Database Migration Planning Guide

End-to-end guide for planning and executing a database migration — assessment, schema mapping, testing, cutover, and post-migration optimisation with expert recommendations for UK businesses.

5	**30+**	**Templates**	**Free**
PLANNING PHASES	EXPERT TIPS	INCLUDED THROUGHOUT	PRINT & USE NO STRINGS

About This Guide

This guide provides practical, actionable advice for UK businesses. Work through each section to build a comprehensive understanding of the topic. Use the information to make informed decisions and implement best practices.

Need Help With Your IT?

Our team can help you implement the recommendations in this resource.

info@cloudswitched.com
+44 2030 043 450
[redacted]

1 Migration Assessment & Strategy

Before touching any data, thoroughly assess why you're migrating, what risks exist, and which approach fits your requirements.

Every database migration begins with a clear understanding of the **business drivers, technical constraints, and risk tolerance**. Whether you're moving from on-premises to the cloud, switching database engines, or consolidating multiple databases, the assessment phase determines whether the project succeeds or becomes a costly failure.

Common Migration Drivers

- **Cost reduction:** Moving from expensive on-premises licensing (Oracle, SQL Server Enterprise) to open-source alternatives (PostgreSQL, MariaDB) or managed cloud services can significantly reduce total cost of ownership.
- **Scalability requirements:** On-premises databases hitting hardware limits may benefit from cloud-native solutions offering elastic scaling without capital expenditure.
- **End of life or support:** Legacy database versions reaching end of extended support create security and compliance risks that necessitate migration.
- **Compliance and data sovereignty:** UK GDPR requirements may drive migrations to UK-based data centres or cloud regions to ensure data residency compliance.
- **Modernisation:** Moving to managed database services (AWS RDS, Azure SQL, Google Cloud SQL) eliminates patching, backup management, and infrastructure overhead.

Migration Approaches

APPROACH	DESCRIPTION	BEST FOR	RISK LEVEL
Lift and shift	Move database as-is to new infrastructure	Same-engine platform changes	Low
Re-platform	Minor adjustments for new platform compatibility	Cloud-managed service adoption	Medium
Re-engine	Convert to a different database engine entirely	Cost reduction, modernisation	High
Phased migration	Migrate schemas/tables incrementally over time	Large databases, zero-downtime needs	Medium
Parallel run	Run old and new simultaneously during validation	Mission-critical systems	Low (but costly)

Start with a Proof of Concept

Before committing to a full migration, run a proof of concept with a representative subset of your data and application queries. This reveals compatibility issues, performance differences, and hidden complexity early when it is cheap to address.

Document your migration strategy in a **formal project charter** that includes scope, objectives, success criteria, timeline, resource requirements, and a clear rollback plan. Secure stakeholder sign-off before proceeding to the mapping phase.

2 Schema & Data Mapping

Mapping source schemas, data types, and application dependencies to the target platform is the most detail-intensive phase of any migration.

Schema mapping requires a **comprehensive inventory of every database object** — tables, views, stored procedures, functions, triggers, indexes, constraints, and scheduled jobs. Missing even one dependency can cause application failures after cutover.

Object Inventory Checklist

- **Tables and columns:** Document every table with column names, data types, nullability, defaults, and computed columns. Note any proprietary data types that require conversion.
- **Relationships and constraints:** Map all primary keys, foreign keys, unique constraints, and check constraints. Verify referential integrity rules are preserved in the target schema.
- **Indexes:** Catalogue all indexes including type (clustered, non-clustered, unique, filtered), included columns, and fill factor settings. Index strategy may need adjustment for the target engine.
- **Stored procedures and functions:** Identify all procedural code that may use engine-specific syntax (T-SQL, PL/SQL, PL/pgSQL). These often require significant rewriting during re-engine migrations.
- **Triggers and scheduled jobs:** Document all database-level automation including triggers, agent jobs, and maintenance plans that must be replicated or replaced on the target platform.

Data Type Mapping

SOURCE TYPE	TARGET CONSIDERATION	COMMON ISSUE
DATETIME / DATETIME2	Precision and timezone handling differ between engines	Loss of sub-second precision
VARCHAR / NVARCHAR	Character set and collation must match	Encoding mismatches, sort order changes
DECIMAL / NUMERIC	Scale and precision limits vary by platform	Rounding errors in financial data
BIT / BOOLEAN	Mapping between integer-based and true boolean types	Application logic compatibility
BLOB / VARBINARY	Maximum size limits and storage mechanisms differ	Large object migration performance
Identity / Serial	Auto-increment implementation varies significantly	Sequence gaps, reseeding requirements

Collation Differences Can Break Queries

Different database engines use different default collations. A query that works correctly with case-insensitive collation on SQL Server may return different results on PostgreSQL with case-sensitive defaults. Test all application queries against the target collation before cutover.

Create a detailed **mapping document** that pairs every source object with its target equivalent, noting any transformations, manual interventions, or code changes required. This document becomes the single source of truth for the migration team.

3 Testing & Validation Strategy

Rigorous testing is the difference between a smooth migration and a business-disrupting disaster. Never shortcut this phase.

Testing must validate three things: **data integrity** (every row migrated correctly), **application compatibility** (every feature works as before), and **performance parity** (response times meet or exceed the source system). Each requires a distinct testing approach.

Data Validation

- **Row count verification:** Compare row counts for every table between source and target. Any discrepancy must be investigated and resolved before proceeding.
- **Checksum comparison:** Generate checksums or hash values for critical columns to verify data integrity beyond simple row counts.
- **Boundary value testing:** Verify that minimum, maximum, and edge-case values (nulls, empty strings, Unicode characters, maximum-length fields) are preserved correctly.
- **Referential integrity validation:** Confirm all foreign key relationships are intact and no orphaned records exist in the target database.

Application Testing

- **Regression testing:** Execute the full application test suite against the target database. Every feature that worked before must continue to work identically.
- **User acceptance testing (UAT):** Key business users test their daily workflows and critical processes end-to-end on the migrated database.
- **Integration testing:** Verify all integrations, APIs, ETL processes, and reporting tools function correctly with the target database.

Performance Testing

- **Baseline comparison:** Run the same workload against source and target, comparing query response times, throughput, and resource utilisation.
- **Load testing:** Simulate peak concurrent user loads to verify the target database handles production-level traffic without degradation.
- **Stress testing:** Push beyond expected peak loads to identify the breaking point and ensure adequate headroom for growth.

Automate Everything

Manual testing is slow, error-prone, and unrepeatable. Invest in automated validation scripts that can be run repeatedly as you refine the migration process. You will likely perform the migration multiple times in rehearsal before the production cutover.

Never Skip UAT

Technical testing catches data and compatibility issues, but only business users can verify that the system **behaves correctly** from an operational perspective. Allocate at least two weeks for UAT with real business data and real users.

4 Cutover Planning & Execution

The cutover window is the highest-risk phase. Meticulous planning and rehearsal are essential for a smooth transition.

A successful cutover requires a **detailed runbook with every step documented, timed, and assigned to a named individual**. Rehearse the entire cutover at least twice in a staging environment before attempting it in production. Measure the actual time for each step and build contingency into your schedule.

Pre-Cutover Preparation

- **Communication plan:** Notify all stakeholders, users, and dependent systems of the migration window. Provide clear timelines and fallback procedures.
- **Freeze period:** Implement a code freeze and schema freeze at least one week before cutover to prevent last-minute changes introducing variables.
- **Final backup:** Take a verified, tested backup of the source database immediately before beginning the cutover. This is your safety net.
- **Connection string preparation:** Pre-configure application connection strings for the target database, ready to switch. Use environment variables or configuration management, not hardcoded strings.

Cutover Execution Steps

- **Step 1:** Place the source database into read-only mode or stop application writes to prevent data changes during migration.
- **Step 2:** Execute the final incremental data sync to bring the target database fully up to date with the source.
- **Step 3:** Run automated validation scripts to verify data integrity, row counts, and checksums between source and target.
- **Step 4:** Switch application connection strings to the target database and restart application services.
- **Step 5:** Execute smoke tests — a predefined set of critical transactions that verify core functionality is working.
- **Step 6:** Monitor application logs, error rates, and database performance metrics intensively for the first 2–4 hours after cutover.

Rollback Plan

Every cutover plan must include a **clearly defined rollback point** — the latest moment at which you can revert to the source database without data loss. Once you pass this point, you are committed. Typical rollback triggers include:

- **Data integrity failures:** Validation scripts detect missing or corrupted data that cannot be resolved within the cutover window.
- **Application errors:** Critical application features fail and cannot be fixed within the allocated maintenance window.
- **Performance degradation:** Response times exceed acceptable thresholds and tuning within the window is not feasible.

Schedule Cutover Outside Business Hours

For UK businesses, Friday evening or Saturday morning cutover windows are ideal — they provide the entire weekend for stabilisation before Monday morning users arrive. Never schedule a cutover on a Thursday or Friday afternoon with no buffer time.

5 Post-Migration Optimisation

The migration is not complete when the cutover finishes. Post-migration tuning and monitoring are critical for long-term success.

A newly migrated database will almost certainly require **performance tuning, index adjustments, and configuration refinement** in the days and weeks following cutover. Plan for a dedicated optimisation period and do not disband the migration team prematurely.

Immediate Post-Migration Tasks (Week 1)

- **Update statistics:** Rebuild all database statistics on the target to ensure the query optimiser has accurate information for the new data distribution.
- **Monitor query performance:** Enable slow query logging and compare execution times against pre-migration baselines. Investigate any queries performing worse than before.
- **Review index usage:** The target engine’s optimiser may use indexes differently. Monitor index usage patterns and adjust the indexing strategy based on actual workload behaviour.
- **Check error logs:** Review database and application error logs daily for the first week, looking for warnings, deprecated feature usage, and connection issues.

Short-Term Optimisation (Weeks 2–4)

- **Tune memory configuration:** Adjust buffer pool size, work memory, and cache settings based on observed workload patterns on the target platform.
- **Optimise problematic queries:** Rewrite any queries that performed well on the source engine but struggle on the target due to optimiser differences.
- **Configure maintenance jobs:** Set up automated index maintenance, statistics updates, integrity checks, and log management on the target platform.
- **Decommission source:** Once you are confident the migration is stable (minimum 2 weeks with no rollback), decommission the source database — but retain a final backup for at least 90 days.

Long-Term Considerations

- **Knowledge transfer:** Ensure the operations team is fully trained on the target platform’s administration, monitoring, and troubleshooting procedures.
- **Documentation update:** Update all runbooks, disaster recovery procedures, and architecture diagrams to reflect the new database platform.
- **Lessons learned:** Conduct a formal retrospective capturing what went well, what could be improved, and recommendations for future migration projects.

Keep the Source Backup

Retain a verified backup of the source database for at least 90 days after decommissioning. Edge cases and rarely used features may surface weeks later, and having the source available for comparison is invaluable for troubleshooting.