DBMigratePro Migration Series

Oracle → MySQL

Simpler, cheaper, web-ready — without losing the workload.

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

Executive summary

Not every Oracle workload needs to land on PostgreSQL. Teams running web applications, catalog databases, CMS backends, or operational OLTP systems often find MySQL a simpler, cheaper, and operationally lighter target than PostgreSQL — especially when the application stack is already MySQL-friendly (LAMP, WordPress, Magento, phpMyAdmin, legacy Java ORMs).

Oracle → MySQL is less common than Oracle → PostgreSQL but well defined: the schema conversion is straightforward, data types have close equivalents, and MySQL's stored-procedure surface is simpler than PostgreSQL's PL/pgSQL. The hard part is the long tail of Oracle-isms — packages, autonomous transactions, sequences, MERGE, hierarchical queries — that MySQL handles differently or not at all.

DBMigrateAIPro automates every phase and, critically, flags every item that can't be auto-translated so the team makes an informed decision. This paper explains what the tool does, where Oracle → MySQL gets tricky, and the outcomes to expect.

DBMigrateAIPro takes Oracle workloads that fit MySQL's sweet spot — operational OLTP, web applications, catalog and content systems — and moves them onto MySQL with full automation and validation.

Why migrate from Oracle to MySQL?

Three reasons dominate Oracle → MySQL projects.

Cost, without the PostgreSQL complexity

MySQL (or MariaDB) is often the cheapest managed database offering on every major cloud. For workloads that don't need PostgreSQL's advanced features, MySQL is simpler to operate, easier to staff, and cheaper to run — a 60-80% licence cost reduction is typical.

Web-application fit

Most web frameworks, CMS platforms, and e-commerce stacks started on MySQL. If the Oracle database sits behind a web application with a MySQL-native ecosystem around it, moving back to MySQL removes friction and opens up a much larger pool of community tooling.

Operational simplicity

MySQL's operational model — replication, backup, failover, monitoring — is well-understood, widely documented, and supported by every cloud provider. Teams that want a lighter database to operate than PostgreSQL (or Oracle) gravitate to MySQL for exactly this reason.

What makes this migration hard

MySQL is capable but less feature-rich than PostgreSQL. The migration tool has to handle the feature gap cleanly and flag workloads that would be better off on PostgreSQL instead.

PL/SQL → MySQL procedures

MySQL's stored-procedure surface is smaller than PL/pgSQL. There are no packages, no autonomous transactions, no FORALL, no BULK COLLECT. DBMigrateAIPro flattens Oracle packages into schema-qualified MySQL procedures, rewrites built-ins, and flags any construct that has no equivalent — autonomous transactions especially — for human review.

Sequences vs AUTO_INCREMENT

Oracle sequences have no direct MySQL equivalent. DBMigrateAIPro maps IDENTITY-style usage to AUTO_INCREMENT and rewrites explicit .NEXTVAL / .CURRVAL references. Non-identity sequence usage (distributed IDs, cross-table ID pools) is flagged — these cases usually need a small application change.

Hierarchical queries (CONNECT BY)

MySQL 8.0+ supports recursive CTEs, so CONNECT BY PRIOR translates cleanly to WITH RECURSIVE. For MySQL 5.7 and earlier, the tool flags the query for rewrite — an application-level fix is usually required.

Data type ceilings

Oracle's NUMBER has no direct MySQL equivalent. DECIMAL matches precision but not performance; BIGINT covers most integer cases. CLOB maps to LONGTEXT, BLOB to LONGBLOB. DBMigrateAIPro ships with default mappings for 50+ Oracle types and per-column overrides.

MERGE and complex SQL

MySQL has no native MERGE; the tool rewrites MERGE statements as INSERT … ON DUPLICATE KEY UPDATE when semantics allow, or flags them for application-level handling when they don't.

How DBMigratePro handles it

Every Oracle → MySQL 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.

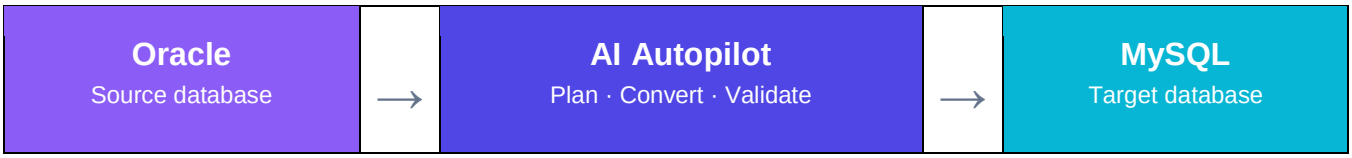

Figure 1 — End-to-end flow. AI Autopilot orchestrates all phases.

Six autonomous phases

1. Pre-flight

Reads the Oracle catalog, counts rows and objects, flags incompatibilities (packages, autonomous transactions, MERGE patterns), and produces a full assessment.

2. Schema conversion

Translates tables, constraints, indexes, views, and sequences to MySQL DDL. Identity columns map to AUTO_INCREMENT; character-set UTF8MB4 by default.

3. Code object transpilation

PL/SQL procedures and functions are parsed and rewritten to MySQL stored procedures. Packages are flattened into schema-qualified procedures.

4. Data streaming

Rows stream from Oracle to MySQL 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 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 Oracle type. You can override any mapping per column from the UI — the table below shows the defaults.

Oracle type	MySQL type	Notes
VARCHAR2(n)	VARCHAR(n)	Same semantics; UTF8MB4 character set by default.
NUMBER	DECIMAL(65,30)	Full-precision NUMBER; override to BIGINT/DECIMAL(p,s) where possible.
NUMBER(p)	BIGINT / INT	Integer-only cases collapse to BIGINT or INT.
NUMBER(p,s)	DECIMAL(p,s)	Exact mapping.
DATE	DATETIME	Oracle DATE includes time; DATETIME preserves that.
TIMESTAMP	DATETIME(n)	Precision preserved.
TIMESTAMP WITH TZ	TIMESTAMP	MySQL TIMESTAMP stores UTC; semantics mostly preserved.
CLOB / NCLOB	LONGTEXT	LONGTEXT holds up to 4 GB; close enough for most workloads.
BLOB	LONGBLOB	Binary data maps cleanly.
RAW(n)	VARBINARY(n)	Length constraint preserved.
ROWID	surrogate PK	Application code referencing ROWID must be rewritten.
XMLTYPE	LONGTEXT + app parsing	MySQL has no native XML type; stored as text with application-level parsing.
BOOLEAN (PL/SQL)	TINYINT(1)	MySQL convention for BOOLEAN.

Code object conversion

DBMigrateAIPro transpiles the PL/SQL surface into MySQL stored procedures. Converted objects are saved side-by-side with originals:

- Procedures and functions — IN/OUT/IN-OUT parameters preserved.
- Packages — flattened into schema-qualified MySQL procedures; package-level state becomes session-scoped variables or tables.
- Triggers — BEFORE/AFTER INSERT/UPDATE/DELETE supported; :NEW / :OLD rewritten.
- Views — DDL translated; materialized views are simulated with tables + scheduled refresh where needed.
- Sequences — mapped to AUTO_INCREMENT where usage matches; flagged for review otherwise.
- Built-in function calls — NVL, DECODE, SYSDATE, TO_DATE, TO_CHAR, TRUNC and 60+ others rewritten to MySQL equivalents.
- Hierarchical queries — CONNECT BY PRIOR rewritten as WITH RECURSIVE (MySQL 8.0+); flagged for 5.7 targets.
- MERGE statements — translated to INSERT … ON DUPLICATE KEY UPDATE where semantics allow.

Validation, safety, rollback

DBMigrateAIPro validates every row it copies — using native hash functions on both sides so the data never leaves either server.

For each table, Autopilot computes an MD5 over a stable canonical row representation on Oracle and on MySQL, 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, per-mismatch detail — is a single artefact you can attach to any cutover sign-off.

Safety guarantees

- Pre-flight assessment is read-only — production Oracle is never touched.
- The target is written only inside the project's database; existing MySQL databases are never modified.
- 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 Oracle → MySQL migrations on estates between 50 GB and 2 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. 3–9 months hand-rolled.
PL/SQL auto-conversion rate	95%+ of procedures transpile without manual edits; remaining 5% (autonomous transactions, MERGE edge cases) 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	60–80% typical annual database spend reduction vs. the prior Oracle 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 Oracle database — read-only.
- Review the generated workspace: converted DDL, transpiled procedures, flagged items (packages, autonomous tx, MERGE).
- Run a dry migration to a test MySQL target; review the row-hash validation report.
- Book a 30-minute migration review with our team (free) to triage flagged items before cutover.
- Schedule the cutover. Shadow-apply handles continuously-updated tables with zero downtime.

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