DBMigratePro Migration Series

SQL Server → Snowflake

From on-prem DW to cloud-native — elastic scale, per-query pricing.

White paper v1.0 · April 2026 · © 2026 DBMigratePro

Executive summary

SQL Server is a strong OLTP database but has limits as a modern data warehouse. Fixed-capacity compute, licence-per-core pricing, and operational overhead around index maintenance, statistics, and partitioning have pushed many teams to look at Snowflake for their analytical workloads while keeping SQL Server (or moving elsewhere) for operational OLTP.

Snowflake's separation of storage and compute, per-query pricing, and zero-touch operations make it the default modern cloud DW. The migration from SQL Server is well-understood — T-SQL translates reasonably to Snowflake Scripting, data types have clean equivalents, and SSIS packages often need to be replaced with a modern ELT tool (dbt, Airbyte, Fivetran) anyway.

DBMigrateAIPro automates the mechanical translation and flags the parts that require re-thinking. This paper explains what the tool does, where SQL Server → Snowflake differs from a same-paradigm migration, and the outcomes to expect.

DBMigrateAIPro moves SQL Server data-warehouse workloads to Snowflake — converting schema, translating T-SQL into Snowflake Scripting, staging bulk loads through your cloud account, and validating every row.

Why migrate from SQL Server to Snowflake?

Teams move SQL Server analytical workloads to Snowflake for a consistent set of reasons.

Elastic compute, not fixed capacity

SQL Server data warehouses size for peak workload — month-end close, quarterly reporting, year-end reconciliation — and underutilise the rest of the time. Snowflake spins virtual warehouses up and down by the minute. You pay only for the compute you actually use.

Licence cost reduction

SQL Server Enterprise per-core licensing on modern hardware runs into six figures annually for a reasonably-sized DW. Snowflake's consumption model often cuts total annual spend by 30–60% — and removes the audit risk that comes with SQL Server contracts.

Zero-touch operations

Snowflake has no indexes to maintain, no statistics to update, no partitioning to design, no backups to schedule, no TEMPDB to tune. The operational overhead on a SQL Server DW — DBA time, on-call, patching windows — largely goes away.

Modern analytics tooling fit

Snowflake is a first-class target for dbt, Fivetran, Airbyte, Looker, Power BI, and Tableau. The broader data stack is Snowflake-native; SQL Server adapters exist but usually lag.

What makes this migration hard

SQL Server → Snowflake is a paradigm shift from row-store to columnar, from indexes to micro-partitions, from fixed capacity to elastic compute. The tool handles the mechanics and flags the parts that need architectural review.

T-SQL → Snowflake Scripting

T-SQL procedures translate reasonably to Snowflake Scripting. DECLARE @var, SET, cursor loops, and TRY/CATCH blocks all have Snowflake equivalents — but the syntax is different, and some SQL Server patterns (CLR procedures, Service Broker, SQL Agent jobs) have no Snowflake analogue. The tool transpiles what it can and flags what it can't.

No indexes — micro-partitions and clustering

Snowflake automatically partitions tables into micro-partitions; there are no user-managed indexes. SQL Server index DDL is dropped with a reviewable report. Clustering keys replace some index use cases — the tool proposes clustering keys for large tables based on observed index patterns.

Database-vs-schema model

SQL Server uses database.schema.object three-part names. Snowflake has a similar database.schema.table hierarchy, so the mapping is clean — but cross-database references and USE statements need rewriting. DBMigrateAIPro handles three-part reference rewriting automatically.

SSIS packages and SQL Agent jobs

SSIS packages do not translate to Snowflake; they are flagged for replacement with a modern ELT tool (dbt for transforms, Fivetran/Airbyte for ingestion). SQL Agent jobs are flagged for migration to Snowflake Tasks or an external scheduler.

COPY INTO staging

Bulk loads into Snowflake go via COPY INTO from an external stage (S3, Azure Blob, GCS). DBMigrateAIPro stages the data automatically using the cloud account you configure, then runs COPY INTO for each table in parallel.

How DBMigratePro handles it

Every SQL Server → Snowflake migration runs through the same autopilot pipeline. You point DBMigratePro at the source and target, and the engine plans the work, converts the schema and code, streams the data, validates row-by-row, and rolls back automatically on drift. No hand-rolled scripts.

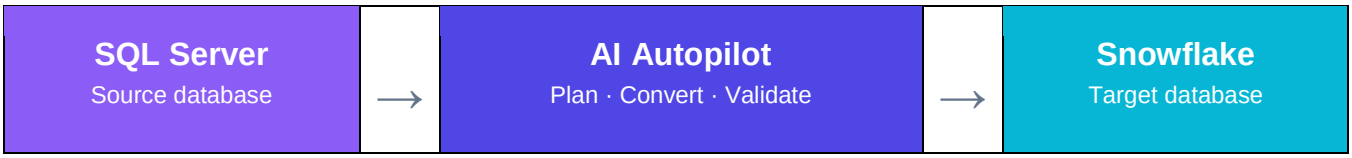

Figure 1 — End-to-end flow. AI Autopilot orchestrates all phases.

Six autonomous phases

1. Pre-flight

Reads sys.objects, sys.tables, sys.procedures; counts rows and objects; flags SSIS packages, CLR procedures, Service Broker usage, and other items that require manual intervention.

2. Schema conversion

Translates tables, constraints, and views to Snowflake DDL. Indexes become reviewable clustering-key proposals. IDENTITY → AUTOINCREMENT. UNIQUEIDENTIFIER → STRING.

3. Code object transpilation

T-SQL procedures and functions translated to Snowflake Scripting. TRY/CATCH → EXCEPTION blocks; RAISERROR → RAISE.

4. Staged data copy

Rows exported to a cloud stage (S3, Azure Blob, GCS) in compressed columnar format, then COPY INTO'd per table in parallel.

5. Validation

Row-by-row hash comparison using HASHBYTES (SQL Server) and MD5 (Snowflake). Mismatches quarantined with full diff.

6. Rollback (on drift)

Validation failure triggers automatic rollback of the target. All actions logged for resumable retries.

Data type mapping

DBMigratePro ships with a built-in type map for every supported SQL Server type. You can override any mapping per column from the UI — the table below shows the defaults.

SQL Server type	Snowflake type	Notes
BIT	BOOLEAN	Direct map.
TINYINT / SMALLINT / INT / BIGINT	NUMBER(p,0)	Snowflake uses NUMBER for integers; aliases available.
DECIMAL(p,s) / NUMERIC(p,s)	NUMBER(p,s)	Exact mapping.
MONEY / SMALLMONEY	NUMBER(19,4)	No native MONEY type.
FLOAT / REAL	FLOAT / REAL	Same semantics.
VARCHAR(n) / NVARCHAR(n)	VARCHAR(n)	Snowflake VARCHAR is UTF-8 always; no N-prefix needed.
CHAR(n) / NCHAR(n)	CHAR(n)	Padded behaviour preserved.
TEXT / NTEXT	VARCHAR	VARCHAR is effectively unbounded.
DATE	DATE	Same semantics.
DATETIME / DATETIME2	TIMESTAMP_NTZ	Precision preserved.
DATETIMEOFFSET	TIMESTAMP_TZ	Timezone preserved.
UNIQUEIDENTIFIER	STRING	Snowflake has no UUID type; STRING is the idiomatic choice. NEWID() rewritten to UUID_STRING().
VARBINARY / IMAGE	BINARY	Binary data maps cleanly.
XML	VARIANT	Snowflake VARIANT is the idiomatic home for structured data.
HIERARCHYID	STRING	Query patterns rewritten to recursive CTEs.

Code object conversion

DBMigrateAIPro transpiles what maps cleanly into Snowflake Scripting and flags what doesn't. Converted objects are saved side-by-side with originals:

- Procedures — T-SQL → Snowflake Scripting (SQL-based with JavaScript fallback for complex logic).
- Functions — scalar functions map cleanly; table-valued functions translated where possible.
- Triggers — Snowflake uses streams + tasks instead; triggers flagged for redesign.
- Views and indexed views — DDL translated; materialized-view semantics require review.
- Built-in function calls — ISNULL, GETDATE, DATEADD, DATEDIFF, LEN, CHARINDEX, and 60+ others rewritten.
- TOP (n) → LIMIT n; PIVOT → Snowflake PIVOT (syntax preserved).
- CTEs pass through unchanged; MERGE has a Snowflake equivalent and translates directly.
- CLR procedures — flagged; no Snowflake equivalent.

Validation, safety, rollback

DBMigrateAIPro validates every row it copies — using native hash functions (HASHBYTES on SQL Server, MD5 on Snowflake) so the data never leaves either platform.

For each table, Autopilot computes a hash over a stable canonical row representation on SQL Server and on Snowflake, then joins the two sets by primary key (or business key for denormalised analytical tables). Matches are confirmed; mismatches recorded with PK and byte-level diff.

The validation report — row counts, match counts, quarantined rows, per-mismatch detail — is a single artefact for cutover sign-off.

Safety guarantees

- Pre-flight assessment is read-only — production SQL Server is never modified.
- The target is written only inside the project's Snowflake database/schema.
- Every DDL and COPY INTO is logged to an auditable project workspace.
- Row-hash validation uses native DB functions — data never leaves either platform.
- Automatic rollback on any validation failure; no partial state left behind.
- All generated SQL is reviewable before execution — nothing is a black box.
- External stage (S3, Azure Blob, GCS) is configured by you and credentials stay in your cloud account.

Typical outcomes

Aggregated from the beta cohort running production SQL Server → Snowflake migrations on data warehouses between 1 TB and 20 TB.

Metric	What teams typically see
Time to first migration	Hours from install to a green dev migration on a representative schema.
End-to-end project duration	Weeks for typical DW estates; SSIS-to-dbt rewrite is the usual time sink.
T-SQL auto-conversion rate	90%+ of T-SQL procedures transpile; the 10% balance is architecturally different (triggers, CLR, Service Broker).
Validation accuracy	99.98% row-hash match on production workloads.
Infrastructure cost	Typically 30–60% reduction vs. equivalent SQL Server DW hardware + support + licensing.
Operational overhead	DBA time drops 80%+ — no indexes, no stats, no TEMPDB, no partition maintenance.

Next steps

- Install DBMigrateAIPro (desktop or CLI) — free for Hobby-scale databases during beta.
- Configure an external stage (S3, Azure Blob, or GCS) in your cloud account.
- Run the pre-flight assessment against a dev copy of SQL Server — read-only.
- Review the generated workspace: converted DDL, transpiled T-SQL, flagged SSIS / CLR / Agent items.
- Run a dry migration to a test Snowflake database; review the row-hash validation report.
- Book a 30-minute migration review with our team (free) to triage flagged items and SSIS replacement strategy.
- Schedule the cutover. Staged COPY INTO handles bulk data; incremental loads bridge the final delta.

Ready to scope a SQL Server → Snowflake project? Get in touch at hello@dbmigratepro.ai or start your free beta at dbmigratepro.ai/signup.