## **Assessment Report <Prepared For> / <Date>**

**Mobilize.Net SnowConvert for Teradata**

The purpose of this document is to summarize the technical considerations and code analysis in migrating SQL to Snowflake from Teradata that either have an impact on the automated code conversion or cannot be handled by automated code conversion, as well as providing a high level inventory and automation capability of the code that will need to be addressed.

*NOTE: The assessment inventory of scripts such as BTEQ, FastLoad, MultiLoad, TPT, and TPump is currently in development, but is not included in this release of Mobilize.Net SnowConvert.*

**-----------------------------------------------------------------------------------------------------------------------------**

**OVERALL CONVERSION SUMMARY**

Total Files: 32

- SQL Files: 31

- Script Files: 1

Total Files Not Generated: 0

Conversion Speed: 43 lines/sec

Conversion Time: 00:00:11

Total Conversion Errors: 1

Total Parsing Errors: 0

Total Warnings: 39

Total Lines of Code (LOC): 508

**SQL LINE CONVERSION SUMMARY**

Lines of Code: 477

LOC Conversion Percentage: 99%

Unrecognized Lines of Code: 0

**SCRIPTS LINE CONVERSION SUMMARY**

Lines of Code: 31

LOC Conversion Percentage: N/A

Unrecognized Lines of Code: N/A

**OBJECT CONVERSION SUMMARY**

Identified Objects: 31

Identified Object Conversion Rate: 100%

Unrecognized Objects: 0

**-------------------------------------------------------------------------------------------------------------------------------**

**Table of Contents**

File and Object Level Breakdown …………………………………………………………… 2

Environment Settings …………………………………………………………………………. 3

Databases and Schema ……….……………………………………………………………… 3

Tables ..…………………………………………………………………………………………. 4

Views, Join Indexes, Macros, and Procedures …………………………………………….. 7

Numeric Data Operations Considerations …………………………………………………… 8

Notes on Output Files Not Generating ………………………………………………………. 10

**FILE AND OBJECT LEVEL BREAKDOWN**

**SQL – Files**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File** | **Conversion Rate** | | **Total Object** | **Total LOC** | **Parsing** | **Conversion** |
|  | **Object** | **LOC** | **Quantity** |  | **Errors** | **Errors** |
| SQL | 100% | 99% | 31 | 477 | 0 | 0 |

**SQL – Identified Objects**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Object** | **Conversion Rate** | | **Total Object** | **Total LOC** | **Parsing** | **Conversion** |
|  | **Object** | **LOC** | **Quantity** |  | **Errors** | **Errors** |
| Tables | 100% | 100% | 14 | 168 | 0 | 0 |
| Views | 100% | 100% | 9 | 114 | 0 | 0 |
| Join Indexes | 0% | 0% | 0 | 0 | 0 | 0 |
| Macros | 100% | 100% | 2 | 16 | 0 | 0 |
| Procedures | 100% | 100% | 6 | 176 | 0 | 0 |

**SCRIPTS – Files**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **File** | **Conversion Rate** | | **Total Object** | **Total LOC** | **Parsing** | **Conversion** |
|  | **Object** | **LOC** | **Quantity** |  | **Errors** | **Errors** |
| BTEQ | - | - | - | 31 | 0 | 1 |
| FastLoad | - | - | - | 0 | 0 | 0 |
| MultiLoad | - | - | - | 0 | 0 | 0 |
| TPT | - | - | - | 0 | 0 | 0 |
| TPump | - | - | - | 0 | 0 | 0 |

**SCRIPTS – Identified Objects**: *Coming soon!*

**ENVIRONMENT SETTINGS**

These are key settings that need to be considered in the Teradata migration.

ANSI Mode versus Teradata Mode - Teradata mode which is typically used ignores only trailing spaces and ANSI mode ignores both leading and trailing spaces by default during comparison. This setting is important to understand when converting code.

<https://docs.teradata.com/reader/Daz9Bt8GiwSdtthYFn~vdw/6oXMWdRsjArKuCs36O7vmA>

*Parameters* -

QUOTED\_IDENTIFIERS\_IGNORE\_CASE (Snowflake) - Should be set to TRUE for the database for a Teradata migration. This is necessary as there are many identifiers that have to use quotes in Snowflake that are not necessary in Teradata (any name with a # symbol for example).

TIMESTAMP\_TYPE\_MAPPING (Snowflake) - Depending on customer requirements this alias mapping will need to be set to TIMESTAMP\_LTZ for the default behavior of the TIMESTAMP function to match the default behavior in Teradata.

ROUNDHALFWAYMAGUP (Teradata) - This Teradata parameter controls banker rounding and should be looked at to determine if banker rounding is being used as Snowflake does not currently support this behavior.

**DATABASES & SCHEMAS**

Teradata does not have the concept of schemas and a decision must be made when migrating Teradata environments and databases. The simplest approach is for a single Teradata environment to equate to a single Snowflake Database. The individual Teradata databases are then deployed as individual schemas within the Snowflake Database. However, there may be cases where workloads will need to split into separate Snowflake Databases for purposes of managing the resources (sizing) dedicated to specific database.

*DATABASES*

Number of Databases Containing Objects: 3

3 databases had objects defined (tables, views, macros, join indexes, procedures). The size of this list is important for conversion strategy if converting Teradata databases to Snowflake databases as opposed to Snowflake schemas within a single database.

No databases were found with the name PUBLIC of INFORMATION\_SCHEMA which are default Snowflake schemas.

**TABLES: 100% Identified Object Conversion Rate**

Conversion Rate by Lines of Code: 100%

Number of Tables: 14

Lines of Code: 168

Total Parsing Errors: 0

Conversion Errors: 0

Successfully converted objects have been converted to an equivalent table and data types capable of storing the same data values as Teradata but some functionality below needs to be further analyzed and considered based upon specific use cases.

*Constraints / Data Values* - Snowflake does not enforce or allow constraints. If your data input/ingestion process relies on these capabilities to validate data prior to being inserted into a table, these processes will need to be rearchitected to prevent duplicate or bad data.

* CHECK (1) - Not supported in Snowflake, checks the value of data being inserted to be a specific list of values or meet certain conditions
* UNIQUE PRIMARY INDEX / PRIMARY KEY (0/3)- Not enforced in Snowflake. Definition is allowed for documentation purposes only.
* FOREIGN KEY (0) - Not enforced in Snowflake. Definition is allowed for documentation purposes only.

*Object Naming*

Invalid column names (0) - These column names are not supported in Snowflake and are converted to XXX\_<column name>\_XXX. This could require changes to external reporting/ETL systems. Examples include ‘current\_date’, ‘localtimestamp’.

* + Identifiers Requiring Double Quotes (0) - These objects require double quotes for defining in Snowflake and as a result are case sensitive unless the parameter QUOTED\_IDENTIFIERS\_IGNORE\_CASE has been set to True.

*SET Tables (6)* - Set tables in Teradata automatically prevent duplicate rows by checking for duplicates upon insert. If a primary key is defined it will base the duplicate check on the primary key, otherwise it compares the entire row - which is very performance intensive. Snowflake does not have this functionality and will require rearchitecting the data ingestion process to check for duplicates prior to insertion if the SET functionality of these tables is being leveraged for de-duplication.

*Temporal Tables (0)* - Temporal tables operate as pre-built slowly changing dimensions, capturing history as records are inserted/updated/deleted. Snowflake does not have an equivalent table type. Some of this functionality can be mimicked utilizing Streams, Tasks and Time Travel but use cases must be considered due to process timing and change capture requirements.

*Global Temporary Tables (0)* - Global temporary tables allow for a table to be permanently defined but the contents of the table are specific and temporary to users’ sessions. Snowflake does not currently have a comparable feature and use of these must be re-architected based upon the use case.

*Queue Tables (0)* - Persistent table type used to handle queue‑oriented data, such as event processing and asynchronous data loading applications. Snowflake does not have a like functionality and a process will need to be defined to emulate the functionality.

*Data Types   
(This assessment can be used as a complement to a full data profile. A complete inventory for this assessment is coming soon.)*

* NUMBER without precision (3) - Number when defined without a scale/precision in Teradata allows for a flexible scale value for any given record from 0 to 38 provided the total precision never exceeds 38. Snowflake does not allow this and always has a fixed scale/precision. Data Profiling results showed….
* INTERVAL (16) - Interval data types are not supported in Snowflake and are converted to varchars and manipulated as such using custom functions. Heavy use of intervals can cause performance issues and may require some re-architecture of the DDL and SQL.
* PERIOD (2) - Period data types are not supported in Snowflake and are converted to varchars. They are a pair of dates or timestamps. Heavy use of periods can cause performance issues and may require some re-architecture of the DDL and SQL.
* BLOB (1) - BINARY can be used instead but has a maximum of 8,388,608 bytes. Data Profiling results showed….
* CLOB (0) - VARCHAR can be used instead but has a maximum of 16,777,216 bytes (for single-byte). Data Profiling results showed….
* GeoSpatial (0) - Geospatial types have recently had support released.  Some specific use cases may require evaluation for workarounds.
* XML/JSON (0) - Special column types that are converted to VARIANT columns in Snowflake and may need additional attention in testing/data loading.

*Data Values*

* Binary Defaults (0) - Binary data types do not currently support a default value being defined.
* TIME Defaults (0)
  + - Time with TimeZone is not a supported type in Snowflake
  + FLOAT Default (0)
    - Timestamp can be defined as a default value for a FLOAT column in Teradata but is not supported in Snowflake. This is typically a column required for an auxiliary system such as SAS and many times will not be an issue once the system has been adjusted to work with Snowflake.
  + IDENTITY Columns (0) - These are converted to sequences and can represent an area that requires additional testing. 0 unique definitions were found. A large number of these can represent an area of increased testing, particularly if sequences are required to be kept in progression at cutover.
  + SESSION Defaults (0) - These columns are defined as integers in Teradata and must be varchars in Snowflake to hold the Snowflake current\_session value.

*Formats (0)* - Default format definitions by column are not supported in Snowflake. This can have implications for reporting or data extracts that are created for consumption. 0 unique format definitions were found.

*Partition By (1)* - Partitions are not supported in Snowflake but have a similar functionality called CLUSTER BY. By default, the PARTITION BY statements are converted to CLUSTER BY but are commented out as CLUSTER BY does not support all the same cases that can be defined in Partitions, so some must be modified before enabling. Each CLUSTER BY statement will need to be individually evaluated based upon the necessity for performance.

*UPPERCASE (0)* - Not supported in Snowflake and must be analyzed to determine if case insensitive collation is sufficient.

**Identified Object Summary (Non Tables)**

**VIEWS: 100% Identified Object Conversion Rate**

Conversion Rate by Lines of Code: 100%

Number of Views: 9

Lines of Code: 114

Parsing Errors: 0

Conversion Errors: 0

**JOIN INDEXES: 0% Identified Object Conversion Rate**

*Join indexes can be converted to Materialized views if only one table is referenced in the join index. This can have cost impacts that need to be analyzed.  Join indexes referencing more than a single table would be converted to a traditional view.*

Conversion Rate by Lines of Code: 0%

Number of Join Indexes: 0

Lines of Code: 0

Parsing Errors: 0

Conversion Errors: 0

**MACROS: 100% Identified Object Conversion Rate**

Conversion Rate by Lines of Code: 100%

Number of Macros: 2

Lines of Code: 16

Parsing Errors: 0

Conversion Errors: 0

**PROCEDURES: 100% Identified Object Conversion Rate**

Conversion Rate by Lines of Code: 100%

Number of Procedures: 6

Lines of Code: 176

Parsing Errors: 0

Conversion Errors: 0

Dynamic Result Sets: 3

Cursors: 1

Dynamic SQL Statements: 12

ACTIVITY\_COUNT Occurrences: 3

**NUMERIC DATA OPERATIONS CONSIDERATIONS**

*Calculation Precision* - Calculations in Teradata round after every step based upon the data types in the operation.  For decimal types, this will keep the larger precision used in the calculation.  NUMBER data types will keep full precision.  Since Snowflake stores everything as a number, it keeps full precision throughout the calculation which can result in different outcomes for any number type (decimal, integer, float...) other than NUMBER. This behavior is not changed by automated code conversion.

Teradata: SELECT (1.00/28) \* 15.00 = 0.60

Snowflake: SELECT (1.00/28) \* 15.00 = 0.535710

*Integer-Integer Division* – Teradata performs a truncation or floor when dividing two integer values in, whereas Snowflake will ultimately perform a round operation. This scenario is accounted for in the automated code conversion by adding a TRUNC statement wherever this happens.

Teradata: SELECT (5/3) = 1

Snowflake: SELECT (5/3) = 1.6666666 = 2

Converted Snowflake: SELECT TRUNC(5/3) = 1

*Banker Rounding* - Teradata has the option to use Banker rounding by setting the parameter ROUNDHALFWAYMAGUP while Snowflake only uses normal rounding.

|  |  |  |
| --- | --- | --- |
| SQL | Teradata | Snowflake |
| **CAST( 1.05 AS DECIMAL(9,1))** | 1.0 | 1.1 |
| **CAST( 1.15 AS DECIMAL(9,1))** | 1.2 | 1.2 |
| **CAST( 1.25 AS DECIMAL(9,1))** | 1.2 | 1.3 |
| **CAST( 1.35 AS DECIMAL(9,1))** | 1.4 | 1.4 |
| **CAST( 1.45 AS DECIMAL(9,1))** | 1.4 | 1.5 |
| **CAST( 1.55 AS DECIMAL(9,1))** | 1.6 | 1.6 |
| **CAST( 1.65 AS DECIMAL(9,1))** | 1.6 | 1.7 |
| **CAST( 1.75 AS DECIMAL(9,1))** | 1.8 | 1.8 |
| **CAST( 1.85 AS DECIMAL(9,1))** | 1.8 | 1.9 |
| **CAST( 1.95 AS DECIMAL(9,1))** | 2.0 | 2.0 |

*Decimal to Integer Conversion* - Teradata truncates the decimal values while Snowflake rounds to the nearest integer.

|  |  |  |
| --- | --- | --- |
| SQL | Teradata | Snowflake |
| **CAST( 1.0 AS INTEGER)** | 1 | 1 |
| **CAST( 1.1 AS INTEGER)** | 1 | 1 |
| **CAST( 1.2 AS INTEGER)** | 1 | 1 |
| **CAST( 1.3 AS INTEGER)** | 1 | 1 |
| **CAST( 1.4 AS INTEGER)** | 1 | 1 |
| **CAST( 1.5 AS INTEGER)** | 1 | 2 |
| **CAST( 1.6 AS INTEGER)** | 1 | 2 |
| **CAST( 1.7 AS INTEGER)** | 1 | 2 |
| **CAST( 1.8 AS INTEGER)** | 1 | 2 |
| **CAST( 1.9 AS INTEGER)** | 1 | 2 |

*Number without Precision/Scale* - Number when defined without a scale/precision in Teradata allows for a flexible scale value for any given record from 0 to 38 provided the total precision never exceeds 38.  Snowflake does not allow this and always has a fixed scale/precision.  An example of numbers defined in a table this way:

CREATE MULTISET TABLE DATABASEXYZ.TABLE\_NUMS

     (NUM\_COL1 NUMBER(\*),

      NUM\_COL2 NUMBER,

      NUM\_COL3 NUMBER(38,\*));

In this table, the example of 2 values below that would not fit into a single Snowflake column but could be found in any of the columns shown in the table above in Teradata:

Value 1:  123,345,678,901,234,567,891,012.0123456789

Value 2:  123.12345678901234567890

These two values would require a fixed precision/scale of NUMBER(42, 20) which exceeds the maximum available in Snowflake precision of 38.  Snowflake is currently developing the functionality for flexible precision/scale.

*Truncation on INSERT for SQL DML Statements* - Teradata will auto-truncate a string value upon insert if the string is too large to fit into the specified field.  SnowConvert will convert fields on a like to like basis (ex: VARCHAR(20) -> VARCHAR(20)).  If an ingestion process is relying on auto-truncation, that process will need to be adjusted manually to truncate the data using a LEFT() function.  SnowConvert does not automatically add this as there are multiple implications to doing this across the entire code base.

*Float Default Issue Example:*

/\* <sc-table> TABLE AUPR\_CDM\_WORK.CH4F694E77A5109C703018F\_003\_RS </sc-table> \*/

/\*\*\*\* WARNING: SET TABLE FUNCTIONALITY NOT SUPPORTED \*\*\*\*/

CREATE TABLE CUSTOMER\_X.AUPR\_CDM\_WORK.CH4F694E77A5109C703018F\_003\_RS (

LOGTYPE INTEGER,

OPERSEQ INTEGER DEFAULT 0,

RUNDATE DATE DEFAULT CURRENT\_DATE() /\*\*\*\* WARNING: FORMAT 'YYYY/MM/DD' NOT SUPPORTED \*\*\*\*/,

RUNTIME FLOAT /\*\*\*\* WARNING: FORMAT '99:99:99' NOT SUPPORTED \*\*\*\*/ **/\*\*\*\* ERROR: DEFAULT CURRENT\_TIME NOT VALID FOR DATA TYPE \*\*\*\*/,**

USERID CHAR(128) COLLATE 'en-ci' DEFAULT CURRENT\_USER NOT NULL,

LOGVERSION CHAR(4) COLLATE 'en-ci' DEFAULT 'V3.0',

CHKPTINFO BINARY,

TOTALAMPS SMALLINT DEFAULT 0,

UPAMPS SMALLINT DEFAULT 0,

DOWNAMPS SMALLINT DEFAULT 0,

AMPARRAY BINARY,

UNIQUE ( LOGTYPE ,OPERSEQ)

);

**Notes on Output Files Not Generating**

If one or more output files were not generated, this may be due to a critical conversion error. Critical errors are serious enough that SnowConvert cannot properly recover and can often lead to output files not being generated. Fear not! All the assessment information gathered by SnowConvert before the critical error occurred is still available in this report. However, data that would have been collected after the error was encountered is not included in the report. As a result, the migration did not complete, and the conversion percentages may not be 100% accurate in this run of SnowConvert.

How should we resolve this problem? Great question. Check out the documentation for SnowConvert that specifically addresses troubleshooting through these issues.

<http://docs.mobilize.net/SnowConvert/articles/issues-troubleshooting.html>