Manan Gupta

Founding Engineer - Multigres

@GuptaManan100

supabase



Pooling the Unpoolable

Architecture, Connection Pooling
Features, and more.

supabase

Manan Gupta // manan@supabase.io

August 2026

What is Multigres?

Multigres is Vitess for PostgreSQL.



Open Source



Sharding



High Availability



Backups And Restore

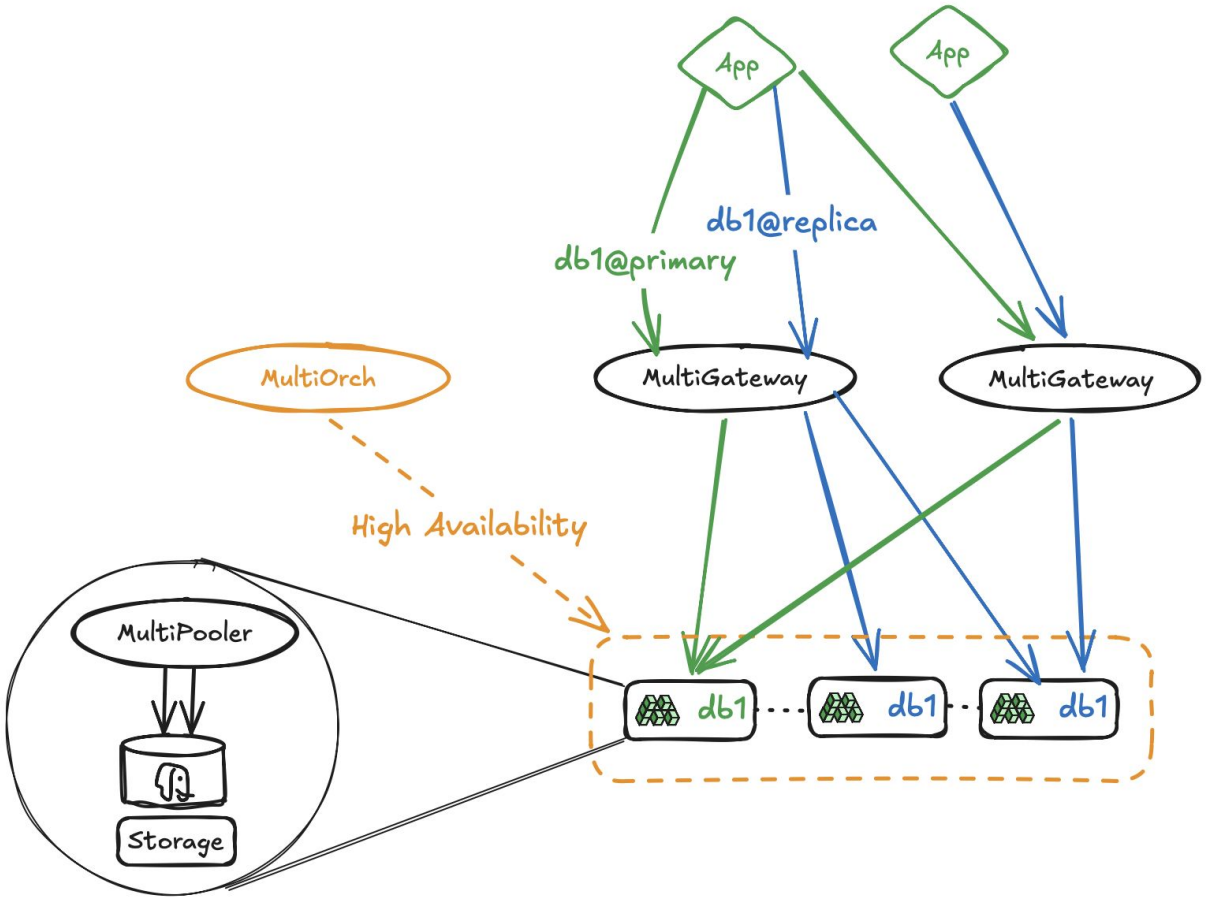


Connection Pooling



Query Analytics

The Architecture



Comparison to PgBouncer

	PgBouncer Statement	PgBouncer Transaction	PgBouncer Session	Multigres
Multi-statement transactions	✗	✓	✓	✓
Temp tables, Advisory locks	✗	✗	✓	✓
LISTEN NOTIFY	✗	✗	✓	✓
Suspended portals / cursors	✗	✗	✓	✓
SET / session GUCs	✗	✗	✓	✓
Prepared statements in simple protocol	✗	✗	✓	✓
Connection reuse	High	High	Low	**High***

** high when no reservation reasons are set; moves to reserved only when needed.*

We don’t make you pick

1 Parse every query *fully*

We know exactly what each statement is doing - BEGIN, CREATE TEMP TABLE, COPY, anything.

2 Track session state *explicitly*

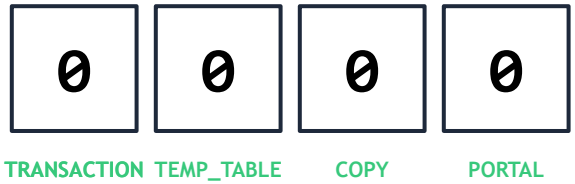
A bitmask per connection. Each bit tracks one reason the connection has to stay reserved.

The pool knows exactly when it's safe to recycle a connection, and when it isn't.

The Reservation reasons

`TRANSACTION`	Set on BEGIN. Cleared on COMMIT or ROLLBACK.
`TEMP_TABLE`	Set when a temp table is created. Cleared on DISCARD TEMP or disconnect.
`COPY`	Set when COPY IN/OUT starts. Cleared when streaming completes.
`PORTAL`	Set when a named portal is suspended. Cleared on full execute or close.

A bitmask per connection



All bits clear → regular pool.

Any bit set → reserved.

Composable - Multiple reasons can hold at once.

One client's session

`BEGIN`	0 0 0 **1**	Reserved	*Transaction starts.*
`CREATE TEMP TABLE t (...)`	0 0 **1** **1**	Reserved	*Two reasons now.*
`INSERT INTO t VALUES (...)`	0 0 **1** **1**	Reserved	*No change.*
`COMMIT`	0 0 **1** 0	Reserved	***TRANSACTION clears. TEMP_TABLE remains.***
`DISCARD TEMP`	0 0 0 0	**Regular**	*All clear. Back to pool.*

Step 4: PgBouncer transaction mode would silently drop the temp table here. Multigres doesn't.

Three pool tiers

Regular

no reasons set

Recyclable. Most queries served from here.

Reserved

≥ 1 reason set

Pinned to one client. Released when all clear.

Admin

always available

Small, separate. Management Ops.

LISTEN/NOTIFY breaks in transaction pooling

Transaction pooling can’t support LISTEN NOTIFY easily

1. **LISTEN foo** on backend #7
2. **Txn ends** pooler returns backend #7
3. **NOTIFY foo** delivered on backend #7
4. Nobody holds backend #7, client is gone
5. **× Notification orphaned!**

Why LISTEN/NOTIFY is different?

The pattern so far

Pin client session until reservation reason is gone

TRANSACTION - Clear on COMMIT
TEMP_TABLE - Clear on DISCARD
COPY - Clear when stream ends
PORTAL - Clear when execution finishes
ADVISORY LOCKS - Clear when locks released

set → work → clear → back to pool

LISTEN breaks it

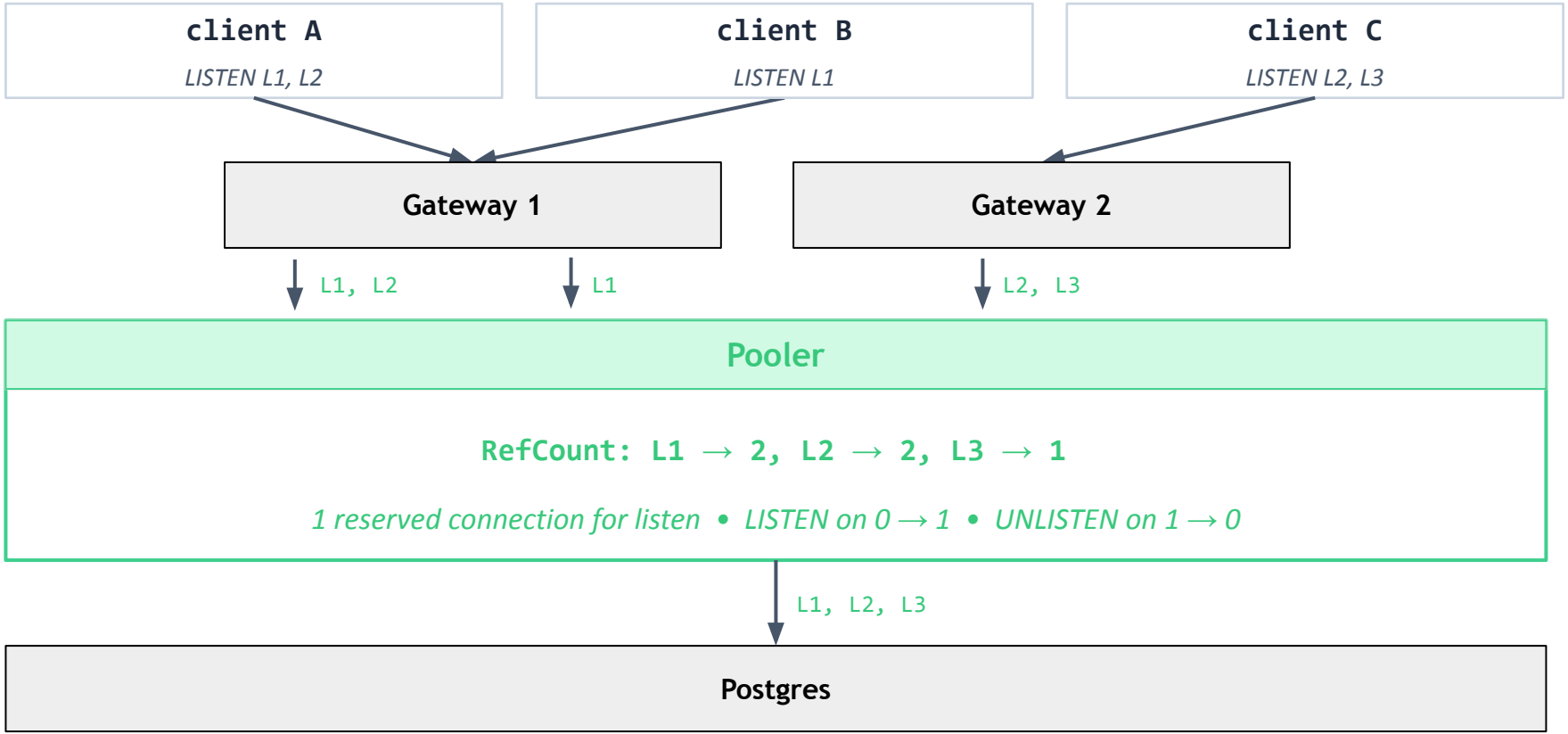
A LISTEN is long lived

Pin one backend per listener and it is held forever!

1,000 idle listeners = 1,000 reserved backends

If we can't pin it, and we can't drop it, what do we do?

The Inversion: Pool the subscription, not the connection



Tradeoffs we accepted

Two services to operate.

Not one. Deploy, monitor, upgrade as a coordinated unit.

Extra hop on the hot path.

Gateway → pooler over gRPC. Small (colocated), but real.

More work in the pool.

Bitmask, parsing, prepared-statement maps.

Detection must be exhaustive.

Missing a reason means a wrong recycle.

Worth it for what we get.

Thank you

Questions?

manan@supabase.com

Manan Gupta • Multigres

github.com/multigres/multigres • multigres.com