DBMigratePro Migration Series

MySQL → PostgreSQL

Feature-complete upgrade — JSONB, CTEs, real transactions, extensions.

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

Executive summary

PostgreSQL and MySQL look similar from a distance, but teams that have run both in production know the differences are meaningful. Real transactional DDL, strict data types, a rich extension ecosystem, first-class JSON (JSONB), window functions, CTEs, and full-text search make PostgreSQL the natural next step when a MySQL workload outgrows what the engine was designed for.

The migration itself is usually less painful than Oracle → PostgreSQL, but the long tail of stored procedures, AUTO_INCREMENT columns, ENUMs, character-set issues, and subtle date/time semantics still eats weeks. DBMigrateAIPro automates the whole pipeline — schema conversion, procedure translation, data streaming, row-hash validation, and automatic rollback.

This paper explains what the tool does, where MySQL → PostgreSQL quietly gets tricky, and the outcomes teams see on real projects.

DBMigrateAIPro moves a MySQL estate to PostgreSQL with automated schema, stored-procedure translation, and row-hash validation — without rewriting a line of application code.

Why migrate from MySQL to PostgreSQL?

Teams move from MySQL to PostgreSQL for three reasons that show up again and again.

Feature ceiling

MySQL has caught up on many fronts, but PostgreSQL still leads on JSONB, GIN/GiST indexing, full-text search, window functions, CTEs, partial indexes, foreign data wrappers, and the extension ecosystem (PostGIS, pg_stat_statements, TimescaleDB, pgvector). Workloads that lean on analytics, search, or geospatial hit a wall on MySQL that PostgreSQL simply doesn't have.

Correctness and strict typing

PostgreSQL rejects invalid data — MySQL has historically been permissive about silently truncating strings, inserting zero dates, and accepting out-of-range numbers. Teams that have been bitten by silent data corruption want strict mode by default.

Transactional DDL and operational safety

PostgreSQL runs schema changes inside transactions; MySQL doesn't. This matters enormously during deployments — a failed migration in PostgreSQL rolls back cleanly; in MySQL it can leave the schema half-applied. For teams running CI/CD with frequent schema changes, this one feature is often the tipping point.

Cloud and ecosystem

Managed PostgreSQL (RDS, Aurora, Cloud SQL, Azure, Supabase, Neon, Crunchy Data) has become the default cloud primitive. The tooling, connectors, ORMs, and community resources around PostgreSQL are no longer a trade-off.

What makes this migration hard

MySQL → PostgreSQL is friendlier than Oracle → PostgreSQL, but a handful of areas still need deliberate handling. DBMigrateAIPro handles each one automatically and flags the rare edge cases for review.

Stored procedures and functions

MySQL stored procedures use a different dialect from PL/pgSQL. DECLARE semantics, cursor handling, error conditions (SQLSTATE vs PostgreSQL exception blocks), and control flow keywords all differ. DBMigrateAIPro transpiles the full MySQL procedure surface into idiomatic PL/pgSQL, preserving parameter semantics and error-handling behaviour.

Data type ceilings and semantics

MySQL's TINYINT(1) is used as BOOLEAN by convention; it maps to PostgreSQL BOOLEAN cleanly. DATETIME vs TIMESTAMP has subtly different semantics around timezone storage. DECIMAL precision rules differ. DBMigrateAIPro's default type map handles all of these and lets you override per column.

AUTO_INCREMENT and identity columns

MySQL AUTO_INCREMENT maps to PostgreSQL's IDENTITY columns on modern PostgreSQL (10+), or SERIAL on older versions. The tool handles both targets and updates any code that explicitly references LAST_INSERT_ID() to use PostgreSQL's RETURNING clause.

ENUM types

MySQL's inline ENUM declarations don't exist in PostgreSQL — you get a named CREATE TYPE enum instead, or a VARCHAR with a CHECK constraint. DBMigrateAIPro defaults to real PostgreSQL enum types (cleaner, faster) but you can override to CHECK constraints if your application already does lax enum handling.

Character set and collation

MySQL databases are often on utf8mb4 with specific collations (utf8mb4_unicode_ci, utf8mb4_general_ci). PostgreSQL uses ICU or libc collations with different naming and slightly different sort rules. The tool preserves case-insensitive sort semantics by default using PostgreSQL's CITEXT extension where appropriate.

How DBMigratePro handles it

Every MySQL → 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.

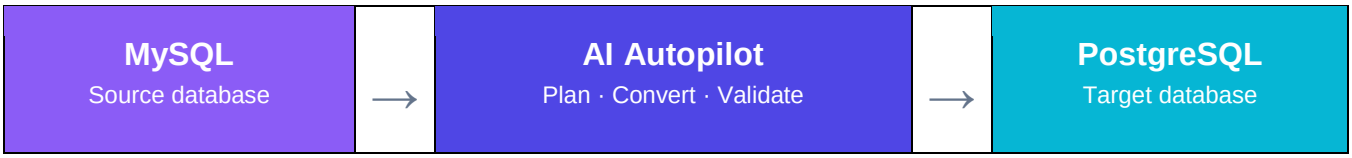

Figure 1 — End-to-end flow. AI Autopilot orchestrates all phases.

Six autonomous phases

1. Pre-flight

Reads the MySQL catalog (information_schema), counts rows and objects, flags incompatibilities, and produces a full assessment and risk score. No production load.

2. Schema conversion

Translates every table, constraint, index, view, and sequence to PostgreSQL DDL. AUTO_INCREMENT becomes IDENTITY, ENUM becomes CREATE TYPE, character-set and collation are mapped.

3. Code object transpilation

MySQL stored procedures, functions, and triggers are parsed and rewritten to PL/pgSQL. Original and converted code are saved side-by-side for review.

4. Data streaming

Rows stream from MySQL 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 MD5/SHA hash comparison using native DB functions. Data never leaves the servers. Mismatches are quarantined with full diff detail.

6. Rollback (on drift)

Any validation failure triggers automatic rollback of the target. Every action is logged so the next attempt resumes with full context.

Data type mapping

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

MySQL type	PostgreSQL type	Notes
TINYINT(1)	BOOLEAN	MySQL convention for BOOLEAN; direct map.
TINYINT	SMALLINT	Signed 8-bit → 16-bit PostgreSQL.
SMALLINT / MEDIUMINT	INTEGER	MySQL MEDIUMINT has no direct PG equivalent.
INT	INTEGER	Same semantics.
BIGINT	BIGINT	Same semantics.
DECIMAL(p,s)	NUMERIC(p,s)	Exact mapping.
FLOAT / DOUBLE	REAL / DOUBLE PRECISION	Same semantics.
VARCHAR(n)	VARCHAR(n)	PostgreSQL allows larger max than MySQL; mapping is safe.
TEXT / LONGTEXT	TEXT	PostgreSQL TEXT is unbounded; no size variants needed.
DATE	DATE	Same semantics.
DATETIME	TIMESTAMP	No timezone storage in either.
TIMESTAMP	TIMESTAMPTZ	MySQL TIMESTAMP stores UTC; TIMESTAMPTZ preserves this.
ENUM(...)	CREATE TYPE ... AS ENUM	Default; can also map to CHECK constraint if preferred.
BLOB / LONGBLOB	BYTEA	Binary data maps cleanly.
JSON	JSONB	JSONB is faster and indexable; default choice.

Code object conversion

DBMigrateAIPro transpiles the full MySQL stored-procedure surface. Every converted object is saved side-by-side with the original for review:

- Procedures and functions — IN/OUT/INOUT parameter semantics preserved.
- Triggers — BEFORE/AFTER INSERT/UPDATE/DELETE; NEW/OLD references rewritten.
- Views — DDL translated; MySQL's WITH CHECK OPTION is preserved.
- Events (MySQL scheduled events) — flagged for migration to pg_cron or an external scheduler.
- Built-in function calls — IFNULL, NOW, CURDATE, STR_TO_DATE, GROUP_CONCAT, and 60+ others rewritten.
- Query rewriting — LIMIT/OFFSET passes through unchanged; INSERT … ON DUPLICATE KEY UPDATE translated to INSERT … ON CONFLICT.
- User-defined variables (@var) — rewritten to CTEs or PL/pgSQL local variables where semantics allow.

Validation, safety, rollback

DBMigrateAIPro validates every row it copies — using native hash functions on both sides so the data never leaves the source or target database.

For each table, Autopilot computes an MD5 (or SHA-256) over a stable canonical row representation on MySQL and on PostgreSQL, then joins the two hash sets by primary key. Every row that matches is confirmed. Every row that doesn't is recorded with its PK and the byte-level difference.

The resulting report — row counts, hash-match counts, quarantined-row counts, and a drill-down for every mismatch — gives you a single artefact you can attach to any cutover sign-off.

Safety guarantees

- Pre-flight assessment is read-only — production MySQL 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 with timestamps.
- Row-hash validation uses native DB functions — data never leaves the source or target server.
- Automatic rollback on any validation failure; partial state is never 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 MySQL → PostgreSQL migrations on estates between 50 GB and 3 TB.

Metric	What teams typically see
Time to first migration	< 30 minutes from install to a green dev migration on a representative schema.
End-to-end project duration	Hours to a few days for typical estates, vs. 2–6 months hand-rolled.
Procedure auto-conversion	99%+ of MySQL procedures transpile without manual edits; remaining 1% flagged for review.
Validation accuracy	99.99% row-hash match on production workloads; mismatches quarantined, never silently written.
Cutover downtime	Near-zero for continuously-updated tables using shadow-apply + atomic switch.
Managed-service cost delta	Typically neutral-to-cheaper; feature gains (JSONB, CTEs, extensions) are the usual ROI.

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 MySQL database — read-only.
- Review the generated workspace: converted DDL, transpiled procedures, flagged 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 MySQL → PostgreSQL project? Get in touch at hello@dbmigratepro.ai or start your free beta at dbmigratepro.ai/signup.