

It works on my database

Regression testing of SQL queries

boringSQL

radim@boringsql.com
Skeptic/Heretic, Turned Believer

Why?

**SQL Queries are the #1
cause of database problems**

SQL is scary 🙈

doesn't feel like "real code"

runs somewhere you might have not seen

might be invisible

developers never "really learned" SQL

Tests for the rescue!

Except we test around it

What is SQL testing

Syntax correctness

Schema validity

Contract

Predictable query behaviour

Who's actually writing your SQL?

Developers who learned just enough

ORMs generating queries you've never seen

LLMs!

84% of developers now use LLMs in some way

LLM might solve the part of problem
Ask for a query. Get a query.

**LLMs make this
worse, not better**

**More SQL written by things
that never see production**

Faster rate of change

**The guardrail isn't better
prompts - it's regression testing**

So what's the problem?

Rate of change might accelerate

How do you scale
the rate of reviews?

A simple change

Before

```
SELECT id, user_id, status, total_amount, created_at
FROM orders
WHERE created_at > now() - interval '1 day'
LIMIT 50;
```

After

```
SELECT id, user_id, status, total_amount, created_at
FROM orders
WHERE created_at > now() - interval '1 day'
ORDER BY total_amount DESC
LIMIT 50;
```

Looks reasonable. Ship it.

**Your tests
passed! Ship it**

For that one query, the planner might choose:

- Index Scan on primary key
- Index Scan on a different index
- Bitmap Index Scan
- Sequential Scan
- Parallel Sequential Scan

The planner is smart

It looks at table statistics, available indexes, memory settings, data patterns

And it makes the best choice.

PostgreSQL is very good at guessing - until it isn't 🤪

Same query, different reality

Factor	Your Laptop	CI	Production
Rows	100	1,000	10,000,000
Distribution	Uniform	Uniform	Heavily skewed
Memory	16GB	4GB	128GB

**It works on
my database**

We solved this for code

"It works on my machine" → Docker

"It works in staging" → Feature flags,
gradual rollouts

"It works with my data" → ???

What about code review?

Pop quiz

```
SELECT o.*, u.name
FROM orders o
JOIN users u ON u.id = o.user_id
WHERE o.created_at > now() - interval '30 days'
ORDER BY o.created_at DESC
OFFSET 2500
LIMIT 100;
```

Will this use indexes? Which ones?

How many rows will it scan?

You can't tell from reading SQL

"Just run EXPLAIN before merging"

We need to test two things

Logical correctness: Does it return the right data?

Performance correctness: Does it return them efficiently?

PERFORMANCE

What you going
to measure?

The timing trap

"Just fail the test if it's slow!"

Run	Time
1	154 ms
2	119 ms
3	105 ms

COST?

Cost is the planner's pre-execution estimate based on statistics and cost parameters

BUFFERS FTW!

**Buffers, reported with
ANALYZE, reflect actual pages
accessed during execution**

Expect some little things...

Buffers don't lie

Machine	Time	Buffers
Hetzner ARM64	154 ms	24,108
Hetzner AMD64	119 ms	24,108
M4 Pro VM	98 ms	24,108

Why not cost? that's estimate

**If your query
suddenly reads just
10% more buffers,
that's a regression**

**If you have reproducible
start point (data)**

Introducing RegreSQL

<https://github.com/boringsql/regresql>

SQL Query regression
testing framework

Getting started

```
regresql init postgres://regresql:can123do@192.168.139.28/my_demo
```

regresql/regress.yaml

```
-- default
root: .
pguri: postgres://regresql:can123do@192.168.139.28/my_demo

-- optional configuration
snapshot:
  path: snapshots/test_data.dump
  schema: regresql/schema.sql

analyze:
  enabled: true
```

Flow

- regresql fixturize
- regresql snapshot
- regresql discover (think `git status`)
- regresql add/remove
- edit plan in your favourite editor (hx)
- regresql update -- generate `expected` data (reference)
- regresql baseline -- generate `baseline` (reference)
- regresql test

discover

```
regresql discover
SQL files in project:
  [ ] internal/services/auth/create_provider_link.sql (1 query)
  [ ] internal/services/auth/delete_failed_login_attempt_by_user_id.sql
(1 query)
  [ ] internal/services/auth/delete_provider_link.sql (1 query)
  [ ] internal/services/auth/get_provider_link.sql (1 query)
  [+]
internal/services/auth/select_failed_login_attempt_by_user_email.sql (1
query)
  [ ] internal/services/auth/select_failed_login_attempt_by_user_id.sql
(1 query)
  [ ] internal/services/auth/upsert_failed_login_attempt.sql (1 query)
```

Plan

```
# Test cases for select_failed_login_attempt_by_user_email query
# Parameters: $1=base_lockout_time, $2=lockout_multiplier, $3=email

# User with failed login attempts - should return user_id, attempts, locked_until
user_with_failed_attempts:
  arg1: 5          # base_lockout_time in seconds
  arg2: 2          # lockout_multiplier (exponential)
  arg3: "user0@example.com"

# User without failed login attempts - should return empty result
user_without_failed_attempts:
  arg1: 5
  arg2: 2
  arg3: "user5001@example.com" # index beyond failed_login_attempts count

# Case insensitive email lookup - should still find the user
case_insensitive_email:
  arg1: 5
  arg2: 2
  arg3: "USER0@EXAMPLE.COM"

# Nonexistent user email - should return empty result
nonexistent_user:
  arg1: 5
  arg2: 2
  arg3: "nobody@nowhere.test"
```

RESULTS:

✓ 8 passing
0.00s total

WARNINGS:

select_failed_login_attempt_by_user_email_select_failed_login_attempt_by_user_email.case_insensitive_email.buffer (56 <= 56 * 102%)

⚠ Sequential scan detected on table 'user_account'

Suggestion: Consider adding an index if this table is large or this query is frequently executed

⚠ Nested loop join with sequential scan detected

Suggestion: Add index on join column to avoid repeated sequential scans

select_failed_login_attempt_by_user_email_select_failed_login_attempt_by_user_email.nonexistent_user.buffer (56 <= 56 * 102%)

⚠ Sequential scan detected on table 'user_account'

Suggestion: Consider adding an index if this table is large or this query is frequently executed

⚠ Nested loop join with sequential scan detected

Suggestion: Add index on join column to avoid repeated sequential scans

select_failed_login_attempt_by_user_email_select_failed_login_attempt_by_user_email.user_with_failed_attempts.buffer (56 <= 56 * 102%)

⚠ Sequential scan detected on table 'user_account'

Suggestion: Consider adding an index if this table is large or this query is frequently executed

⚠ Nested loop join with sequential scan detected

Suggestion: Add index on join column to avoid repeated sequential scans

select_failed_login_attempt_by_user_email_select_failed_login_attempt_by_user_email.user_without_failed_attempts.buffer (56 <= 56 * 102%)

⚠ Sequential scan detected on table 'user_account'

Suggestion: Consider adding an index if this table is large or this query is frequently executed

⚠ Nested loop join with sequential scan detected

Suggestion: Add index on join column to avoid repeated sequential scans

What data to test?

I thought that's easy
part; turns out it's NOT

Fixturize

github.com/boringSQL/fixturize

Production query plans without production data

`pg_restore_relation_stats`

`pg_restore_attribute_stats`

<https://boringsql.com/posts/portable-stats/>

pg_regresql extension - PGXN / source

https://github.com/boringSQL/regresql/tree/master/pg_ext

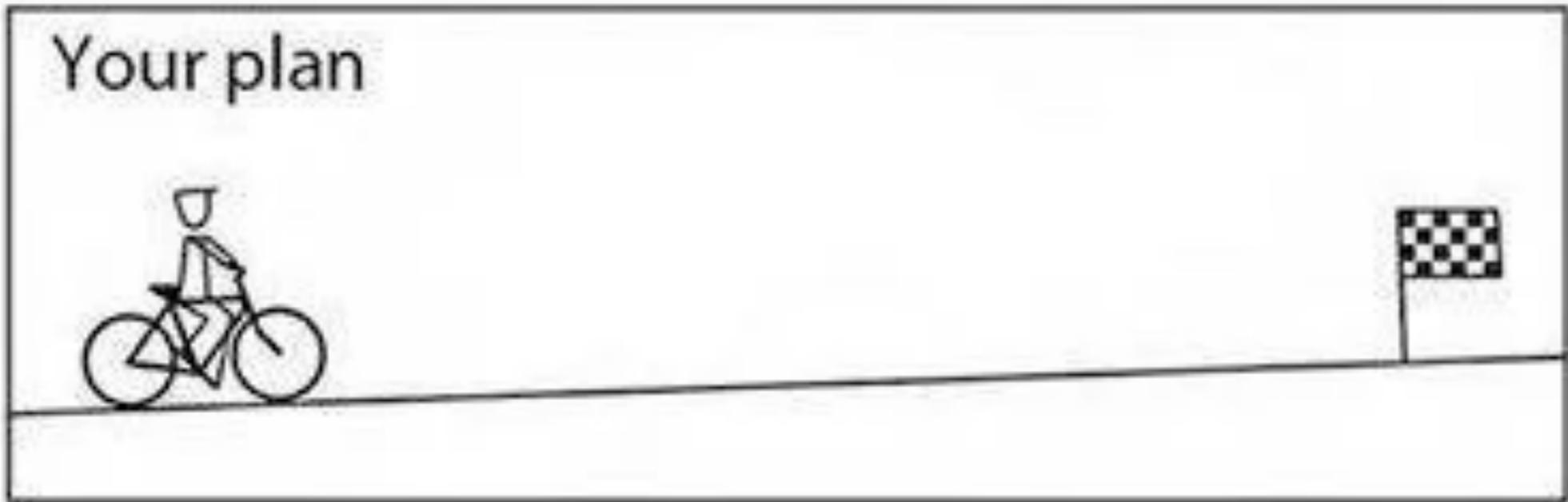
```
SELECT pg_restore_relation_stats('schemaname', %L, 'relname', %L,
'relpages', %s, 'reltuples', %s::real, 'relallvisible', %s);',
      n.nspname,
      c.relname,
      c.relpages,
      c.reltuples,
      c.relallvisible
)
FROM pg_class c
JOIN pg_namespace n ON c.relnamespace = n.oid
WHERE c.relkind IN ('r', 'm')
      AND n.nspname NOT IN ('pg_catalog', 'information_schema',
'pg_toast')
```

RegreSQL 2.0

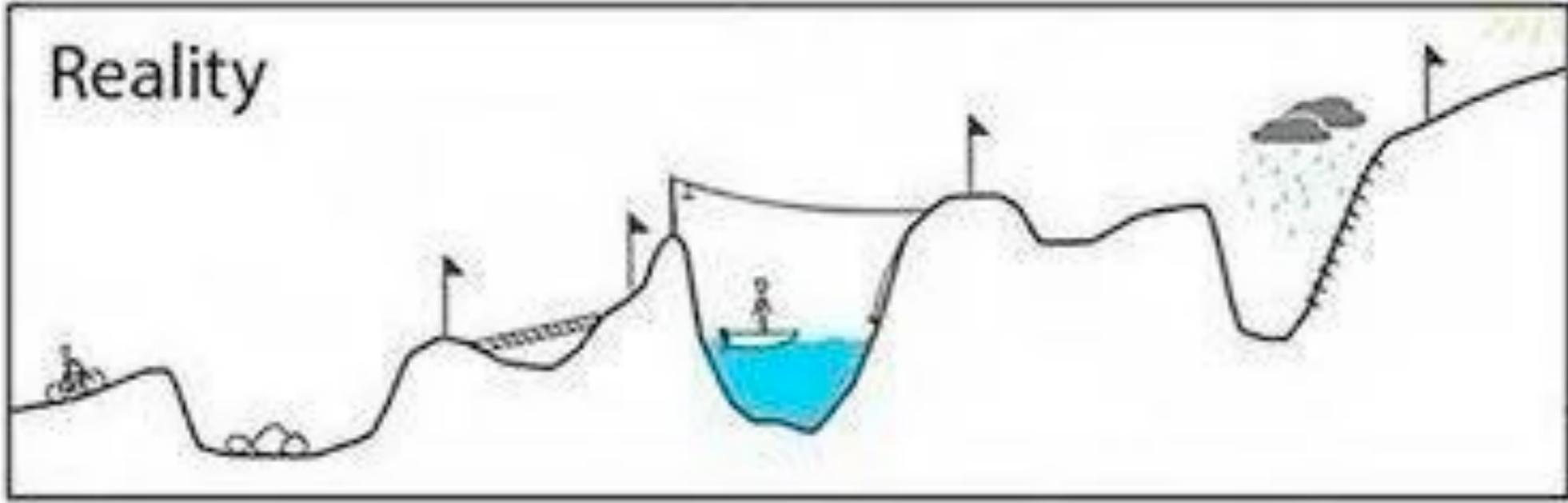
v2.0.0-beta1

full release in coming weeks (*)

Your plan



Reality



What about ORMs?

The N+1 elephant in the in the room

The invisible SQL

ORM Library support vs Query Detection

`pg_stat_statements`

query attribution

Work-in-progress

ActiveRecord/SQLAlchemy support; just ask for it

What ORMs do you use?

Beyond?

- Query capture, ORM support, more CI/CD, automations
- PlanAdvisors integration
- better developer experience
- Improve the contract governance aspect of the tool
- Policy kits (template on what's critical)
- Next Post: Your database does not know it's a bank

Thank you!

Find 1 - 4 queries to start with

Don't be afraid to talk to me

radim@boringsql.com

RegreSQL

Fixturize

DryRun PostgreSQL MCP (AI, yay!)

SQL Labs

Monthly workshops (04/2026)

Newsletter with tips/behind scenes/and thoughts

