DBMigratePro Migration Series

SQL Server → PostgreSQL

Cut licence spend, shift to Linux, modernise the stack.

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

Executive summary

Microsoft SQL Server is a capable database, but its per-core licensing, Windows-first deployment story, and tight coupling to the Microsoft stack have become expensive constraints for teams standardising on Linux, Kubernetes, and open-source tooling. PostgreSQL is the dominant destination when teams decide to move off SQL Server.

The migration is not free. T-SQL and PL/pgSQL are different dialects. SQL Server's database-vs-schema model doesn't map 1:1 to PostgreSQL. DATETIME2, UNIQUEIDENTIFIER, NVARCHAR, linked servers, and SQL Agent jobs each have PostgreSQL equivalents — but the translation takes weeks if done by hand.

DBMigrateAIPro automates the whole pipeline: schema conversion, T-SQL → PL/pgSQL translation, data streaming, row-hash validation, and automatic rollback. This paper explains what the tool does, where SQL Server → PostgreSQL gets tricky, and the outcomes to expect.

> DBMigrateAIPro moves SQL Server workloads to PostgreSQL — translating T-SQL to PL/pgSQL, rewriting identity columns and schemas, and validating every row before cutover.

Why migrate from SQL Server to PostgreSQL?

Teams move off SQL Server for a short, predictable list of reasons.

Per-core licensing cost

SQL Server Enterprise licensing on modern hardware can exceed $7,000 per core. PostgreSQL has no per-core licence. For teams running multi-socket database servers, the annual saving often justifies the migration project by itself.

Linux / Kubernetes alignment

Running SQL Server on Linux is supported but uncommon in practice. Most teams standardising on Linux, containers, and Kubernetes find PostgreSQL a much more natural fit — it runs everywhere, the operator ecosystem is mature, and the tooling is uniform across environments.

Cloud portability

Every major cloud provides a first-class managed PostgreSQL service. Moving off SQL Server removes a vendor concentration risk and opens up cheaper managed options (RDS, Cloud SQL, Supabase, Neon, Aurora) that are often hard to match on price for SQL Server workloads.

Open standards and talent

PostgreSQL skills are widely available, the documentation is free, and there is no vendor relationship to manage. The open-source ecosystem around PostgreSQL — ORMs, observability, migration tools, extensions — keeps pace with modern application development patterns.

What makes this migration hard

T-SQL and PL/pgSQL are different enough that naive translation fails. DBMigrateAIPro handles each of these areas automatically.

T-SQL → PL/pgSQL translation

T-SQL uses DECLARE @var, SET statements, and PRINT — none of which map to PL/pgSQL directly. Cursor syntax differs, TRY/CATCH vs EXCEPTION blocks use different structures, and RAISERROR / THROW have to be rewritten. DBMigrateAIPro transpiles the full T-SQL surface into idiomatic PL/pgSQL and emits a side-by-side diff.

Database vs schema model

SQL Server uses separate databases as the primary isolation unit, with a flat schema model below. PostgreSQL uses one database with multiple schemas. Cross-database queries, USE statements, and three-part names (db.schema.table) all need translation. DBMigrateAIPro maps each SQL Server database to a PostgreSQL schema by default and rewrites all three-part references.

IDENTITY, UNIQUEIDENTIFIER, DATETIME2

IDENTITY columns map to PostgreSQL IDENTITY (10+) or SERIAL. UNIQUEIDENTIFIER maps to UUID with the same generation semantics. DATETIME2 maps to TIMESTAMP(n). NEWID() and NEWSEQUENTIALID() are rewritten to gen_random_uuid() with pgcrypto. All automatic, all reviewable.

Linked servers and distributed queries

SQL Server linked servers translate to PostgreSQL foreign data wrappers (postgres_fdw, tds_fdw, oracle_fdw). The tool emits FDW server and foreign-table definitions automatically for every linked server referenced in the source.

SQL Agent jobs and SSIS

SQL Server Agent scheduled jobs need an equivalent scheduler — typically pg_cron, an external scheduler (Airflow, dbt Cloud, Dagster), or the cloud provider's scheduler. SSIS packages usually need to be re-implemented; DBMigrateAIPro flags every SQL Agent job and SSIS reference in the assessment.

How DBMigratePro handles it

Every SQL Server → PostgreSQL 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.

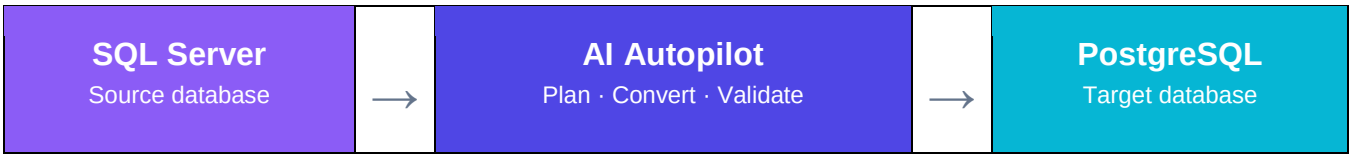

Figure 1 — End-to-end flow. AI Autopilot orchestrates all phases.

Six autonomous phases

1. Pre-flight

Reads sys.objects, sys.tables, sys.columns, sys.procedures; counts rows and objects; flags linked servers, SSIS packages, Service Broker usage, and other manual-intervention items.

2. Schema conversion

Translates tables, constraints, indexes, views, types, and schemas. IDENTITY → IDENTITY, UNIQUEIDENTIFIER → UUID, DATETIME2 → TIMESTAMP, database → schema mapping applied consistently.

3. Code object transpilation

T-SQL procedures, functions, and triggers are parsed and rewritten to PL/pgSQL. TRY/CATCH becomes EXCEPTION blocks; RAISERROR becomes RAISE; cursor syntax rewritten.

4. Data streaming

Rows stream from SQL Server to PostgreSQL in parallel. FKs deferred during load, re-enabled at the end. Shadow-apply mode available for zero-downtime cutover.

5. Validation

Row-by-row hash comparison using native DB functions on both sides. Data never leaves the servers. Mismatches quarantined with full diff detail.

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	PostgreSQL type	Notes
BIT	BOOLEAN	SQL Server BIT (0/1) maps cleanly to PostgreSQL BOOLEAN.
TINYINT	SMALLINT	SQL Server TINYINT is unsigned; PG SMALLINT is signed but range is sufficient.
SMALLINT	SMALLINT	Same semantics.
INT	INTEGER	Same semantics.
BIGINT	BIGINT	Same semantics.
DECIMAL(p,s) / NUMERIC(p,s)	NUMERIC(p,s)	Exact mapping.
MONEY / SMALLMONEY	NUMERIC(19,4)	No native MONEY type in PostgreSQL.
FLOAT / REAL	DOUBLE PRECISION / REAL	Same semantics.
VARCHAR(n)	VARCHAR(n)	Same semantics.
NVARCHAR(n) / NTEXT	TEXT	PostgreSQL TEXT is always UTF-8; no N-prefix needed.
CHAR(n) / NCHAR(n)	CHAR(n)	Length-padded behaviour preserved.
DATE	DATE	Same semantics.
DATETIME	TIMESTAMP(3)	SQL Server DATETIME has ~3.33ms precision; TIMESTAMP(3) is close.
DATETIME2(n)	TIMESTAMP(n)	Precision preserved.
DATETIMEOFFSET	TIMESTAMPTZ	TIMESTAMPTZ stores UTC; semantics preserved.
UNIQUEIDENTIFIER	UUID	Direct map; NEWID() rewritten to gen_random_uuid().
VARBINARY / IMAGE	BYTEA	Binary data maps cleanly.
XML	XML	Native PostgreSQL XML type; XPath/XQuery supported.
HIERARCHYID	LTREE	Requires the ltree extension; queries rewritten to ltree operators.

Code object conversion

DBMigrateAIPro transpiles the full T-SQL procedural surface. Every converted object is saved side-by-side with the original:

- Procedures and functions — full T-SQL → PL/pgSQL translation; OUTPUT parameters preserved.
- Triggers — AFTER/INSTEAD OF; INSERTED/DELETED pseudo-tables rewritten to NEW/OLD.
- Views and indexed views — DDL translated; materialized views offered for indexed-view workloads.
- User-defined table types — translated to PostgreSQL composite types or temp tables.

SQL Server → PostgreSQL

- Built-in function calls — ISNULL, GETDATE, DATEADD, DATEDIFF, LEN, CHARINDEX, STUFF, and 80+ others rewritten.
- TOP (n) → LIMIT n; PIVOT / UNPIVOT → CASE / UNION patterns.
- CTEs — pass through unchanged; MERGE translated where semantics require.
- Error handling — TRY/CATCH blocks become BEGIN/EXCEPTION blocks with equivalent state.

Validation, safety, rollback

DBMigrateAIPro validates every row it copies — using native hash functions (HASHBYTES on SQL Server, md5/digest on PostgreSQL) so the data never leaves either server.

For each table, Autopilot computes a hash over a stable canonical row representation on SQL Server and on PostgreSQL, then joins the two sets by primary key. Matches are confirmed; mismatches are recorded with PK and byte-level diff.

The validation report — row counts, match counts, quarantined rows, and per-mismatch detail — is a single attachable 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 schema; existing PostgreSQL schemas are never touched.
- Every DDL and DML action is logged to an auditable project workspace.
- Row-hash validation uses native DB functions — data never leaves the source or target server.
- Automatic rollback on any validation failure; no partial state left behind.
- Shadow-apply mode allows cutover with zero downtime on continuously-updated source tables.
- All generated SQL is reviewable before execution — nothing is a black box.

Typical outcomes

Aggregated from the beta cohort running production SQL Server → PostgreSQL migrations on estates between 100 GB and 5 TB.

Metric	What teams typically see
Time to first migration	< 1 hour from install to a green dev migration on a representative schema.
End-to-end project duration	Days to a few weeks for typical estates, vs. 4–12 months hand-rolled.
T-SQL auto-conversion rate	98%+ of procedures transpile without manual edits; remaining 2% flagged for review.
Validation accuracy	99.98% row-hash match on production workloads; mismatches quarantined, never silently written.
Cutover downtime	Near-zero for continuously-updated tables using shadow-apply + atomic switch.
License cost reduction	50–80% typical annual database spend reduction vs. the prior SQL Server contract.

Next steps

- Install DBMigrateAIPro (desktop or CLI) — free for Hobby-scale databases during beta.
- Run the pre-flight assessment against a dev copy of your SQL Server database — read-only.
- Review the generated workspace: converted DDL, transpiled T-SQL, flagged SSIS / Agent / linked-server items.
- Run a dry migration to a test PostgreSQL target; review the row-hash validation report.
- Book a 30-minute migration review with our team (free) before the production cutover.
- Schedule the cutover. Shadow-apply handles continuously-updated tables with zero downtime.

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