PostgreNoSQL

OpenSource Days 2013
Copenhagen, Denmark

Magnus Hagander
magnus@hagander.net

Magnus Hagander

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

PostgreSQL

- Relational database
- "Old world"
- Nobody wants SQL?

Nobody wants SQL

- Really?

What do people actually dislike?

- SQL
- Or relational
- Or "ACID guarantees"
- or...?

What can we do

- We can't take away SQL
- But we can minimize use of it
- Still very useful for many things
  - Even in typical NoSQL scenarios

ACID guarantees

- By default, PostgreSQL guarantees everything
- Turning off fsync risks your data
  - Almost guaranteed data corruption
  - Hard to detect, even harder to fix

Almost-ACID

- Instead, use synchronous_commit=off
- Guarantees data integrity
- Guarantees data consistency
- But may loose small amounts of durability

Eventual consistency

- Often used for write scaling
- Don't really care about data consistency
- We don't do this at this point
  - There are people experimenting...

Relational model

- Tables and relations
  - And scary joins!
- Does not always match application model
  - Though more often than you think
- Something we can work with

Potential issues with relational

- Relational model has columns
  - Need to know which
  - Very sparse data sets are bad
- Anti-solution: EAV
  - Please don't go there

Dealing with non-relational

- key/value store
- General schemaless data
  - Put all requirements on application
- Mix and match!

hstore

- Generic key-value store
- Fully indexable!
- Typeless
- Available in all supported versions

Installing hstore

```
postgres=# CREATE EXTENSION hstore;
CREATE EXTENSION
```

Defining hstore columns

```
postgres=# CREATE TABLE items (
postgres(#  itemid serial NOT NULL PRIMARY KEY,
postgres(#  itemname text NOT NULL,
postgres(#  tags hstore);
CREATE TABLE
```

Creating hstore values

```
postgres=# INSERT INTO items (itemname, tags)
postgres-# VALUES ('item1', 'color => red, category => stuff');
INSERT 0 1
```

Query by hstore

```
postgres=# SELECT itemname FROM items
postgres-# WHERE tags->'color' = 'red';
 item1
```

Indexed access

- Create normal expression index on column

```
CREATE INDEX foo ON
  ((items->'color'))
```

- Requires one index per key
- That's what we wanted to avoid...

Dynamic GiST indexing

- Create index covering all keys

```
CREATE INDEX hstoreidx
 ON items
 USING gist(tags)
```

- Available for multiple operators
  - All types of containment
- Must use these operators

Querying with GiST

```
postgres=# EXPLAIN
postgres-# SELECT itemname FROM items
postgres-# WHERE tags @> 'color=>red';

 Index Scan using hstoreidx on items  (cost=0.12..8.14 rows=1 width=32)
   Index Cond: (tags @> '"color"=>"red"'::hstore)
```

Querying for tag presence

```
postgres=# EXPLAIN
postgres-# SELECT itemname FROM items
postgres-# WHERE tags ? 'color';

 Index Scan using hstoreidx on items  (cost=0.12..8.14 rows=1 width=32)
   Index Cond: (tags ? 'color'::text)
```

Downsides of hstore

- Values are not typed
  - Just strings
- No hierarchy
- No key compression
- Still slower than "normal columns"
  - But very useful with sparse data!

Fully schemaless

- "Document storage"
- And of course, processing
- "Everybody uses JSON"
  - Yup, they stopped using XML!

JSON

- JavaScript Object Notation
- Text-based data
- Schemaless
- Hierarchical
- PostgreSQL has native support (since 9.2)!

JSON in PostgreSQL

```
CREATE TABLE jsontable (
   id serial PRIMARY KEY,
   j json
);
```

Storing JSON

```
postgres=# INSERT INTO jsontable (j) VALUES ('{
postgres'#  "id":"mha",
postgres'#  "name":"Magnus Hagander",
postgres'#  "country": "Sweden"
postgres'# }');
INSERT 0 1
```

- Validates json syntax
- Maintains formatting

Mapping JSON

```
postgres=# SELECT row_to_json(schedule)
postgres-# FROM schedule WHERE id=1;
 {"id":1,"employee_id":1,"t":"[\"2013-02-08 13:00:00+00\",
   \"2013-02-08 17:00:00+00\")"}
```

Mapping JSON

```
postgres=# SELECT row_to_json(schedule)
postgres-# FROM schedule WHERE id=1;
 {"id":1,"employee_id":1,"t":"[\"2013-02-08 13:00:00+00\",
   \"2013-02-08 17:00:00+00\")"}

postgres=# SELECT row_to_json(t) FROM (
postgres-#  SELECT id, employee_id
postgres-#  FROM schedule) t;
 {"id":2,"employee_id":1}
 {"id":3,"employee_id":2}
 {"id":1,"employee_id":1}
```

Using JSON

- That's really all there is to JSON
  - At least in 9.2
- For full power, use with pl/v8

PL/V8

- Combines PostgreSQL and V8
- Outside extension

```
postgres=# CREATE EXTENSION plv8;
CREATE EXTENSION
```

JSON extraction

```
CREATE or replace FUNCTION jmember (j json, key text )
 RETURNS text LANGUAGE plv8 IMMUTABLE
AS $function$
  if (typeof j != 'object')
      return NULL;
  return JSON.stringify(j[key]);
$function$;
```

JSON extraction

```
v8test=# SELECT jmember(j, 'name')
v8test-# FROM jsontable;
 "Magnus Hagander"
```

JSON indexing

```
v8test=# CREATE INDEX idx_nameid
v8test-# ON jsontable (jmember(j, 'id'));
CREATE INDEX
```

JSON indexing

```
v8test=# EXPLAIN
v8test-# SELECT * FROM jsontable
v8test-# WHERE jmember(j, 'id') = 'mha';

 Index Scan using idx_nameid on jsontable  (cost=0.38..8.39 rows=1 width=36)
   Index Cond: (jmember(j, 'id'::text) = 'mha'::text)
```

JSON indexing

- Still need one index per key
- Can use arbitrary expression
  - Including processing hierarchical data
- Can transform key on lookup as well

Javascript functions

- Uses "javascript way"
- E.g., subtransactions by:

```
CREATE OR REPLACE FUNCTION public.st()
 RETURNS void LANGUAGE plv8
AS $function$
plv8.subtransaction(function() {
  plv8.execute("INSERT INTO tbl VALUES(1)");
  plv8.execute("INSERT INTO tbl VALUES(2)");
});
$function$
```

JSON/JS future in PostgreSQL

- More built-in operators for JSON
  - Likely showing up in 9.3
  - Extraction etc
- No current plans to include PL/V8 in core
  - C++, etc
  - But we like extensions!

JSON/JS future in PostgreSQL

- Generic JSON indexing
- With hierarchy support
- No code yet
- Plans are in progress
- Not in 9.3, but maybe 9.4

Thank you!

Magnus Hagander
magnus@hagander.net
@magnushagander