THE NOSQL STORE EVERYONE IGNORED

BY ZOHAIB SIBTE HASSAN @ DOORDASH

ABOUT ME

Zohaib Sibte Hassan

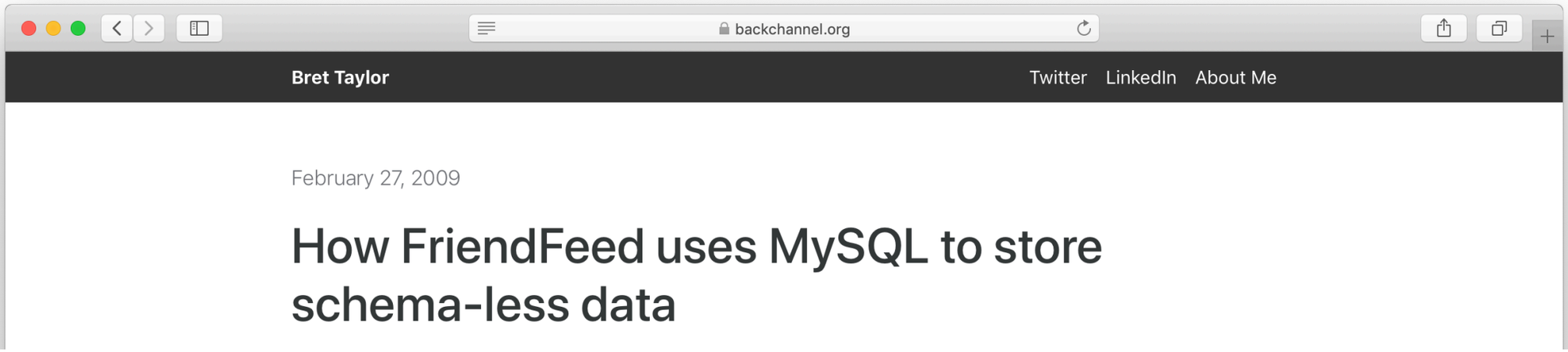
@zohaibility

Dad, engineer, hacker, philosopher, troublemaker, love open source!

EVERYTHING NOSQL WAS A HYPE

HISTORY

2009 - Friend Feed blog



HISTORY

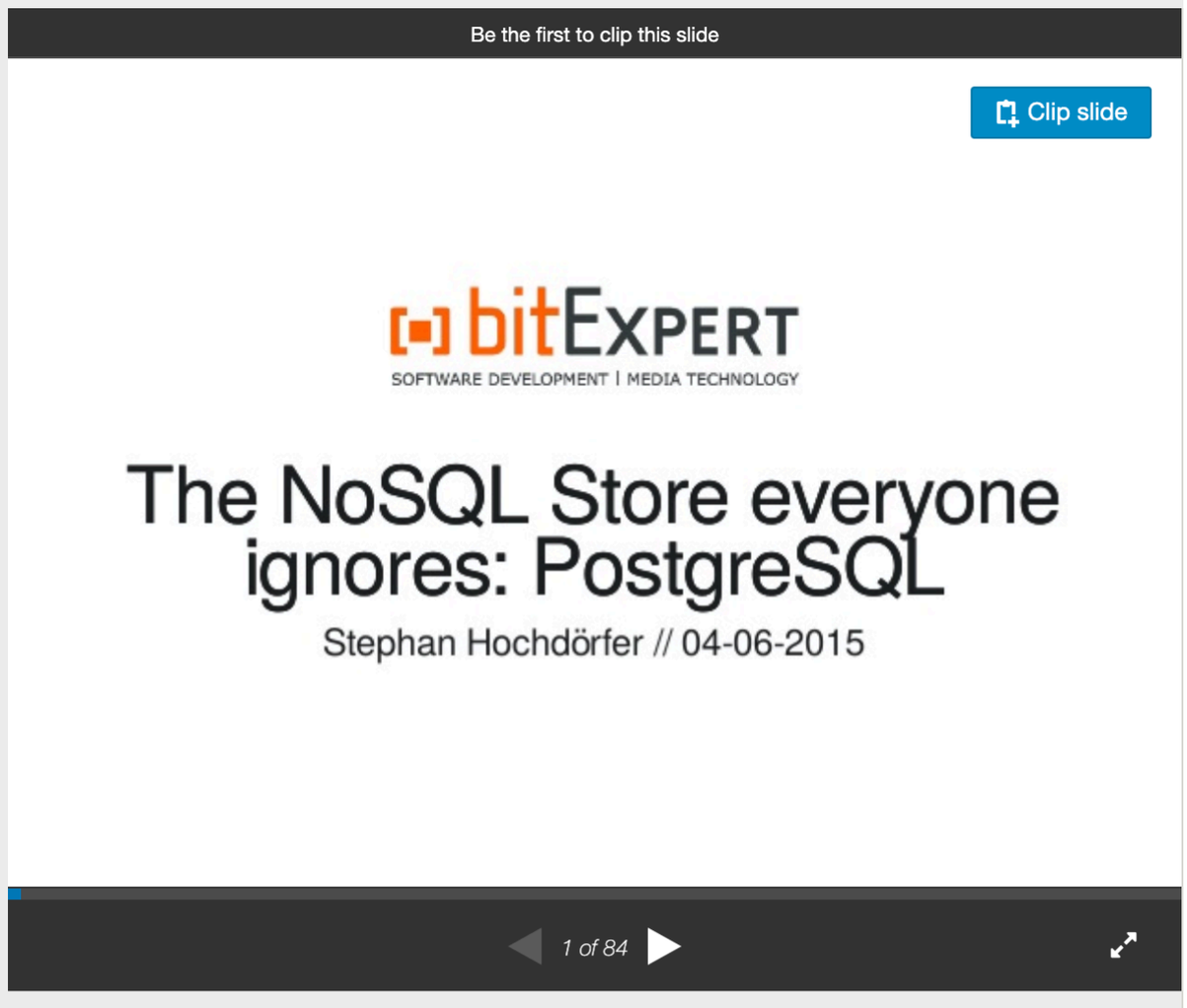
2011 - Discovered HSTORE and blogged about it

HISTORY

2012 - Revisited imagining FriendFeed on Postgres & HSTORE

HISTORY

2015 - Talk with same title in Dublin



Stefan Hochdörfer - The NoSQL Store everyone ignores: PostgreSQL - NoSQL matters Dublin 2015

988 views

HISTORY

2016 - Uber talks about how they built a schema-less store

Architecture

Designing Schemaless, Uber Engineering's Scalable Datastore Using MySQL

Jakob Holdgaard Thomsen January 12, 2016

OUR ROADMAP TODAY

- A brief look at FriendFeed use-case
- Warming up with HSTORE
- Taking it to next level:
  - JSONB
  - Complex yet simple queries
  - Partitioning our documents

POSTGRES HAS EVOLVED

- Robust schemaless-types:
  - Array
  - HSTORE
  - XML
  - JSON & JSONB
- Improved storage engine
- Improved Foreign Data Wrappers
- Partitioning support

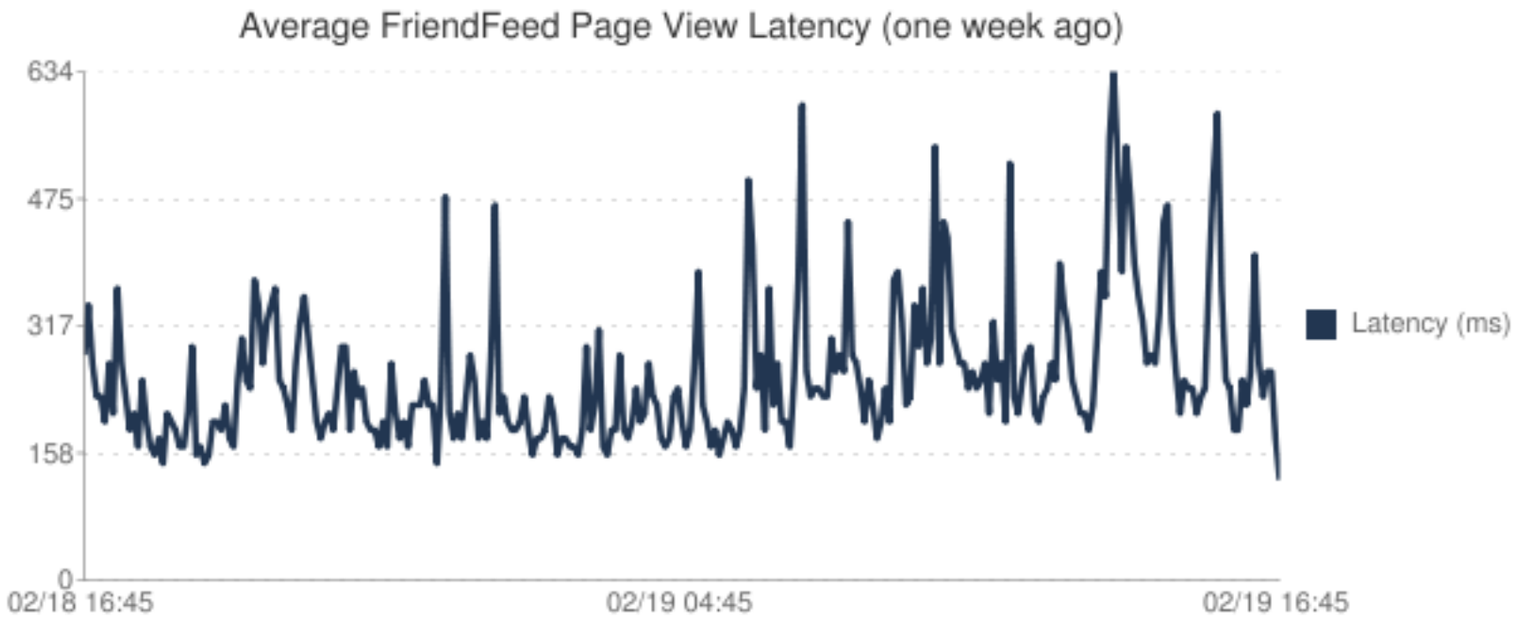
USING SQL TO BUILD NOSQL

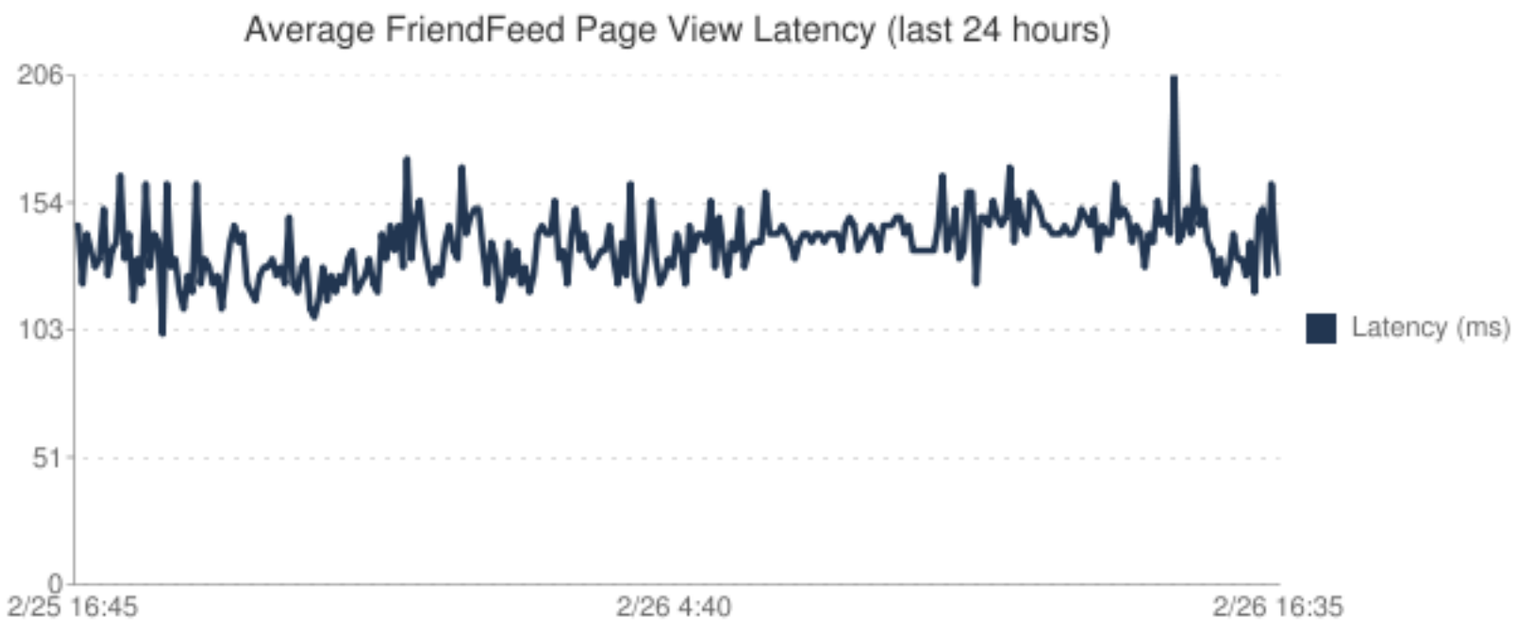
- https://backchannel.org/blog/friendfeed-schemaless-mysql

WHY FRIENDFEED?

- Good example of understanding available technology and problem at hand.
- Did not cave in to buzzword, and started using something less known/reliable.
- Large scale problem with good example on how modern SQL tooling solves the problem.
- Using tool that you are comfortable with.
- Read blog post!

WHY FRIENDFEED?





FRIENDFEED


```
{
    "id": "71f0c4d2291844cca2df6f486e96e37c",
    "user_id": "f48b0440ca0c4f66991c4d5f6a078eaf",
    "feed_id": "f48b0440ca0c4f66991c4d5f6a078eaf",
    "title": "We just launched a new backend system for FriendFeed!",
    "link": "http://friendfeed.com/e/71f0c4d2-2918-44cc-a2df-6f486e96e37c",
    "published": 1235697046,
    "updated": 1235697046,
}
```

FRIENDFEED

```
CREATE TABLE entities (
    added_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    id BINARY(16) NOT NULL,
    updated TIMESTAMP NOT NULL,
    body MEDIUMBLOB,
    UNIQUE KEY (id),
    KEY (updated)
) ENGINE=InnoDB;
```

FRIENDFEED INDEXING

- Create tables for each indexed field.
- Have background workers to populate newly created index.
- Complete language framework to ensure documents are indexed as they are inserted.

```
CREATE TABLE index_user_id (
    user_id BINARY(16) NOT NULL,
    entity_id BINARY(16) NOT NULL UNIQUE,
    PRIMARY KEY (user_id, entity_id)
) ENGINE=InnoDB;
```

CODING FRAMEWORK

```
user_id_index = friendfeed.datastore.Index(
    table="index_user_id", properties=["user_id"], shard_on="user_id")
datastore = friendfeed.datastore.DataStore(
    mysql_shards=["127.0.0.1:3306", "127.0.0.1:3307"],
    indexes=[user_id_index])

new_entity = {
    "id": binascii.a2b_hex("71f0c4d2291844cca2df6f486e96e37c"),
    "user_id": binascii.a2b_hex("f48b0440ca0c4f66991c4d5f6a078eaf"),
    "feed_id": binascii.a2b_hex("f48b0440ca0c4f66991c4d5f6a078eaf"),
    "title": u"We just launched a new backend system for FriendFeed!",
    "link": u"http://friendfeed.com/e/71f0c4d2-2918-44cc-a2df-6f486e96e37c",
    "published": 1235697046,
    "updated": 1235697046,
}
datastore.put(new_entity)
entity = datastore.get(binascii.a2b_hex("71f0c4d2291844cca2df6f486e96e37c"))
entity = user_id_index.get_all(datastore, user_id=binascii.a2b_hex("f48b0440ca0c4
```

THE KEY-VALUE STORE EVERYONE IGNORED

HSTORE

Branch: master ▾ **postgres** / contrib / **hstore** /

Create new file | Upload files | Find file | History

michaelpq Fix inconsistencies and typos in the tree, take 11 ···

Latest commit c96581a 19 days ago

..

data	Add GIN support for pg_trgm. From Guillaume Smet <guillaume.smet@gmai...	13 years ago

HSTORE

```
CREATE TABLE feed (
  id        varchar(64) NOT NULL PRIMARY KEY,
  doc       hstore
);
```

HSTORE

```
INSERT INTO feed VALUES (
  'ff923c93-7769-4ef6-b026-50c5a87a79c5',
  'id⇒zohaibility, post⇒hello'::hstore
);
```

HSTORE

```
SELECT doc→'post' as post, doc→'undefined_field' as should_be_null
FROM feed
WHERE doc→'id' = 'zohaibility';
```

```
 post  | should_be_null
-------+----------------
 hello |
(1 row)
```

HSTORE

```sql
EXPLAIN SELECT *
FROM feed
WHERE doc→'id' = 'zohaibility';
```

```
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on feed  (cost=0.00..1.03 rows=1 width=178)
   Filter: ((doc → 'id'::text) = 'zohaibility'::text)
(2 rows)
```

HSTORE

```
CREATE INDEX feed_user_id_index
ON feed ((doc→'id'));
```

HSTORE ❤️ GIST

```
CREATE INDEX feed_gist_idx
ON feed
USING gist (doc);
```

HSTORE ❤️ GIST

```
SELECT doc→'post' as post, doc→'undefined_field' as undefined
FROM feed
WHERE doc @> ‘id⇒zohaibility';
```

```
 post  | undefined
-------+-----------
 hello |
(1 row)
```

MORE OPERATORS!

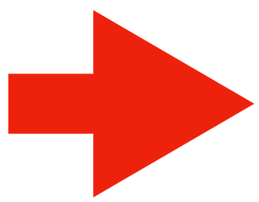
https://www.postgresql.org/docs/current/hstore.html

Operator	Description	Example	Result
`hstore -> text`	get value for key (`NULL` if not present)	`'a=>x, b=>y'::hstore -> 'a'`	`x`
`hstore -> text[]`	get values for keys (`NULL` if not present)	`'a=>x, b=>y, c=>z'::hstore -> ARRAY['c','a']`	`{"z","x"}`
`hstore || hstore`	concatenate `hstore`s	`'a=>b, c=>d'::hstore || 'c=>x, d=>q'::hstore`	`"a"=>"b", "c"=>"x", "d"=>"q"`
`hstore ? text`	does `hstore` contain key?	`'a=>1'::hstore ? 'a'`	`t`
`hstore ?& text[]`	does `hstore` contain all specified keys?	`'a=>1,b=>2'::hstore ?& ARRAY['a','b']`	`t`
`hstore ?| text[]`	does `hstore` contain any of the specified keys?	`'a=>1,b=>2'::hstore ?| ARRAY['b','c']`	`t`
`hstore @> hstore`	does left operand contain right?	`'a=>b, b=>1, c=>NULL'::hstore @> 'b=>1'`	`t`
`hstore <@ hstore`	is left operand contained in right?	`'a=>c'::hstore <@ 'a=>b, b=>1, c=>NULL'`	`f`
`hstore - text`	delete key from left operand	`'a=>1, b=>2, c=>3'::hstore - 'b'::text`	`"a"=>"1", "c"=>"3"`
`hstore - text[]`	delete keys from left operand	`'a=>1, b=>2, c=>3'::hstore - ARRAY['a','b']`	`"c"=>"3"`
`hstore - hstore`	delete matching pairs from left operand	`'a=>1, b=>2, c=>3'::hstore - 'a=>4, b=>2'::hstore`	`"a"=>"1", "c"=>"3"`
`record #= hstore`	replace fields in `record` with matching values from `hstore`	see Examples section	
`%% hstore`	convert `hstore` to array of alternating keys and values	`%% 'a=>foo, b=>bar'::hstore`	`{a,foo,b,bar}`
`%# hstore`	convert `hstore` to two-dimensional key/value array	`%# 'a=>foo, b=>bar'::hstore`	`{{a,foo},{b,bar}}`

REIMAGINING FRIENDFEED

```
CREATE TABLE entities (
    id BIGINT PRIMARY KEY,
    updated TIMESTAMP NOT NULL,
    body HSTORE,
    …
);
```

```
CREATE TABLE index_user_id (
    user_id BINARY(16) NOT NULL,
    entity_id BINARY(16) NOT NULL UNIQUE,
    PRIMARY KEY (user_id, entity_id)
) ENGINE=InnoDB;
```



```
CREATE INDEX CONCURRENTLY entity_id_index
ON entities ((body→'entity_id'));
```

JSONB

TO INFINITY AND BEYOND

WHY JSON?

- Well understood, and goto standard for almost everything on modern web.
- “Self describing”, hierarchical, and parsing and serialization libraries for every programming language
- Describes a loose shape of the object, which might be necessary in some cases.

TWEETS

- JSON
  - id : 1155874848175984600
  - geo : null
  - lang : "en"
  - text : "With @SeleniumHQ (the industry standard for web #automation), combined with #Python's @pytestdotorg (a top unit… https://t.co/2pA0fM9ZlW"
  - user
  - place : null
  - id_str : "1155874848175984641"
  - source : " Twitter for iPhone"
  - entities
  - favorited : false
  - retweeted : false
  - truncated : true
  - created_at : "Mon Jul 29 16:16:53 +0000 2019"
  - coordinates : null
  - quote_count : 0
  - reply_count : 0
  - contributors : null
  - filter_level : "low"
  - timestamp_ms : "1564417013578"
  - retweet_count : 0
  - extended_tweet
  - favorite_count : 0
  - is_quote_status : false
  - display_text_range
  - possibly_sensitive : false
  - in_reply_to_user_id : null
  - in_reply_to_status_id : null
  - in_reply_to_screen_name : null
  - in_reply_to_user_id_str : null
  - in_reply_to_status_id_str : null

TWEETS TABLE

```
CREATE TABLE tweets (
    id varchar(64) NOT NULL PRIMARY KEY,
    content jsonb NOT NULL
);
```

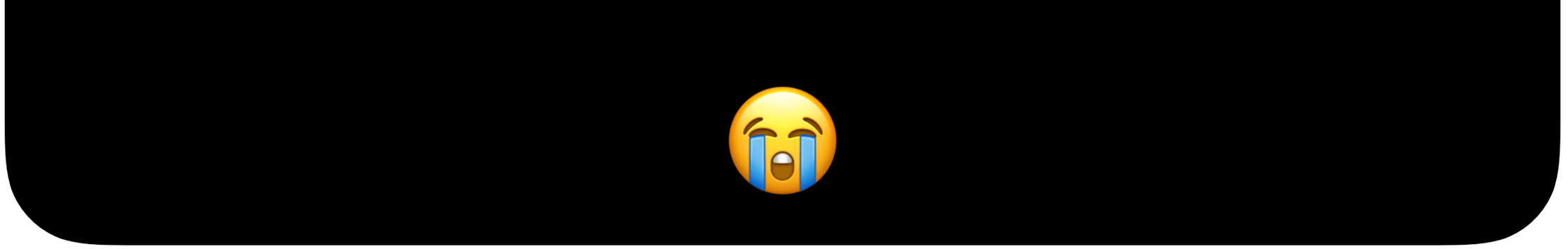
BASIC QUERY

```
SELECT "content"→'text' as txt, "content"→'favorite_count' as cnt
FROM tweets
WHERE “content"→'id_str' == ‘…’
```

AND YES YOU CAN INDEX THIS!!!

PEEKIN INTO STRUCTURE

```sql
SELECT *
FROM tweets
WHERE (content->>'favorite_count')::integer >= 1;
```



```sql
EXPLAIN SELECT *
FROM tweets
WHERE (content→'favorite_count')::integer ⩾ 1;
```

```
                                 QUERY PLAN
-----------------------------------------------------------------------------
 Seq Scan on tweets  (cost=0.00..2453.28 rows=6688 width=718)
   Filter: (((content -»' 'favorite_count'::text))::integer ⩾ 1)
(2 rows)
```

BASIC INDEXING

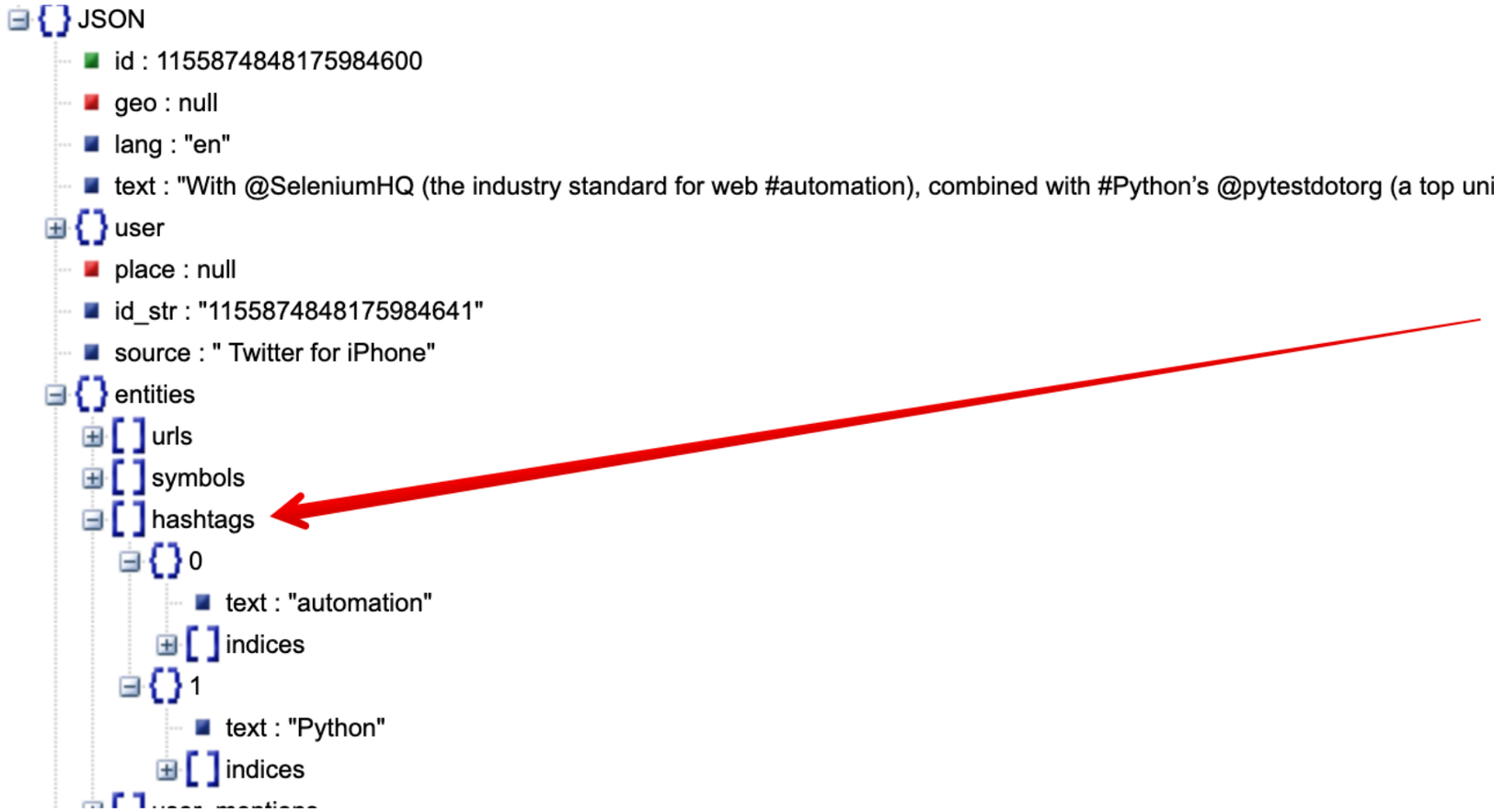
```
CREATE INDEX fav_count_index
ON tweets (((content→'favorite_count')::INTEGER));
```

BASIC INDEXING

```sql
EXPLAIN SELECT *
FROM tweets
WHERE (content->'favorite_count')::integer >= 1;
```

```
                                                                  QUERY PLAN
----------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on tweets  (cost=128.12..2297.16 rows=6688 width=718)
   Recheck Cond: (((content -> 'favorite_count'::text))::integer >= 1)
   ->  Bitmap Index Scan on fav_count_index  (cost=0.00..126.45 rows=6688 width=0)
         Index Cond: (((content -> 'favorite_count'::text))::integer >= 1)
(4 rows)
```

DEEP INTO THE RABBIT HOLE




```sql
SELECT content#>>'{text}' as txt
FROM tweets
WHERE (content#>'{entities,hashtags}') @> '[{"text": "python"}]'::jsonb;
```

JSON OPERATORS

Table 9-40. `json` and `jsonb` Operators

Operator	Right Operand Type	Description	Example	Example Result
`->`	`int`	Get JSON array element (indexed from zero, negative integers count from the end)	`'[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2`	`{"c":"baz"}`
`->`	`text`	Get JSON object field by key	`'{"a": {"b":"foo"}}'::json->'a'`	`{"b":"foo"}`
`->>`	`int`	Get JSON array element as `text`	`'[1,2,3]'::json->>2`	3
`->>`	`text`	Get JSON object field as `text`	`'{"a":1,"b":2}'::json->>'b'`	2
`#>`	`text[]`	Get JSON object at specified path	`'{"a": {"b":{"c": "foo"}}}'::json#>'{a,b}'`	`{"c": "foo"}`
`#>>`	`text[]`	Get JSON object at specified path as `text`	`'{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}'`	3

JSONB OPERATORS

Table 9-41. Additional `jsonb` Operators

Operator	Right Operand Type	Description	Example
`@>`	**`jsonb`**	Does the left JSON value contain the right JSON path/value entries at the top level?	`'{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb`
`<@`	**`jsonb`**	Are the left JSON path/value entries contained at the top level within the right JSON value?	`'{"b":2}'::jsonb <@ '{"a":1, "b":2}'::jsonb`
`?`	**`text`**	Does the string exist as a top-level key within the JSON value?	`'{"a":1, "b":2}'::jsonb ? 'b'`
`?|`	**`text[]`**	Do any of these array strings exist as top-level keys?	`'{"a":1, "b":2, "c":3}'::jsonb ?| array['b', 'c']`
`?&`	**`text[]`**	Do all of these array strings exist as top-level keys?	`'["a", "b"]'::jsonb ?& array['a', 'b']`
`||`	**`jsonb`**	Concatenate two **`jsonb`** values into a new **`jsonb`** value	`'["a", "b"]'::jsonb || '["c", "d"]'::jsonb`
`-`	**`text`**	Delete key/value pair or string element from left operand. Key/value pairs are matched based on their key value.	`'{"a": "b"}'::jsonb - 'a'`
`-`	**`integer`**	Delete the array element with specified index (Negative integers count from the end). Throws an error if top level container is not an array.	`'["a", "b"]'::jsonb - 1`
`#-`	**`text[]`**	Delete the field or element with specified path (for JSON arrays, negative integers count from the end)	`'["a", {"b":1}]'::jsonb #- '{1,b}'`

MATCHING TAGS

```sql
SELECT content#>>'{text}' as txt
FROM tweets
WHERE (content#>'{entities,hashtags}') @> '[{"text": "python"}]'::jsonb;
```

INDEXING

```sql
CREATE INDEX idx_gin_hashtags
ON tweets
USING GIN ((content#>'{entities,hashtags}') jsonb_ops);
```

COMPLEX SEARCH

```
CREATE INDEX idx_gin_rt_hashtags
ON tweets
USING GIN ((content#>'{retweeted_status,entities,hashtags}') jsonb_ops);
```

```
SELECT content#>'{text}' as txt
FROM tweets
WHERE (
  (content#>'{entities,hashtags}') @> '[{"text": “postgres"}]'::jsonb
  OR
  (content#>'{retweeted_status,entities,hashtags}') @> '[{"text": “postgres"}]'::jsonb
);
```

JSONB + ECOSYSTEM

THE POWER OF ALCHEMY

JSONB + TSVECTOR

```sql
CREATE INDEX idx_gin_tweet_text
ON tweets
USING GIN (to_tsvector('english', content->>'text') tsvector_ops);
```

```sql
SELECT content->>'text' as txt
FROM tweets
WHERE to_tsvector('english', content->>'text') @@ to_tsquery('english', 'python');
```

JSONB + PARTITION

```sql
CREATE TABLE part_tweets (
  id       varchar(64) NOT NULL,
  content  jsonb NOT NULL
) PARTITION BY hash (md5(content->'user'->>'id'));

CREATE TABLE part_tweets_0 PARTITION OF part_tweets FOR
VALUES WITH (MODULUS 4, REMAINDER 0);

CREATE TABLE part_tweets_1 PARTITION OF part_tweets FOR
VALUES WITH (MODULUS 4, REMAINDER 1);

CREATE TABLE part_tweets_2 PARTITION OF part_tweets FOR
VALUES WITH (MODULUS 4, REMAINDER 2);

CREATE TABLE part_tweets_3 PARTITION OF part_tweets FOR
VALUES WITH (MODULUS 4, REMAINDER 3);
```

JSONB + PARTITION + INDEXING

```
CREATE INDEX pidx_gin_hashtags ON part_tweets USING GIN
((content#>'{entities,hashtags}') jsonb_ops);

CREATE INDEX pidx_gin_rt_hashtags ON part_tweets USING GIN
((content#>'{retweeted_status,entities,hashtags}') jsonb_ops);

CREATE INDEX pidx_gin_tweet_text ON tweets USING GIN
(to_tsvector('english', content->>'text') tsvector_ops);



INSERT INTO part_tweets SELECT * from tweets;
```

JSONB + PARTITION + INDEXING

```sql
EXPLAIN SELECT content#>'{text}' as txt
FROM part_tweets
WHERE (content#>'{entities,hashtags}') @> '[{"text": "postgres"}]'::jsonb;
```

```
                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=24.26..695.46 rows=131 width=32)
   ->  Bitmap Heap Scan on part_tweets_0  (cost=24.26..150.18 rows=34 width=32)
         Recheck Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "postgres"}]'::jsonb)
         ->  Bitmap Index Scan on part_tweets_0_expr_idx  (cost=0.00..24.25 rows=34 width=0)
               Index Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "postgres"}]'::jsonb)
   ->  Bitmap Heap Scan on part_tweets_1  (cost=80.25..199.02 rows=32 width=32)
         Recheck Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "postgres"}]'::jsonb)
         ->  Bitmap Index Scan on part_tweets_1_expr_idx  (cost=0.00..80.24 rows=32 width=0)
               Index Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "postgres"}]'::jsonb)
   ->  Bitmap Heap Scan on part_tweets_2  (cost=28.25..147.15 rows=32 width=32)
         Recheck Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "postgres"}]'::jsonb)
         ->  Bitmap Index Scan on part_tweets_2_expr_idx  (cost=0.00..28.24 rows=32 width=0)
               Index Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "postgres"}]'::jsonb)
   ->  Bitmap Heap Scan on part_tweets_3  (cost=76.26..198.46 rows=33 width=32)
         Recheck Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "postgres"}]'::jsonb)
         ->  Bitmap Index Scan on part_tweets_3_expr_idx  (cost=0.00..76.25 rows=33 width=0)
               Index Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "postgres"}]'::jsonb)
(17 rows)
```

JSONB + PARTITION + INDEXING

```sql
EXPLAIN SELECT content#>'{text}' as txt FROM tweets WHERE (
  (content#>'{entities,hashtags}') @> '[{"text": "python"}]'::jsonb
  OR
  (content#>'{retweeted_status,entities,hashtags}') @> '[{"text": "python"}]'::jsonb
);
```

```
                                                                                         QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Append  (cost=48.54..1326.10 rows=262 width=32)
   ->  Bitmap Heap Scan on part_tweets_0  (cost=48.54..291.87 rows=68 width=32)
         Recheck Cond: (((content #> '{entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb) OR ((content #> '{retweeted_status,entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb))
         ->  BitmapOr  (cost=48.54..48.54 rows=68 width=0)
               ->  Bitmap Index Scan on part_tweets_0_expr_idx  (cost=0.00..24.25 rows=34 width=0)
                     Index Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb)
               ->  Bitmap Index Scan on part_tweets_0_expr_idx1  (cost=0.00..24.25 rows=34 width=0)
                     Index Cond: ((content #> '{retweeted_status,entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb)
   ->  Bitmap Heap Scan on part_tweets_1  (cost=144.51..370.90 rows=63 width=32)
         Recheck Cond: (((content #> '{entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb) OR ((content #> '{retweeted_status,entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb))
         ->  BitmapOr  (cost=144.51..144.51 rows=63 width=0)
               ->  Bitmap Index Scan on part_tweets_1_expr_idx  (cost=0.00..80.24 rows=32 width=0)
                     Index Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb)
               ->  Bitmap Index Scan on part_tweets_1_expr_idx1  (cost=0.00..64.24 rows=32 width=0)
                     Index Cond: ((content #> '{retweeted_status,entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb)
   ->  Bitmap Heap Scan on part_tweets_2  (cost=52.52..286.00 rows=64 width=32)
         Recheck Cond: (((content #> '{entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb) OR ((content #> '{retweeted_status,entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb))
         ->  BitmapOr  (cost=52.52..52.52 rows=65 width=0)
               ->  Bitmap Index Scan on part_tweets_2_expr_idx  (cost=0.00..28.24 rows=32 width=0)
                     Index Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb)
               ->  Bitmap Index Scan on part_tweets_2_expr_idx1  (cost=0.00..24.24 rows=32 width=0)
                     Index Cond: ((content #> '{retweeted_status,entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb)
   ->  Bitmap Heap Scan on part_tweets_3  (cost=136.54..376.02 rows=67 width=32)
         Recheck Cond: (((content #> '{entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb) OR ((content #> '{retweeted_status,entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb))
         ->  BitmapOr  (cost=136.54..136.54 rows=67 width=0)
               ->  Bitmap Index Scan on part_tweets_3_expr_idx  (cost=0.00..76.25 rows=33 width=0)
                     Index Cond: ((content #> '{entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb)
               ->  Bitmap Index Scan on part_tweets_3_expr_idx1  (cost=0.00..60.25 rows=33 width=0)
                     Index Cond: ((content #> '{retweeted_status,entities,hashtags}'::text[]) @> '[{"text": "python"}]'::jsonb)
(29 rows)
```

LIMIT IS YOUR
IMAGINATION

LINKS & RESOURCES

- https://www.postgresql.org/docs/current/datatype-json.html
- https://www.postgresql.org/docs/current/functions-json.html
- https://www.postgresql.org/docs/current/gin-builtin-opclasses.html
- https://www.postgresql.org/docs/current/ddl-partitioning.html
- https://www.postgresql.org/docs/current/textsearch-tables.html
- https://blog.creapptives.com/post/14062057061/the-key-value-store-everyone-ignored-postgresql
- https://blog.creapptives.com/post/32461917960/migrating-friendfeed-to-postgresql
- https://pgdash.io/blog/partition-postgres-11.html
- https://talks.bitexpert.de/dpc15-postgres-nosql/#/
- https://www.postgresql.org/docs/current/hstore.html
- https://heap.io/blog/engineering/when-to-avoid-jsonb-in-a-postgresql-schema

THANK YOU!
QUESTIONS?

zohaib.hassan@doordash.com

@zohaibility