$ _

Is `'{JSONB}'` a Silver Bullet?

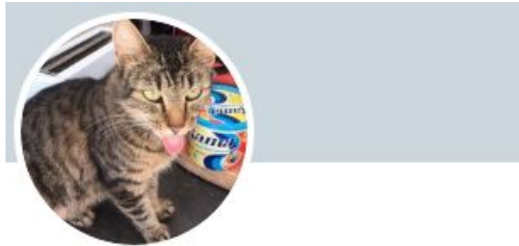
Angus Dippenaar

Who is this guy?

DevOps / Full stack developer at Quant Solutions / GoodX

We make medical / financial software using Python and Postgres

We really like Postgres



Intro

- What is JSON(B)?
- JSON vs JSONB
- Some cool things
- Some limitations
- JSON vs JSONB vs Tables
- When could you use JSONB?
- When not to use JSONB
- Conclusion

$ _

What is JSON(B)?

Imagine JSON, but you can query it

```
{
    "introduced": {
        "JSON": "Postgres 9.4",
        "JSONB": "Postgres 9.5"
    },
    "JSON": "Checks if string valid JSON, stores as text",
    "JSONB": "Stores as a decomposed Binary format"
}
```

$ _

JSONB -> Postgres types

```
    JSONB      |       PostgreSQL         |                    Notes
---------------+--------------------------+----------------------------------------
 string        | text                     | Remember that it conforms to the database encoding
 number        | numeric                  | NaN and infinity values are not allowed
 boolean       | boolean                  | Only lowercase `true` and `false` spelling is valid
 null          | Does not translate       | SQL NULL is a different concept.
(4 rows)

$ SELECT 'null'::jsonb IS NULL, 'null'::jsonb = 'null';

 is_null | = null
---------+--------
 f       | t
(1 row)
```

$ _

JSON(B) example

From https://www.postgresql.org/docs/11/datatype-json.html

```
-- Simple scalar/primitive value
-- Primitive values can be numbers, quoted strings, true, false, or null
SELECT '5'::json;

-- Array of zero or more elements (elements need not be of same type)
SELECT '[1, 2, "foo", null]'::json;

-- Object containing pairs of keys and values
-- Note that object keys must always be quoted strings
SELECT '{"bar": "baz", "balance": 7.77, "active": false}'::json;

-- Arrays and objects can be nested arbitrarily
SELECT '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}'::json;
```

$ _

JSON vs JSONB

JSON	JSONB
Stores as if it was a TEXT field	Stores as a binary format
Doesn’t modify the input string Preserves whitespace. Including new lines	Trims whitespace and re-orders keys for more efficient searching
Duplicate keys are preserved	Duplicate keys are removed
Only a few operators and functions	Many operators
No indexes, no constraints	Indexes and constraints
Inserts/updates are fast	Inserts/updates a little bit slower Has to parse and convert types

(6 rows)

$ _

Some cool things of JSONB

```
                          cool things
--------------------------------------------------------------
 You can filter the data
 You can index specific keys
 You can index the whole thing (jsonb_path_ops)
 You can add constraints
 You can interact with JSON(B) and normal table columns in the same query
 You can use JSON Path from Postgres 12
(6 rows)
```

$ _

Some limitations of JSONB

```
                              limitations
------------------------------------------------------------------------
 No statistics
 The concat operator (||) is only top-level
 Max row size of 268 435 455 bytes (268.43 MB)
 NaN and infinity not allowed
 null IS NOT NULL
 \u0000 not allowed
(6 rows)
```

$ _

Experiment time

1. Figure out how Twitter streaming API works. ✅
2. Over a period of a few days, save up about 4M Tweets (Avengers and GoT). ✅
3. Save these 4M Tweets into a table, one for Avengers, one for GoT. ✅
4. Join `tweets_avengers` and `tweets_got` into one table, still in JSON. ❌
5. Convert to JSONB.
6. Model single table with all 2nd level JSON objects as JSONB fields.
7. Model the Tweet fully.
8. Run experiments.

What went wrong?

$ _

What went wrong?

\u0000 not allowed

$ _

What went wrong?

\u0000 not allowed

```
INSERT INTO tweets_json (SELECT * FROM tweets_avengers);

ERROR:  22P05: unsupported Unicode escape sequence
DETAIL:  \u0000 cannot be converted to text.
CONTEXT:  JSON data, line 1: ...hing  | ALL I WANNA DO","url":null,"description":...
LOCATION:  json_lex_string, json.c:882
Time: 15465.960 ms (00:15.466)
```

$ _

Let’s remove the NULL bytes

```
INSERT INTO tweets_json (
    SELECT regexp_replace(tweet::text, '\\u0000', '', 'g')::json
    FROM tweets_avengers
);

ERROR:  22P02: invalid input syntax for type json
DETAIL:  Token "screen_name" is invalid.
CONTEXT:  JSON data, line 1: ...tr":"1043592081766326277","name":"\","screen_name...
LOCATION:  report_invalid_token, json.c:1248
Time: 743264.154 ms (12:23.264)
```

$ _

Let’s remove the NULL bytes

Part 2

But don’t remove the best possible Twitter account name

```
INSERT INTO tweets_json (
    SELECT regexp_replace(tweet::text, '(?<!\\)\\u0000', '', 'g')::json
    FROM tweets_avengers
);

INSERT 0 2564034
Time: 795585.674 ms (13:15.586)
```

YES! IT WORKED!

$ _

What did \u0000 have to say?

```
SELECT
    tweet->'user'->>'screen_name' AS screen_name,
    tweet->>'text' AS text
FROM tweets_json
WHERE tweet->'user'->>'name' = '\u0000';

  screen_name   |                    text
----------------+---------------------------------------
 lordofthedykes | woke up this early to go see endgame... again
(1 row)

Time: 109700.848 ms (01:49.701)
```

$ _

Experiment time

1. Figure out how Twitter streaming API works. ✅
2. Over a period of a few days, save up about 4M Tweets (Avengers and GoT). ✅
3. Save these 4M Tweets into a table, one for Avengers, one for GoT. ✅
4. Join `tweets_avengers` and `tweets_got` into one table, still in JSON. ❌ ❌ ✅
5. Convert to JSONB. ✅
6. Model single table with all 2nd level JSON objects as JSONB fields. ✅
7. Model the Tweet fully. ✅
8. Run experiments.

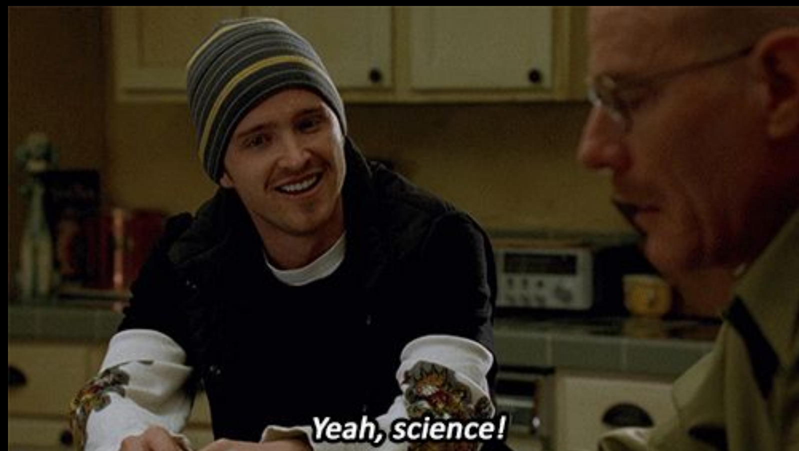
$ _

Experiment time

Method

max_connections = 200
shared_buffers = 4GB
effective_cache_size = 12GB
maintenance_work_mem = 1GB
checkpoint_completion_target = 0.7
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
work_mem = 5242kB
min_wal_size = 1GB
max_wal_size = 2GB
max_worker_processes = 8
max_parallel_workers_per_gather = 4
max_parallel_workers = 8

Postgres 12 on Ubuntu 18.04
Python scripts ran 12 times, 4 concurrently
Values in slides are averages without min/max



$ _

JSON vs JSONB vs Tables

Compare table sizes

```
        List of relations
         Name           |  Size
------------------------+-------
 tweets_json            | 11 GB
 tweets_jsonb           | 13 GB
 tweets_mix_normalized  | 17 GB

 place                  | 3240 kB
 tweet                  | 4118 MB
 user                   | 835 MB
                 Total    4955 MB
(6 rows)
```

$ _

JSON vs JSONB vs Tables

Find \u0000 (unindexed)

Query time

```
SELECT * FROM tweets_json WHERE tweet->'user'->>'name' = '\u0000';
    Time: 129739.053 ms (02:09.739)
```

67

```
SELECT * FROM tweets_jsonb WHERE tweet->'user'->>'name' = '\u0000';
    Time: 53394.158 (00:53.394)
```

27

```
SELECT * FROM tweets_mix_normalized WHERE "user"->>'name' = '\u0000';
    Time: 195003.180 ms (03:15.003)
```

100

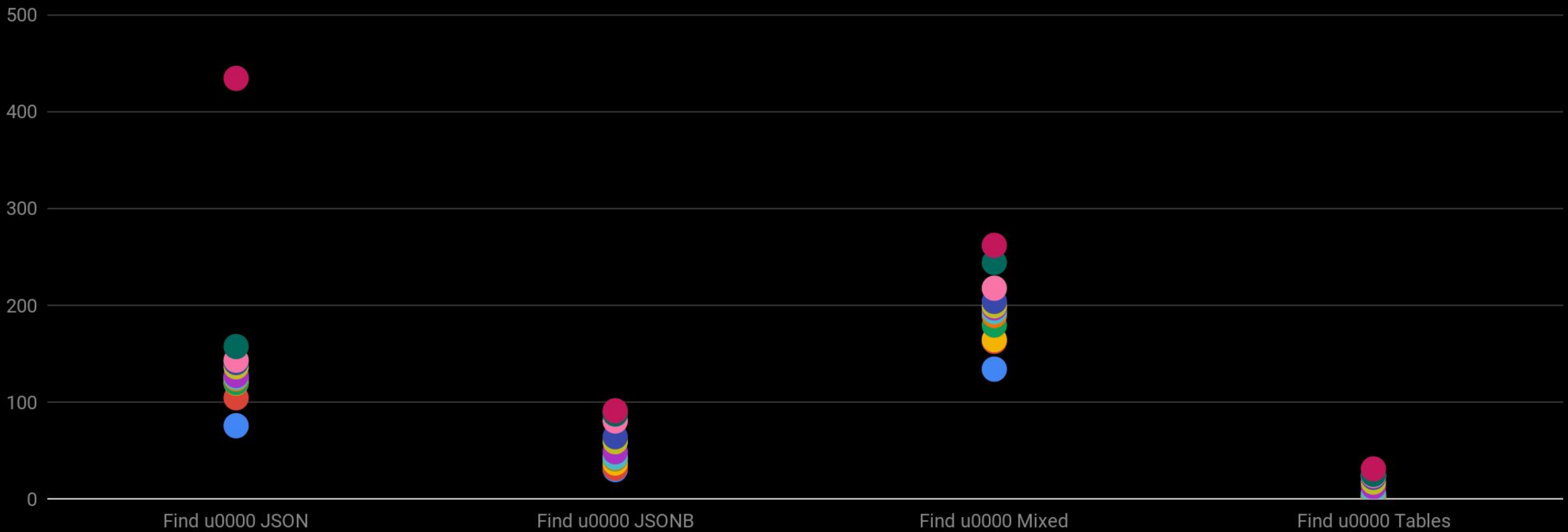
```
SELECT * FROM tweet
JOIN "user" ON tweet.user_id = "user".id
WHERE "user".name = '\u0000';
    Time: 11753.515 ms (00:11.754)
```

6

$ _

JSON vs JSONB vs Tables

Find \u0000 (unindexed)



$ _

JSON vs JSONB vs Tables

Real life example: Inspectability

1.

```
\d tweets_jsonb
          Table "public.tweets_jsonb"
 Column | Type  | Collation | Nullable | Default
--------+-------+-----------+----------+---------
 tweet  | jsonb |           |          |
```

2.

```
\d tweets_mix_normalized
                  Table "public.tweets_mix_normalized"
           Column           |  Type   | Collation | Nullable | Default
----------------------------+---------+-----------+----------+---------
 created_at                 | text    |           |          |
 id                         | bigint  |           |          |
 id_str                     | text    |           |          |
 text                       | text    |           |          |
 source                     | text    |           |          |
 truncated                  | boolean |           |          |
 in_reply_to_status_id      | bigint  |           |          |
 in_reply_to_status_id_str  | text    |           |          |
 in_reply_to_user_id        | bigint  |           |          |
 in_reply_to_user_id_str    | text    |           |          |
 in_reply_to_screen_name    | text    |           |          |
 user                       | jsonb   |           |          |
 geo                        | jsonb   |           |          |
 coordinates                | jsonb   |           |          |
 place                      | jsonb   |           |          |
 contributors               | jsonb   |           |          |
 retweeted_status           | jsonb   |           |          |
 quoted_status_id           | bigint  |           |          |
 quoted_status_id_str       | text    |           |          |
 quoted_status              | jsonb   |           |          |
 quoted_status_permalink    | jsonb   |           |          |
 is_quote_status            | boolean |           |          |
 quote_count                | bigint  |           |          |
 reply_count                | bigint  |           |          |
 retweet_count              | bigint  |           |          |
 favorite_count             | bigint  |           |          |
 entities                   | jsonb   |           |          |
 favorited                  | boolean |           |          |
 retweeted                  | boolean |           |          |
 possibly_sensitive         | boolean |           |          |
 filter_level               | text    |           |          |
 lang                       | text    |           |          |
 timestamp_ms               | text    |           |          |
```

3.

```
\d tweet user place
                         Table "public.tweet"
          Column           |  Type   | Collation | Nullable | Default
---------------------------+---------+-----------+----------+---------
 created_at                | text    |           |          |
 id                        | bigint  |           | not null |
 id_str                    | text    |           |          |
 text                      | text    |           |          |
 source                    | text    |           |          |
 truncated                 | boolean |           |          |
 in_reply_to_status_id     | bigint  |           |          |
 in_reply_to_status_id_str | text    |           |          |
 in_reply_to_user_id       | bigint  |           |          |
 in_reply_to_user_id_str   | text    |           |          |
 in_reply_to_screen_name   | text    |           |          |
 user_id                   | bigint  |           |          |
 geo                       | jsonb   |           |          |
 coordinates               | jsonb   |           |          |
 place_id                  | text    |           |          |
 retweeted_status_id       | bigint  |           |          |
 quoted_status_id          | bigint  |           |          |
 quoted_status_id_str      | text    |           |          |
 is_quote_status           | boolean |           |          |
 quote_count               | bigint  |           |          |
 reply_count               | bigint  |           |          |
 retweet_count             | bigint  |           |          |
 favorite_count            | bigint  |           |          |
 entities                  | jsonb   |           |          |
 favorited                 | boolean |           |          |
 retweeted                 | boolean |           |          |
 possibly_sensitive        | boolean |           |          |
 filter_level              | text    |           |          |
 lang                      | text    |           |          |
 timestamp_ms              | text    |           |          |
Indexes:
    "tweet_pkey" PRIMARY KEY, btree (id)
Foreign-key constraints:
    "tweet_place_id_fkey" FOREIGN KEY (place_id) REFERENCES place(id)
    "tweet_user_id_fkey" FOREIGN KEY (user_id) REFERENCES "user"(id)

\d place
               Table "public.place"
    Column    | Type  | Collation | Nullable | Default
--------------+-------+-----------+----------+---------
 id           | text  |           | not null |
 url          | text  |           |          |
 place_type   | text  |           |          |
 name         | text  |           |          |
 full_name    | text  |           |          |
 country_code | text  |           |          |
 country      | text  |           |          |
 bounding_box | jsonb |           |          |
 attributes   | jsonb |           |          |
Indexes:
    "place_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "tweet" CONSTRAINT "tweet_place_id_fkey" FOREIGN KEY (place_id) REFERENCES place(id)
```

```
\d user
                      Table "public.user"
         Column          |  Type   | Collation | Nullable | Default
-------------------------+---------+-----------+----------+---------
 id                      | bigint  |           | not null |
 id_str                  | text    |           |          |
 name                    | text    |           |          |
 screen_name             | text    |           |          |
 location                | text    |           |          |
 derived                 | jsonb   |           |          |
 url                     | text    |           |          |
 description             | text    |           |          |
 protected               | boolean |           |          |
 verified                | boolean |           |          |
 followers_count         | integer |           |          |
 friends_count           | integer |           |          |
 listed_count            | integer |           |          |
 favourites_count        | integer |           |          |
 statuses_count          | integer |           |          |
 created_at              | text    |           |          |
 profile_banner_url      | text    |           |          |
 profile_image_url_https | text    |           |          |
 default_profile         | boolean |           |          |
 default_profile_image   | boolean |           |          |
 withheld_in_countries   | text[]  |           |          |
 withheld_scope          | text    |           |          |
Indexes:
    "user_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "tweet" CONSTRAINT "tweet_user_id_fkey" FOREIGN KEY (user_id) REFERENCES "user"(id)
```

$ _

JSON vs JSONB vs Tables

Real life example: Maintainability

1. JSONB

```
SELECT
    tweet->>'id' AS id,
    tweet->>'text' AS text,
    tweet->>'created_at' AS created_at,
    tweet->>'source' AS source,
    tweet->>'retweet_count' AS retweet_count,
    tweet->>'favorite_count' AS favorite_count,
    tweet->'user'->>'name' AS name,
    tweet->'user'->>'screen_name' AS screen_name,
    tweet->'user'->>'verified' AS verified,
    tweet->'user'->>'profile_image_url_https' AS profile_image_url_https
FROM tweets_jsonb
ORDER BY (tweet->>'timestamp_ms')::BIGINT
LIMIT 10;
```

2. Mixed

```
SELECT
    id,
    text,
    created_at,
    source,
    retweet_count,
    favorite_count,
    "user"->>'name' AS name,
    "user"->>'screen_name' AS screen_name,
    "user"->>'verified' AS verified,
    "user"->>'profile_image_url_https' AS profile_image_url_https
FROM tweets_mix_normalized
ORDER BY timestamp_ms::BIGINT
LIMIT 10;
```

3. Tables

```
SELECT
    tweet.id,
    tweet.text,
    tweet.created_at,
    tweet.source,
    tweet.retweet_count,
    tweet.favorite_count,
    "user".name,
    "user".screen_name,
    "user".verified,
    "user".profile_image_url_https
FROM tweet
JOIN "user" ON tweet.user_id = "user".id
ORDER BY timestamp_ms::BIGINT
LIMIT 10;
```

$ _

JSON vs JSONB vs Tables

Real life example: List last 10 tweets (unindexed)

1. JSONB

```
SELECT
    tweet->>'id' AS id,
    tweet->>'text' AS text,
    tweet->>'created_at' AS created_at,
    tweet->>'source' AS source,
    tweet->>'retweet_count' AS retweet_count,
    tweet->>'favorite_count' AS favorite_count,
    tweet->'user'->>'name' AS name,
    tweet->'user'->>'screen_name' AS screen_name,
    tweet->'user'->>'verified' AS verified,
    tweet->'user'->>'profile_image_url_https' AS profile_image_url_https
FROM tweets_jsonb
ORDER BY (tweet->>'timestamp_ms')::BIGINT
LIMIT 10;
```

311245.580 ms (05:11.246)

100

2. Mixed

```
SELECT
    id,
    text,
    created_at,
    source,
    retweet_count,
    favorite_count,
    "user"->>'name' AS name,
    "user"->>'screen_name' AS screen_name,
    "user"->>'verified' AS verified,
    "user"->>'profile_image_url_https' AS profile_image_url_https
FROM tweets_mix_normalized
ORDER BY timestamp_ms::BIGINT
LIMIT 10;
```

180532.100 ms (03:00.532)

58

3. Tables

```
SELECT
    tweet.id,
    tweet.text,
    tweet.created_at,
    tweet.source,
    tweet.retweet_count,
    tweet.favorite_count,
    "user".name,
    "user".screen_name,
    "user".verified,
    "user".profile_image_url_https
FROM tweet
JOIN "user" ON tweet.user_id = "user".id
ORDER BY timestamp_ms::BIGINT
LIMIT 10;
```

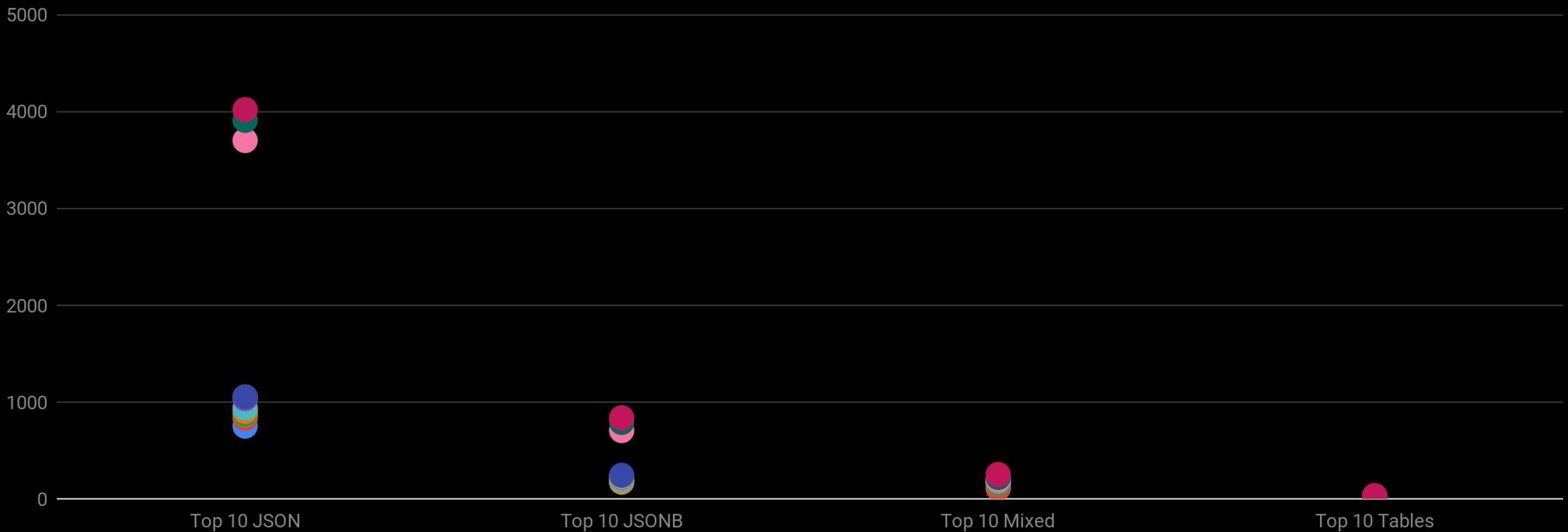
12479.723 ms (00:12.480)

4

$ _

JSON vs JSONB vs Tables

Real life example: List last 10 tweets (unindexed)



$ _

JSON vs JSONB vs Tables

Updates

But first. Some context

$ _

Message

Remind Me

Jason

Friday 13th is my favourite

$ _

I Am Managr
@iammanagr

push to production
on Fridays because I
promised the client we
would

$ _

JSON vs JSONB vs Tables

Updates

But Jason made a mistake.
His script to scrape Twitter saved everyone with less than 1000 followers with 200 too few.

Let's fix Jason's mistake.

$ _

JSON vs JSONB vs Tables

Updates

Query time

1. JSONB

```
UPDATE tweets_jsonb
SET tweet = jsonb_set(
    tweet,
    '{user, followers_count}',
    (
        (
            (
                tweet->'user'->>'followers_count'
            )::INT + 200
        )::TEXT
    )::jsonb
)
WHERE (tweet->'user'->>'followers_count')::INT < 1000
```

897754.2313 ms (14:57.754)

100

2. Mixed

```
UPDATE tweets_mix_normalized
SET "user" = jsonb_set(
    "user",
    '{followers_count}',
    (
        (
            (
                "user"->>'followers_count'
            )::INT + 200
        )::TEXT
    )::jsonb
)
WHERE ("user"->>'followers_count')::INT < 1000
```

460230.4723 ms (07:40.230)

51

3. Tables

```
UPDATE "user"
SET followers_count = followers_count + 200
WHERE followers_count < 1000
```

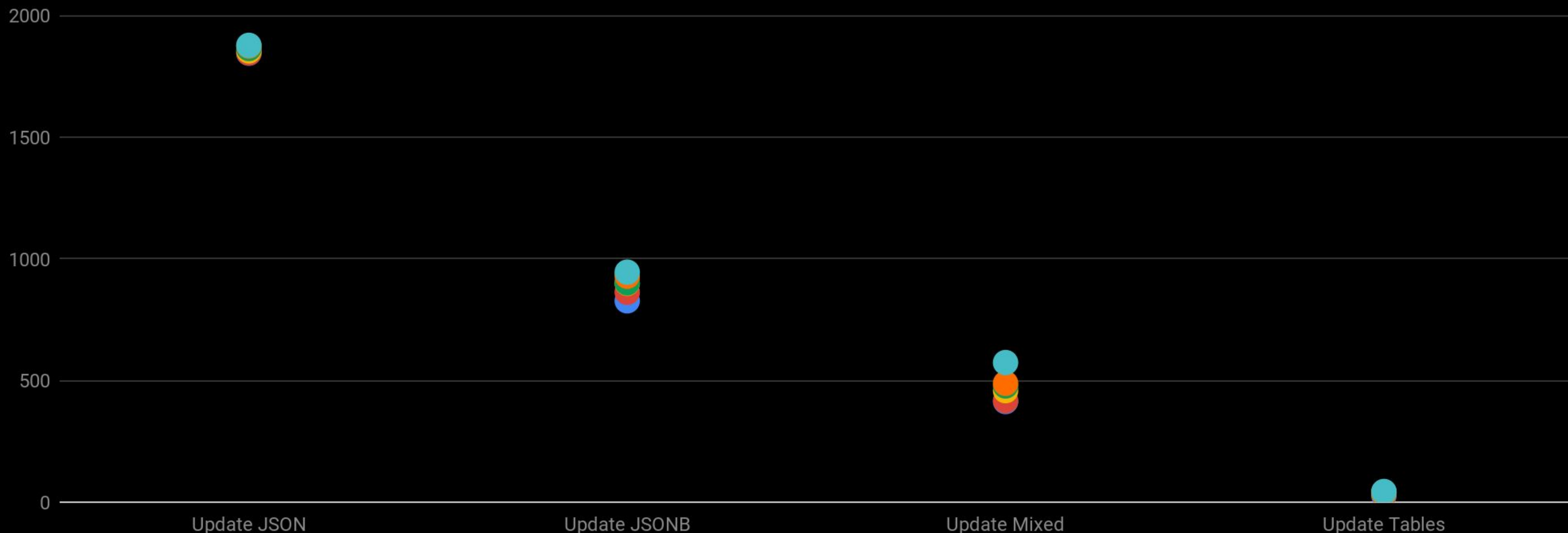
38254.552 ms (00:38.255)

4

$ _

JSON vs JSONB vs Tables

Updates



$ _

Me: JSONB slower than tables
Everyone:
ACKCHYUALLY

$ _

JSON vs JSONB vs Tables

Creating Indexes on user’s name

Index time

```
CREATE INDEX idx_json_user_name ON tweets_json ((tweet->'user'->>'name'));
    Time: 139393.493 ms (02:19.393)
```

100

```
CREATE INDEX idx_jsonb_user_name ON tweets_jsonb ((tweet->'user'->>'name'));
    Time: 24268.704 ms (00:24.269)
```

17

```
CREATE INDEX idx_mixed_user_name ON tweets_mix_normalized (("user"->>'name'));
    Time: 31793.163 ms (00:31.793)
```

23

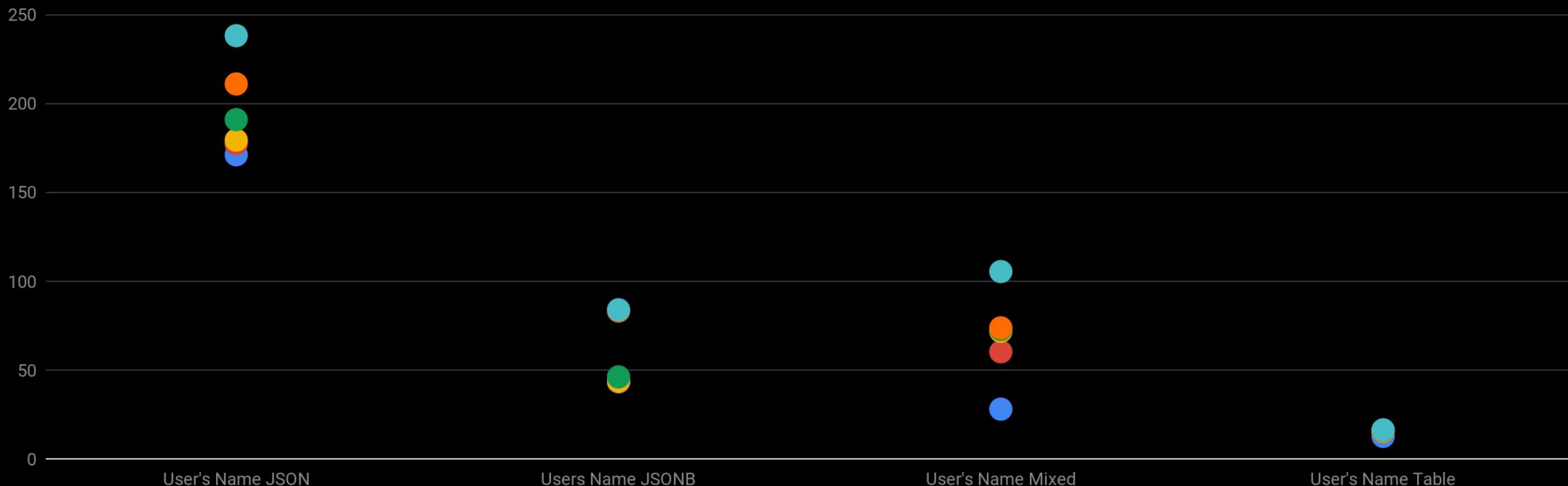
```
CREATE INDEX idx_user_name ON "user" (name);
    Time: 12967.147 ms (00:12.967)
```

9

$ _

JSON vs JSONB vs Tables

Creating Indexes on user's name



$ _

JSON vs JSONB vs Tables

Compare Indexed queries

Query time

```
SELECT * FROM tweets_json WHERE tweet->'user'->>'name' = '\u0000';
    Time: 1.031 ms
```

67

```
SELECT * FROM tweets_jsonb WHERE tweet->'user'->>'name' = '\u0000';
    Time: 1.161 ms
```

76

```
SELECT * FROM tweets_mix_normalized WHERE "user"->>'name' = '\u0000';
    Time: 1.108 ms
```

72

```
SELECT * FROM tweet
JOIN "user" ON tweet.user_id = "user".id
WHERE "user".name = '\u0000';
    Time: 1.535 ms
```

100

$ _

JSON vs JSONB vs Tables

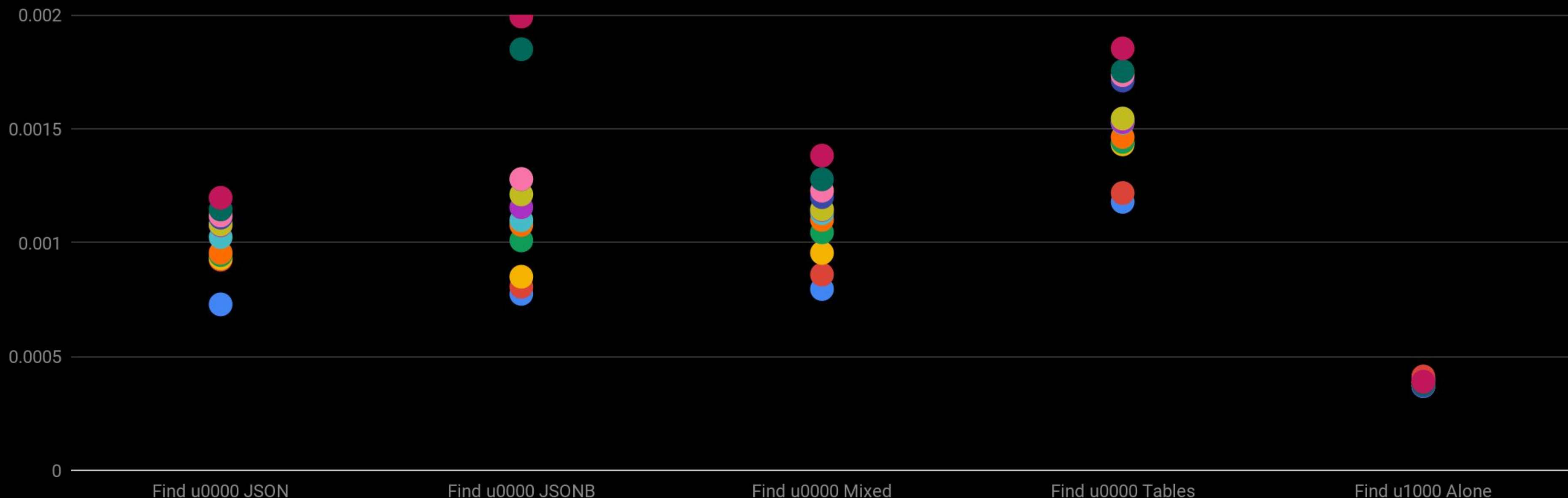
Compare Indexed queries

```
$ EXPLAIN SELECT tweet.text FROM tweet
JOIN "user" ON tweet.user_id = "user".id
WHERE "user".name = '\u0000';

                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.86..75.52 rows=16 width=119)
   ->  Index Scan using idx_user_name on "user"  (cost=0.43..11.59 rows=9 width=8)
         Index Cond: (name = '\u0000'::text)
   ->  Index Scan using ids_user_rel on tweet  (cost=0.43..7.05 rows=5 width=127)
         Index Cond: (user_id = "user".id)
(5 rows)

Time: 0.450 ms
```

$ _

JSON vs JSONB vs Tables

Compare Indexed queries



$ _

JSON vs JSONB vs Tables

Creating Indexes

Index time

```
CREATE INDEX idx_json_timestamp_ms ON tweets_json (((tweet->>'timestamp_ms')::BIGINT));
    Time: 139393.493 ms (02:19.393)
```

100

```
CREATE INDEX idx_jsonb_timestamp_ms ON tweets_jsonb (((tweet->>'timestamp_ms')::BIGINT));
    Time: 24268.704 ms (00:24.269)
```

17

```
CREATE INDEX idx_mixed_timestamp_ms ON tweets_mix_normalized (CAST (timestamp_ms AS BIGINT));
    Time: 31793.163 ms (00:31.793)
```

23

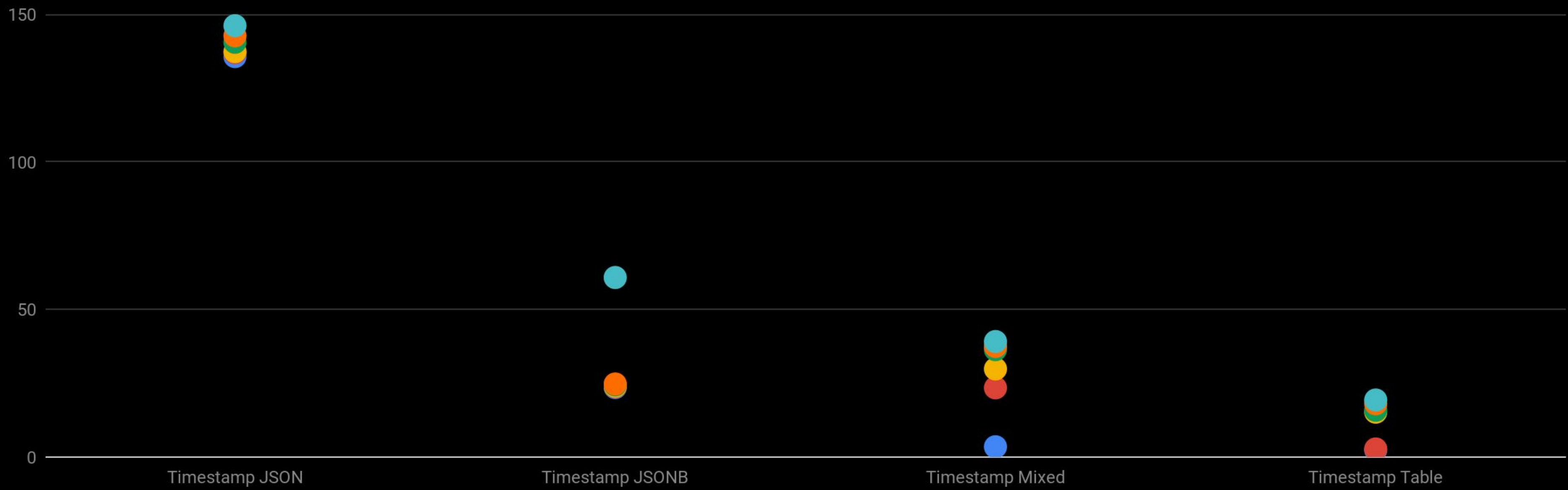
```
CREATE INDEX idx_tweet_timestamp_ms ON tweet (CAST (timestamp_ms AS BIGINT));
    Time: 12967.147 ms (00:12.967)
```

9

$ _

JSON vs JSONB vs Tables

Creating Indexes



$ _

JSON vs JSONB vs Tables

Real life example: Last 10 Tweets (indexed)

Indexed QT
Unindexed QT

1.

```
SELECT
    tweet->>'id' AS id,
    tweet->>'text' AS text,
    tweet->>'created_at' AS created_at,
    tweet->>'source' AS source,
    tweet->>'retweet_count' AS retweet_count,
    tweet->>'favorite_count' AS favorite_count,
    tweet->'user'->>'name' AS name,
    tweet->'user'->>'screen_name' AS screen_name,
    tweet->'user'->>'verified' AS verified,
    tweet->'user'->>'profile_image_url_https' AS profile_image_url_https
FROM tweets_jsonb
ORDER BY (tweet->>'timestamp_ms')::BIGINT
LIMIT 10;
```

Time: 3.053 ms

100
100

2.

```
SELECT
    id,
    text,
    created_at,
    source,
    retweet_count,
    favorite_count,
    "user"->>'name' AS name,
    "user"->>'screen_name' AS screen_name,
    "user"->>'verified' AS verified,
    "user"->>'profile_image_url_https' AS profile_image_url_https
FROM tweets_mix_normalized
ORDER BY timestamp_ms::BIGINT
LIMIT 10;
```

Time: 1.613 ms

53
51

3.

```
SELECT
    tweet.id,
    tweet.text,
    tweet.created_at,
    tweet.source,
    tweet.retweet_count,
    tweet.favorite_count,
    "user".name,
    "user".screen_name,
    "user".verified,
    "user".profile_image_url_https
FROM tweet
JOIN "user" ON tweet.user_id = "user".id
ORDER BY timestamp_ms::BIGINT
LIMIT 10;
```

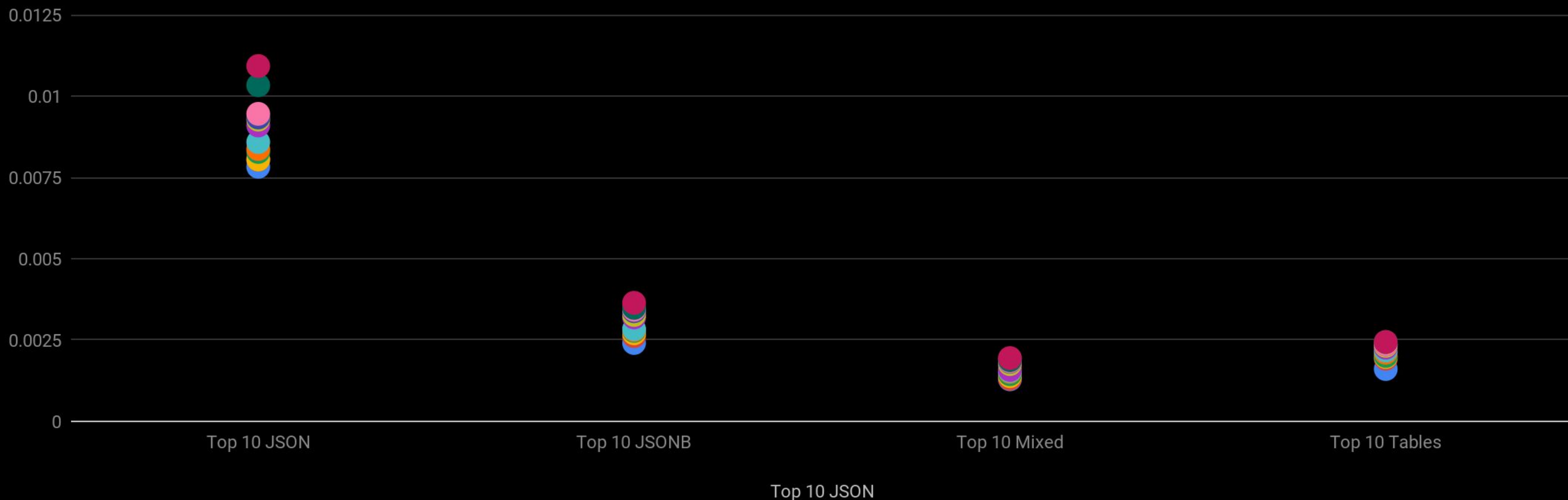
Time: 2.198 ms

72
4

$ _

JSON vs JSONB vs Tables

Real life example: Last 10 Tweets (indexed)



$ _

JSON vs JSONB vs Tables

Index size

Name	Table	Size
idx_json_timestamp_ms	tweets_json	**87 MB**
idx_json_user_name	tweets_json	132 MB
idx_jsonb_timestamp_ms	tweets_jsonb	**87 MB**
idx_jsonb_user_name	tweets_jsonb	132 MB
idx_mixed_timestamp_ms	tweets_mix_normalized	**87 MB**
idx_mixed_user_name	tweets_mix_normalized	132 MB
idx_tweet_timestamp_ms	tweet	**88 MB**
idx_tweet_user_id_fkey	tweet	88 MB
tweet_pkey	tweet	88 MB
place_pkey	place	352 kB
idx_user_name	user	73 MB
user_pkey	user	50 MB

(12 rows)

$ _

JSON vs JSONB vs Tables

Index size

Name	Table	Size
idx_json_timestamp_ms	tweets_json	87 MB
idx_json_user_name	tweets_json	**132 MB**
idx_jsonb_timestamp_ms	tweets_jsonb	87 MB
idx_jsonb_user_name	tweets_jsonb	**132 MB**
idx_mixed_timestamp_ms	tweets_mix_normalized	87 MB
idx_mixed_user_name	tweets_mix_normalized	**132 MB**
idx_tweet_timestamp_ms	tweet	88 MB
idx_tweet_user_id_fkey	tweet	88 MB
tweet_pkey	tweet	88 MB
place_pkey	place	352 kB
idx_user_name	user	**73 MB**
user_pkey	user	**50 MB**

(12 rows)

$ _

JSON vs JSONB vs Tables

Real life example: Backups

Backup time
Backup size

```
$ pg_dump --db twitter --table tweets_json -f tweets_json.sql
real  2m59.762s
30G tweets_json.sql
```

Backup time: 50
Backup size: 100

```
$ pg_dump --db twitter --table tweets_jsonb -f tweets_jsonb.sql
real  6m3.020s
30G tweets_jsonb.sql
```

Backup time: 100
Backup size: 99

```
$ pg_dump --db twitter --table tweets_mix_normalized -f tweets_mix_normalized.sql
real  5m29.481s
26G tweets_mix_normalized.sql
```

Backup time: 91
Backup size: 86

```
$ pg_dump --db twitter_normalized --table tweet --table user --table place -f tweets_normalized.sql
real  0m43.653s
4.1G tweets_normalized.sql
```

Backup time: 12
Backup size: 14

$ _

JSON vs JSONB vs Tables

Real life example: Restores

```
$ psql backup -f tweets_json.sql
real  14m45.819s

$ psql backup -f tweets_jsonb.sql
real  22m54.010s

$ psql backup -f tweets_mix_normalized.sql
real  27m18.312s

$ psql backup -f twitter_normalized.sql
real  6m23.690s
```

$ _

When could you use JSONB?

```
                                 usage
----------------------------------------------------------------------
(0 rows)
```

$ _

When could you use JSONB?

```
                                 usage
--------------------------------------------------------------------------
 Got NOSQL, but you need some ACID
 Storing document data - like settings, API queries (HAR format?)...
 Audit trail
 Fast-changing data
 Data with many optional columns/keys (Could inherited tables work?)
 As an aggregate
 When you have JSON data and quickly need to do some queries on the data
(7 rows)
```

$ _

When to not use JSONB

```
                          Usage
--------------------------------------------------
 When you need to filter the data
 When you need indexes
 When you need constraints
 When you need a key-value store
 When you could use a table
 When other people need to know what is going on
 When you need to store a lot of data in one row
(7 rows)
```

$ _

Conclusion Prelude

Programming, like all other engineering, is about compromises - there is no absolute answer.

But we need to do our best to assess the options.

$ _

Conclusion

$ _

Conclusion