How do you change a database schema without downtime?

20 November 2026 · 5 min read · blogs.iictp.org/zero-downtime-database-migrations

Split every breaking schema change into small steps that each work with both the old and the new version of the code. This is often called expand and contract, or parallel change: first add the new structure, then move code and data over, and only remove the old structure once nothing uses it.

TL;DR The short version

- Change a schema without downtime by splitting each breaking change into steps that work with both old and new code: expand and contract.
- A simple rename breaks because old and new application versions run side by side during deploys and rollbacks.
- Backfill in small batches, watch for operations that lock or rewrite tables, and version migrations separately from application deploys.

Read online, listen to it, or change the text size: https://blogs.iictp.org/zero-downtime-database-migrations

Why does a simple rename break things?

During a deployment, old and new versions of the application run at the same time, and a rollback brings the old version back. If you rename a column in one step, whichever version expects the other name fails. Every schema change has to be compatible with the code on both sides of it.

What are the steps of expand and contract?

Expand: add the new column or table, without removing anything. Migrate: deploy code that writes to both old and new, backfill existing rows, then switch reads to the new structure. Contract: once no running code uses the old structure, remove it in a later release. Each step can be deployed and rolled back on its own.

How do you backfill a large table safely?

In small batches, with a pause between them, so the database is not locked or overloaded and replicas keep up. Make the backfill restartable and idempotent, so it can stop and resume without duplicating work, and watch database load while it runs.

Which operations are risky even when compatible?

Some changes lock a table or rewrite it in full, depending on the database and version: adding a column with certain defaults, changing a column type, or building an index without the database's concurrent option. Check your database's documentation and test the migration on a copy with production-sized data.

How do you keep this manageable?

Run migrations through a tool that versions them alongside the code, separate schema migrations from application deploys, and keep each migration small. A checklist in code review, asking whether the change works with the previous release, catches most problems before they ship.

> Related credential: Certified Technical Practitioner. Explore the exam facts and syllabus.

Sources and further reading

1. Ambler, S. and Sadalage, P. Refactoring Databases: Evolutionary Database Design. Addison-Wesley, 2006.
2. Sato, D. ParallelChange. martinfowler.com, 2014.

Published by IICTP, an independent technical credential body, on 20 November 2026. The current version of this post is at https://blogs.iictp.org/zero-downtime-database-migrations. More guides: blogs.iictp.org.