Federating Queries Using postgres_fdw

john melesky

Rentrak, Inc

September 17, 2013

Who Am I?

- A long-time programmer, working with PostgreSQL in the cloud

Who Am I?

- A long-time programmer, working with PostgreSQL in ~~the cloud~~ my butt

Who Am I?

- A long-time programmer, working with PostgreSQL in ~~the cloud~~ my butt
- Now, a DBA, working with PostgreSQL on real machines with real disks

Who Am I?

- A long-time programmer, working with PostgreSQL in ~~the cloud~~ my butt
- Now, a DBA, working with PostgreSQL on ~~real machines~~ VMWare with real disks

Who Am I?

- A long-time programmer, working with PostgreSQL in ~~the cloud~~ my butt
- Now, a DBA, working with PostgreSQL on ~~real machines~~ VMWare with ~~real disks~~ NetApps

PostgreSQL inheritance partitioning

```
create table transactions (
        id serial,
        user_id bigint,
        time_utc timestamp,
        int_value bigint,
        txt_value text,
        primary key (id)
);

create table transactions_201306 (
        like transactions including indexes,
        check
          (time_utc >= '2013-06-01' and
           time_utc < '2013-07-01')
) inherits (transactions);
```

PostgreSQL inheritance partitioning

```
create table transactions (
        id serial,
        user_id bigint,
        time_utc timestamp,
        int_value bigint,
        txt_value text,
        primary key (id)
);

create table transactions_201306 (
        like transactions including indexes,
        check
          (time_utc >= '2013-06-01' and
           time_utc < '2013-07-01')
) inherits (transactions);
```

You know this already

Old-school partitioning

```
create view transactions as (
        select * from transactions_201301
          union all
        select * from transactions_201302
          union all
        select * from transactions_201303
          union all
        select * from transactions_201304
          union all
        ...
);
```

Why don't we still use this?

Why don't we still use this?

1. No insert triggers on views

Why don't we still use this?

1. No insert triggers on views
2. No "inherit indexes" without additional misdirection

Why don't we still use this?

1. No insert triggers on views
2. No "inherit indexes" without additional misdirection
3. Basically, we have a better option with inheritence partitioning

Postgres Foreign Data Wrapper

```
-- just once
create extension postgres_fdw;

-- once per data node
create server node0 foreign data wrapper postgres_fdw
  options (connection stuff);
create user mapping for app_user server node0;

-- once per table per node
create foreign table transactions_node0
  (table definition)
  server node0
  options (table_name 'transactions');
```

Federating, Old-school

```
create view transactions as (
        select * from transactions_node0
          union all
        select * from transactions_node1
          union all
        select * from transactions_node2
          union all
        select * from transactions_node3
          union all
        ...
);
```

Querying

```
primary=# explain select count(*) from transactions;
                                                QUERY PLAN
-----------------------------------------------------------------------------------------
 Aggregate  (cost=1767.38..1767.39 rows=1 width=0)
   ->  Append  (cost=100.00..1699.12 rows=27304 width=0)
         ->  Foreign Scan on transactions_node0
               (cost=100.00..212.39 rows=3413 width=0)
         ->  Foreign Scan on transactions_node1
               (cost=100.00..212.39 rows=3413 width=0)
         ->  Foreign Scan on transactions_node2
               (cost=100.00..212.39 rows=3413 width=0)
         ->  Foreign Scan on transactions_node3
               (cost=100.00..212.39 rows=3413 width=0)
         ->  Foreign Scan on transactions_node4
               (cost=100.00..212.39 rows=3413 width=0)
         ...
(10 rows)

Time: 1.226 ms
```

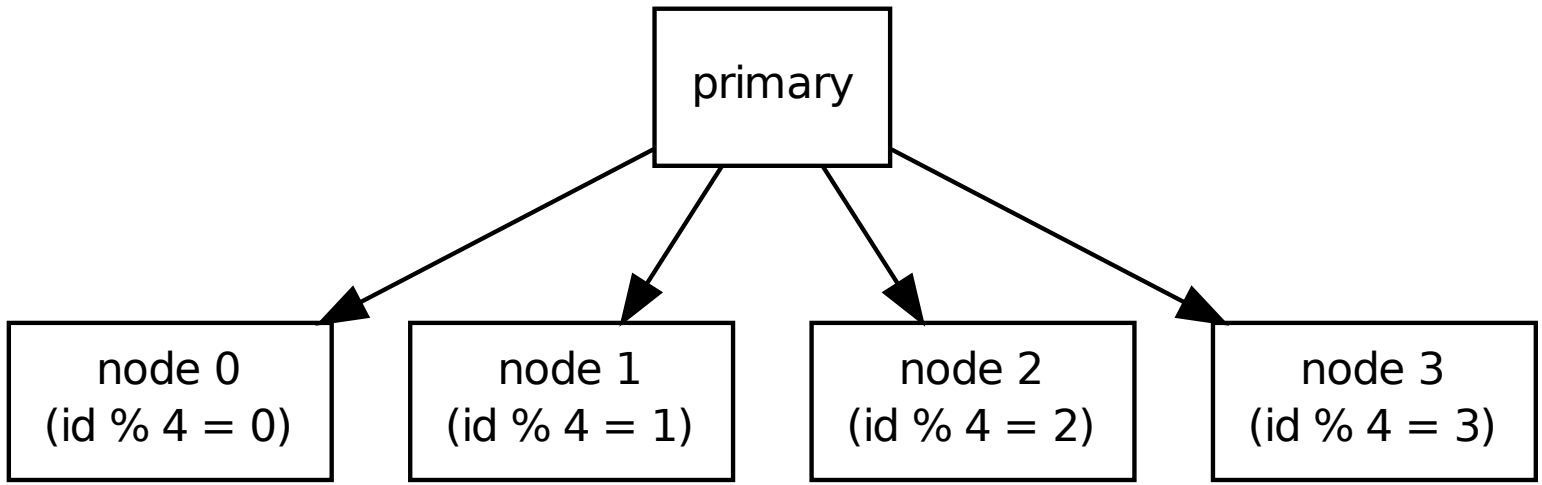
Querying

```
primary=# explain verbose select count(*) from transactions;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------
 Aggregate  (cost=1767.38..1767.39 rows=1 width=0)
   Output: count(*)
   ->  Append  (cost=100.00..1699.12 rows=27304 width=0)
         ->  Foreign Scan on public.transactions_node0
               (cost=100.00..212.39 rows=3413 width=0)
                Remote SQL: SELECT NULL FROM public.transactions
         ->  Foreign Scan on public.transactions_node1
               (cost=100.00..212.39 rows=3413 width=0)
                Remote SQL: SELECT NULL FROM public.transactions
         ->  Foreign Scan on public.transactions_node2
               (cost=100.00..212.39 rows=3413 width=0)
                Remote SQL: SELECT NULL FROM public.transactions
         ...
(19 rows)

Time: 1.273 ms
```

Querying

```
primary=# select count(*) from transactions;
   count
---------
 1095336
(1 row)

Time: 3035.054 ms
```

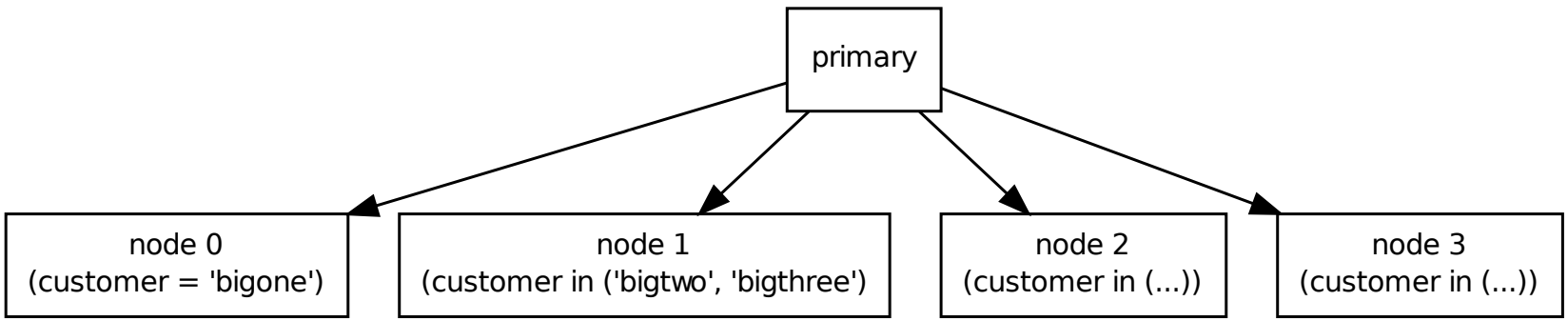
Round-robin



Round-robin

```
primary=# create foreign table transactions_node0 (
primary(#         id serial,
primary(#         user_id bigint,
primary(#         time_utc timestamp,
primary(#         int_value bigint,
primary(#         txt_value text,
primary(#         check ((id % 8) = 0)
primary(# ) server node0
primary(# options (table_name 'transactions');
ERROR:  constraints are not supported on foreign tables
LINE 6:         check ((id % 8) = 0)) server node0 ...
```

Domain-based (aka "sharding")



Range-based

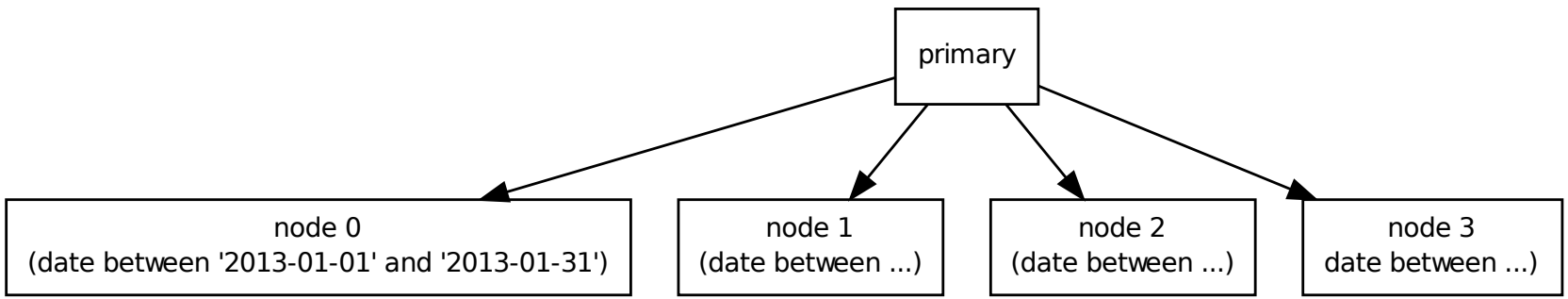
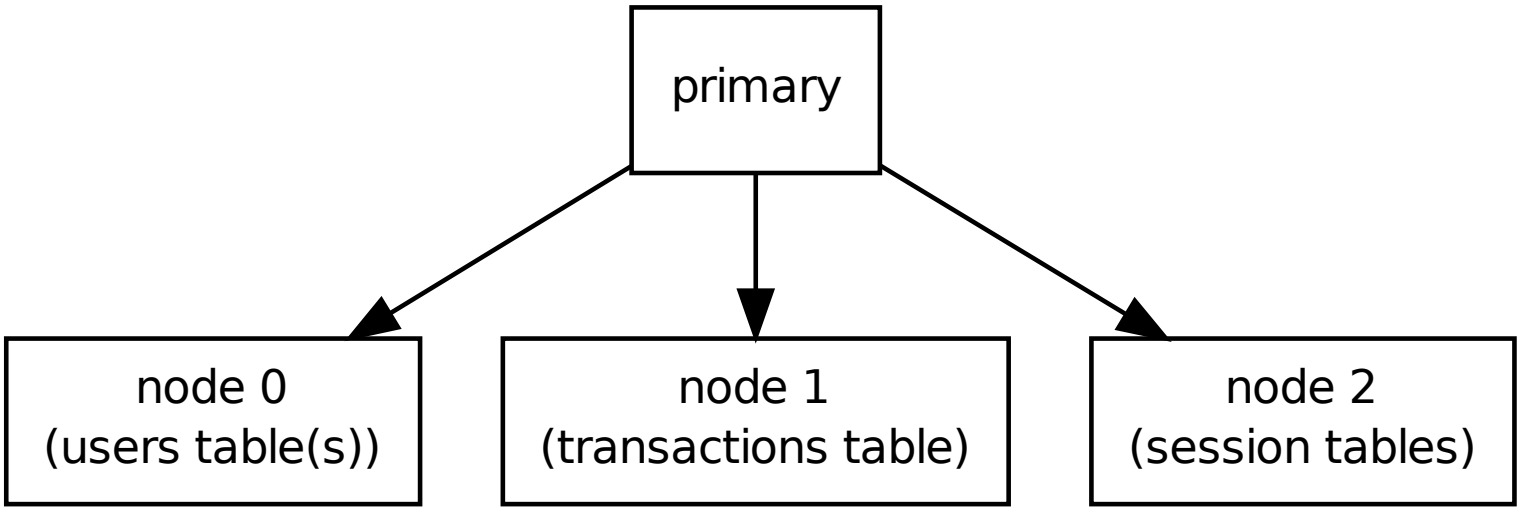
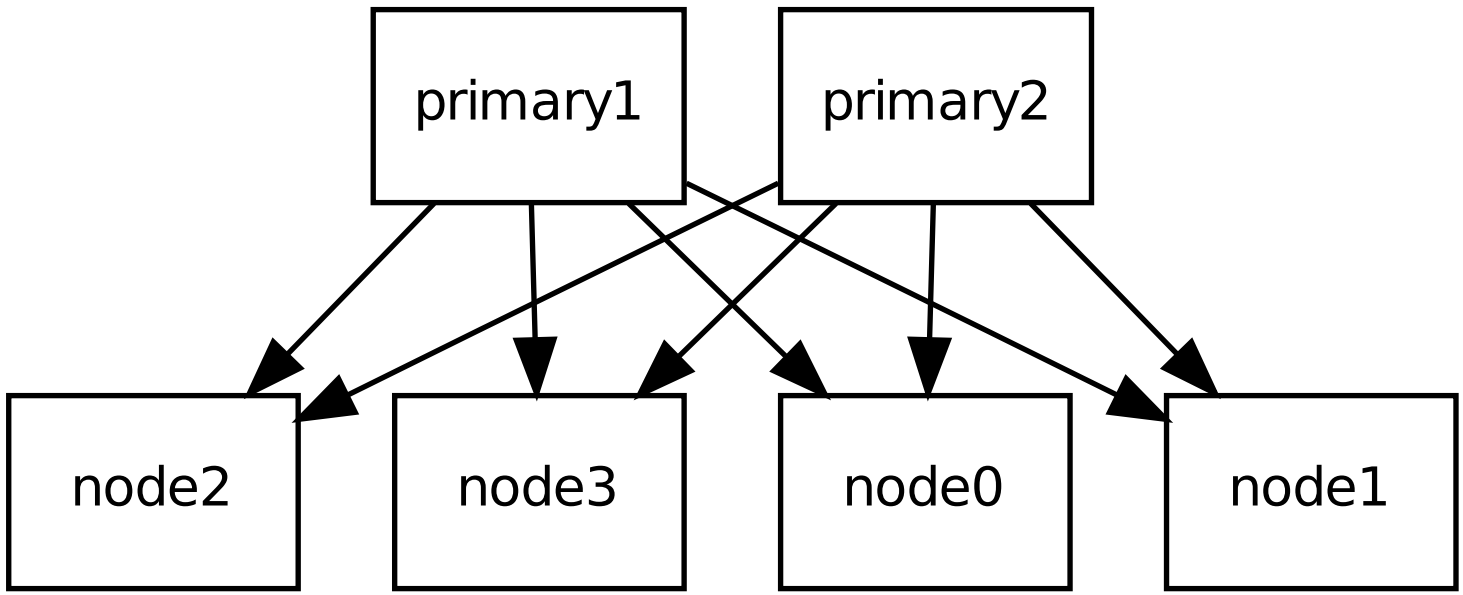

primary
node 0
(date between '2013-01-01' and '2013-01-31')
node 1
(date between ...)
node 2
(date between ...)
node 3
date between ...)

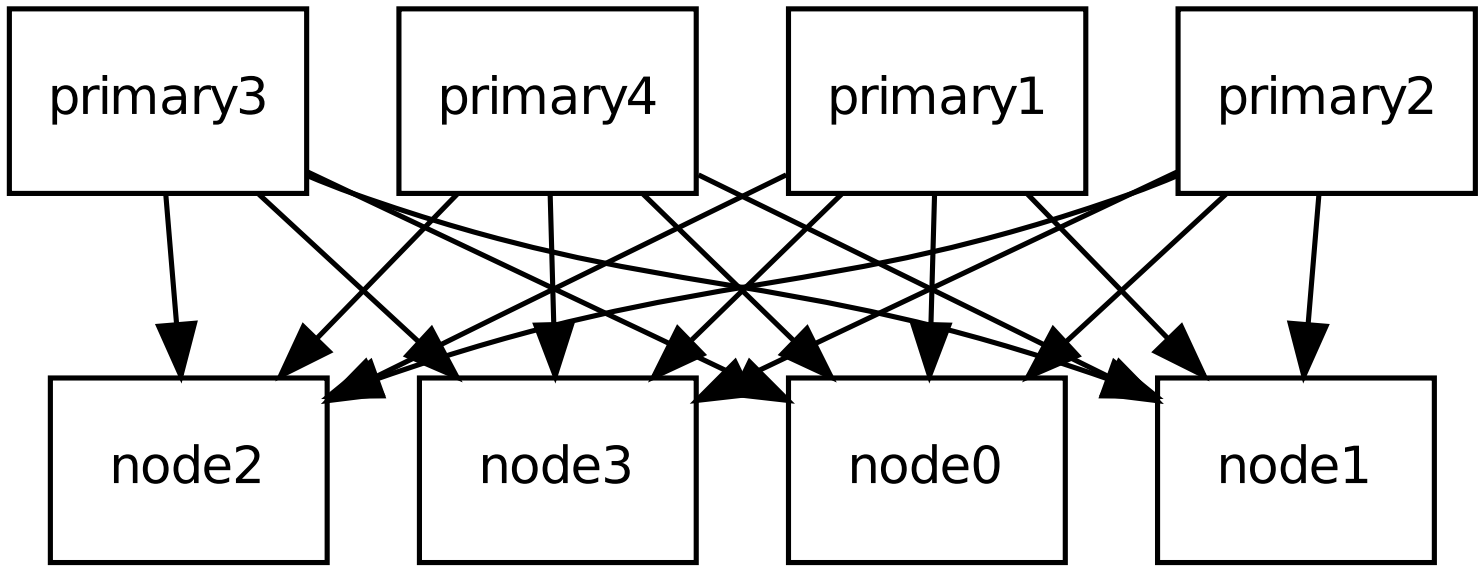
Table-based


primary
node 0
(users table(s))
node 1
(transactions table)
node 2
(session tables)

Multi-head



Multi-head


primary3
primary4
primary1
primary2
node2
node3
node0
node1

Demo time

Limitations: Network traffic

Limitations: Network traffic

```
primary=# select count(*) from transactions_local;
  count
---------
 1095336
(1 row)

Time: 209.097 ms
```

Limitations: Network traffic

```
primary=# select count(*) from transactions_local;
  count
---------
 1095336
(1 row)

Time: 209.097 ms
primary=# select count(*) from transactions_primary;
  count
---------
 1095336
(1 row)

Time: 2867.385 ms
```

Limitations: Dumb queries

Limitations: Dumb queries

```
primary=# explain verbose select count(*) from transactions;
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------
 Aggregate  (cost=1767.38..1767.39 rows=1 width=0)
   Output: count(*)
   ->  Append  (cost=100.00..1699.12 rows=27304 width=0)
         ->  Foreign Scan on public.transactions_node0
               (cost=100.00..212.39 rows=3413 width=0)
                Remote SQL: SELECT NULL FROM public.transactions
         ...
```

Limitations: Dumb queries

```
primary=# explain verbose select count(*) from transactions;
                                                QUERY PLAN
-----------------------------------------------------------------------------------
 Aggregate  (cost=1767.38..1767.39 rows=1 width=0)
   Output: count(*)
   ->  Append  (cost=100.00..1699.12 rows=27304 width=0)
         ->  Foreign Scan on public.transactions_node0
               (cost=100.00..212.39 rows=3413 width=0)
                Remote SQL: SELECT NULL FROM public.transactions
         ...
primary=# explain verbose select avg(int_value) from transactions;
                                                QUERY PLAN
-----------------------------------------------------------------------------------
 Aggregate  (cost=1545.60..1545.61 rows=1 width=8)
   Output: avg(transactions_node0.int_value)
   ->  Append  (cost=100.00..1494.40 rows=20480 width=8)
         ->  Foreign Scan on public.transactions_node0
               (cost=100.00..186.80 rows=2560 width=8)
                Output: transactions_node0.int_value
                   Remote SQL: SELECT int_value FROM public.transactions
         ...
```

Limitations: Dumb queries

```
select type, count(*)
from users
group by type
order by 2 desc;
```

Limitations: Joins

Limitations: Joins

```
select count(*)
from transactions t, users u
where t.user_id = u.id
 and u.type = 'mistaken';
```

Limitations: Keys

Limitations: Keys

'Nuff said

Limitations: Constraint exclusion

Remember this?

```
ERROR:  constraints are not supported on foreign tables
LINE 6:         check ((id % 8) = 0)) server node0 ....
```

Limitations: Single-threaded executer

Limitations: Single-threaded executer

How many nodes do you have?

Limitations: Single-threaded executer

How many nodes do you have?
Do you know what they're doing?

Strategies

- Large working set, small nodes
- Node-level partitioning
- Heavy distributed processing
- Multi-head

Strategy: Large working set, small nodes

Strategy: Large working set, small nodes

- Your working set is larger than one node's RAM

Strategy: Large working set, small nodes

- Your working set is larger than one node's RAM
- ... but you have lots of nodes

Strategy: Large working set, small nodes

- Your working set is larger than one node's RAM
- ... but you have lots of nodes
- (and network is faster than disk)

Strategy: Large working set, small nodes

- Your working set is larger than one node's RAM
- ... but you have lots of nodes
- (and network is faster than disk)
- This might be worth looking into if you're on AWS, but please, please test it first

Strategy: Node-level partitioning

Like parititioning, but with a separate node per partition group!

Strategy: Node-level partitioning

Like parititioning, but with a separate node per partition group! As a total strategy, this is probably not worthwhile. However, it can work with a fast "current data" node combining with slower "archived data" nodes.

Heavy distributed processing

Heavy distributed processing

- Take advantage of lots of CPUs

Heavy distributed processing

- Take advantage of lots of CPUs
- Works well when you have node-discrete workloads

Heavy distributed processing

- Take advantage of lots of CPUs
- Works well when you have node-discrete workloads
- Lock management can become a bit hairier

Heavy distributed processing

- Take advantage of lots of CPUs
- Works well when you have node-discrete workloads
- Lock management can become a bit hairier
- This might actually be a useful use case

Multi-headed

Multi-headed

- Like replication, but with no overhead or delay!

Multi-headed

- Like replication, but with no overhead or delay!
- Also, no storage overhead!

Multi-headed

- Like replication, but with no overhead or delay!
- Also, no storage overhead!
- Might work well with the distributed processing setup

Multi-headed

- Like replication, but with no overhead or delay!
- Also, no storage overhead!
- Might work well with the distributed processing setup
- In fact, given the overhead that lands on the head node, it might be necessary for a working FDW federation setup

Pan-Strategy Advice

Pan-Strategy Advice

- Think very carefully about what tables should live where

Pan-Strategy Advice

- Think very carefully about what tables should live where
- Think very carefully about tuning settings (especially on your head node)
  - work_mem
  - shared_buffers
  - temp_buffers

Pan-Strategy Advice

- Think very carefully about what tables should live where
- Think very carefully about tuning settings (especially on your head node)
  - work_mem
  - shared_buffers
  - temp_buffers
- Think very carefully about how many data nodes you want

Pan-Strategy Advice

- Think very carefully about what tables should live where
- Think very carefully about tuning settings (especially on your head node)
  - work_mem
  - shared_buffers
  - temp_buffers
- Think very carefully about how many data nodes you want
- Think very carefully about network vs. disk vs. dumb-query costs

Pan-Strategy Advice

- Think very carefully about what tables should live where
- Think very carefully about tuning settings (especially on your head node)
  - work_mem
  - shared_buffers
  - temp_buffers
- Think very carefully about how many data nodes you want
- Think very carefully about network vs. disk vs. dumb-query costs
- Think very carefully!

Thanks!

Questions?

Any questions?

Questions?

Any questions?
John, do you use this approach for your databases?

Questions?

Any questions?
John, do you use this approach for your databases?
Why not?

Thanks!

Thanks!

Plug: Stephen Frost has another postgres_fdw talk tomorrow

Thanks!

Plug: Stephen Frost has another postgres_fdw talk tomorrow
Also: Rentrak is hiring: programmers, sysadmins, and devops

Federating Queries Using postgres_fdw