New and cool in PostgreSQL

ConFoo 2016
Montreal, Canada

Magnus Hagander
magnus@hagander.net

Magnus Hagander

- Redpill Linpro
  - Infrastructure services
  - Principal database consultant
- PostgreSQL
  - Core Team member
  - Committer
  - PostgreSQL Europe

PostgreSQL

The World's Most Advanced Open Source Database

Yeah?

PostgreSQL

- Long history
  - Berkeley Postgres: 1986
  - Postres95: 1994
  - PostgreSQL: 1996

PostgreSQL

- High speed of development
- No longer just "catchup"
- Many brand new things
- Some catchup too, of course
  - Oracle's been around since 1978-1979...

PostgreSQL

- Releases approx 1 / year
- 5 year support lifecycle
    - Current: 9.5 (Jan 2016)
    - Oldest: 9.1 (Sep 2011)

What's new and cool

- Big things in every version
- Let's pick a couple
- Across 9.2, 9.3 and 9.4 (mainly)
- Mix of developer and DBA
  - Mostly developer today!

Foreign Data Wrappers

Foreign Data Wrappers

- Access data in remote databases
- As regular tables

postgres_fdw

- No more dblink required (still supported!)
- Access remote PostgreSQL servers "properly"
- Supports remote cost estimates
- Pushes down quals (when possible)
  - 9.6 will push down joins

Writeable FDWs

- Ability to update foreign tables
- INSERT/UPDATE/DELETE
- Transaction aware (*of course*)
- Can be slow for complicated updates/deletes
- Requires FDW specific support

Foreign Data Wrappers

```
CREATE SERVER remotepg
 FOREIGN DATA WRAPPER postgres_fdw
 OPTIONS (host 'localhost', dbname 'pagila', port '5432')
```

```
CREATE USER MAPPING FOR mha SERVER remotepg
```

```
CREATE FOREIGN TABLE actor (
  actor_id int, first_name varchar(45),
  last_name varchar(45), last_update timestamp
)
SERVER remotepg
```

Foreign Data Wrappers

```
postgres=# select * FROM actor WHERE first_name='BOB';
 actor_id | first_name | last_name |        last_update
----------+------------+-----------+----------------------------
       19 | BOB        | FAWCETT   | 2012-03-15 14:53:16.211411
(1 row)

postgres=# UPDATE actor SET first_name='BOBBY'
postgres=-   WHERE first_name='BOB';
UPDATE 1

postgres=# SELECT count(*) FROM actor INNER JOIN localnames
postgres-#  ON actor.first_name=localnames.first_name;
 count
-------
     2
(1 row)
```

Other FDWs

- csv files
- Oracle
- MySQL
- Redis
- MongoDB
- ODBC
- ...

Range types

Range types

- Store ranges of *something*
  - Generic framework
  - Built-in for int, numeric, timestamp, date
- **Query** ranges of *something*
- **Constrain** ranges of *something*

Storing ranges

Traditional way

```
CREATE TABLE meetings (
   start_time timestamptz NOT NULL,
   end_time timestamptz NOT NULL,
   room int NOT NULL REFERENCES rooms,
   title text NOT NULL
);
```

Storing ranges

Using range types

```
CREATE TABLE meetings_r (
   blocktime tstzrange NOT NULL,
   room int NOT NULL REFERENCES rooms,
   title text NOT NULL
);
```

Inserting range values

```
INSERT INTO meetings VALUES (
  '2015-06-11 14:00', '2015-06-11 15:00',
  1, 'First meeting')
```

```
INSERT INTO meetings_r VALUES (
  tstzrange('2015-06-11 14:00', '2015-06-11 15:00'),
  1, 'First meeting')
```

Querying ranges

- Find any overlapping entries
  - Is the conference room free?
  - Let's say from 14:30-15:30
- Rapidly becomes complicated

Querying ranges

```
SELECT * FROM meetings
WHERE room = 1 AND (
  '2015-06-11 14:30' <= start_time AND
  '2015-06-11 15:30' >= start_time AND
  '2015-06-11 14:30' <= end_time AND
  '2015-06-11 15:30' <= end_time
)
OR (
...
```

- And what about open/closed ranges?

Querying ranges

```
SELECT * FROM meetings_r
WHERE room=1 AND
  blocktime &&
    tstzrange('2015-06-11 14:30','2015-06-11 15:30')
```

Querying ranges

- *&&* is an indexable operator!
- GiST and SP-GiST indexes
- Including multi-key!

Constraining ranges

- Concurrency for check
- Is the room available or not?
- But someone books it while we're typing!
- Can use EXCLUSION constraints

Constraining ranges

```
ALTER TABLE meetings_r
  ADD CONSTRAINT no_double_bookings
  EXCLUDE USING GIST (
    room WITH =,
    blocktime WITH &&
  )
```

Over to queries

Ordered set aggregates

Ordered set aggregates

- "Offset in group" aggregates
- *WITHIN GROUP*
- Also *hypothetical aggregates*

Ordered set aggregates

Most common value in group

```
SELECT a,
       mode() WITHIN GROUP (ORDER BY b)
FROM agg GROUP BY a
```

Ordered set aggregates

Percentiles

```
SELECT a,
       percentile_cont(0.3) WITHIN GROUP (ORDER BY b),
       percentile_disc(0.3) WITHIN GROUP (ORDER BY b)
FROM agg GROUP BY a
```

Ordered set aggregates

Hypothetical rows

```
SELECT a,
       rank(4) WITHIN GROUP (ORDER BY b),
       percent_rank(4) WITHIN GROUP (ORDER BY b)
FROM agg GROUP BY a
```

Unstructured data

JSON

JSONB

- "Binary json"
- Parsed JSON data
- Previous json datatype just stores text
- Basic datatyping

JSONB

- Key-independent indexes
- Nested structure support
- Containment operators (and others)

JSONB

- Just another datatype
- Max 1GB, automatically compressed

```
CREATE TABLE jsontable (
    .. columns ..,
    j JSONB
)
```

JSONB

```
INSERT INTO jsontable (..., j)
VALUES (...,
'{"name": "Magnus",
  "skills": {
   "database": ["sql", "postgres"],
   "other": ["something"],
  }
}')
```

JSONB operators

- Key and sub-object access is easy

```
SELECT j->>'name' FROM jsontable
```

```
SELECT j->'skills' FROM jsontable
```

```
SELECT j->'skills'->'database' FROM jsontable
```

```
SELECT j#>'{skills,database,0}' FROM jsontable
```

JSONB searching

- Easy key searching
- Requires key per index
  - Similar to MongoDB etc

```
SELECT * FROM jsontable
WHERE j->>'name' = 'Magnus'
```

- But you didn't need *jsonb* for that

JSONB searching

- Path operators are more powerful!

```
SELECT * FROM jsontable
  WHERE j @> '{"name":"Magnus"}'
```

- Support for deeper paths

```
SELECT * FROM jsontable
  WHERE j @> '{"skills": {
   "database": ["postgresql"]
   }
}'
```

JSONB indexing

- Generic jsonb index

```
CREATE INDEX json_idx
  ON jsontable USING gin(j)
```

- Path operator only index

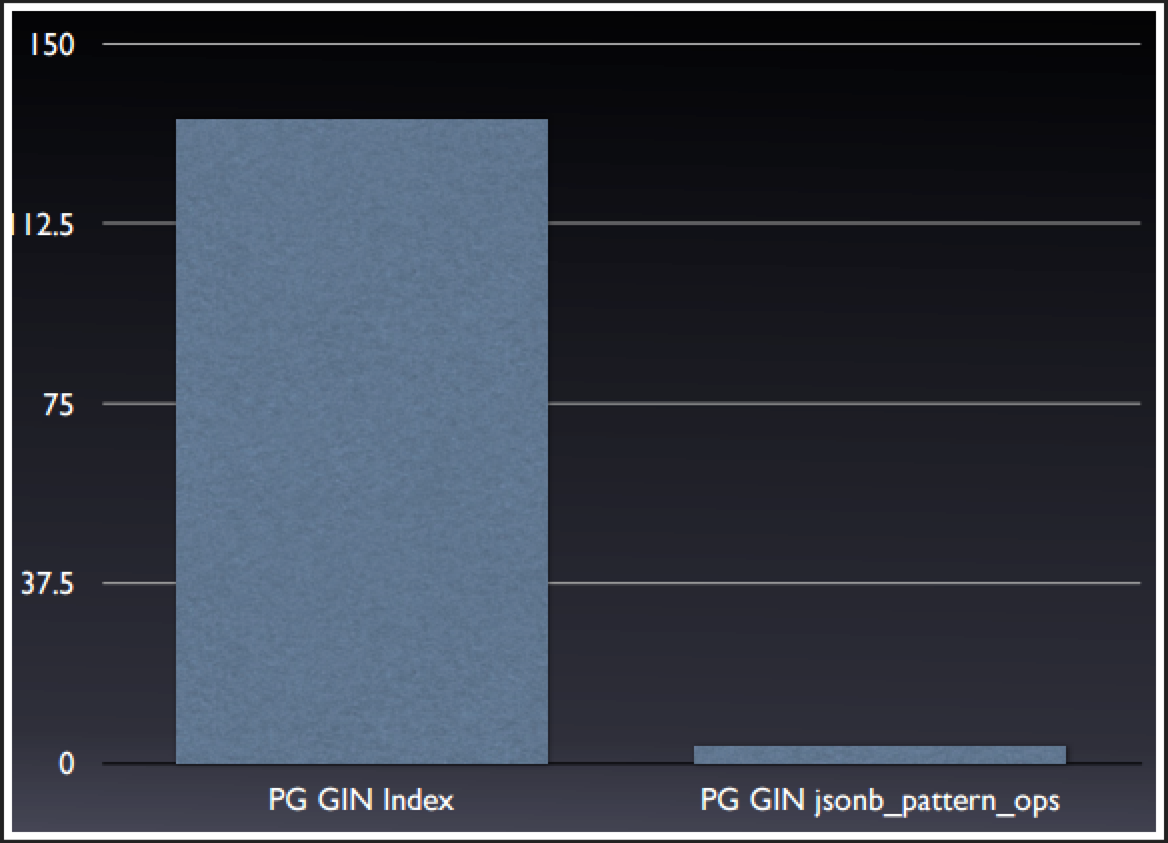
```
CREATE INDEX json_idx
  ON jsontable USING gin(j jsonb_path_ops)
```

JSONB indexes

- Very efficient indexes!
- Small!
- Fast (especially jsonb_path_ops)
- Some benchmarks:
    - 1 million rows
    - 200 fields in json
    - (Thanks Christophe Pettus!)

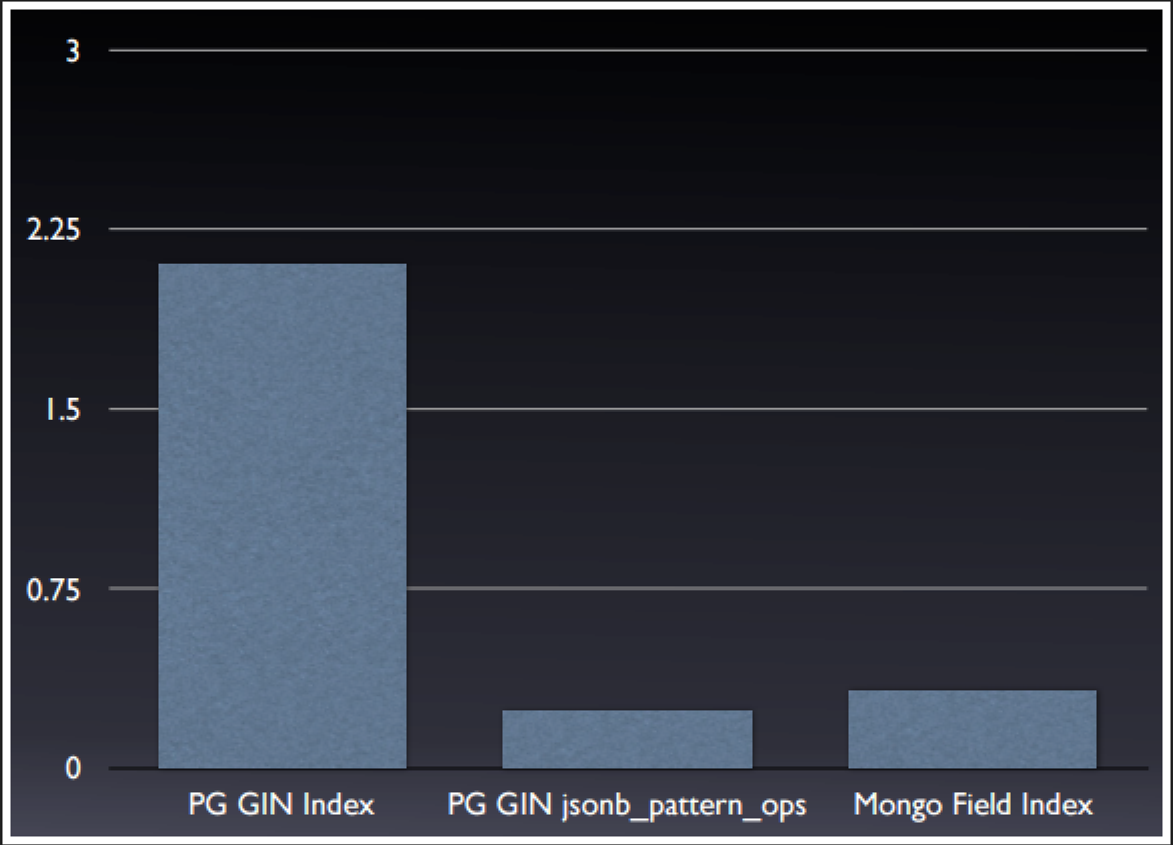
JSONB indexes

- avg query time (ms)



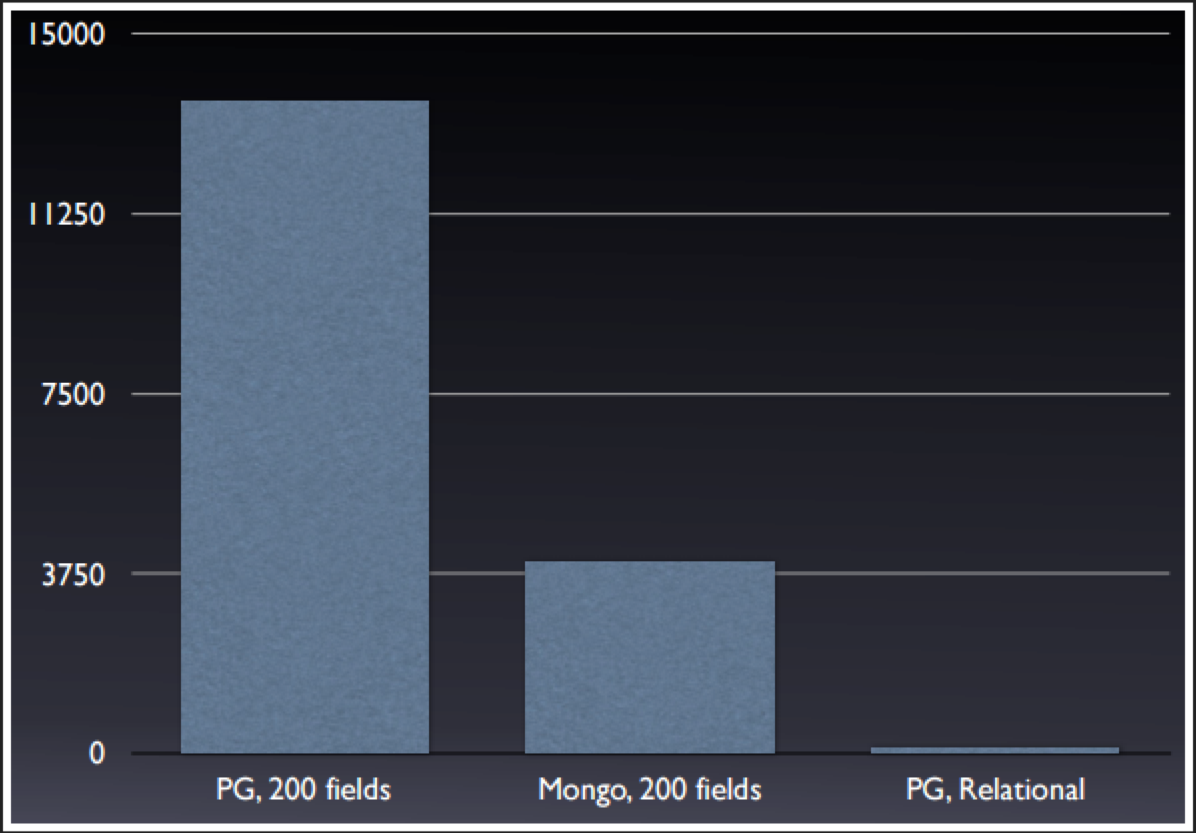
JSONB indexes

- avg query time (ms)



Relational still wins

- avg query time (ms)



Be smart!

- Relational still faster
- *If* your data fits relational model
- Combine both!
- Known fields get a column
- Dynamic fields share a *jsonb*

Summary

New and cool in PostgreSQL

- Plenty of new things!
- Plenty of cool things!
- Go try it out!

http://www.postgresql.org/download/

Thank you!

Magnus Hagander
magnus@hagander.net
@magnushagander
http://www.hagander.net/talks/