Delegating SQL Parsing to PostgreSQL

Greg Potter

FOSDEM 2026

How do you build a tool that understands a Postgres schema?

Option A: Write a SQL parser 😰

Option B: Ask Postgres 🤔

```
CREATE FUNCTION calculate_weighted_average(
   measurements jsonb,
   weights numeric[] DEFAULT ARRAY[1.0],
   ignore_nulls boolean DEFAULT true
) RETURNS TABLE (
   category text,
   weighted_avg numeric(10,4),
   sample_count bigint,
   quality text
) LANGUAGE sql STABLE PARALLEL SAFE
BEGIN ATOMIC
   SELECT
       m->>'category',
       sum((m->>'value')::numeric * w) / nullif(sum(w), 0),
       count(*),
       CASE
           WHEN count(*) >= 100 THEN 'high'
           WHEN count(*) >= 10 THEN 'medium'
           ELSE 'low'
       END
   FROM jsonb_array_elements(measurements) WITH ORDINALITY AS t(m, i)
   LEFT JOIN unnest(weights) WITH ORDINALITY AS w(w, wi) ON t.i = w.wi
   WHERE NOT ignore_nulls OR m->>'value' IS NOT NULL
   GROUP BY m->>'category';
END;
```

What Postgres Knows Now

```
postgres=# SELECT proname, proargtypes, proallargtypes, proargnames, prorettype, provolatile,
proparallel, prosqlbody FROM pg_proc WHERE proname ='calculate_weighted_average';


-[ RECORD 1 ]--+-------------------------------------------------------------------------------
proname        | calculate_weighted_average
proargtypes    | 3802 1231 16                    -- jsonb, numeric[], boolean
proallargtypes | {3802,1231,16,25,1700,20,25}  -- jsonb, numeric[], boolean, text, numeric, ...
proargnames    | {measurements,weights,ignore_nulls,category,weighted_avg, sample_count,quality}
prorettype     | 2249
provolatile    | s                               -- STABLE
proparallel    | s                               -- PARALLEL SAFE
prosqlbody     | (parsed query tree)
```

Shadow Database

System Catalogs

pg_class	tables, views, indexes, sequences, materialized views
pg_attribute	columns
pg_proc	functions, procedures, aggregates
pg_type	types (built-in, composite, enum, domain, range)
pg_constraint	primary keys, foreign keys, check, unique, exclusion
pg_index	index details (columns, expressions, predicates)

Querying the catalogs

```
SELECT attname,
       format_type(atttypid, atttypmod) as type,
       NOT attnotnull as nullable,
       pg_get_expr(d.adbin, d.adrelid) as default
FROM pg_attribute a
LEFT JOIN pg_attrdef d ON a.attrelid = d.adrelid AND a.attnum = d.adnum
WHERE a.attrelid = 'orders'::regclass
  AND a.attnum > 0;
```

```
 attname  |     type     | nullable |         default
----------+--------------+----------+-------------------------
 id       | integer      | f        |
 status   | order_status | f        | 'pending'::order_status
 customer | text         | f        |
```

Order Matters

```
postgres=# DROP TABLE orders;
ERROR:  cannot drop table orders because other objects depend on it
DETAIL:  function pending_orders() depends on table orders
view pending_order_count depends on function pending_orders()
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

```
postgres=# DROP TYPE order_status;
ERROR:  cannot drop type order_status because other objects depend on it
DETAIL:  column status of table orders depends on type order_status
function pending_orders() depends on type order_status
view pending_order_count depends on function pending_orders()
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
```

```
pending_order_count (view)
    ↓ uses
pending_orders() (function)
    ↓ queries
orders (table)
    ↓ uses
order_status (type)
```

pg_depend stores the graph

```
 classid    | objid                | refobjid       | deptype
------------+----------------------+----------------+---------
 pg_rewrite | pending_order_count | pending_orders | n
 pg_proc    | pending_orders      | orders         | n
 pg_class   | orders              | order_status   | n
```

From graph to order

pg_depend gives you edges:

pending_order_count → pending_orders
pending_orders → orders
orders → order_status

Topological sort gives you order:

CREATE: order_status → orders → pending_orders → pending_order_count
DROP: pending_order_count → pending_orders → orders → order_status

deptype

deptype = 'n' (normal)
    view → function → table → type
    Real dependencies. These are yours.

deptype = 'a' (automatic)
    sequence → SERIAL column
    index → table
    Created together. Linked.

deptype = 'i' (internal)
    TOAST table → parent
    Postgres internals.

deptype = 'e' (extension)
    st_distance → postgis
    Belongs to an extension. Not yours.

What pg_depend can't see

```
-- pg_depend CAN'T see inside this (string-literal bodies)
CREATE FUNCTION pending_orders() RETURNS SETOF orders
LANGUAGE sql STABLE
AS $$
    SELECT * FROM orders WHERE status = 'pending';
$$;
```

pg_depend tracks:
- function → return type (`SETOF orders`) ✓

pg_depend doesn't track:
- function body → orders table ✗
- function body → status column ✗

BEGIN ATOMIC fixes this

```
-- pg_depend CAN see inside this (BEGIN ATOMIC)
CREATE FUNCTION pending_orders() RETURNS SETOF orders
LANGUAGE sql STABLE
BEGIN ATOMIC
    SELECT * FROM orders WHERE status = 'pending';
END;
```

pg_depend tracks:

- function → return type (`SETOF orders`) ✓
- function body → orders table ✓
- function body → status column ✓

Implicit objects

```
CREATE TABLE my_table (
    id SERIAL PRIMARY KEY
);
 objid    | refobjid        | deptype
----------+-----------------+---------
 my_table | my_table_id_seq | a
 my_table | my_table_pkey   | a

Creates:
    table: orders
    sequence: orders_id_seq   (deptype 'a')
    index: orders_pkey        (deptype 'a')
```

Array types

```
CREATE TYPE order_status AS ENUM (...);
Creates:
    type: order_status
    type: _order_status       (array type, depends on order_status, deptype ‘i’);

CREATE TABLE array_table (statuses order_status[]);

 objid          | refobjid       | deptype
----------------+----------------+---------
 array_table    | _order_status | n
 _order_status | order_status   | i
```

Trade Offs

You need a running Postgres
- Shadow database = real database

You have to build support explicitly
- There are a lot of catalog tables

Postgres versions matter
- Catalogs evolve
- prosqlbody requires Postgres 14+

Review your output
- Only as complete as your queries

The toolkit

Shadow database
→ Postgres parses, you query

System catalogs
→ Structure, types, columns

pg_depend
→ Dependency graph
→ Filter by deptype
→ Know the blind spots
→ Watch for implicit objects

Topological sort
→ Edges → correct order

pgmt

I built pgmt using all of this.

Schema-as-code for Postgres.
Edit .sql files, pgmt generates migrations.

pgmt.dev

github.com/gdpotter/pgmt

Not just for migrations

Drift detection
- Compare expected catalogs to live state

Schema linters
- Query catalogs for problems
- (unused indexes, missing FKs, naming violations)

CI validation
- Apply to shadow database
- If it succeeds, the schema parses

Visualization
- pg_depend is the dependency graph
- Draw it

Postgres already knows your schema.

Just ask.