DBMigratePro Migration Series

MongoDB → PostgreSQL

From documents to a real schema — with JSONB, ACID, and analytics on tap.

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

Executive summary

MongoDB makes early product iteration fast. Schema-on-read, flexible documents, and horizontal scaling all remove friction when the data model is still changing daily. But most systems that started on MongoDB reach a point where the lack of schema discipline, the difficulty of running SQL analytics, the cost of Atlas at scale, and the operational weight of a purpose-built document store outweigh the original benefits.

PostgreSQL answers most of these concerns without giving up flexibility. JSONB columns carry semi-structured data with full indexing support; a real relational model captures the parts that are actually stable; ACID transactions, window functions, CTEs, and a mature analytics ecosystem handle the rest.

Migrating is not free. You have to decide what becomes a column, what stays in JSONB, and how to handle arrays and nested documents. DBMigrateAIPro automates the mechanical parts — schema inference, data streaming, index mapping, row-hash validation — and gives you a reviewable schema proposal you can edit before cutover.

DBMigrateAIPro infers a schema from your MongoDB collections, converts them to PostgreSQL tables (with JSONB for the genuinely flexible parts), and validates every document-to-row migration with native hash checks.

Why migrate from MongoDB to PostgreSQL?

Teams move from MongoDB to PostgreSQL for a consistent set of reasons.

Schema discipline

MongoDB's schemaless model is valuable early on, but at scale it means every application code path carries defensive checks for missing fields, type drift, and orphaned references. PostgreSQL enforces the schema in one place — the database — and removes the long tail of data-quality bugs.

SQL analytics

Analytics on MongoDB means the aggregation framework — powerful, but a different model from SQL and hard to share with non-MongoDB analysts. PostgreSQL gives every BI tool, every ORM, and every analyst immediate access to the data with the SQL they already know. JSONB columns keep the flexible fields queryable.

ACID and real transactions

MongoDB supports multi-document transactions since 4.0 but performance and operational cost are high. PostgreSQL's MVCC transaction model is cheap, well-understood, and reliable. Workloads with meaningful cross-document invariants (accounting, inventory, orders, billing) are often the trigger for migration.

Cost and operational weight

MongoDB Atlas at production scale is not cheap, and self-hosted MongoDB comes with operational weight (replica set management, backups, version upgrades). Managed PostgreSQL is usually cheaper per GB and supported by a larger talent pool.

What makes this migration hard

MongoDB → PostgreSQL is the most qualitatively different migration we support. It's less about translation and more about modelling — DBMigrateAIPro makes the modelling choices visible and reviewable rather than hidden in the tool.

Schema inference

MongoDB collections have no declared schema. DBMigrateAIPro samples each collection (configurable sample size), infers a proposed PostgreSQL schema from field presence and type, and shows you every field's coverage percentage. You can accept the proposal, drop columns, promote JSONB fields to columns, or demote columns to JSONB — all before any data moves.

Nested documents and arrays

A nested sub-document can become (a) a JSONB column, (b) a separate child table with an FK, or (c) a flattened set of columns. Arrays can become JSONB, a child table, or a PostgreSQL array column. DBMigrateAIPro proposes a default based on depth and variance and lets you override per field.

ObjectId and document identity

MongoDB ObjectIds are 12-byte identifiers, typically rendered as 24-character hex strings. DBMigrateAIPro maps ObjectIds to VARCHAR(24) by default, or to UUID with a deterministic transform if the application is being rewritten to use UUIDs.

References and $lookup

MongoDB references are an application-level convention, not an enforced constraint. During migration, referenced IDs become foreign keys — and referential integrity problems that were silently tolerated in MongoDB surface as constraint violations. DBMigrateAIPro quarantines orphaned documents and generates a cleanup report before enforcing FKs.

Indexes and query patterns

MongoDB indexes translate reasonably well — B-tree indexes map directly, compound indexes preserve column order, and text indexes become GIN on tsvector or trigram. Geospatial indexes map to PostGIS. $lookup queries become JOINs; aggregation pipelines usually need a rewrite, which the tool flags but does not attempt automatically.

How DBMigratePro handles it

Every MongoDB → 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.

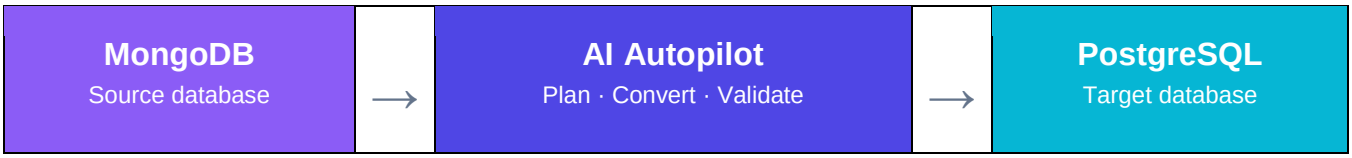

Figure 1 — End-to-end flow. AI Autopilot orchestrates all phases.

Six autonomous phases

1. Pre-flight

Samples each collection, infers a proposed schema, counts documents, flags orphaned references and type drift, and produces a reviewable schema proposal.

2. Schema design

You review the proposed PostgreSQL schema. Accept the defaults, promote/demote JSONB fields to columns, split nested documents into child tables, or add indexes. All via the UI or a config file.

3. Schema apply

Applies the approved schema to PostgreSQL: tables, constraints, indexes (B-tree, GIN on JSONB, GIN on tsvector), foreign keys. All DDL saved to the project workspace.

4. Data streaming

Documents stream from MongoDB to PostgreSQL in parallel. Nested documents fan out into the configured target shape (JSONB column, child table, or flattened columns).

5. Validation

Document-by-document hash comparison. Orphaned references quarantined. Type-drift rows flagged. Full diff report per collection.

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 MongoDB type. You can override any mapping per column from the UI — the table below shows the defaults.

MongoDB type	PostgreSQL type	Notes
String	TEXT	Unbounded; use VARCHAR(n) with CHECK if length is meaningful.
Int32 / Int64	INTEGER / BIGINT	Direct map.
Double	DOUBLE PRECISION	Same semantics.
Decimal128	NUMERIC	Precision preserved.
Boolean	BOOLEAN	Direct map.
Date	TIMESTAMPTZ	MongoDB Date stores UTC ms since epoch; TIMESTAMPTZ preserves this.
ObjectId	VARCHAR(24)	Default; override to UUID if application is rewriting identifiers.
Array	JSONB / child table / ARRAY	Default is JSONB; promote to child table for relational queries.
Embedded document	JSONB / child table / flattened columns	Default is JSONB; tool proposes per-field based on variance.
Null / missing	NULL + nullable column	Missing fields become NULL; tool reports field coverage.
Binary (BinData)	BYTEA	Binary data maps cleanly.
Regex	TEXT (with / delimiters) + flags	Stored as text; query rewriting required.

Code object conversion

MongoDB has no stored procedures to transpile, but several application-layer patterns need attention — the tool documents them in the assessment:

- Aggregation pipelines — flagged for rewrite as SQL queries; the tool does not auto-translate these (the gap is too wide to do reliably).
- $lookup joins — flagged; become SQL JOINs in the rewritten application.
- Map-reduce and server-side JavaScript — flagged; usually rewritten in application code or as PL/pgSQL.
- Change streams — translated to PostgreSQL logical replication + a Debezium-style downstream if CDC is needed.
- TTL indexes — preserved as a scheduled pg_cron job that deletes expired rows.
- Full-text indexes — mapped to GIN on tsvector.
- Geospatial indexes — mapped to PostGIS GiST indexes.

Validation, safety, rollback

DBMigrateAIPro validates every document it migrates. Because documents map to one or more rows (depending on your schema choices), validation is document-centric: every source document must produce the expected target rows.

For each collection, Autopilot computes a hash over a canonical JSON representation on MongoDB and reconstructs the same canonical JSON from the PostgreSQL rows. The two hash sets are joined by ObjectId. Documents that match are confirmed; mismatches are quarantined with full diff detail.

The validation report — document counts, match counts, quarantined documents, orphaned references, type drift — is a single artefact for cutover sign-off.

Safety guarantees

- Pre-flight sampling is read-only — production MongoDB is never modified.
- Schema proposals are reviewable and editable before any data moves.
- 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.
- Document-level validation catches type drift and orphaned references that MongoDB tolerated silently.
- Automatic rollback on any validation failure; no partial state left behind.
- All generated SQL is reviewable before execution — nothing is a black box.

Typical outcomes

Aggregated from the beta cohort running production MongoDB → PostgreSQL migrations on estates between 100 GB and 4 TB.

Metric	What teams typically see
Time to schema proposal	Hours — the tool samples collections and produces a reviewable PostgreSQL schema you can edit.
End-to-end project duration	Weeks for typical estates; schema review is the time sink, not the data copy.
Validation accuracy	99.9%+ document-hash match; orphaned references and type drift surface as quarantined rows.
Cutover downtime	Low — change streams + shadow-apply allow continuous sync up to cutover.
Data-quality improvement	Typical projects find 0.1–2% orphaned references and type drift — visible for the first time.
Atlas cost reduction	50–70% typical cost reduction when moving from Atlas to managed PostgreSQL.

Next steps

- Install DBMigrateAIPro (desktop or CLI) — free for Hobby-scale databases during beta.
- Run the pre-flight schema inference against a read-only replica of MongoDB.
- Review the proposed PostgreSQL schema; edit field-by-field as needed (promote/demote JSONB, split nested docs, add indexes).
- Run a dry migration to a test PostgreSQL target; review the document-hash validation report.
- Book a 30-minute schema review with our team (free) before the production cutover.
- Schedule the cutover. Change-stream + shadow-apply handles continuously-updated collections with near-zero downtime.

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