

DEVELOPMENT · 2026-09

When two workers create the database at once

A three-line September CodeSeal fix shows where PostgreSQL advisory locks can help with coordinated startup work.



By Edwin Rodriguez

Published Oct 14, 2026 · 4 min read

September 2026 retrospective · Story context 2026-09-30



Startup code often looks harmless because it runs before the interesting work begins. Create a table if it doesn't exist. Add an index. Check a small piece of schema. Then accept requests.

Now start several workers together.

CodeSeal's September 30 correction added a transaction-scoped advisory lock around PostgreSQL schema initialization. The change addressed a narrow coordination problem: multiple workers could reach the same catalog setup at the same time. It's a small patch with a useful question behind it: who gets to do the setup while everybody else is starting?

| “If missing” still needs coordination

A conditional creation statement describes what should happen if an object already exists. That doesn't automatically make an entire multi-statement initialization sequence safe under every concurrent interleaving.

Imagine two workers checking related objects, creating tables, and preparing indexes. The second worker can enter while the first is partway through the sequence. The application needs a clear answer to who owns that setup work and when another worker may proceed.

One option is a separate migration step before normal workers start. Another is a carefully bounded coordination mechanism around startup initialization. The right choice depends on the application and deployment process.

| What an advisory lock contributes

[PostgreSQL offers application-defined advisory locks](#). The database doesn't infer the meaning of the lock key; participating code agrees on that meaning.

The September patch used a transaction-scoped lock before executing the schema setup. At that scope, the lock is released when the transaction ends. The intended sequence is straightforward:

```
BEGIN;
SELECT pg_advisory_xact_lock(4242, 1);
-- Perform the agreed, bounded schema initialization here.
COMMIT;
```

The numbers above are illustrative, not the application's key. Choose a deliberate key convention for your system. Every competing initializer must participate in the same convention for the lock to coordinate them.

Check your connection wrapper too. If it commits between the lock and the protected statements, your assumed protected interval may be different from the actual transaction. Transaction lifetime is part of the correctness argument.

| Keep the protected work small

A startup lock can become a bottleneck when the protected section grows without review. Avoid slipping downloads, long scans, or unrelated work into the same interval.

Make the initialization repeatable. A worker arriving after the first one commits should find a valid schema and finish quickly. A worker that fails should leave an understandable error and a recoverable transaction state.

Consider a bounded lock wait or a deployment timeout appropriate to your service. Otherwise, a new instance can wait quietly while operators see only slow startup. Log enough to distinguish waiting for coordination from executing schema changes, without exposing credentials or user data.

| Exercise the race you intended to fix

A useful test starts multiple separate connections against a disposable database and triggers initialization concurrently. Check that every successful participant ends with the expected schema.

Then fail one initializer during the protected work. Confirm the transaction rolls back and another participant can proceed. Test the actual connection path used by the application, including wrappers and automatic commits.

A unit test that merely asserts the SQL string appears in the source won't establish the behavior. You need to see separate connections compete for the same setup work. Keep the fixture isolated; schema race tests should never target a production database.

| Decide whether startup owns migrations

This pattern can be appropriate for a narrow existing initialization path. As a service grows, a dedicated migration process may make deployment responsibilities clearer. That process still needs coordination, error handling, and a rollback strategy.

Document the lock key near the initializer and the migration procedure. That gives a future maintainer a way to recognize another code path that must join the same coordination scheme before it touches the protected schema.

Before adding a lock, sketch the setup sequence and circle the part that only one worker should enter. Check where the transaction starts and ends. Once those pieces are clear, a small lock can do a useful job without becoming a mystery the next maintainer has to untangle.

| Sources & further reading

Checked 2026-10-14 · Story context 2026-09-30

1. [PostgreSQL explicit locking](#) (2026-10-14)
2. [CodeSeal](#) (2026-10-14)

[Read the current article online →](#)

Support the workshop

Explore software and learning offers at [StackSocial](#). Affiliate link: ProgreTech may earn a commission from purchases. Check current price, license and renewal terms before buying.

Find our projects at [progretech.com](#).

ProgreTech LLC · Build / Learn / Improve / Together

This file contains its text, illustrations and fonts. No account, scripts or network connection required to read. External reference links need internet access.