DBMigrateAIPro · White Paper

The 14-Step AI Migration Methodology

How a structured AI playbook turns Oracle → PostgreSQL from a months-long project into hours of supervised work.

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

Executive summary

Most database migration tools stop at moving bytes. They get data from A to B, ideally with low lag — and that is the entirety of what they do. The real work — deciding what to move, how to translate it, when it is safe, what to optimise after — sits on the engineer running the migration.

DBMigrateAIPro inverts that. The pipe is a commodity. The intelligence is the product. The model is autopilot-first: the engineer approves, the engine drives. At the heart of that engine is a structured, 14-step methodology — a playbook that the AI follows on every migration and that the customer reviews end-to-end before sign-off.

This paper explains the methodology, why each step exists, and how the discipline of running every migration through the same 14 steps is what delivers 95% automated conversion — with the remaining 5% named, explained, and accompanied by working alternative code rather than silently failing.

> Most tools translate. We have a methodology. The first looks like a feature; the second is a moat — and it is what makes the difference between a delivered project and a hopeful one.

Why methodology matters

Database migrations fail for the same handful of reasons across every industry: schema decisions made without inspecting the data, code translations that look correct but diverge in edge cases, validation that confirms row counts but not row contents, and post-migration performance that nobody owns. Each of these is a process failure, not a tooling failure.

AWS Database Migration Service, Oracle GoldenGate, Striim, Qlik Replicate, and Fivetran are excellent at the parts they do — log-based capture, row-level streaming, managed ergonomics. None of them ship a methodology. The decisions a methodology would make are left to whoever is sitting in front of the console.

The four shared blind spots

Blind spot	What it means	Who pays
Schema as a heuristic, not a study	Type maps from declared types, never inspecting actual data shape	The DBA in week 3, when target storage is 4× source
Code translation is binary	Either it works or it stops — no graded confidence	The engineer rewriting 200 PL/SQL packages by hand
No post-cutover intelligence	Once data is on target, the tool's job is done	The team finding out their indexes are wrong a quarter later
Decisions are rule-based, not adaptive	Same answer every run, regardless of data	Workloads where the right answer depends on data shape

The 14-step methodology is a direct response to all four. Every step produces an artefact. Every artefact is human-reviewable. Every decision is traceable to a source-system fact. Nothing happens silently.

The 14 steps

Each step is a discrete prompt to a specialist AI persona — a schema analyst, a type-mapping specialist, a PL/SQL translator, a CDC architect, a performance tuner, a risk officer, an adversarial self-reviewer. Each persona has a role, context, inputs, tasks, constraints, and a strict output format.

Step 1 — Migration Architect (Master Orchestrator)

Holistic schema analysis; defines the migration contract end-to-end; sets the rollback and monitoring strategy.

Step 2 — Schema Analysis

Object inventory, complexity grading, dependency graph, risk register from the source catalog.

Step 3 — Datatype Mapping (Usage-Based)

The differentiating step. Maps every column based on observed data — not declared types — for storage and performance gains.

Step 4 — Schema Conversion (DDL Generation)

Emits executable PostgreSQL DDL from the approved type map. Three split files for parallel data load.

Step 5 — PL/SQL → PL/pgSQL Conversion

Per-object code translation with a confidence score (HIGH / MEDIUM / LOW). Side-by-side diff for every converted object.

Step 6 — Data Migration Strategy

Tooling choice, snapshot consistency, parallelism, batch sizing, network path, encryption, egress modelling.

Step 7 — CDC (Change Data Capture) Design

Log-based capture, supplemental logging, per-tier latency SLA, DLQ handling, cutover runbook.

Step 8 — Validation Query Generator

Five tiers: row counts → checksums → row hashes → sample diff → schema metadata. Catches every class of divergence.

Step 9 — Performance Optimization

Index, partition, query, configuration, autovacuum, statistics, pooling. Every recommendation tied to a measured query.

Step 10 — Pre & Post Migration Steps

Idempotent runbooks, backup verification, rollback testing, app smoke-test, monitoring confirmation.

Step 11 — Risk & Mitigation

14-row risk register covering data, code, performance, operations, compliance, and skills.

Step 12 — Final Migration Report

Executive deliverable consolidating every prior phase. Audience: technical stakeholders + executive sponsors.

Step 13 — Self-Validation

Adversarial review of every artefact. Issues + corrected artefacts re-emitted. Cross-prompt consistency check.

Step 14 — Super Prompt (Autopilot)

Orchestrates all 13 steps. Decision audit trail logs every non-trivial choice with inputs, alternatives, and rationale.

Three steps no competitor ships

1. Usage-based datatype mapping (Step 3)

Every other tool maps Oracle types to PostgreSQL types using the declared definitions. NUMBER(38,10) becomes NUMERIC(38,10). VARCHAR2(4000) becomes VARCHAR(4000) or TEXT. The math is mechanical and the result fits — but it is wrong.

Oracle schemas in real-world production are systematically over-allocated. A NUMBER(38) column storing 0/1 booleans. A VARCHAR2(4000) column whose longest value is 14 characters. A NUMBER(p,s) column whose actual magnitude is < 10^9. The naive map preserves the over-allocation. The PostgreSQL target ends up with the same storage waste — and the same index-size penalty, the same query-plan cost, the same buffer-cache thrash.

DBMigrateAIPro runs a profiling query against every column before deciding the target type. MIN, MAX, MAX(LENGTH), distinct count, NULL ratio, sample magnitude. The result drives the mapping:

Observed pattern	Declared	Naive map	Smart map
100% integer, max < 32K	NUMBER	NUMERIC	SMALLINT
100% integer, max < 2B	NUMBER	NUMERIC	INTEGER
Cardinality < 256, strings	VARCHAR2(4000)	TEXT	Native enum or surrogate FK
pct_null > 85%, low cardinality	NUMBER	NUMERIC	Sparse + partial index
RAW(16) holding GUIDs	RAW(16)	BYTEA	UUID
Date column, time always 00:00	DATE	TIMESTAMP(0)	DATE

The customer sees the recommendation and overrides per-column from the UI before any DDL runs. The decision is documented in the decision audit trail (Step 14) with the profiling evidence that drove it. Storage savings of 30–60% on the target are typical.

2. Code translation with confidence scoring (Step 5)

Existing tools either translate a piece of code or refuse to. Failures are invisible: the engineer downloads the converted output, sees fifty TODO stubs, and starts manually rewriting.

DBMigrateAIPro returns a confidence score with every converted object. HIGH means mechanical translation, tested against the reference suite. MEDIUM means logically equivalent but worth a review — typically because of a built-in function with subtle semantic differences. LOW means probably correct but with an edge case the engineer should validate (e.g., NULL ordering, empty-string semantics, transaction-boundary effects).

The customer's review effort is then directed at the LOW and MEDIUM objects, not at every line of the converted output. For the rare construct with no clean PostgreSQL equivalent, the tool emits REDESIGN REQUIRED with a complete worked example of the recommended pattern — not just a flag.

3. Performance recommendations from observed workload (Step 9)

After cutover, every other tool's job is done. DBMigrateAIPro's begins a new phase. The Performance Recommender reads pg_stat_statements, pg_stat_user_tables, and pg_stat_user_indexes continuously, and emits structured recommendations grounded in actual query patterns:

- Insert-heavy tables — partition by date, switch to BRIN where sequential, drop unused indexes that cost write throughput.

- Update-heavy tables — lower fillfactor for HOT updates, tune autovacuum scale factor per-table, move updated columns out of indexes.
- Read-heavy tables — covering indexes for slow queries, materialised views for repeated aggregations, vertical splits for hot/cold access.
- Top-N queries — index suggestions tied to actual WHERE clauses, plan-shape comparisons against the source AWR baseline.

Every recommendation comes with the implementation SQL, the expected impact, the risk classification, and the measurement query that will confirm the improvement after applying it.

The 5% honesty contract

Every migration tool that claims 100% conversion is hiding the failures. We claim 95%, document the 5% explicitly, and provide working alternatives with sample code. This is the most important part of the methodology — and the one that earns the trust of senior DBAs the fastest.

Construct	Why it doesn't auto-convert	What we provide instead
Bitmap indexes	No PostgreSQL equivalent	BRIN for sequential low-cardinality data; B-tree otherwise — explained per index
Autonomous transactions	No PostgreSQL primitive	dblink with a separate connection — sample code in the deliverable
Pipelined functions	Different iteration semantics	RETURNS SETOF with explicit type — pattern documented
MODEL clause	Often unportable	Recursive CTE or window-function rewrite — case-by-case
DBMS_* package calls	Not all have PG equivalents	Mapped where possible; flagged + suggested package otherwise
External tables	Different file-access model	Foreign data wrapper alternative documented
BFILE columns	External file references	Migrate file content to BYTEA / lo, or external storage with reference table
Custom CONTEXT / FGAC	Manual	Security-policy approach explained per-case

> The 5% is not a failure. It is the contract. The customer gets 95% automated and 5% well-documented and quoted — every gap with a name, a reason, and a working alternative.

What customers see in practice

Aggregated outcomes from the beta cohort running production Oracle → PostgreSQL migrations on estates between 200 GB and 8 TB:

Metric	Typical outcome
Time to first dev migration	< 1 hour from install
End-to-end project duration	Hours to days for mid-size estates (vs 6–18 months hand-rolled)
PL/SQL auto-conversion rate	99%+ on common patterns; 85%+ on the hardest schemas
Storage savings from smart mapping	30–60% on the target vs naive 1:1 mapping
Validation accuracy	99.98% row-hash match (mismatches quarantined, never silently written)
Cutover downtime	Near-zero with shadow-apply and CDC drain
License cost reduction	60–80% annual database spend reduction vs prior Oracle contract
DBA time on operations	70%+ reduction post-cutover (no Oracle patching, less index maintenance)

How an engagement runs

Every engagement follows the same shape, regardless of estate size:

Day 1 — Pre-flight

Read-only connection to a development copy of the source. Steps 1, 2, and 3 run silently and produce a HTML assessment report: object inventory, complexity grades, risk register, and the smart-mapping delta versus naive mapping. No production load.

Day 2 — Review and approve

The customer's architecture review board reviews the assessment, approves or overrides the type map per column, and signs off on the migration plan. The tool emits the executable DDL (Step 4), the PL/pgSQL conversions with confidence scores (Step 5), and the validation suite (Step 8).

Day 3+ — Dry migration

The full pipeline runs against a non-production target. Validation (Step 8 + Step 13) confirms equivalence. Performance recommendations (Step 9) are tested against representative workload.

Cutover day

The pre-migration runbook (Step 10) executes. Bulk load + CDC drain. Validation suite runs to gate the cutover. On any failure the automatic rollback (Step 1's contract) restores the prior state.

Post-cutover (continuous)

The Performance Recommender runs forever, surfacing structured tuning advice as workload patterns evolve. The final report (Step 12) is delivered with the lessons-learned section. The 90-day rollback window keeps the source available read-only for emergencies.

Why this wins

Dimension	Existing tools	DBMigrateAIPro
Schema decisions	Heuristic — declared types only	Data-aware — observed values drive the map
Code conversion	Trivial procedures only; manual stubs	Full PL/SQL surface; confidence scores; side-by-side diffs
Validation	Row counts	Five tiers including row hashes (no data leaves source/target)
Post-cutover	Nothing	Continuous performance recommender
The 5% un-converted	Silent failure or // TODO	Named, explained, with worked alternative code
Methodology	Implicit, varies by engineer	Explicit, 14 steps, every artefact traceable

The market is over-served on data-movement tooling. It is dramatically under-served on migration intelligence. The 14-step methodology is the system that closes that gap — and the document that proves we have a plan, not a hope.

Ready to scope a real migration? Get in touch at hello@medaxai.com or start a free pre-flight assessment at medaxai.com/signup.