Graph Walking in PostgreSQL

For fun and (possible) profit

Building a system to sift and correlate large volumes of data

Mark Bracher
bracher@gmail.com

About me

Mark Bracher (bracher@gmail.com)

makr17 on various platforms (HN, github, gitlab, keybase)

Software Engineer

Data Nerd

How we got here

The Existing Business

- Data discovery and analysis for small/medium-sized businesses
  - Find/fix incorrect data
  - Textual analysis on reviews

The Challenge

- Who are their customers, *really*?

Graphs

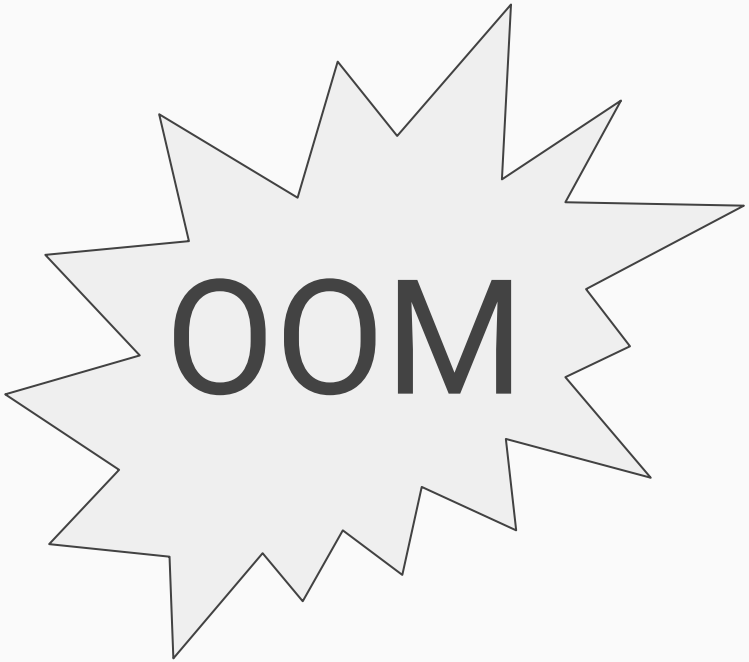
Solve any problem by cutting it into pieces

The Graph

- **Nodes** - pieces of data
- **Edges** - nodes are related (probability)
- **Graph Neighborhood** - enumerate connected nodes from some point

- Test data set
  - 10M nodes
  - 50M edges

What about <this> graph system?

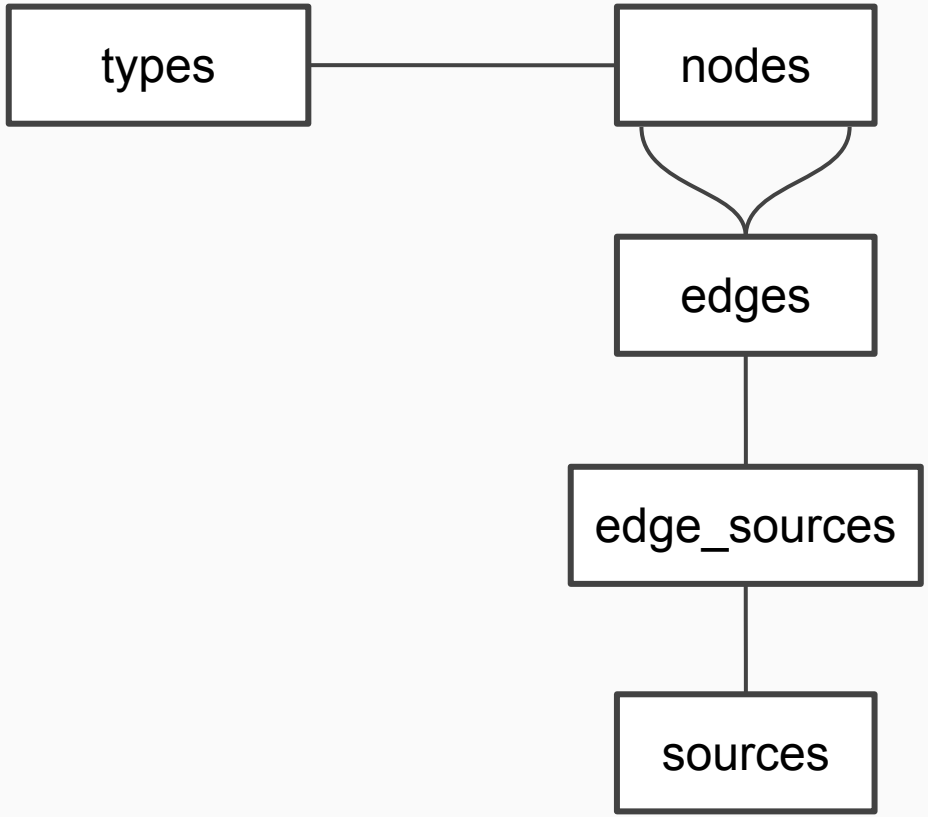


Postgres

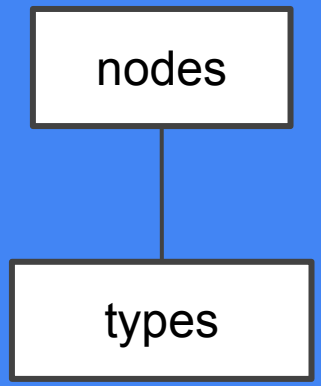
If you squint, any problem starts to look like a **database** problem

Postgres

- **Nodes**
  - **JSON payload**
- **Edges** connect Nodes
- A Node has a **Type**
- Edges have **Sources**



Postgres

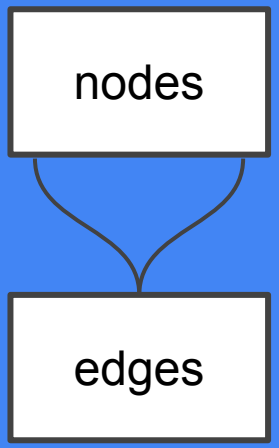



```
create table types (
  id       bigserial    not null  primary key,
  name     varchar(64)  not null,
  created  timestamptz  not null  default current_timestamp
);


create table nodes (
  id          bigserial    not null  primary key,
  type_id     bigint       not null  references types(id),
  normalized  jsonb        not null,
  pretty      jsonb        not null,
  created     timestamptz  not null  default current_timestamp
);

create unique index on nodes (md5(normalized::text));
create index normalized_idx on nodes
  using gin(normalized jsonb_path_ops);
```

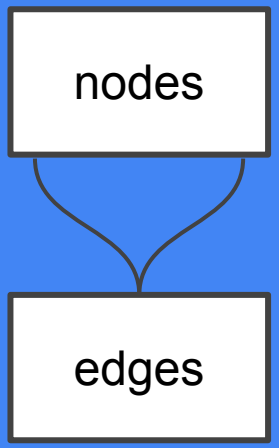
Postgres



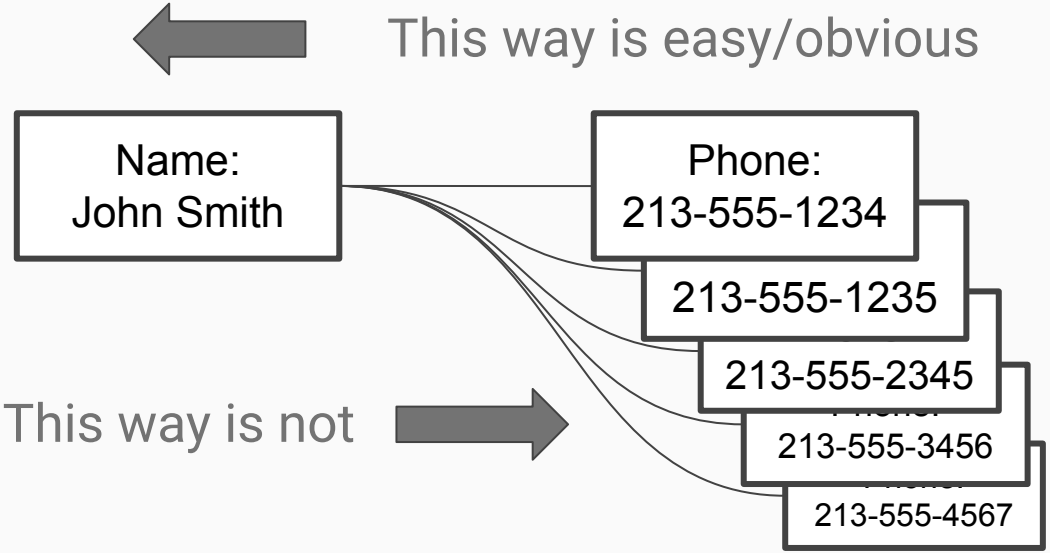

```
create table edges (
  id            bigserial        not null  primary key,
  from_node_id  bigint           not null  references nodes(id),
  to_node_id    bigint           not null  references nodes(id),
  probability  double precision  not null
    check(probability > 0.0 and probability < 1.0),
  created      timestamptz       not null  default current_timestamp,
);

create unique index on edges (from_node_id, to_node_id);
create index on edges (from_node_id, probability desc);
```

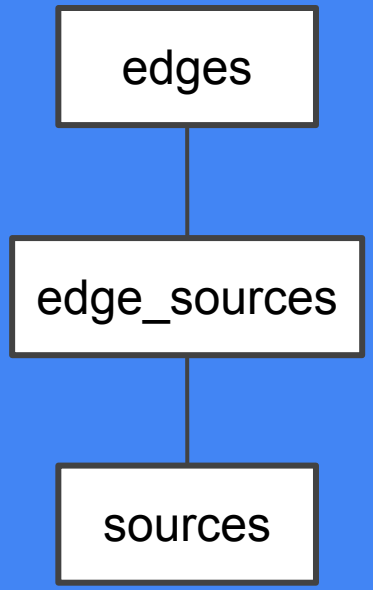
Postgres



Probability Adjustments



Postgres




```
create table sources (
  id                bigserial         not null  primary key,
  base_probability  double precision  not null
    check(base_probability > 0.0 and base_probability < 1.0),
  name              varchar(64)       not null,
  uri               varchar           not null,
  created           timestamptz       not null  default current_timestamp
);


create table edge_sources (
  id                bigserial         not null  primary key,
  edge_id           bigint            not null  references edges(id),
  source_id         bigint            not null  references sources(id),
  base_probability  double precision  not null
    check(base_probability > 0.0 and base_probability < 1.0),
  uri               varchar           not null,
  created           timestamptz       not null  default current_timestamp
);

create unique index on edge_sources (edge_id, source_id);
```

JSON Document

nodes

Pasadena Convention Center
300 E Green St, Pasadena, CA 91101

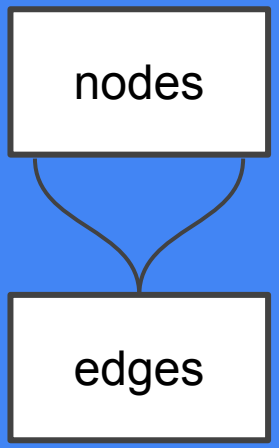
“Pretty”

```
{
  “address”: “300”,
  “predirAbbrev”: “East”,
  “streetName”: “Green”,
  “streetTypeAbbrev”: “Street”,
  “location”: “Pasadena”,
  “stateAbbrev”: “CA”,
  “zip”: “91101”,
  “plus4”: “2399”,
  “latitude”: 34.1436537,
  “longitude”: -118.14520848
}
```

Normalized

```
{
    “address”: “300”,
    “predirAbbrev”: “E”,
    “streetName”: “GREEN”,
    “streetTypeAbbrev”: “ST”,
    “location”: “PASADENA”,
    “stateAbbrev”: “CA”,
    “zip”: “91101”,
    “plus4”: “2399”,
    “latitude”: 34.1436537,
    “longitude”: -118.14520848
}
```

Resulting Document



Walk the graph neighborhood using a recursive CTE

```
with recursive neighborhood (node_id, path, p) as (
  select
    id
    , array_append('{}'::bigint[], id)
    , 1.0::double precision
  from nodes
  where normalized @> '{"phone":"6265551234"}' -- starting node
  union all
  select
    to_node_id
    , array_append(nbr.path, to_node_id)
    , nbr.p * e.probability
  from
    edges e
    join neighborhood nbr
      on e.from_node_id = nbr.node_id
     and e.probability >= 0.45/nbr.p      -- stay above min p, 0.45
     and not e.to_node_id = any(nbr.path) -- avoid cycles
)
select
  nbr.node_id
  , n.pretty
  , max(nbr.p)
from
  neighborhood nbr
  join nodes n on nbr.node_id = n.id
group by 1,2
;
```

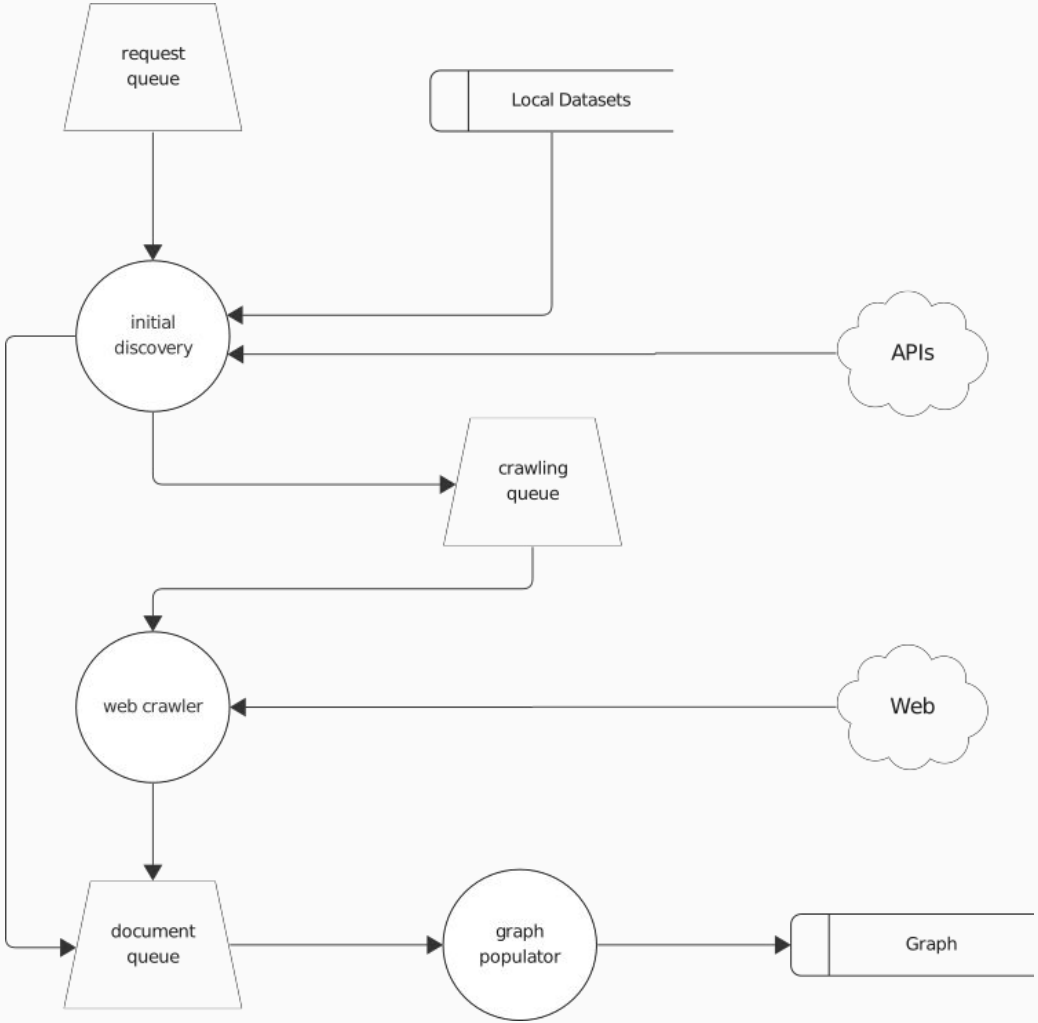
The hardware

“One sign of a good job is that they buy you nice shiny toys to play with.”
– Arthur Bracher, Jr. (dad)

MVP, and It (mostly) Works

It Works?!?

And we start shoveling in data as fast as we can…



Still (Mostly) Working

500 Million Nodes… and the sharp edges start to show

Graph Maintenance Performance


```
create or replace function node_to_type( _node_id bigint ) returns bigint as $$
declare
  _type_id bigint;
begin
  select type_id from nodes into _type_id where id = _node_id;
  return _type_id;
end;
$$ immutable language plpgsql;

create index on edges (from_node_id, node_to_type(to_node_id));
```

Cross-Contaminated Profiles?

- Looks self-inflicted, but ultimately is a legitimate problem
  - Add `types.identifying`

```
alter table types
  add column identifying boolean default false;

update  types
  set   identifying = true
  where name in ('email', 'phone', 'facebook', 'twitter', ... , 'address');
```

  - And only walk out from nodes that are identifying

Cross-Contaminated Profiles

Extend the recursive CTE, only walk out from identifying nodes

```
with recursive neighborhood (node_id, type_id, path, p) as (
  select
    id as node_id
    , type_id
    , array_append('{}'::bigint[], id) as path
    , 1.0::double precision as p
  from nodes
  where normalized @> '{"phone":"6265551235"}' -- starting node
  union all
  select
    to_node_id as node_id
    , node_to_type(e.to_node_id)
    , array_append(nbr.path, to_node_id)
    , nbr.p * e.probability
  from
    edges e
    join neighborhood nbr
      on e.from_node_id = nbr.node_id
     and e.probability >= 0.45/nbr.p      -- stay above min p, 0.45
     and not e.to_node_id = any(nbr.path) -- avoid cycles
  where
    nbr.type_id in (select id from types where identifying)
)
select
  nbr.node_id
  , n.pretty
  , max(nbr.p)
from
  neighborhood nbr
  join nodes n on nbr.node_id = n.id
group by 1,2
;
```

Changing the Normalization Rules

- This one is just a ***bad*** idea…

It Works, and kept growing

> 1.5 Billion Nodes
> 8 Billion Edges
~ 20 Billion Edge Sources
> 1000 Sources

OK, It Works, but Should I Have Built It?

PII - Personally Identifiable Information

GDPR - General Data Protection Regulation in the EU

CCPA - California Consumer Privacy Act

Questions or Comments?

https://gitlab.com/makr17/graph-walking-for-fun-and-possible-profit

Mark Bracher
bracher@gmail.com

More on Cross-contaminated Profiles