WHITEPAPER

Enabling Data Agility: A Comprehensive Approach to Modernize your Data, AI, and Application Workloads with Snowflake

Author

Muthusivan Vellapandian

Contents

Abstract 3

Introduction 4

Transitioning SAP HANA workloads to Snowflake 5

- Evaluating the migration strategy 6
- Migration object inventory 8
- Migration accelerators 9
- Optimization 11
- Code change management 13
  1. Environment strategy 13
  2. Extracting changed models from SAP HANA for code reconciliation 13
  3. Code freeze period 14
  4. Synchronized code management 14

Transitioning the SAP BW workload to Snowflake 15

- Analyzing the SAP BW landscape 16
- Re-designing for optimal performance 16
- One big table vs star schema approach 19
- Materialization strategy 20

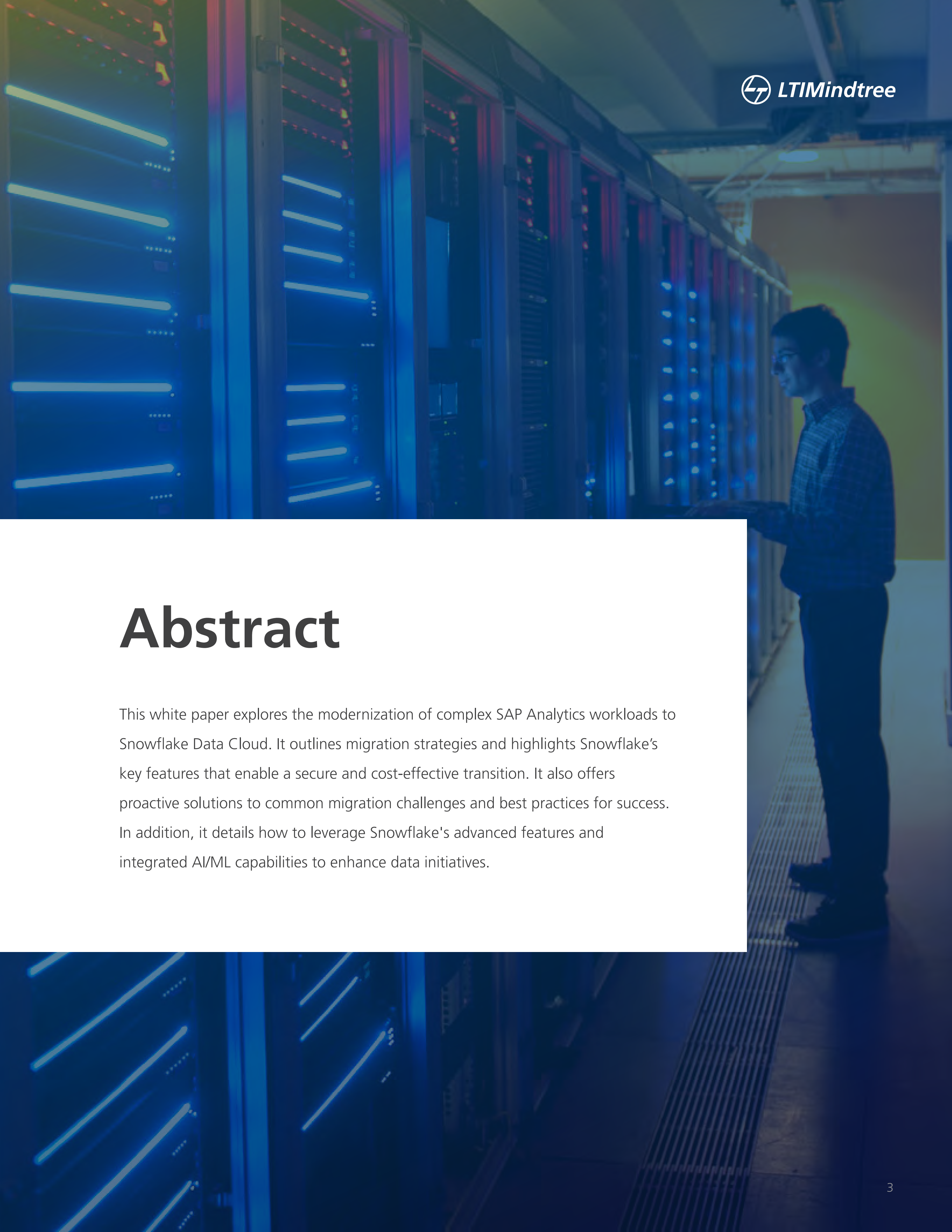
Handling currency conversion in Snowflake 21

Level up the migration with Snowflake features 23

- Dynamic tables for pipeline simplification 24
- Tags for governance 25
- Streamlit apps for data quality 25
- AI/ML for SAP use cases 26

Conclusion 27

About the Author 28



Abstract

This white paper explores the modernization of complex SAP Analytics workloads to Snowflake Data Cloud. It outlines migration strategies and highlights Snowflake's key features that enable a secure and cost-effective transition. It also offers proactive solutions to common migration challenges and best practices for success. In addition, it details how to leverage Snowflake's advanced features and integrated AI/ML capabilities to enhance data initiatives.

Introduction

As the digital world advances, businesses need diverse, adaptable, efficient, and performant workloads. Traditional on-premises solutions often fail to meet modern data demands due to limited scalability, high costs, and slow query speeds. This has driven interest in modernizing to Snowflake, capable of handling data, AI, and application workloads on a single platform.

Snowflake manages diverse data tasks from data warehousing and data lakes to real-time processing and AI/ML for data science. The Snowflake platform has evolved, adding features that simplify the transition. However, it usually requires migration accelerator utilities to accelerate, reduce risk, and build confidence. These accelerators analyze the analytics object base, convert the code, migrate and validate data, and provide testing evidence. While they accelerate the code and data migration to Snowflake, ensuring compatibility with new features for optimal use is crucial.

Migration from legacy databases to Snowflake has challenges. Despite migration accelerators minimizing risk, careful planning, understanding features and capabilities, and a well-defined strategy are essential to minimize disruption and ensure optimization. Key considerations include mapping equivalent features, leveraging advanced cloud capabilities, transforming data, redesigning schemas, optimizing queries, and ensuring data integrity and consistency. This white paper details transitioning two challenging workloads: SAP HANA and SAP BW to Snowflake.

Transitioning SAP HANA workloads to Snowflake

SAP HANA is well known for its in-memory computing power. It has been a cornerstone of success for enterprises seeking real-time analytics and transactional processing. While it enables lightning-fast computation, it often demands a significant investment upfront in terms of hardware and licensing. Despite all these capabilities in SAP HANA, the complex processing that arises during the month-end and year-end raises concurrency issues where multiple user queries and processes lead to conflicts and delays. It requires prioritizing the SQL queues, assigning dedicated computing resources for complex queries, etc. To overcome resource contention, large complex models are broken into multiple sub-processes to overcome memory limits. Such workarounds increase the number of objects and lead to greater environment complexity. In certain circumstances, the SAP HANA infrastructure must be upgraded to process complex and voluminous data models to meet business expectations.
Snowflake's agility and its capabilities provide a compelling alternative to SAP HANA's on-premises constraints. By leveraging the Snowflake cloud platform capabilities, customers can accelerate their analytics initiatives, driving innovation and maintaining a competitive edge in today's environment.

A typical SAP HANA data architecture includes the following layers.

RAW→ODS→INTEGRATION/REPORTING→PRESENTATION/CONSUMPTION

The data is physically stored in the RAW layer and the models built across other layers are typically just views in most scenarios. These models act as a virtual definition on how they get processed at the runtime for near real-time reporting. Here are some key considerations for migrating such SAP HANA semantics to Snowflake.

Evaluating the migration strategy

Selecting the right migration strategy is crucial to ensure a smooth transition, minimize disruptions and maximize benefits. Depending on the complexity, interdependency and business impact, the right migration approach must be chosen that reduces risk. There are two main strategies for any cloud migration: Migration in one-go and migration in waves. Depending on the nature of the SAP HANA analytics landscape, an appropriate strategy needs to be chosen. The common strategy is to do the migration by waves as it reduces risk and provides enough flexibility for the migration team to adjust to the dynamics. However, the typical challenges in wave migration for SAP HANA are:

01 Cross dependency between apps or data products

02 KPI borrowing across models results in model nesting and interdependence

03 Requiring a two-step approach for cross-dependent models

- Temporary model data replication
- Actual model code and data migration

If the models are highly interdependent across applications or data products, then migration in wave is not feasible. There is always temporary model replication effort such as loading the final output of the interdependent model to Snowflake temporarily to complete a certain wave. The actual model migration happens during the actual application migration it belongs to. The higher the magnitude of interdependencies, the greater the temporary efforts with a higher risk and potential migration failure. Determining the interdependency factor is key for a smooth and successful migration.

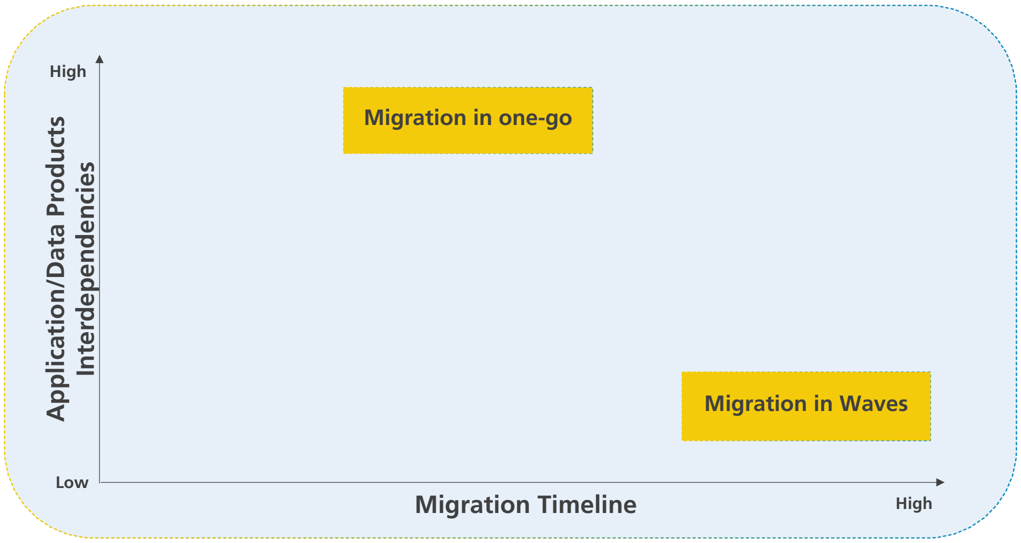


Figure 1: Migration timelines for interdependencies

Similarly, there are two migration approaches for SAP HANA: Lift-optimize-shift and re-engineering. A detailed due diligence and a proof of concept (MVP) leveraging both approaches and evaluating the key parameters with the POC results is mandatory and crucial for the way forward. The table below is a comparison of critical parameters for decision-making.

Parameters	Lift-Optimize-Shift	Re-engineering
Analysis & Design	Minimum client SME involvement overall	Additional ~20-30% of effort of client's SME's
Build	~30-40% automation possible	~15 % automation possible, involvement of Clients SME's
Validation	As models match AS-IS validation at aggregate level can be done to proceed with Dashboard validation	Model needs to be evaluated at each step, needs involvement of Client's SME's as well
Optimization	Identify the bottlenecks and tune for the forklifted models. It needs to be a dedicated track/POD	No dedicated optimization effort is required
Data loads	2 times (initial build and during UAT). No technical issues as structures will match with HANA AS-IS	Multiple times, it will also require additional effort as structures will vary in Snowflake
Speed	Automation friendly approach with ~30-40% automation possible	Manual efforts Based on our past migration POCs, effort in Reengineering is 1.5 times or greater
Inter-Dependencies	Model deployment follows inter-dependencies	Re-engineering may lead to drop and build new models to overcome inter-dependencies

Table 1: Comparison of critical parameters for lift-optimize-shift vs re-engineering

While lift-optimize-shift accelerates the migration, not all the SAP HANA objects can be migrated in this fashion. There will always be re-engineering required, especially relying on native features in models such as star join, attribute views, etc. Each approach has its pros and cons. However, the key difference is the quantum of effort needed to optimize the models–this is a critical success factor. The diagram below illustrates the optimization effort vs timeline in SAP HANA to Snowflake migration.

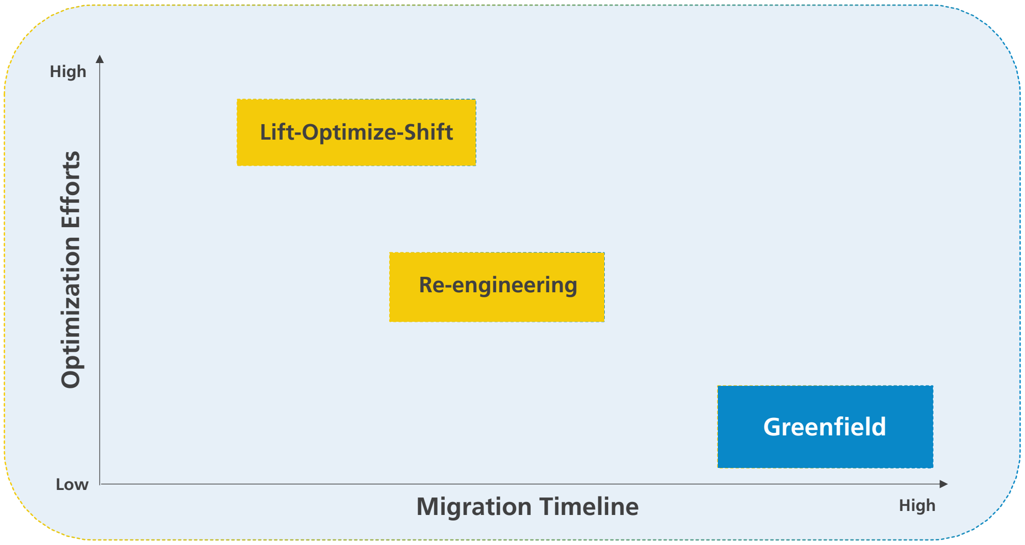


Figure 2: Optimization effort vs timeline in SAP HANA to Snowflake migration

Migration object inventory

The object inventory is essential for a smooth lift-optimize-shift migration to Snowflake, driven by collecting system details from SAP HANA. The objective includes:

01 Identifying the number of objects involved (scope)

02 Object count by data products/subject area

03 Current migration status

04 Object inventory rationalization (decommissioning)

Key information in the object inventory should include:

01 SAP HANA object name and path

02 Equivalent Snowflake database/schema, strategically mapped by architects

03 Application/data products mapping

04 Object type: Defines the object type (e.g., calc views etc.)

05 Complexity

06 Cross dependency

07 Scope/out of scope

Migration accelerators

Migration accelerators play a significant part in the lift-optimize-shift migration approach. The purpose of such accelerators is to speed up migration activities, reduce effort and timelines, and consistently convert the SAP HANA objects to equivalent Snowflake objects. Snowflake provides a conversion utility to speed up the migration, providing a good automation rate with minimal issues.

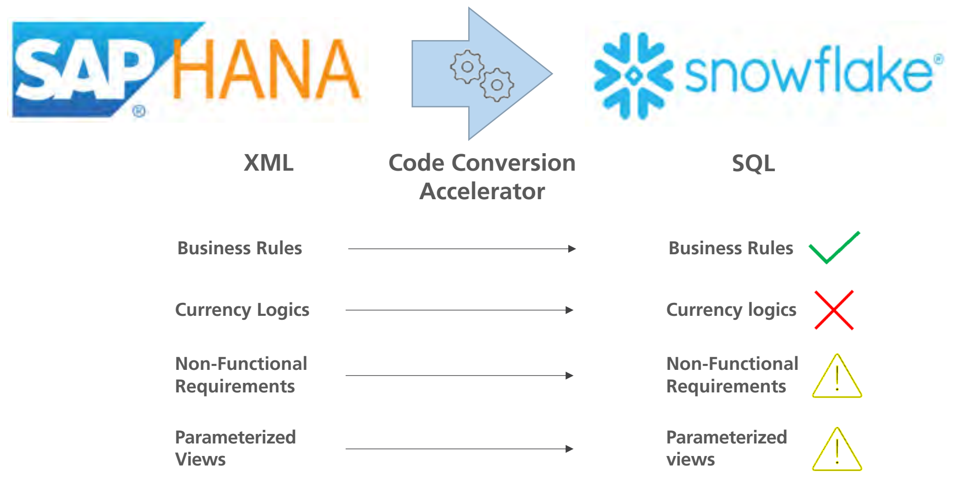


Figure 3: Role of migration accelerators

While the accelerators convert the code to equivalent Snowflake objects, they will provide a workable version on Snowflake. Significant remediation efforts are required to unit test the converted code and match the data across platforms. The code functionality gets carried forward to Snowflake such as business logic and rules, but greater effort is required to optimize the code in Snowflake to meet non-functional requirements. As illustrated in the diagram above, not all the functionality gets converted by the accelerators. The semantics set in the graphical UI and any properties such as cardinality, SAP HANA native aggregation on a measure such as SUM/MAX, etc. do not get converted to their SQL equivalent but require significant remediation. The diagram below illustrates some of the manual intervention and remediation required while using accelerators.

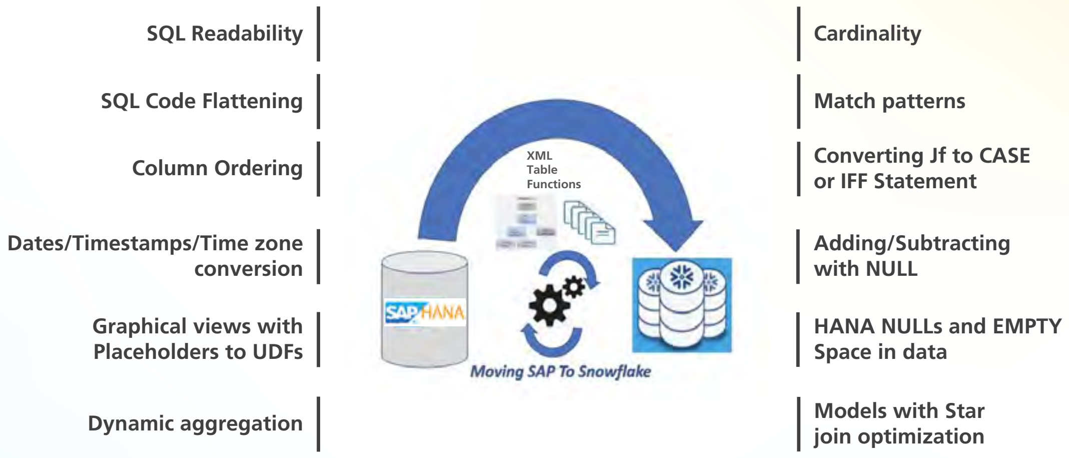

SQL Readability
SQL Code Flattening
Column Ordering
Dates/Timestamps/Time zone conversion
Graphical views with Placeholders to UDFs
Dynamic aggregation
XML Table Functions
SAP HANA
Moving SAP To Snowflake
Cardinality
Match patterns
Converting Jf to CASE or IFF Statement
Adding/Subtracting with NULL
HANA NULLs and EMPTY Space in data
Models with Star join optimization

Figure 4: Manual intervention and remediation required while using the accelerators

Optimization

Optimization is crucial for migrating from SAP HANA to Snowflake due to their architectural differences. Without it, Snowflake workloads may suffer from poor performance and higher costs due to inefficient resource use. The lift-optimize-shift approach needs dedicated optimization efforts, as forklifted models only transfer code functionality, not non-functional aspects. Two key optimization types are:

Code-length optimization

SAP HANA models use graphical nodes for data processing, which become lengthy and complex SQL queries after conversion. These subqueries can be hard to manage and may lead to compilation issues. Eliminating subqueries and using common table expressions (CTEs) are essential for code optimization. While some code-length optimization can be automated, manual validation and fixes are often needed.

Performance optimization

The conversion process does not guarantee completely optimized code for Snowflake's environment. In simpler terms, the code base can be forklifted from SAP HANA with minimal fine-tuning to run on Snowflake's architecture. As many scenarios arise there could be bottlenecks in migration, and they can cause sluggish performance during process execution in Snowflake. Resolving this involves tuning the models in a way that leverages Snowflake's strengths, resulting in a significant performance boost. Sample patterns identified in the migration scenarios are as follows:

Pattern 1 Aggregation against the primary key

When SQL includes the primary key in the SELECT statement and uses aggregate functions like AVG, SUM, or MAX, it results in the same number of rows with or without a GROUP BY, making aggregation irrelevant and inefficient. This is particularly problematic for large tables like BSEG, CDPOS, and MSEG, as Snowflake scans all rows unnecessarily. Removing aggregate functions and GROUP BY statements in such cases improves query performance.

Pattern 2 Repeating HANA graphical view nodes

Forklifting HANA Graphical views (XMLs) to SQL results in repeating identical subqueries, causing Snowflake to generate costly query plans. Each subquery is compiled individually, leading to high compilation and execution times. To address this, use reusability components like CTEs or temporary tables. For stored procedures, create temporary tables and reuse them in subqueries.

Pattern 3 Very high compilation time in the migration scenarios

When SQL queries have a compilation time significantly higher execution time (2x or more), performance issues arise as the execution is much faster but the compilation remains costly. This problem often occurs with lengthy SQLs from forklifted graphical views, which involve many nested views and complex dependencies. To improve performance, rewrite and flatten the SQL by referring to tables directly, reducing or eliminating complex dependent views, and ensuring top-level views only compute necessary objects and columns.

Pattern 4 Sub-optimal master data look up in the migration scenarios

Sub-optimal master data lookups occur when querying complex and highly nested models, causing Snowflake to compile and compute all logic, unlike SAP HANA. This results in inefficiency as it involves unnecessary views and tables. To improve performance, rewrite the SQL to directly fetch master data from base models or raw tables instead of using complex calculation views.

Code change management

Especially in a lift-optimize-shift migration approach, popular for transitioning SAP HANA analytics to Snowflake, a well-planned strategy to manage the ongoing code changes in the models is crucial. Depending on the landscape, the strategy varies but cannot be ignored in the planning. Key aspects to consider for code changes are as follows:

Environment strategy

Code, raw data, transformations and BI reports and dashboards should be from the same snapshot, and a dedicated and static environment (SAP HANA and Snowflake) should be assigned for development and unit testing. The difference in the versions and misalignment in any of these layers cause cascading issues and result in a longer time to de-bug and perform root cause analysis (RCA). These issues turn into potential risks.

Extracting changed models from SAP HANA for code reconciliation

It is crucial to extract and identify code changes from SAP HANA for a given time frame and make this information publicly available via Power BI/Tableau for stakeholders. Changes can vary, including enhancements or new data products. To ensure a smooth transition to Snowflake, follow these steps after extraction:

- Use a comparator utility to identify changes in graphical nodes by comparing XML or SQL versions.
- Collaborate with SAP HANA developers to detect changes in calculations or joins.
- Monitor row counts before and after extraction.
- Compare KPI aggregation values and graphical node counts before and after code reconciliation.

Code freeze period

Establishing a code freeze strategy is vital for smooth migration especially for the lift-optimize-shift migration approach. The following are the key migration scenarios for which freeze is crucial:

- During user acceptance testing and cutover period, the code freeze including the critical fix is important.
- Data product migration with too many interdependencies.
- Finance data products during quarter/year-end reconciliation process.

Synchronized code management

Code freeze is not ideal during the migration due to business impact and demands. Critical bug fixes or external dependencies (e.g., with invoice system integrations) are a few scenarios that demand synchronized development between SAP HANA and Snowflake. Such a dual maintenance strategy not only helps maintain the code in sync between both the systems but also helps the developers to adopt the new platform and align the data product teams to the new development life cycle. It is not recommended to adopt this strategy during the beginning stage of the migration, rather adopt it during system integration testing (SIT) phase or during a dedicated code reconciliation phase as per the master plan.

Transitioning the SAP BW workload to Snowflake

While SAP HANA migration focuses on moving memory operations and real-time analytics to Snowflake, SAP BW migration to Snowflake involves transitioning a classical data warehouse environment with precomputed data models and processes. For building data models, SAP BW relies heavily on its native features such as info cubes/Data Store Objects (DSO) for transformations, it involves re-thinking the data architecture layers that align to standard Snowflake data architectural layers. Unlike SAP HANA, the lift-optimize-shift migration approach for SAP BW is likely to bring more technical debts to Snowflake due to the number of layers involved in transforming the data from a raw state to being consumption-ready. Transitioning BW architecture to Snowflake may require re-design for Snowflake on how the data is transformed for better performance and cost. The picture below is an illustration of mapping the SAP BW layers to Snowflake data layers.

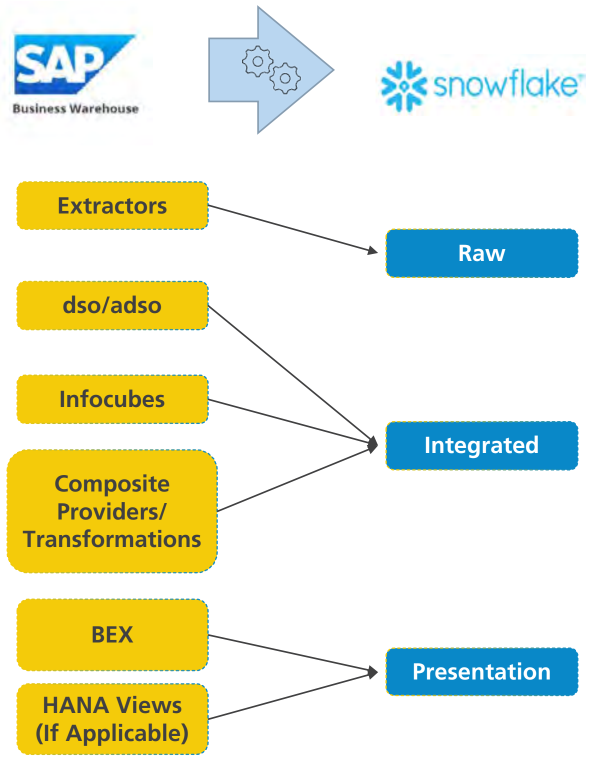


Figure 5: Mapping SAP BW layers to Snowflake data layers

Analyzing the SAP BW landscape

Performing due diligence in the SAP BW landscape is the key phase in the modernization journey. It requires cataloging all the objects in an inventory sheet listing the object types, associated application teams, owner, etc. The outcome of the due diligence should include, but is not limited to, the following:

- Object inventory sheet.
- Functional specification for each data product/application.
  1. Revisiting the functional logics that need fixing.
  2. Refine the calculations for process improvements as applicable.
  3. Add new KPIs/calculation logic to meet the business expectations.
- Technical specifications for each data product/application.
  1. Extractor decoding (standard/custom extractors). Standard extractor decoding is more complex than custom. Hence, it requires a dedicated analysis effort.
  2. Source to target mapping.
  3. Naming conventions.
  4. Incremental data loading of models and design.
  5. Identifying the primary key for the models to enable incremental data load.
  6. Define clustering keys.
  7. View vs table option for the logical steps involved in the model building.
  8. Query tagging configurations.
  9. Non-functional requirements and alignment.

Re-designing for optimal performance

SAP BW multi-tier architecture does not translate directly to Snowflake data architecture standards. Significant design considerations are required to bridge the gap between the two platforms to unlock the Snowflake's potential. Here are some of the re-design aspects.

01

Naming convention

It is important to rename the objects with business relevance and descriptions in Snowflake, aligning to the target platform standards instead of following the BW mindset. In particular, the L1, L2, etc. references to identify the BW layers should be eliminated. The naming convention should be consistent across the project teams. It should be enforced programmatically by DevOps pipelines.

02

Model layer optimization

It is quite common in SAP BW to break a large process into dedicated subprocesses for each region or source ID to overcome performance or concurrency issues. Forklifting such subprocesses to Snowflake produces suboptimal and expensive processing. The key design decision should be to unify such subprocesses into a single process, with rows separated by a new indicator in Snowflake. This design increases the row count, and it requires natural clustering e.g., sorting the data chronologically based on key date columns used in consumption. Here are some sample design considerations.

1. Combining multiple region-specific extractors into a simple stored procedure.
2. Combining data store objects (DSO), transformations, etc. into a simple process.

03

Data modelling considerations

With the capabilities of tools like data build tool (DBT), the data modelling should be efficient and reliable by leveraging the declarative, modular SQL queries and version control them. In SAP BW or HANA to Snowflake migration, it is common design thinking to optimize the multiple layers into a single stored procedure by loading the intermediate result set into a transient table. However, the modularity, reusability, lineage graphs, and data quality frameworks are key aspects that should be considered in the data modelling instead of 10,000 lines of SQL code wrapped in a Snowflake stored procedure sequence.

The table below compares the modular vs monolithic ways of data modelling in the context of Snowflake.

Monolithic data modelling	Modular data modelling
Data Processing Running the entire process in one stored procedure executes tasks sequentially, not utilizing Snowflake's multi-threading and massively parallel processing (MPP) capabilities, leading to longer completion times and keeping the cluster active for extended periods.	**Data Processing** Logically separate contexts into intermediate CTEs to enable parallel execution across cluster nodes, optimizing performance.
Cost Sequential execution keeps the compute cluster active longer, increasing costs. Re-processing due to failures or issues requires running the entire workflow again, resulting in higher expenses and potential cloud wastage.	**Cost** Relatively cost-effective. During re-processing or retrofitting, not all the workflows need to be executed again (can be customized by selecting the required jobs in DBT).
Data Quality Problems in one context cascade require reprocessing all workflows, creating a single point of failure and complicating debugging.	**Data Quality** The DQ issues can be narrowed down easily and the required workflow along with the upstream objects can be reprocessed.
Cross Dependency Management Post-migration, managing interdependencies requires creating views with the necessary columns if granularity remains consistent.	**Cross Dependency Management** Re-usability by calling dbt macros serves the purpose along with lineage.
DataOps and maintainability Lengthy code makes debugging and addressing data issues complex and time-consuming. Retrofitting data necessitates re-running the entire flow.	**DataOps and maintainability** Comprehensive coding as the SQL gets broken into intermediate Common Table Expressions (CTE). Retrofitting or reprocessing can be customized by selecting required jobs in Data Build Tool (DBT).

One big table vs star schema approach

When it comes to the consumption layer and end-user experience, meeting the non-functional requirement and ease of self-service capabilities are key success measures. In the SAP BW transition journey, it is quite common to have:

- Denormalized data set for consumption.
- Dimensions and fact tables in a star schema that can be joined during consumption in Snowflake or inside PowerBI modelling.

01

One big table

1. Queries are straightforward without complex joins, making them easier to write and understand. However, de-normalized data can lead to redundancy, larger storage needs, and potential consistency issues.
2. Benefits from Snowflake's columnar storage and compression but requires careful clustering for efficient query pruning. Maintaining a single table can be complex due to data duplication, with variable load times depending on the KPI context and transaction tables involved.

02

Star schema

1. Queries involve joins between fact and dimension tables, which can be more complex but efficient for large datasets. The normalized structure reduces redundancy and improves data integrity, making performance generally better for large datasets.
2. More complex to maintain with multiple tables but ensures better consistency and easier updates. Load times can be faster with a normalized design, though careful ETL processes are needed to populate multiple tables. Suitable for large, complex datasets where data integrity and performance are crucial.

Materialization strategy

While materialization is one of the key aspects in snowflake, it is not meant to solve performance problems. However, it is a key design component to get the best out of Snowflake. Here are the parameters and considerations for materialization.

01

Data latency

Align materialization frequency and schedule with raw table updates to minimize latency. Ensure consistency by aligning KPIs' latency and maintaining data snapshots for reliable reporting and analytics.

02

Leverage cache

Utilize Snowflake's cache for identical downstream queries to improve performance.

03

Load pattern

Full load: Suitable for smaller datasets; manage blackouts with explicit transaction handling (`BEGIN/COMMIT`).
Incremental load: Ideal for large datasets with identified primary keys; maintain cardinality and manage duplicates with an append-only, delete/insert, merge pattern.

04

Materialization-query performance

Convert frequent queries to CTEs for efficiency.
Use predicate pushdown to move filters to inner queries.
Eliminate dead code and optimize joins.

LTIMindtree

05

Nested models

Flatten complex nested views and use CTEs to improve performance and reduce compilation time.

06

Cost-efficiency and user experience

Persisting data in tables enhances performance and is cost-effective. Precompute outputs on a schedule to manage costs effectively, especially if underlying data changes infrequently.

Handling currency conversion in Snowflake

While forklifting models from SAP HANA to Snowflake, the currency conversion logics do not get migrated. They require a new design in Snowflake that can be invoked in the join as when required. Hence, it is important to understand the conversion approaches in SAP HANA:

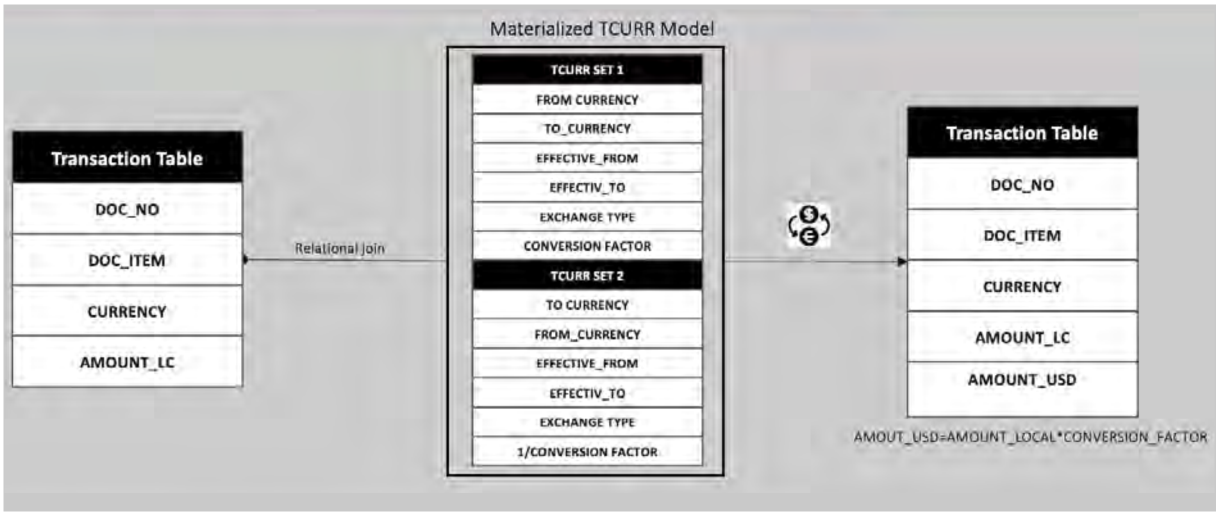
- Global currency conversion models
- Region specific currency conversion models
- A global/domain specific currency conversion semantics

In Snowflake, it is always recommended to build a new model leveraging TCURR and its associated tables. As conversion can happen in multiple columns in the same models, the result should be precomputed for better execution time. The important design aspect is to add "START DATE" and "END DATE" to each row in the conversion table to cover all the missing dates scenarios in TCURR. The conversion model can be consumed by other models leveraging these dates for accurate conversion rates. Depending on the landscape, the pre-computation of the conversion model in Snowflake shall potentially cover scenarios such as:

- USD to non-USD
- Non-USD to USD
- Non-USD to non-USD
- Decimal shift
- Exception dates
- Future date conversion

Currency conversion model in Snowflake

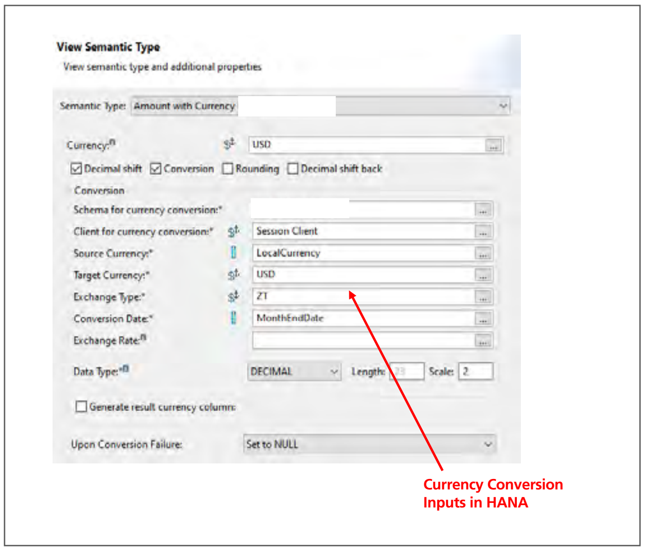
The diagram below illustrates the currency conversion model in Snowflake for set operations. The same can be a user defined function (UDF) with necessary input parameters that can be leveraged for row-by-row operations in API calls or OLTP processes, etc.



Process currency conversion in batches	Conversion for future dates/missing dates such as weekends	Special conversion scenarios such as decimal shift, USD ←→ NON-USD etc.	High performant during SET based consumption

Figure 6: Currency conversion model in Snowflake for SET operations

HANA



View Semantic Type

View semantic type and additional properties

Semantic Type:	Amount with Currency
Currency:	USD

[✓ Decimal shift] [✓ Conversion] Rounding Decimal shift back

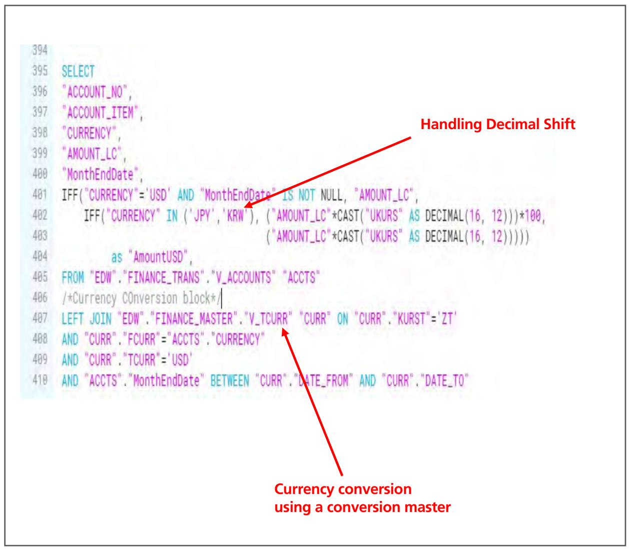
Conversion

Schema for currency conversion:*	
Client for currency conversion:*	Session Client
Source Currency:*	LocalCurrency
Target Currency:*	USD
Exchange Type:*	ZT
Conversion Date:*	MonthEndDate
Exchange Rate:	
Data Type:*	DECIMAL Length: 23 Scale: 2

Generate result currency column:

Upon Conversion Failure:	Set to NULL

SNOWFLAKE



```
SELECT
"ACCOUNT_NO",
"ACCOUNT_ITEM",
"CURRENCY",
"AMOUNT_LC",
"MonthEndDate",
IFF("CURRENCY"='USD' AND "MonthEndDate" IS NOT NULL, "AMOUNT_LC",
    IFF("CURRENCY" IN ('JPY','KRW'), ("AMOUNT_LC"*CAST("UKURS" AS DECIMAL(16, 12)))*100,
                                     ("AMOUNT_LC"*CAST("UKURS" AS DECIMAL(16, 12)))))
          as "AmountUSD",
FROM "EDW"."FINANCE_TRANS"."V_ACCOUNTS" "ACCTS"
/*Currency COnversion block*/
LEFT JOIN "EDW"."FINANCE_MASTER"."V_TCURR" "CURR" ON "CURR"."KURST"='ZT'
AND "CURR"."FCURR"="ACCTS"."CURRENCY"
AND "CURR"."TCURR"='USD'
AND "ACCTS"."MonthEndDate" BETWEEN "CURR"."DATE_FROM" AND "CURR"."DATE_TO"
```

Figure 7: Sample before and after currency conversion scenario in migration

Level up the migration with Snowflake features

A successful migration to Snowflake involves more than just replacing existing functionality; it should also leverage Snowflake's advanced features to future-proof your data and analytics strategy. Utilizing Snowflake's native features is crucial for optimal price-performance. Ignoring these features or deviating from Snowflake standards can lead to scalability issues, poor performance, and higher costs.

Proper planning to incorporate Snowflake's features can enhance the migration process and adhere to timelines. For SAP HANA or BW analytics migrations, several specific Snowflake features can be utilized effectively.

Dynamic tables for pipeline simplification

Dynamic tables are a key feature that simplify and automate data pipelines. They streamline the process of transforming and managing the data transformation logic with automated refresh. This capability makes them ideal for SAP HANA to Snowflake migration scenarios. To improve performance, it is important persist the model data in a table–this is a key design aspect in such a SAP migration. The dynamic table feature is a tailor-made feature to address requirements such as:

- Materializing complex database view data in a table
- Incremental refresh
- Automated refresh based on lag parameter
- Eliminating the need for setting up tasks/stream's orchestrations
- Reducing pipeline build times

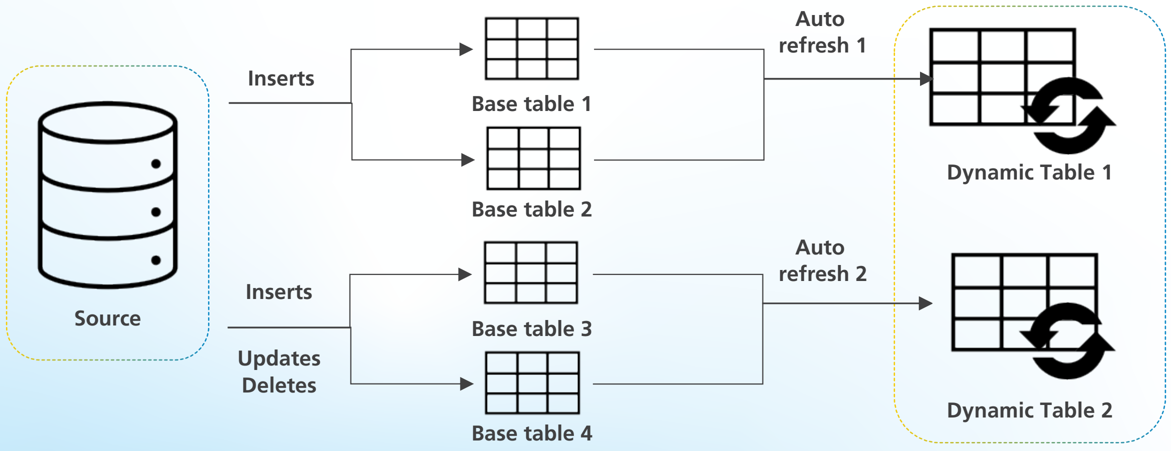


Figure 8: Simplifying and automating data pipelines with dynamic tables

Tags for governance

Object and query tagging are essential for data governance and cost management but are often overlooked. Tags help classify objects and logs during use case onboarding and are crucial for:

- Cost slicing and dicing
- Identifying bottlenecks
- Traceability in Snowflake query history

Incorporating tags from the design phase is critical; adding them later is complex and affects existing query logs, resulting in lost cost control. Therefore, tagging should be a mandatory part of the design checklist.

Streamlit apps for data quality

The streamlit apps can be used to manage user language tables, data quality rules, and lookup flat files via Snowflake native app. Traditionally, these mapping and lookup tables are updated manually by SQL scripts using insert and update statements. Such manual processes can be automated and improved via a Streamlit app framework with ease alongside migration, especially in a re-engineering approach.

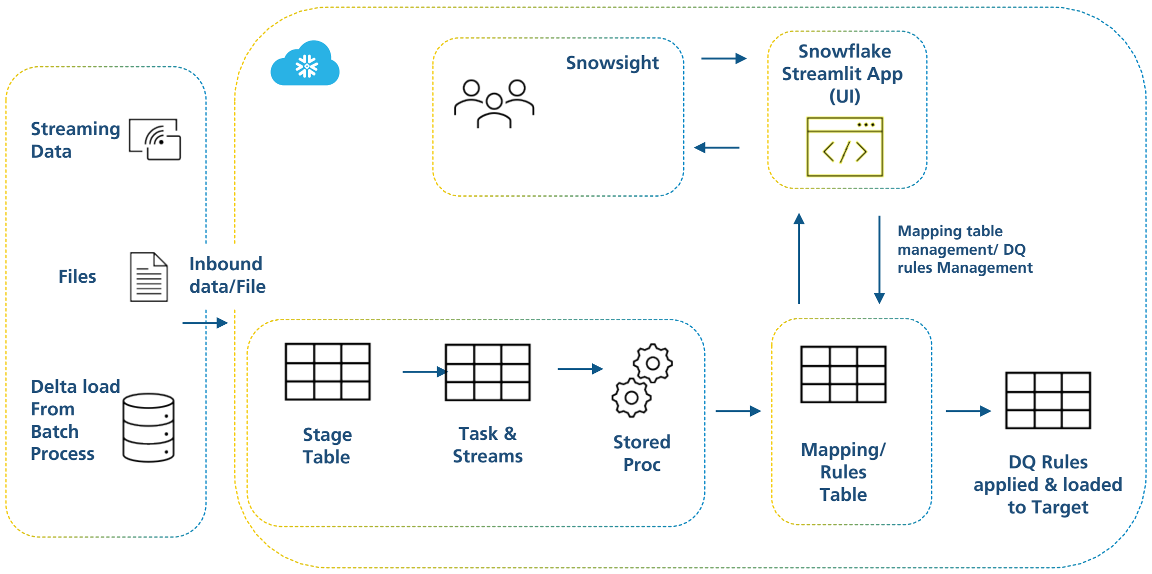

Streaming Data
Files
Delta load From Batch Process
Inbound data/File
Snowsight
Snowflake Streamlit App (UI)
Mapping table management/ DQ rules Management
Stage Table
Task & Streams
Stored Proc
Mapping/ Rules Table
DQ Rules applied & loaded to Target

Figure 9: Automating processes to improve data quality using the Streamlit app during migration

AI/ML for SAP use cases

Snowflake offers its users many AI/ML capabilities such as Cortex–a fully Managed AI/ML solution. Here are some of the sample SAP use cases that can be solved using Cortex's capabilities.

Use case 1	Enterprise fraud management to evaluate the vendors, vendor duplicity, and financial transactions to identify patterns, predict potential frauds, anomaly detection, etc.
Use case 2	Optimizing and predicting the ROP (re-order points) and MAX stock levels to manage the inventory (avoid overstocking /understocking) in the warehouse.
Use case 3	Evaluate the non-stock transactions in EKPO that can be a potential stock material in SAP.
Use case 4	Handling multiple language text and convert them to English to perform work cloud analysis on STXL table texts, purchase order texts, etc.
Use case 5	Document AI in Snowflake helps evaluate the service order documents (SOW) and highlight missing legal statement, word cloud, etc.

Conclusion

While SAP HANA and BW are robust enterprise data platforms, their architecture and on-premises nature result in high costs, limited scalability, and complex infrastructure. Migrating these workloads to Snowflake offers benefits but requires careful planning.

Choosing the right migration strategy is key. The lift-optimize-shift approach moves existing objects directly to Snowflake for faster implementation, minimizing disruption but possibly duplicating inefficiencies, affecting performance and cost. Re-engineering, on the other hand, involves rebuilding data models to fully leverage Snowflake's capabilities, resulting in a future-proof platform with better performance, but with longer duration.

Migrating from SAP HANA and BW to Snowflake transforms data infrastructure into a more agile, cost-effective, and scalable environment. This transition reduces operational overhead and empowers organizations with advanced capabilities and features to drive innovation and stay competitive in a data-driven landscape.

References

i Simplifying Your Analytics Landscape by Migrating from SAP to Snowflake , Technical Guide, Snowflake, 2021

https://www.snowflake.com/resource/migrating-from-sap-to-snowflake/

ii Definitive Guide to Managing Spend in Snowflake , Technical Guide, Snowflake, 2023

https://www.snowflake.com/wp-content/uploads/2023/10/Definitive-Guide-to-Managing-Spend-in-Snowflake.pdf

iii Snowflake Dynamic Table Complete Guide , Alexander, Snowflake Medium

https://medium.com/snowflake/snowflake-dynamic-table-complete-guide-1-7b27925e099d

iv Use AI in Seconds with Snowflake Cortex, Pieter Verhoeven, Snowflake, 2023

https://www.snowflake.com/blog/use-ai-snowflake-cortex/

About the Author

Muthusivan Vellapandian

Solution architect–Snowflake tech-consulting, LTIMindtree

Muthusivan has 17 years of experience in data practice in solutioning and technology consulting for customers across the globe. He is a specialist consultant in devising cost-performance optimization strategies and performing gap analysis in Snowflake. He is passionate about Snowflake, solving complex migration problems in large transformational programs, and solving business problems through data, AI and apps.