```
$ pgmi deploy
```

pgmi: What If PostgreSQL Was the Deployment Engine?

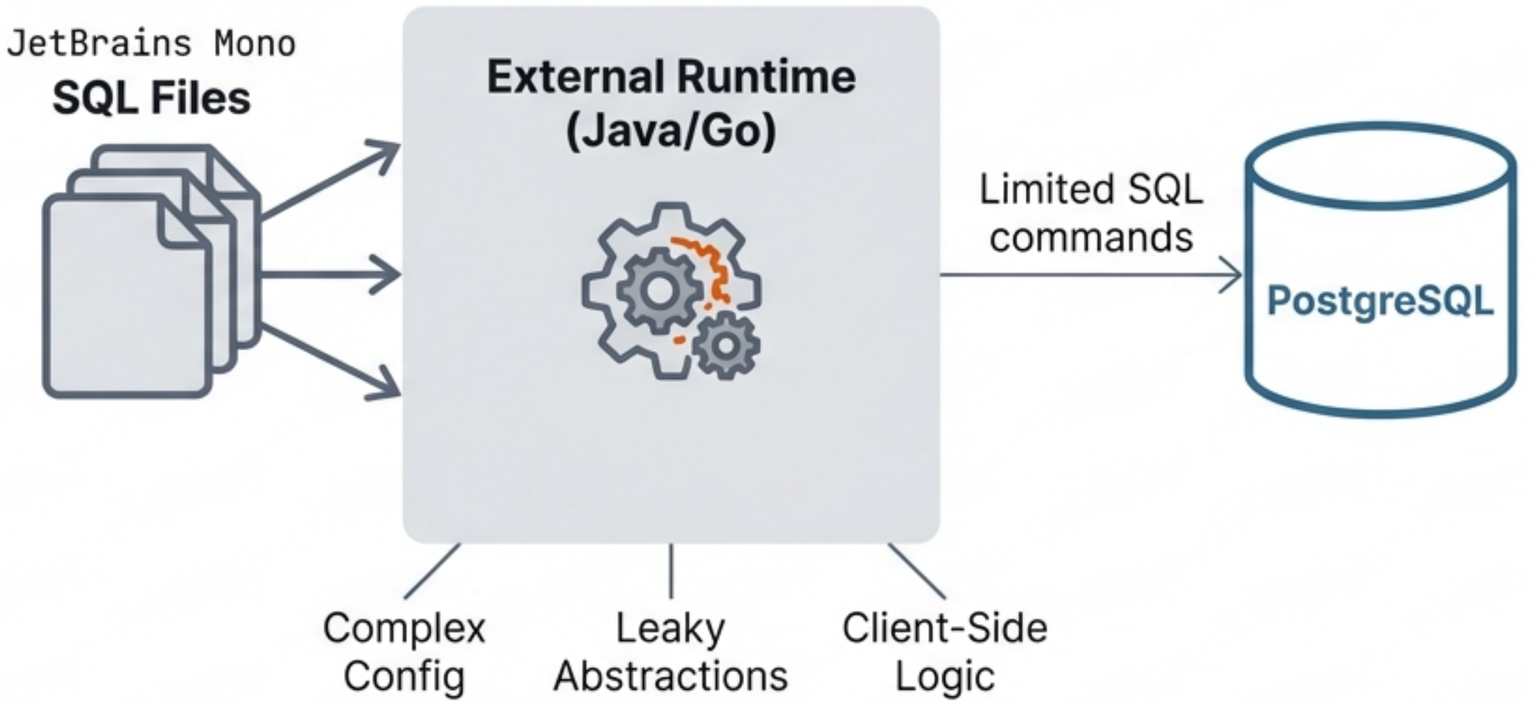
An in-depth introduction to the inversion of control for database migrations.

2026-02-05 | Technical Deep Dive | Intended for PostgreSQL Professionals

The Friction of the External Engine

Current tools (Flyway, Liquibase, Go-migrate) act as opaque execution engines. They sit between your files and the database, attempting to abstract the runtime.

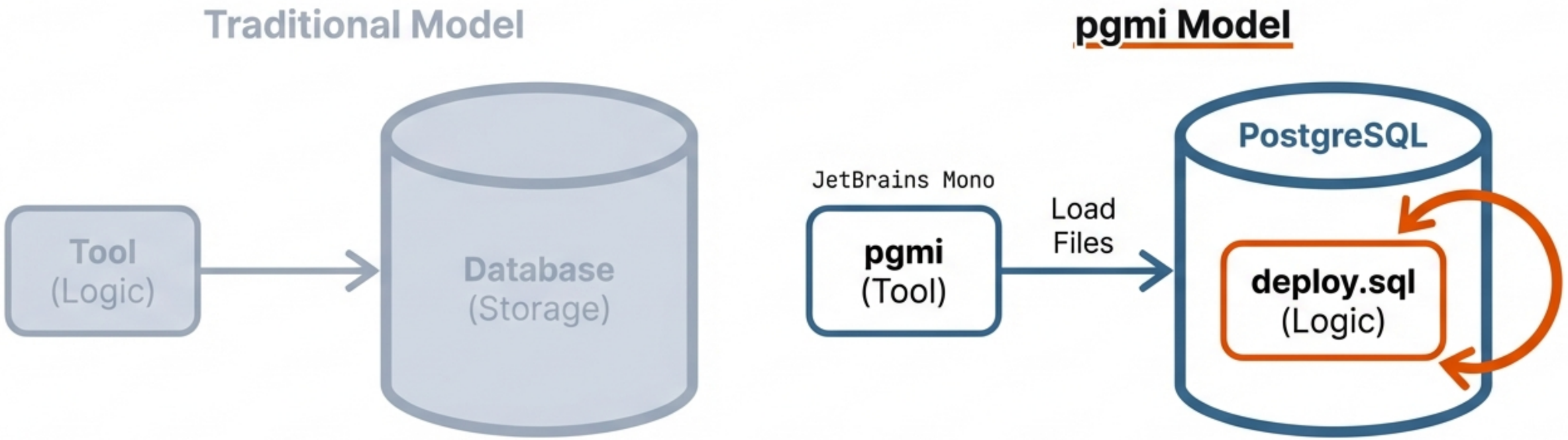
The “Gap”: Developer intent is lost in the translation between filesystem state and database state.



“You have a new lamp that you need to hang on the ceiling, but you're doing that without turning off electricity.” — Vadim Kravcenko

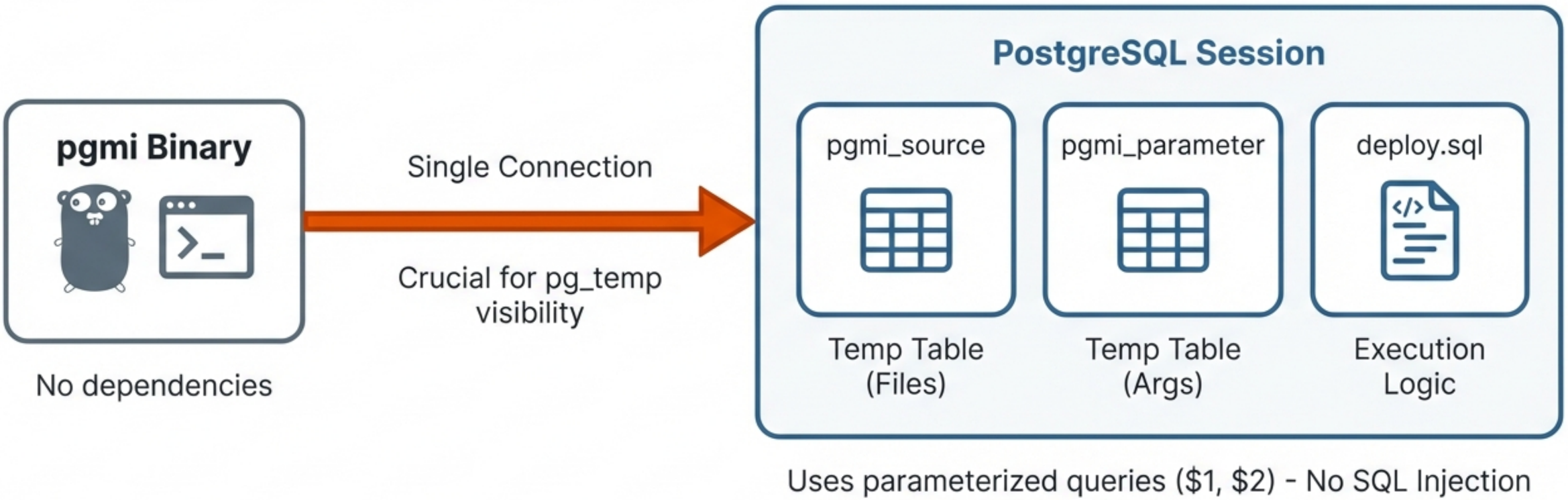
The Inversion of Control

The tool is the fabric. SQL is the logic.



Instead of an external tool deciding execution logic, the tool simply loads files into the DB and lets PL/pgSQL decide.

Architecture: The Execution Fabric

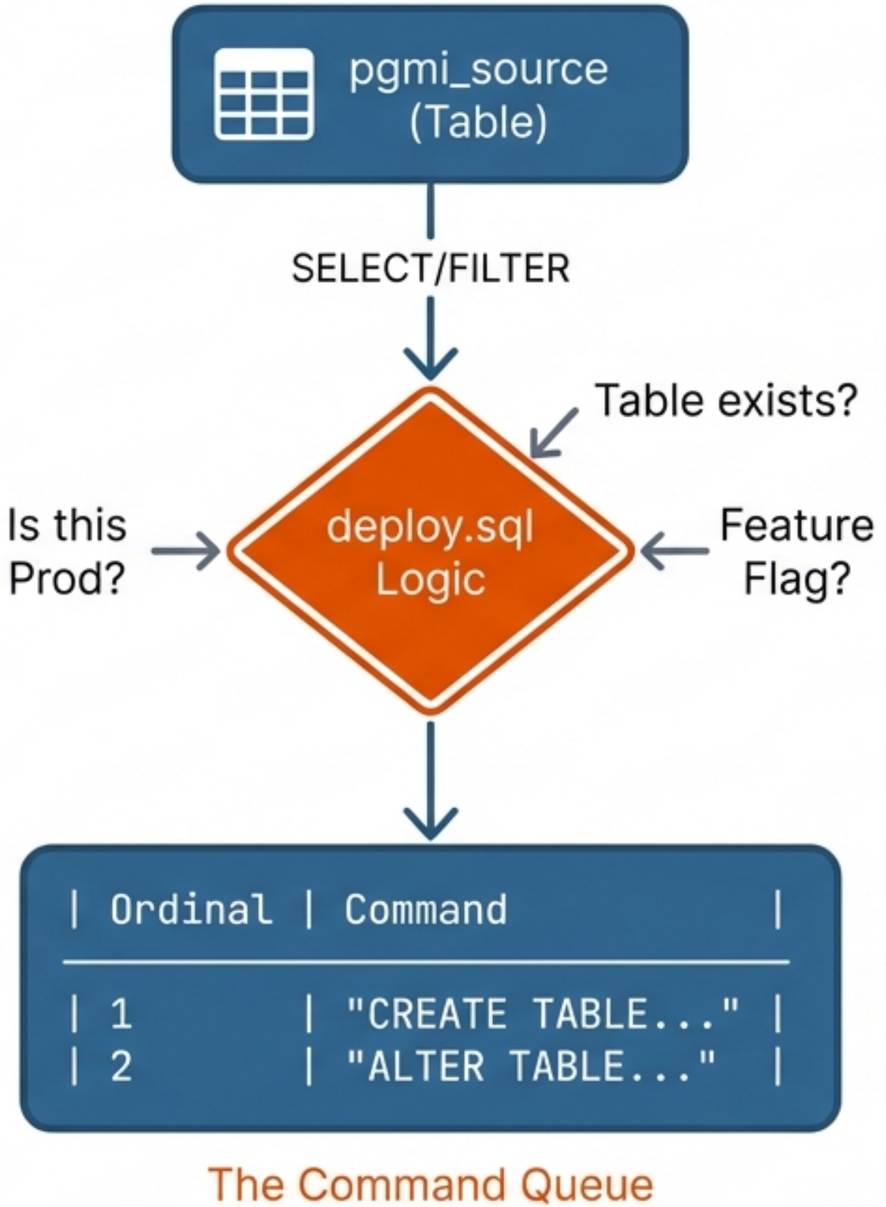


The binary is thin. Its only job is to bridge the filesystem to the database session.

Phase 1: Planning and Logic

The `deploy.sql` script runs as standard PL/pgSQL. It queries the source files, checks runtime conditions, and calls helper functions.

Crucially, it does not execute migrations immediately. It builds a queue.

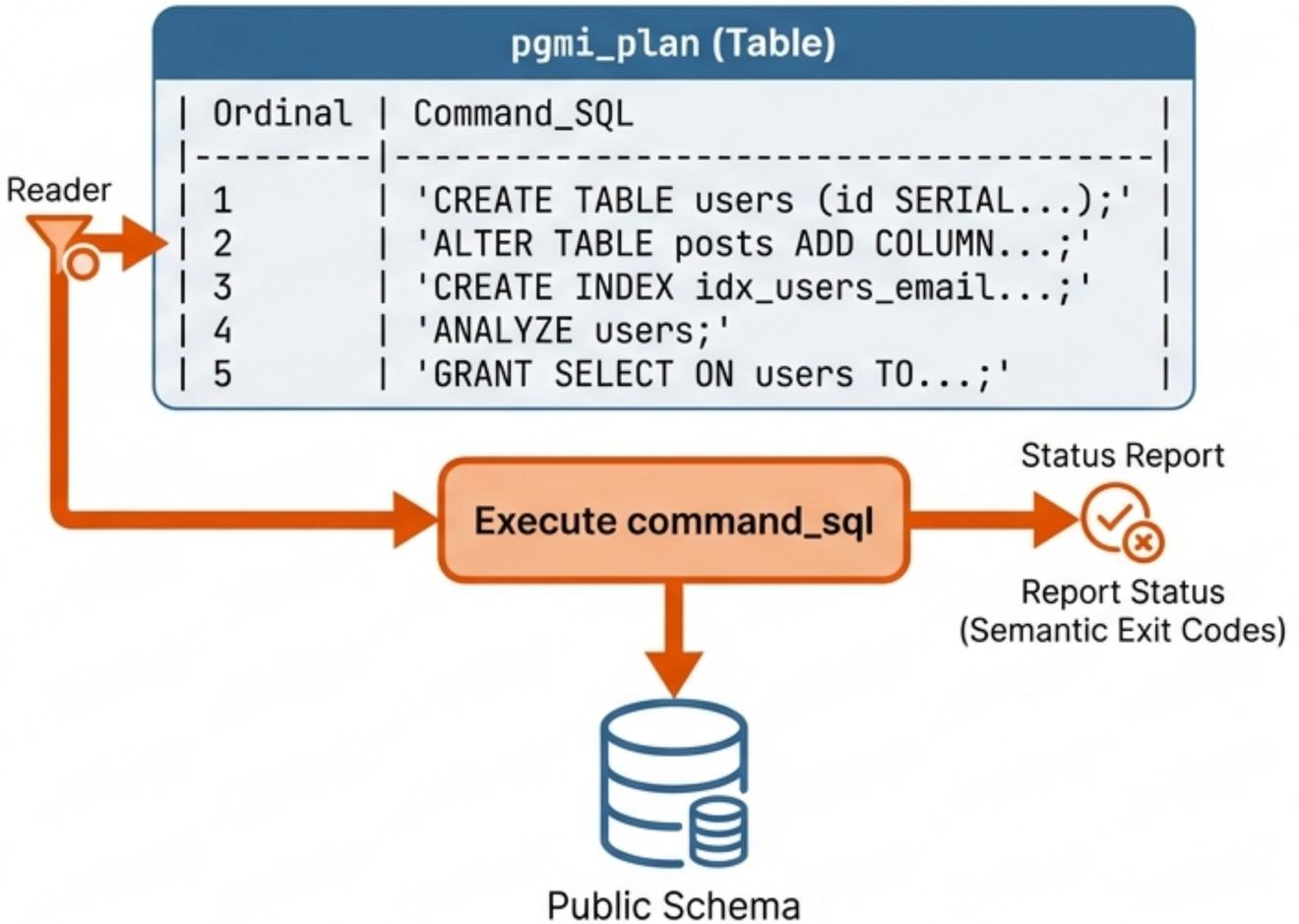


The Command Queue

Phase 2: Deterministic Execution

Once `deploy.sql` finishes, `pgmi` reads the plan table and executes commands in strict ordinal order.

Separating Planning from Execution allows for complex logic validation before any state changes.



Key Insight: The plan is queryable data (`SELECT * FROM pgmi_plan`) before it is executed.

The Session API: Files as Data

The filesystem is exposed as a SQL table.

pg_temp.pgmi_source (Table)		
Column	**Type**	**Description**
path	text	Normalized path starting with './'
content	text	Full file content
checksum	text	SHA-256 hash
extension	text	File extension (e.g., .sql)
depth	integer	Nesting level (0 = project root)
parent_folder_name	text	Metadata for sophisticated filtering

pgmi_parameter

CLI args (`--param env=prod`) map to `pgmi_parameter` rows and `pgmi.env` session variables.

"Code is King": The deploy.sql Template

```
-- 1. Concurrency Control
PERFORM pg_advisory_lock(hashtext(current_database()));

-- 2. Transactional Boundaries
BEGIN;

    -- 3. Dynamic Plan Construction
    FOR rec IN
        SELECT * FROM pgmi_source
        WHERE path LIKE './migrations/%'
        ORDER BY name
    LOOP
        PERFORM pgmi_plan_file(rec.path);
    END LOOP;

COMMIT;
```

Serializes concurrent deployments

Explicit transaction control

Deterministic ordering via SQL

Advanced Capabilities: Beyond Linear Migrations

The Challenge

- CREATE INDEX CONCURRENTLY (Cannot run in transaction block)
- Conditional Seed Data (Dev only)
- Dynamic Feature Flags

The Native SQL Solution

```
-- Outside the main transaction block
PERFORM pgmi_plan_command('CREATE INDEX CONCURRENTLY ...');
BEGIN; ... COMMIT;
```

```
IF current_setting('pgmi.env') = 'dev' THEN
  PERFORM pgmi_plan_file('./seeds/dev_users.sql');
END IF;
```

```
IF NOT EXISTS (SELECT 1 FROM information_schema.tables WHERE
 ...) THEN
   PERFORM pgmi_plan_file('./migrations/01_init.sql');
END IF;
```

Integrated Testing with Savepoints

Zero-pollution integration testing against the exact schema being deployed.

Tests run inside the deployment transaction.
Failure aborts the commit. Success leaves a clean state.

Security & Operations Pipeline

Security

- **No SQL Injection**: All file loading uses parameterized queries ($1, $2).
- **Ephemeral Secrets**: Secrets exist only in session memory; discarded on connection close.
- **Enterprise Auth**: Support for Azure Entra ID, mTLS, and standard connection strings.
- **Race Condition Safety**: Built-in Advisory Locks.

Operations

- **Semantic Exit Codes**:
  - `0` : Success
  - `11` : DB Connection Fail
  - `13` : SQL Execution Fail
- **CI/CD Ready**: `--params-file` support to prevent process list exposure.

Scope and Constraints: What pgmi is NOT

No Multi-DB Support
Optimized strictly for PostgreSQL. Uses native features like transactional DDL and PL/pgSQL.

No Auto-Rollbacks
Down-migrations are often unsafe. We rely on Transactional DDL for atomic failures, and compensating transactions for logic errors.

No Drift Detection
It executes the plan you build. Use pg_catalog queries in deploy.sql if you need state-aware logic.

No GUI
CLI-only. Designed for terminals, scripts, and pipelines.

The Right Tool for the Job

Feature	Flyway / Liquibase	pgmi
Control Paradigm	Managed / Convention-based	Engineering Precision
Language	XML / YAML / Java	Native SQL / PL/pgSQL
Database Support	Multi-Database (Generic)	PostgreSQL Specific (Native)
Best For	Teams wanting 'magic' handling	Complex logic & Heavy PG usage
Philosophy	Abstract the database	'PostgreSQL is the Engine'

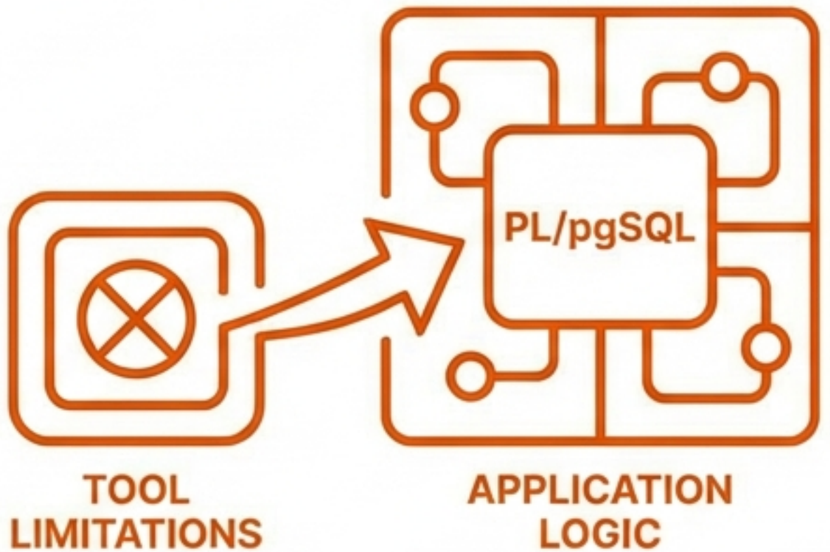
Use pgmi when you need the precision of a scalpel, not the coverage of a net.

The Evolutionary Database

> “*Frequency reduces difficulty.*” — Martin Fowler

By using PL/pgSQL, we make the deployment logic as robust, testable, and expressive as the application code itself.

The Shift: Stop fighting the tool’s limitations. Remove the constraint.



Summary & Resources

Recap Bullets

- **Postgres is the Engine**: Use the runtime you already pay for.
- **Plan -> Execute**: Deterministic, queryable deployment queues.
- **Code-First**: Full PL/pgSQL expressiveness for complex orchestration.

Call to Action

- **GitHub**: github.com/vvka-141/pgmi
- **Documentation**: pgmi.dev
- **Install**:

```
$ go install
github.com/vvka-141/pgmi@latest
```

The plan is data, not opaque state.