Interactive®

Database Migration Checklist

This checklist is designed for:
DBAs, cloud engineers and architects responsible for planning and executing database migrations.

Database migration is one of the most technically demanding forms of data migration. Unlike file or application moves, database migrations must preserve the integrity of the entire data ecosystem: schema, business logic, performance characteristics and application behaviour. In practice, that means tables, views, stored procedures, triggers, functions and constraints must migrate together without breaking dependent systems.

This database migration checklist addresses the platform-specific challenges involved in moving databases such as SQL Server, Oracle, PostgreSQL and MySQL. It covers common scenarios including platform modernisation (such as Oracle to PostgreSQL), major version upgrades and migrations to managed cloud services like Amazon RDS and Azure SQL.

Schema migration is central to success. Migrating data alone is not enough if database logic and performance characteristics are lost in the process.

This guide provides a structured, practical approach, supported by Interactive's experience delivering database migrations for Australian organisations.

What Makes Database Migration Different

Database migration introduces complexity that does not exist in other migration types. Schema objects such as tables, indexes, views, stored procedures, triggers and functions must all migrate accurately. This demands careful analysis and consideration, as data types rarely map one-to-one between platforms.

Applications rely on very specific database behaviour. How transactions are handled, how data is locked, how isolation levels are enforced and how queries are executed all directly affect how systems perform in the real world.

In many environments, stored procedures and triggers also contain critical business logic. When databases move between platforms, this logic often needs to be rewritten to suit the target service.

For mission-critical systems, zero-downtime requirements add another layer of complexity. Throughout migration and cutover, referential integrity must be preserved to avoid data inconsistencies. These are the kinds of challenges that require a deliberate, database-specific migration approach, which this checklist addresses in detail.

Most importantly, all of this must be achieved while improving (or at the very least, maintaining) database performance.

Step-by-Step Database Migration Process

Database migrations follow a structured workflow from assessment through to optimisation.

Phase 1: Assessment and Planning (20%)

Complete a pre-migration assessment, develop the migration plan and identify key risks and dependencies.

Phase 2: Schema Migration and Testing (30%)

Convert database schema, migrate objects and validate changes with application teams.

Phase 3: Data Migration Testing (30%)

Migrate test data and execute data validation, integrity checks and performance testing.

Phase 4: Production Migration (10%)

Execute production cutover, validate reconciliation and confirm application functionality.

Phase 5: Post-Migration Optimisation (10%)

Tune database performance, implement monitoring and document outcomes.

Your Complete Database Migration Checklist

Database migrations are complex and high-risk if not carefully managed.

This checklist provides a practical framework to help your organisation plan effectively, control risk and maintain performance throughout your database migration.

Pre-Migration Assessment

A thorough pre-migration assessment exposes key complexities, dependencies and migration risks. Done well, it enables informed decisions on the migration approach. Done poorly, it can derail the entire migration.

Tick off these items before selecting a migration approach.

Document the full database structure: Capture tables, views, indexes, constraints and relationships.

Inventory all database objects: Identify stored procedures, triggers, functions, sequences and other executable logic.

Map dependencies across the environment: Document all applications, reports, integrations and scheduled jobs that rely on the database.

Assess data volumes and growth patterns: Analyse table sizes, growth rates and any existing partitioning strategies.

Baseline current performance: Measure query latency, throughput and resource usage to establish a pre-migration benchmark.

Identify database-specific features: Flag features, extensions or behaviours that may require refactoring or special handling on the target platform.

Review database security controls: Document users, roles, permissions and any row-level security implementations.

Document backup and recovery requirements: Confirm existing backup processes and current RPO and RTO targets.

Assess embedded business logic complexity: Review stored procedures and triggers to understand refactoring effort and risk.

Identify regulatory and compliance requirements: Confirm obligations under the Australian Privacy Principles, APRA and other applicable frameworks.

Estimate migration effort and duration: Develop a realistic timeline based on database size, complexity and downtime constraints.

Define success criteria upfront: Agree measurable outcomes for performance, data integrity and application behaviour post-migration.

Schema Migration Planning

Schema migration is often the most complex part of a database migration. It requires detailed mapping between the source and target platforms.

Map data types between platforms: Define source-to-target data type mappings, such as VARCHAR to TEXT and NUMBER to NUMERIC.

Plan stored procedure conversion: Rewrite database-specific procedural logic to align with the target platform.

Convert triggers and functions: Adapt trigger definitions and function logic to the target database syntax and capabilities.

Map indexes and constraints: Recreate indexes, primary keys, foreign keys and unique constraints in the correct order.

Plan view migration: Convert views to the target database SQL dialect and execution model.

Handle sequences and identity columns: Map sequence behaviour and auto-increment mechanisms to the target platform.

Convert partitioning strategies: Redesign partitioning - where it's used in the source system - to suit the target database.

Plan for unsupported features: Identify source features not available on the target platform and design appropriate workarounds.

Document all schema transformations: Maintain clear mapping documentation to support validation, audit and rollback.

Review changes with application teams: Confirm schema changes do not break application logic or assumptions.

Plan performance optimisation: Define indexing strategies and query optimisation approaches for the target platform.

Design rollback procedures: Document how schema changes will be reversed if migration issues occur.

MySQL Migration

MySQL migrations are common in modernisation programs, particularly when moving to cloud-native platforms or when migrating from MySQL to PostgreSQL.

These are the common considerations for MySQL migrations, which are frequently encountered in database modernisation projects.

For MySQL to PostgreSQL migration:

Convert MySQL-specific data types: Address types such as TINYINT, MEDIUMINT and ENUM that do not map directly to PostgreSQL.

Rewrite stored procedures: Convert procedural logic to PL/pgSQL.

Convert AUTO_INCREMENT columns: Replace AUTO_INCREMENT with SERIAL or IDENTITY columns as appropriate.

Adapt trigger syntax and behaviour: Adjust trigger definitions and execution timing to match PostgreSQL semantics.

Rewrite MySQL functions: Refactor functions to ensure PostgreSQL compatibility.

For all MySQL migrations:

Export data correctly: Use mysqldump or MySQL Workbench with options that preserve data integrity and consistency.

Ensure character set consistency: Confirm UTF-8 encoding across source and target environments.

Review storage engine assumptions: Remove or refactor logic tied to MySQL-specific storage engines.

Test replication-based cutover: Validate replication workflows if used to support zero-downtime migrations.

Validate MySQL-specific features: Confirm behaviour of JSON fields, full-text search and geospatial data on the target platform.

Australian considerations:

Validate timezone handling: Confirm correct behaviour across AEST and AEDT, including daylight saving transitions.

Plan local cutover support: Ensure DBA availability during Australian business hours for migration and cutover activities.

Testing and Validation:

Thorough testing ensures the migration doesn't disrupt data, system behaviour or performance.

Validate schema completeness: Confirm all tables, views, indexes, constraints and database objects are present and correctly defined.

Reconcile data accuracy: Perform record counts and checksum validation to confirm source and target data consistency.

Validate referential integrity: Confirm all relationships and constraints behave as expected post-migration.

Test stored procedures and logic: Execute stored procedures and validate outputs against known results.

Test triggers and functions under load: Validate trigger behaviour and function execution under realistic workloads.

Validate application connectivity and behaviour: Confirm applications connect successfully and behave consistently against the migrated database.

Execute performance testing: Compare query latency, throughput and resource usage against pre-migration baseline metrics.

Test backup and restore processes: Validate backup execution and recovery procedures on the target platform.

Validate security controls: Confirm users, roles, permissions and access controls are enforced correctly.

Test integrations: Validate ETL pipelines, reporting tools, downstream systems and APIs.

Execute disaster recovery testing: Confirm failover and recovery processes meet defined RPO and RTO targets.

Conduct user acceptance testing: Validate database-dependent workflows with business and application stakeholders.

Perform volume and concurrency testing: Confirm the database behaves correctly under expected peak loads and concurrent access patterns.

Zero-Downtime Migration:

Zero-downtime database migrations rely on replication and tightly controlled cutover processes to minimise disruption to production systems.

Configure replication between source and target databases: Establish continuous replication to keep data aligned during the migration window.

Monitor replication lag continuously: Track lag in real time to ensure the target remains close to the source before cutover.

Plan a minimal cutover window: Schedule cutover activities during low-traffic periods to reduce operational risk.

Design a controlled read-only window where required: Temporarily restrict writes, if necessary, to allow final synchronisation.

Define rollback criteria and procedures: Document clear conditions and steps for reverting to the source system if issues arise.

Configure application failover mechanisms: Prepare connection switching, DNS updates or configuration changes required at cutover.

Test cutover processes in non-production: Rehearse the full cutover sequence to validate timing, roles and recovery steps.

Monitor cutover execution in real time: Actively monitor database health, replication status and application behaviour during cutover.

Validate zero data loss before final cutover: Confirm all data changes have been replicated and verified before committing to the new platform.

Post-Migration Optimisation:

The job isn't finished at go-live. Post-migration optimisation focuses on confirming the migration achieved its intended outcomes and optimising performance in the new environment.

Refresh database statistics: Update table and index statistics so the query optimiser has accurate information.

Rebuild or reorganise indexes: Optimise index structures to suit the target platform and post-migration data layout.

Tune database configuration: Adjust memory allocation, connection pooling and caching settings based on workload.

Optimise slow queries: Identify poorly performing queries and refactor them for the target database engine.

Monitor performance metrics: Compare post-migration performance against baseline metrics and investigate any degradation.

Implement database monitoring: Configure alerts for performance issues, capacity thresholds and resource constraints.

Review execution plans: Analyse query execution plans and add or refine indexes where required.

Schedule ongoing maintenance tasks: Establish routines for backups, index maintenance and statistics refreshes.

Related Data Migration Resources

- **Data Migration Framework:**
  Strategic guidance for complex programs.
- **Data Migration Checklist Library:**
  Coverage of multiple migration types.
- **Cloud Data Migration Checklist:**
  For cloud-hosted databases.
- **ERP Data Migration Checklist:**
  For ERP database backends.

Planning a database migration?

Interactive's data migration services support Australian enterprises with certified database migration professionals. We deliver proven methodologies and 24/7 local support across Australia.

CONTACT US