

The escalation process needs to document each issue, who is responsible for working on the issue, who is responsible for communicating the progress on the issue, and a list of contacts from your

business, from Snowflake, and from any other parties involved in resolving the issue. Be sure everyone involved has access to, and knows how to, log a support ticket in the [Snowflake Community](#). Likewise, they should know how to ask questions and find resources in the [Snowflake Community](#) and on Stack Overflow.

With the list of issues documented, establish a regular cadence for reviewing the issues and getting an updated status on resolving each issue.

You may also identify issues during the migration that don't need resolving during the migration. Document and prioritize them, so you can work on them post-migration.

COMMUNICATE MIGRATION BENEFITS

Use the high-level outcomes captured while preparing for the migration to document the actual, corresponding benefits that occurred. Publish these results to stakeholders, so they clearly understand the benefits of the migration.

NEED HELP MIGRATING?

Snowflake is available to accelerate your migration; structure and optimize your planning and implementation activities; and apply customer best practices to meet your technology and business objectives. Snowflake's Professional Services Team deploys a powerful combination of data architecture expertise and advanced technical knowledge of the platform to deliver high performing data strategies, proof of concepts and migration projects.

Our global and regional solution partners also have extensive experience performing proof of concepts and platform migrations. They offer services ranging from high-level architectural recommendations to manual code conversions. A number of Snowflake partners have also built tools to automate and accelerate the migration process.

Whether your organization is fully staffed for a platform migration or you need additional expertise, Snowflake Professional Services and/or our solution partners have the skills and tools to accelerate your journey to cloud-built data analytics, so you can reap the full benefits of Snowflake quickly. To find out more, please contact the Snowflake sales team or visit the [Snowflake's Community](#).



APPENDIX A: TERADATA DATABASES TO EXCLUDE WHEN MIGRATING TO SNOWFLAKE

The following list of databases are needed for Teradata only and shouldn't be migrated to Snowflake:

- DBC
- Crashdumps
- dbcmngr
- External_AP
- EXTUSER
- LockLogShredder
- QCD
- SQLJ
- Sys_Calendar
- SysAdmin
- SYSBAR
- SYSJDBC
- SYSLIB
- SYSSPATIAL
- SystemFE
- SYSUDTLIB
- SYSUIF
- TD_SERVER_DB
- TD_SYSFNLIB
- TD_SYSGPL
- TD_SYSXML
- TDPUSER
- TDQCD
- TDStats
- tdwm

APPENDIX B: CONVERTING TERADATA DATA TYPES TO SNOWFLAKE DATA TYPES

Teradata Column Type	Teradata Data Type	Snowflake Data Type
++	TD_ANYTYPE	TD_ANYTYPE data type isn't supported in Snowflake.
A1	ARRAY	ARRAY
AN	ARRAY	ARRAY
AT	TIME	TIME
BF	BYTE	BINARY
BO	BLOB	BLOB data type isn't directly supported but can be replaced with BINARY (limited to 8MB).
BV	VARBYTE	BINARY
CF	CHAR	VARCHAR
CO	CLOB	CLOB data type isn't directly supported but can be replaced with VARCHAR (limited to 16MB).
CV	VARCHAR	VARCHAR
D	DECIMAL	NUMBER
DA	DATE	DATE
DH	INTERVAL DAY TO HOUR	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
DM	INTERVAL DAY TO MINUTE	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
DS	INTERVAL DAY TO SECOND	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
DT	DATASET	DATASET data type isn't supported in Snowflake.
DY	INTERVAL DAY	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
F	FLOAT	FLOAT
HM	INTERVAL HOUR TO MINUTE	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
HR	INTERVAL HOUR	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
HS	INTERVAL HOUR TO SECOND	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
I1	BYTEINT	NUMBER
I2	SMALLINT	NUMBER
I8	BIGINT	NUMBER
I	INTEGER	NUMBER
JN	JSON	VARIANT
LF	CHAR	This data type is in DBC only and can't be converted to Snowflake.
LV	VARCHAR	This data type is in DBC only and can't be converted to Snowflake.

APPENDIX B: CONVERTING TERADATA DATA TYPES TO SNOWFLAKE DATA TYPES (CONT'D)

Teradata Column Type	Teradata Data Type	Snowflake Data Type
MI	INTERVAL MINUTE	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
MO	INTERVAL MONTH	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
MS	INTERVAL MINUTE TO SECOND	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
N	NUMBER	NUMBER
PD	PERIOD(DATE)	Can be converted to VARCHAR or split into 2 separate dates.
PM	PERIOD(TIMESTAMP WITH TIME ZONE)	Can be converted to VARCHAR or split into 2 separate timestamps (TIMESTAMP_TZ).
PS	PERIOD(TIMESTAMP)	Can be converted to VARCHAR or split into 2 separate timestamps (TIMESTAMP_NTZ).
PT	PERIOD(TIME)	Can be converted to VARCHAR or split into 2 separate times.
PZ	PERIOD(TIME WITH TIME ZONE)	Can be converted to VARCHAR or split into 2 separate times but WITH TIME ZONE isn't supported for TIME.
SC	INTERVAL SECOND	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
SZ	TIMESTAMP WITH TIME ZONE	TIMESTAMP_TZ
TS	TIMESTAMP	TIMESTAMP_NTZ
TZ	TIME WITH TIME ZONE	TIME WITH TIME ZONE isn't supported because TIME is stored using "wall clock" time only without a time zone offset.
UF	CHAR	This data type is in DBC only and can't be converted to Snowflake.
UT	UDT	UDT data type isn't supported in Snowflake.
UV	VARCHAR	This data type is in DBC only and can't be converted to Snowflake.
XM	XML	VARIANT
YM	INTERVAL YEAR TO MONTH	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).
YR	INTERVAL YEAR	INTERVAL data types aren't supported in Snowflake but date calculations can be done with the date comparison functions (e.g. DATEDIFF and DATEADD).

APPENDIX C: QUERY TO EVALUATE TERADATA DATA TYPE USAGE

The query below provides counts for each Teradata data type in use on the system. You can update the WHERE clause of the query to include the specific list of databases migrated from Teradata to Snowflake. Use the results of this query, along with the list of Teradata and Snowflake data types, to identify the prevalence of data types that may pose challenges during the migration.

```

SELECT
ColumnType,
CASE
WHEN ColumnType = '+' THEN 'TD_ANYTYPE'
  WHEN ColumnType = 'A1' THEN 'ARRAY'
  WHEN ColumnType = 'AN' THEN 'ARRAY'
  WHEN ColumnType = 'AT' THEN 'TIME'
  WHEN ColumnType = 'BF' THEN 'BYTE'
  WHEN ColumnType = 'BO' THEN 'BLOB'
  WHEN ColumnType = 'BV' THEN 'VARBYTE'
  WHEN ColumnType = 'CF' THEN 'CHAR'
  WHEN ColumnType = 'CO' THEN 'CLOB'
  WHEN ColumnType = 'CV' THEN 'VARCHAR'
  WHEN ColumnType = 'D' THEN 'DECIMAL'
  WHEN ColumnType = 'DA' THEN 'DATE'
  WHEN ColumnType = 'DH' THEN 'INTERVAL DAY TO HOUR'
  WHEN ColumnType = 'DM' THEN 'INTERVAL DAY TO MINUTE'
  WHEN ColumnType = 'DS' THEN 'INTERVAL DAY TO SECOND'
  WHEN ColumnType = 'DT' THEN 'DATASET'
  WHEN ColumnType = 'DY' THEN 'INTERVAL DAY'
  WHEN ColumnType = 'F' THEN 'FLOAT'
  WHEN ColumnType = 'HM' THEN 'INTERVAL HOUR TO MINUTE'
  WHEN ColumnType = 'HR' THEN 'INTERVAL HOUR'
  WHEN ColumnType = 'HS' THEN 'INTERVAL HOUR TO SECOND'
  WHEN ColumnType = 'I1' THEN 'BYTEINT'
  WHEN ColumnType = 'I2' THEN 'SMALLINT'
  WHEN ColumnType = 'I8' THEN 'BIGINT'
  WHEN ColumnType = 'I' THEN 'INTEGER'
  WHEN ColumnType = 'JN' THEN 'JSON'
  WHEN ColumnType = 'MI' THEN 'INTERVAL MINUTE'
  WHEN ColumnType = 'MO' THEN 'INTERVAL MONTH'
  WHEN ColumnType = 'MS' THEN 'INTERVAL MINUTE TO SECOND'
  WHEN ColumnType = 'N' THEN 'NUMBER'
  WHEN ColumnType = 'PD' THEN 'PERIOD(DATE)'
  WHEN ColumnType = 'PM' THEN 'PERIOD(TIMESTAMP WITH TIME ZONE)'
  WHEN ColumnType = 'PS' THEN 'PERIOD(TIMESTAMP)'
  WHEN ColumnType = 'PT' THEN 'PERIOD(TIME)'
  WHEN ColumnType = 'PZ' THEN 'PERIOD(TIME WITH TIME ZONE)'
  WHEN ColumnType = 'SC' THEN 'INTERVAL SECOND'
  WHEN ColumnType = 'SZ' THEN 'TIMESTAMP WITH TIME ZONE'
  WHEN ColumnType = 'TS' THEN 'TIMESTAMP'
  WHEN ColumnType = 'TZ' THEN 'TIME WITH TIME ZONE'
  WHEN ColumnType = 'UT' THEN 'UDT'
  WHEN ColumnType = 'XM' THEN 'XML'
  WHEN ColumnType = 'YM' THEN 'INTERVAL YEAR TO MONTH'
  WHEN ColumnType = 'YR' THEN 'INTERVAL YEAR'
END AS Data_Type,
COUNT(*) AS Data_Type_Count
FROM DBC.ColumnsV
WHERE DatabaseName NOT IN ('DBC', 'Crashdumps', 'dbcmngr', 'External_AP',
'EXTUSER', 'LockLogShredder', 'QCD', 'SQLJ', 'Sys_Calendar', 'SysAdmin', 'SYSBAR',
'SYSJDBC', 'SYSLIB', 'SYSSPATIAL', 'SystemFE', 'SYSUDTLIB', 'SYSUIF', 'TD_SERVER_
DB', 'TD_SYSFNLIB', 'TD_SYSGPL', 'TD_SYSEXML', 'TDPUSER', 'TDQCD', 'TDStats',
'tdwm')
GROUP BY 1,2
ORDER BY 1;

```

APPENDIX D: OTHER KNOWN MIGRATION ISSUES FROM TERADATA TO SNOWFLAKE

CASE SENSITIVITY

Teradata is case insensitive by default. While Teradata supports data being stored as CASESPECIFIC, data in Teradata is commonly stored without being case specific. Since Snowflake is case sensitive (e.g. 'Glass', 'GLASS', and 'glass' are 3 different values), during the migration, give attention to checking for comparison issues in queries. One simple solution to this issue is to use UPPER on both sides of a comparison (e.g. WHERE UPPER(COLUMNNAME)=UPPER(COLUMNNAME)), if the desired result is for any differences in case to be ignored.

ENFORCEMENT OF PRIMARY KEYS AND FOREIGN KEYS

Teradata enforces Primary Keys and Foreign Key constraints. While Snowflake supports the syntax to define Primary Keys and Foreign Keys, they aren't enforced within Snowflake. This means you'll need to re-engineer load processes that depend on constraints to prevent duplicate entries and orphaned records from being entered into the data warehouse.

DATE VS. TO_DATE()

Teradata has the capability to put DATE in front of a string in order to treat it as a date value (e.g. DATE '2018-12-31'). In Snowflake, the syntax is TO_DATE() (e.g. TO_DATE('2018-12-31')). It isn't necessary to use DATE or TO_DATE() in many situations, since both Teradata and Snowflake can interpret the date values

stored in a string. When migrating SQL from Teradata to Snowflake, it may be more desirable to replace DATE with TO_DATE() rather than dropping DATE altogether. rather than dropping DATE altogether.

TERADATA PERIOD DATA TYPE

Snowflake doesn't support the PERIOD data type, so when migrating from Teradata to Snowflake, PERIOD columns will need to be split into two separate date or timestamp columns or changed to a VARCHAR. The load process for those columns will need to account for this change.

UPDATING DATA THROUGH A VIEW

Teradata allows inserts, updates and deletes to be executed against a view, which will then update the underlying table. In Snowflake, inserts, updates and deletes must be executed against a table and can't be executed against a view. Again, load processes may need to be re-engineered to account for this.

UPDATE SYNTAX

Teradata allows the FROM in an UPDATE statement to come before the SET statement. In Snowflake, the UPDATE syntax requires that the FROM comes after the SET statement.

DELETE ALL SYNTAX

Teradata supports adding ALL to the end of a DELETE statement. In Snowflake, adding ALL to the end of a DELETE statement isn't supported and needs to be removed.

TERADATA-SPECIFIC SYNTAX

Teradata has SQL syntax for creating tables (DDL) that isn't used in Snowflake:

- SET/MULTISET
- FALLBACK
- PRIMARY INDEX
- PARTITION BY
- COMPRESS
- FORMAT
- INDEXES

TERADATA HAS SQL SYNTAX WITH VIEWS (DDL) THAT ISN'T USED IN SNOWFLAKE:

- LOCKING ROW FOR ACCESS
- SEL (must be spelled out as SELECT)
- DEL (must be spelled out as DELETE)

Also REPLACE VIEW syntax in Teradata should be updated to CREATE OR REPLACE VIEW in Snowflake.

APPENDIX E: COMPARING DATA FROM TERADATA WITH SNOWFLAKE

Use row counts and sums of numeric data to validate data matches between Teradata and Snowflake. Another way to confirm you've successfully loaded all data into Snowflake is to get unique values from columns in Teradata and compare those unique values with Snowflake.

For use cases where you require more in-depth validation, add an MD5 hash to the data extracted from Teradata. Construct this MD5 hash using columns that won't change when the data is loaded into Snowflake (e.g. Include key columns and attributes in the hash. But exclude insert and update dates/timestamps that can change based on when the data is loaded into Snowflake). As you load data into Snowflake, generate another MD5 hash across the same set of columns, so you can compare it with the MD5 hash from Teradata. This allows you to compare the contents of the row on the MD5 hash from Teradata with the MD5 hash from Snowflake, rather than comparing each column individually.



ABOUT SNOWFLAKE

Snowflake Cloud Data Platform shatters the barriers that prevent organizations from unleashing the true value from their data. Thousands of customers deploy Snowflake to advance their businesses beyond what was once possible by deriving all the insights from all their data by all their business users. Snowflake equips organizations with a single, integrated platform that offers the only data warehouse built for any cloud; instant, secure, and governed access to their entire network of data; and a core architecture to enable many other types of data workloads, including a single platform for developing modern data applications. Snowflake: Data without limits. Find out more at [snowflake.com](https://www.snowflake.com).

