$ _

Is '{JSONB}' a Silver Bullet?

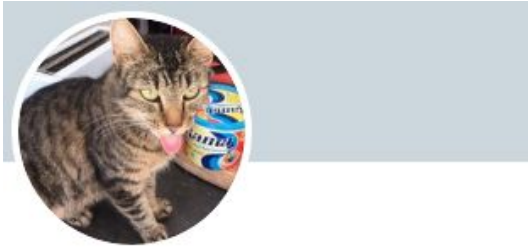
Angus Dippenaar

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



Quant SOLUTIONS

GoodX SAGTEWARE | SOFTWARE

Who is this guy?

Who is this guy?

Who is this guy?

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/12/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, last value kept
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 jsonpath 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.

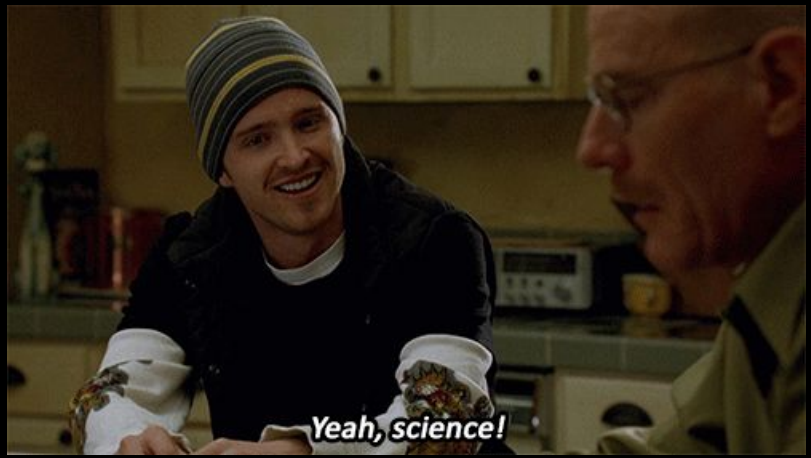
$ _

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

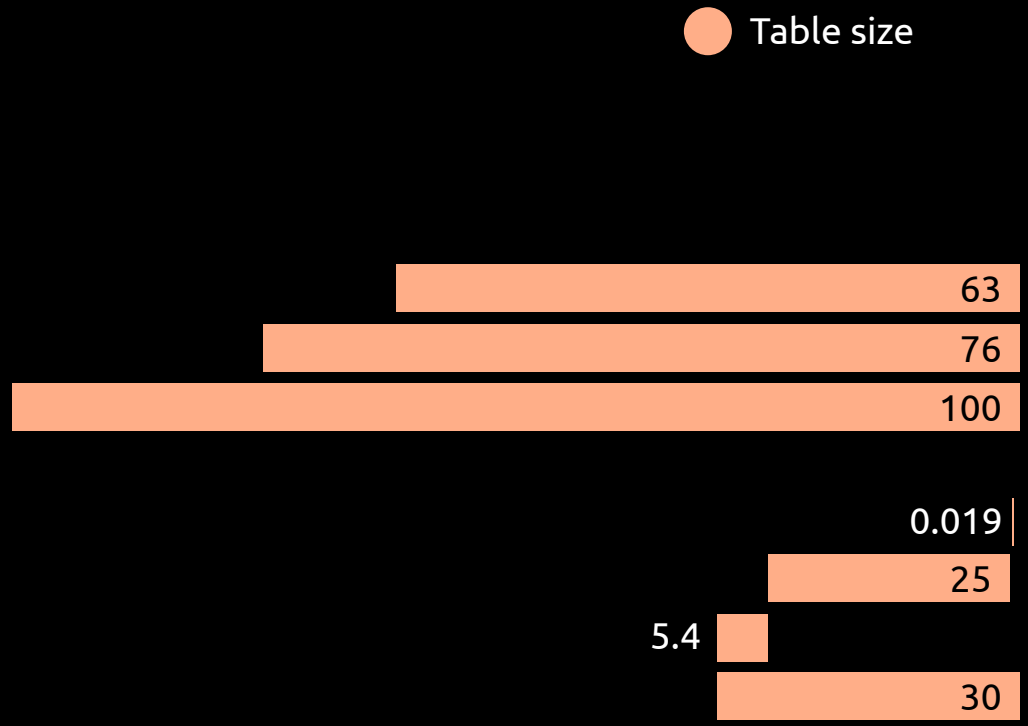


$ _

JSON vs JSONB vs Tables

Compare table sizes

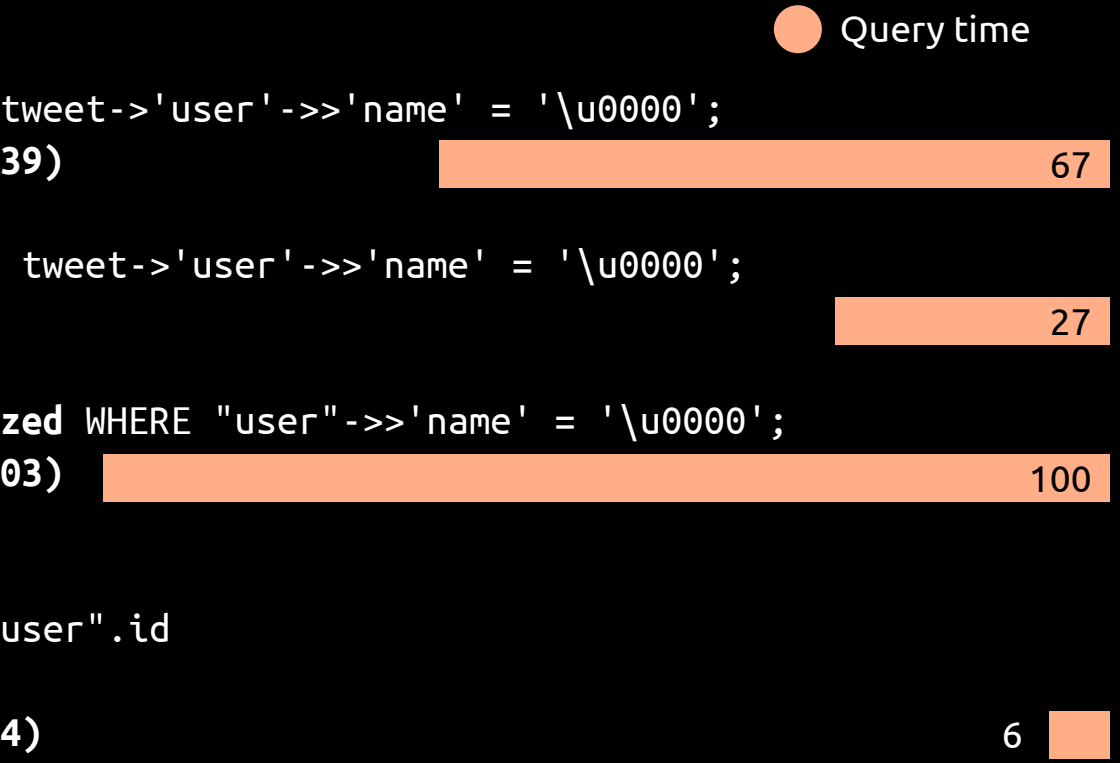
```
        List of relations
         Name           |  Size
------------------------+-------
 tweets_json            | 11 GB
 tweets_jsonb           | 13 GB
 tweets_mix_normalized  | 17 GB

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



$ _

JSON vs JSONB vs Tables

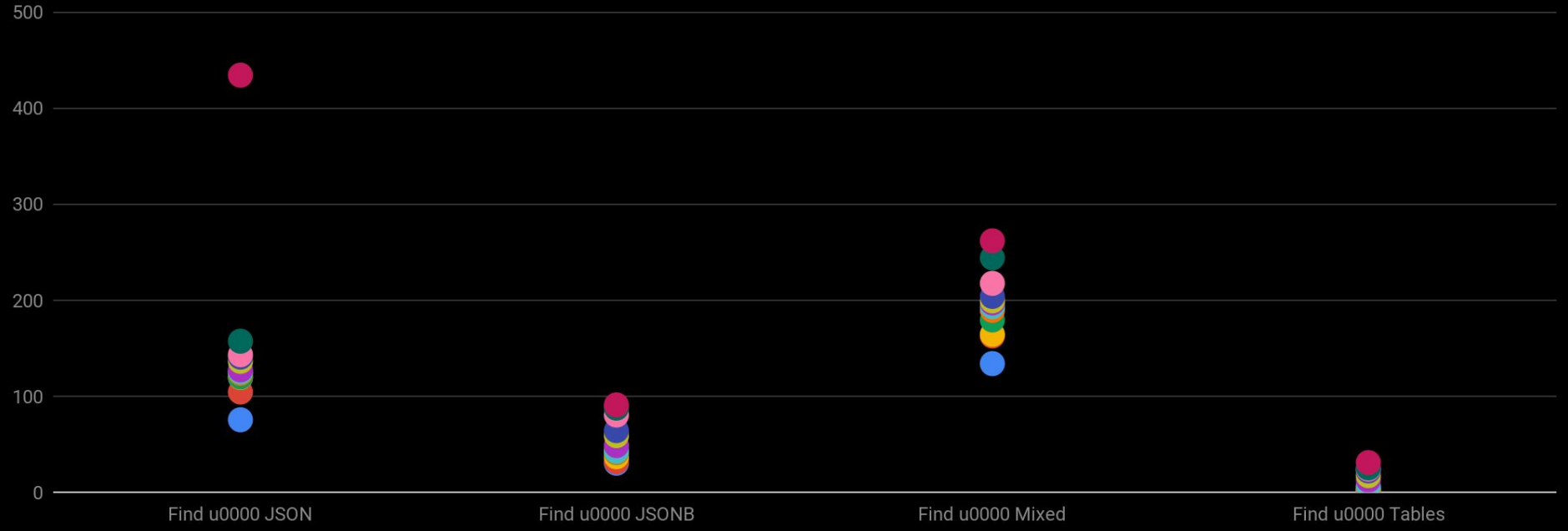
Find \u0000 (unindexed)



$ _

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)

Query time

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;
```

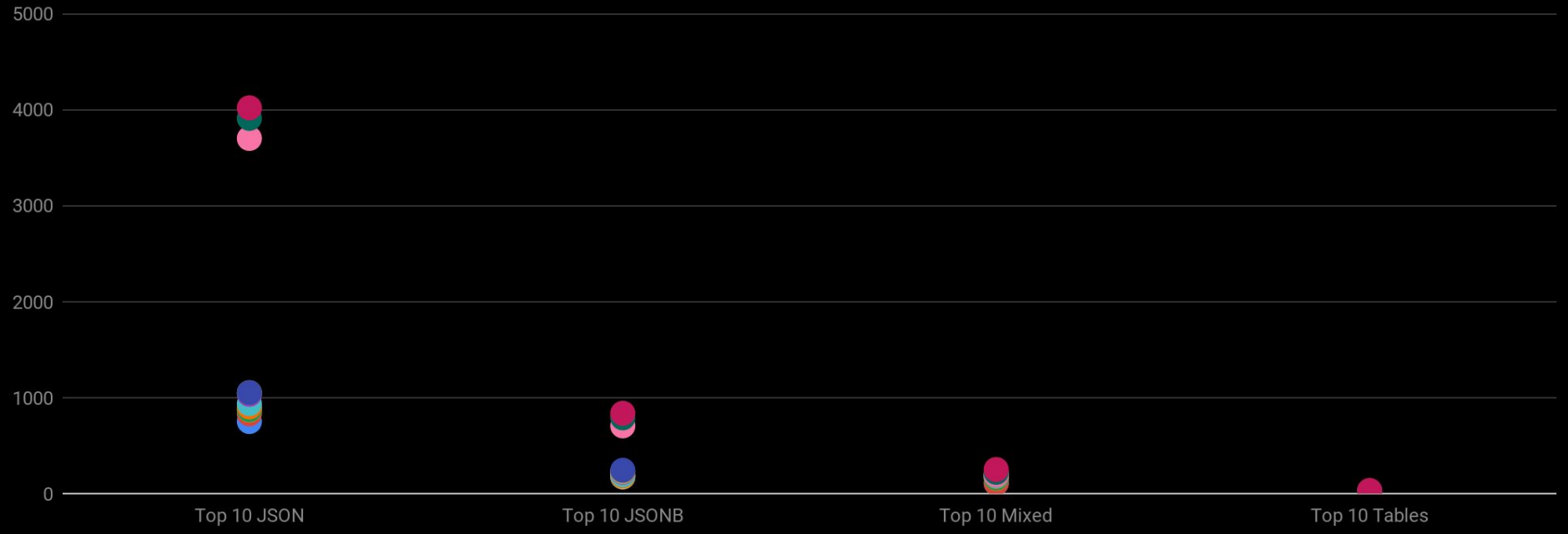
12479.723 ms (00:12.480)

4

$ _

JSON vs JSONB vs Tables

Real life example: List last 10 tweets (unindexed)



$ _

JSON vs JSONB vs Tables

Real life example: Tweets from Czech Republic

Query time

1. JSONB

```
SELECT
    tweet->>'id' AS id,
    tweet->>'text' AS text,
    tweet->'place'->>'full_name' AS city_name
FROM tweets_jsonb
WHERE tweet->'place'->>'country_code' = 'CZ';
```

16584.3412 ms (00:16.584)

2. Mixed

```
SELECT
    id,
    text,
    place->>'full_name' AS city_name
FROM tweets_mix_normalized WHERE
place->>'country_code' = 'CZ';
```

4640.1155 ms (00:04.640)

3. Tables

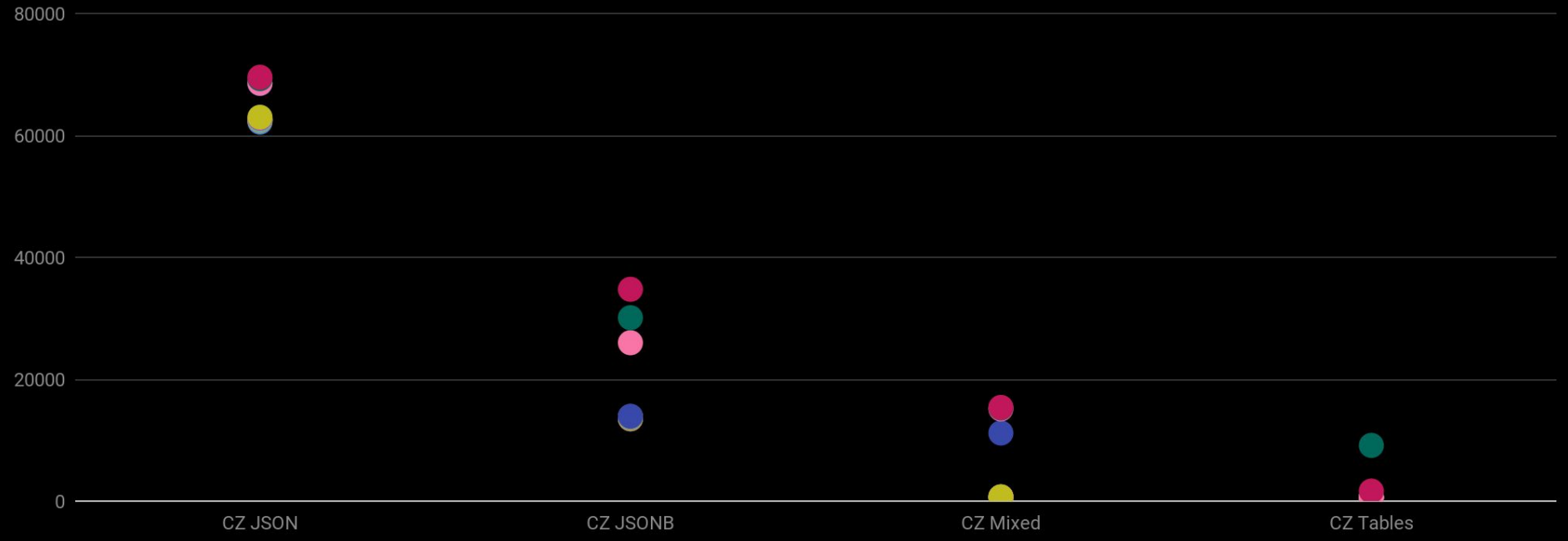
```
SELECT
    tweet.id,
    tweet.text,
    place.full_name AS city_name
FROM tweet
JOIN place
    ON tweet.place_id = place.id
WHERE country_code = 'CZ';
```

1476.8881 ms (00:01.477)

$ _

JSON vs JSONB vs Tables

Real life example: Tweets from Czech Republic



$ _

JSON vs JSONB vs Tables

Real life example: Tweets from Czech Republic

id	text	full_name
1122529003611533312	shrek umře v end game	Plzeň, Česká republika
1123574690981863424	Avengers! (@ CineStar Gold Class in Praha) https://t.co/DrwqXDCVD6	Prague, Czech Republic
1122171554664714249	#avengersendgame are fucking awsome!!! 😭😭😍😍#avengers #marvel #MarvelStudios #cinema https://t.co/3VjDSqbtMx	Prague, Czech Republic
1123666038682521601	But seriously. She is so, so much better at this than most of her colleagues, and yet they can't resist making thei…	Prague, Czech Republic
1123456535764586496	This has so much truth to it! I feel this exact same way! https://t.co/0QKsR0VOdJ	Předboj, Česká republika
1122615391069642753	Sorry about the previous post 😞 I didn't realize it could be taken as spoiler 🙈🙊 I never done that before and never…	Prague, Czech Republic
1123679242905116672	Tak dělám, že nový Avengers se nestali...	Prague, Czech Republic
1123487951646859265	@MantisScar Endgame!	Prague, Czech Republic
1123238181598638080	How do I rid my timeline of any and all things related to both avengers and game of thrones	Prague, Czech Republic
1123328560096325632	Going to see endgame for the second time tomorrow, woo! ❤️	Prague, Czech Republic
1123719917340188674	Byť u nových Avengers všechno fungovalo jak mělo (což je samo o sobě obdivuhodné), nemohl jsem celou dobu přestat m…	Prague, Czech Republic
1123883439290232832	ENDGAME has been "swallowed up" by GAME OF THRONES b/c they came out the same weekend is a thing I just heard some…	Prague, Czech Republic
1123203614577766400	Jedete do Brna. Nastoupí děti a začnou křičet spoilery z Avengers. https://t.co/O9AsaQCifU	Česká Třebová, Česká republika
1123307497727488000	What if I told you that there is no point to watch new avengers? Read more in the fresh article on the website.…	Prague, Czech Republic
1123831038948646918	Captain Marcel byla v Endgame dohromady deset minut a byly to nejhorší okamžiky mýho života. Netroufám si představo…	Prague, Czech Republic
1123895812763869184	Já vím, #Endgame a #GameofThrones ale zítra je na Náplavce Festival Českých Vín ❤️🍷	Louňovice, Česká republika
1122944864587603973	The internet today: 'OMG, Game of Thrones Battle of Winterfell the best thing ever, such long, many wow' Me, an in…	Prague, Czech Republic
1123303433262501888	Hey! Hallo @DiazCanelB existe una cosa, no un ejercicio, si, así creo que se llama. Muchos le llaman atribuir. 🧐… https://t.co/ovzmrJKEsx	Plzeň, Česká republika
1123300621262757890	Mé druhé já. #melisandre #got #hraotrůny #redwoman #lordofthelight #nottoday #valarmorghulis #caricevanhouten #epic…	Desná, Česká republika
1123293369478930432	Tremendo actor!	Prague, Czech Republic
1123268471385493506	Night king is a made of broken ice cubes - jon snow membuka usaha es batu jang bulan puasa #GOTS08E03…3	Cekov, Česká republika
1123560253864448001	Kavede takım elbiseyle kurtlar vadisi izleyen dayılardan ne farkı bu malların hahahs	Prague, Czech Republic
1123329560530051072	Mi líder, siempre 🔥	Prague, Czech Republic
1123528001373835265	I stan! Details l'omo	Prague, Czech Republic
1123550916051963904	Waaaw 😮	Zlín, Česká republika
1123780363007004672	Don't know Game of Thrones at all, but I do recognize the "I liked the show for the wrong reasons?!?!" reaction tha…	Prague, Czech Republic

(26 rows)

$ _

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)

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)

3. Tables

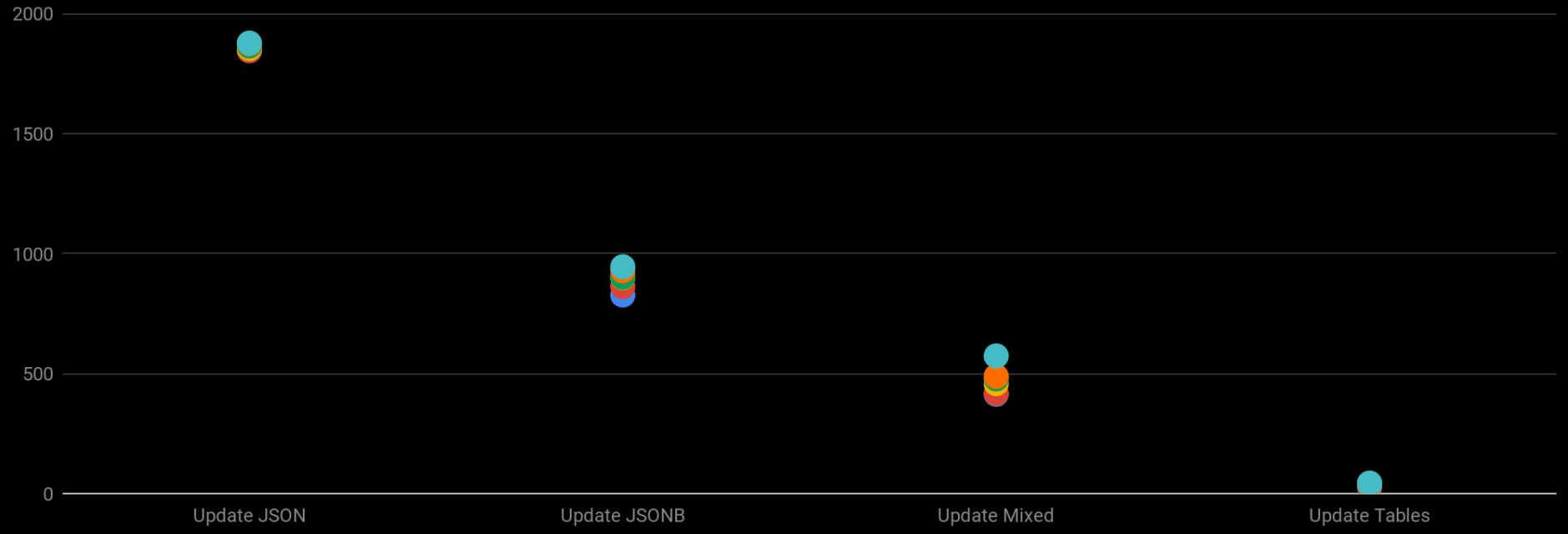
```
UPDATE "user"
SET followers_count = followers_count + 200
WHERE followers_count < 1000
```

38254.552 ms (00:38.255)

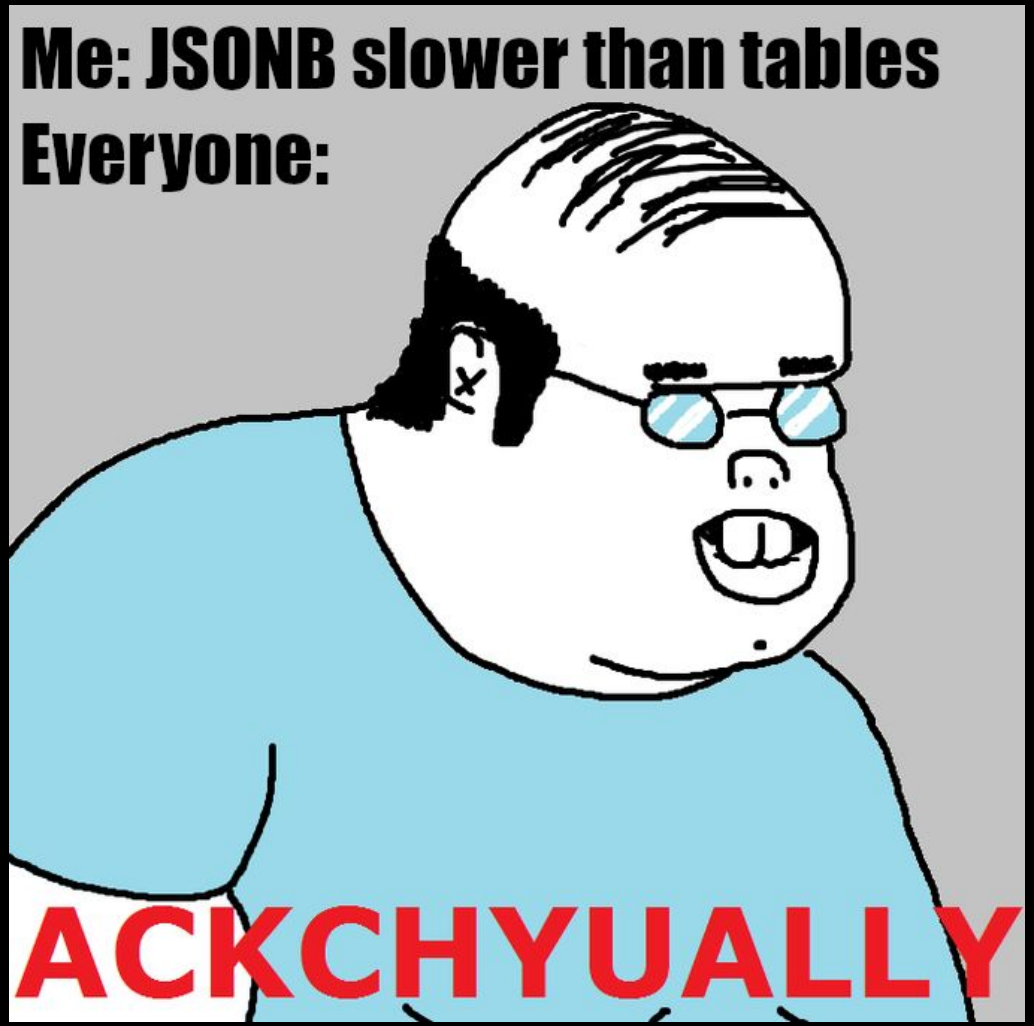
$ _

JSON vs JSONB vs Tables

Updates



$ _

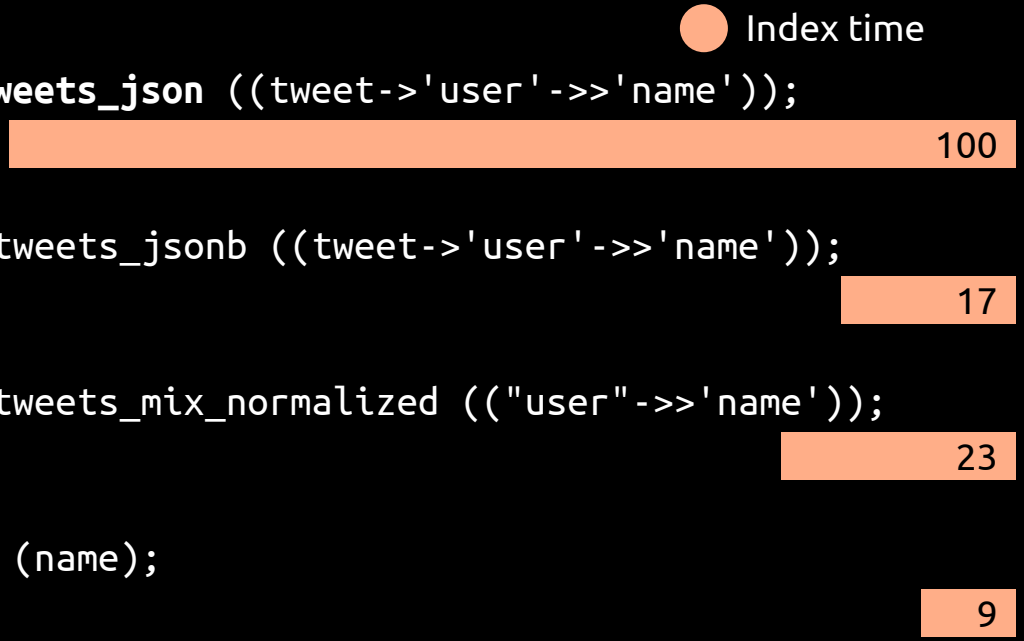

Me: JSONB slower than tables
Everyone:
ACKCHYUALLY

$ _

JSON vs JSONB vs Tables

Creating Indexes on user's name

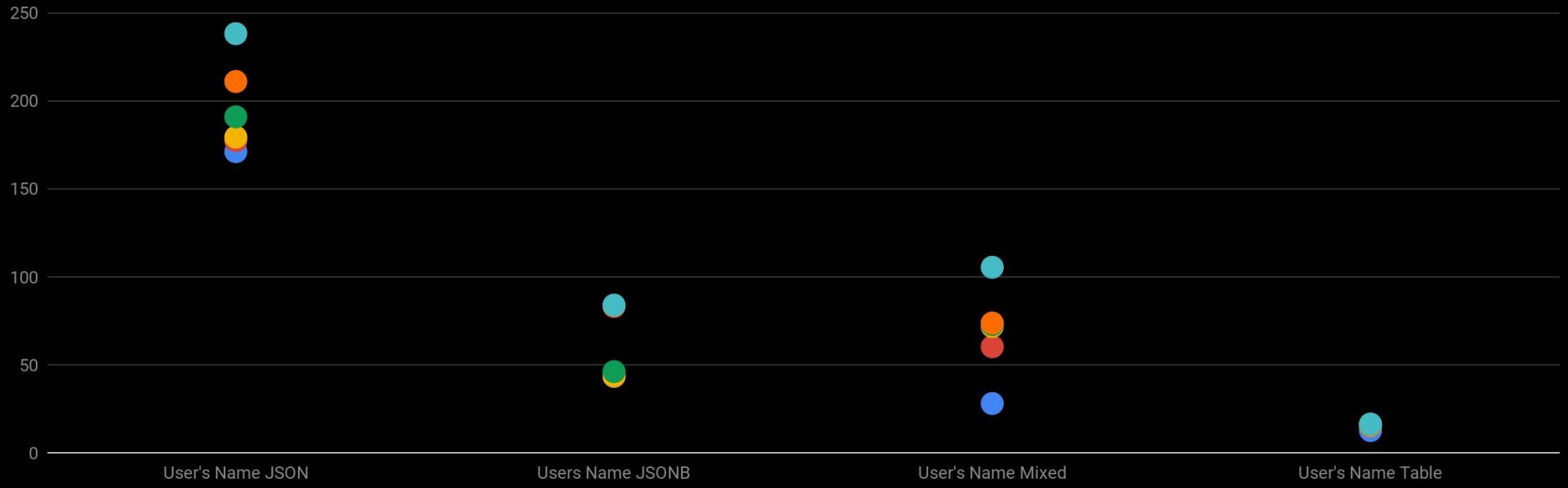
```
CREATE INDEX idx_json_user_name ON tweets_json ((tweet->'user'->>'name'));
    Time: 139393.493 ms (02:19.393)

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

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

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



$ _

JSON vs JSONB vs Tables

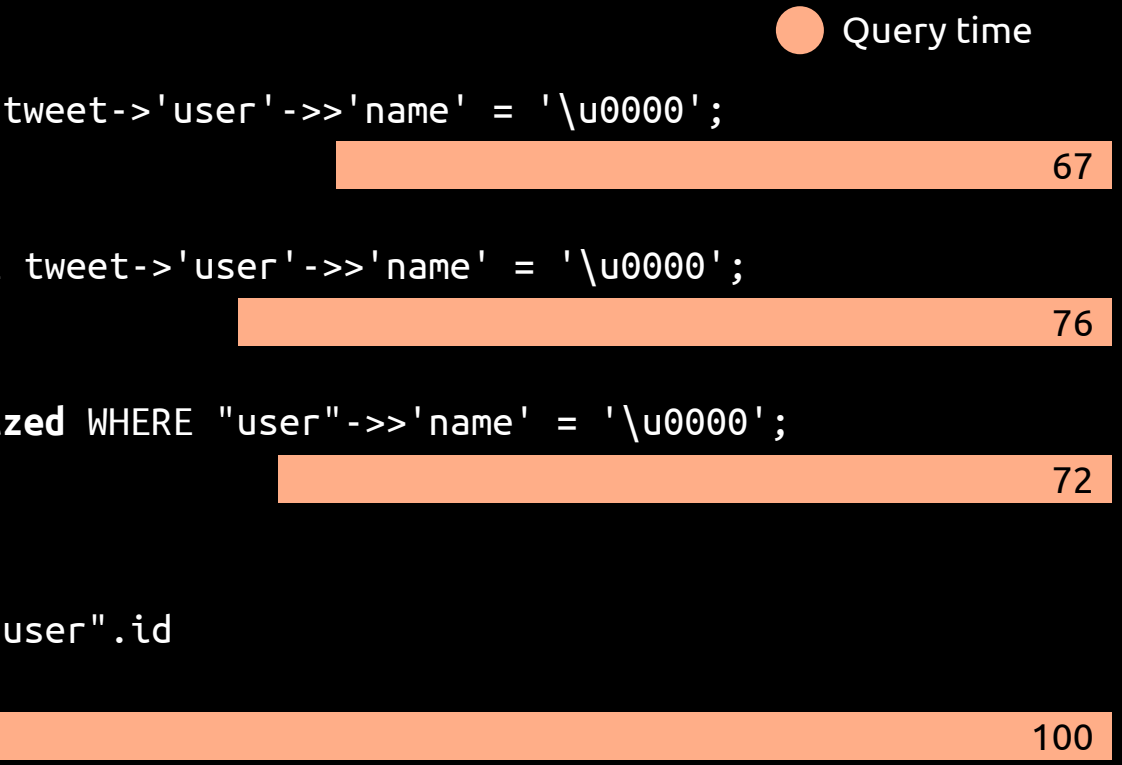
Creating Indexes on user's name



$ _

JSON vs JSONB vs Tables

Compare Indexed queries



$ _

JSON vs JSONB vs Tables

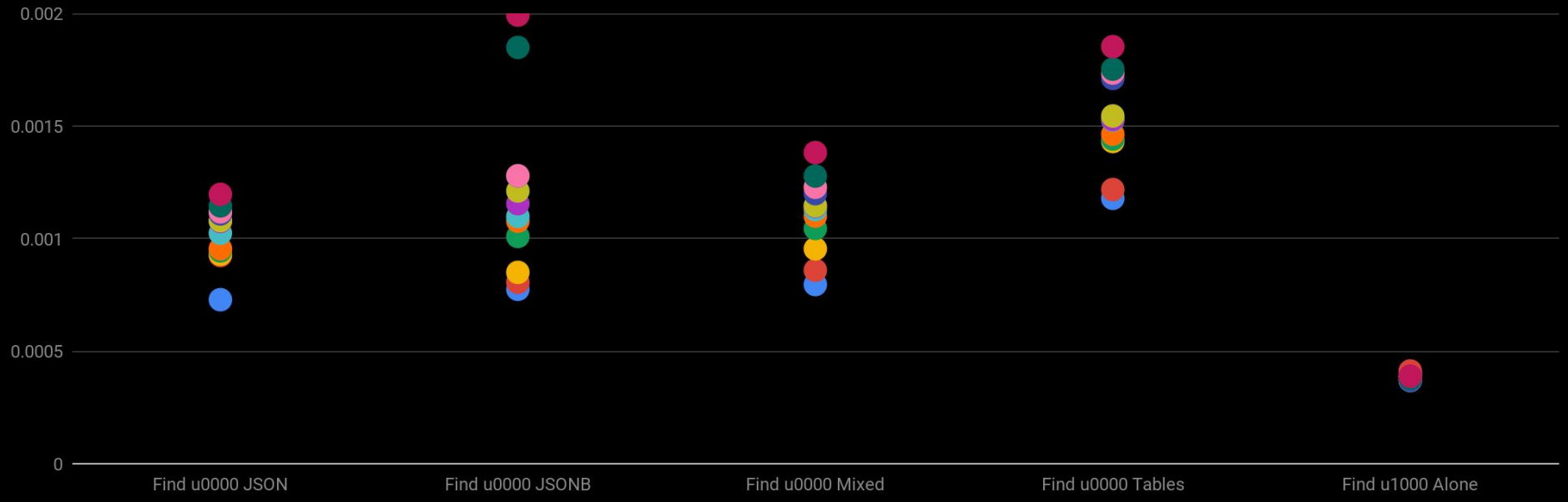
Compare Indexed queries

```
$ 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



$ _

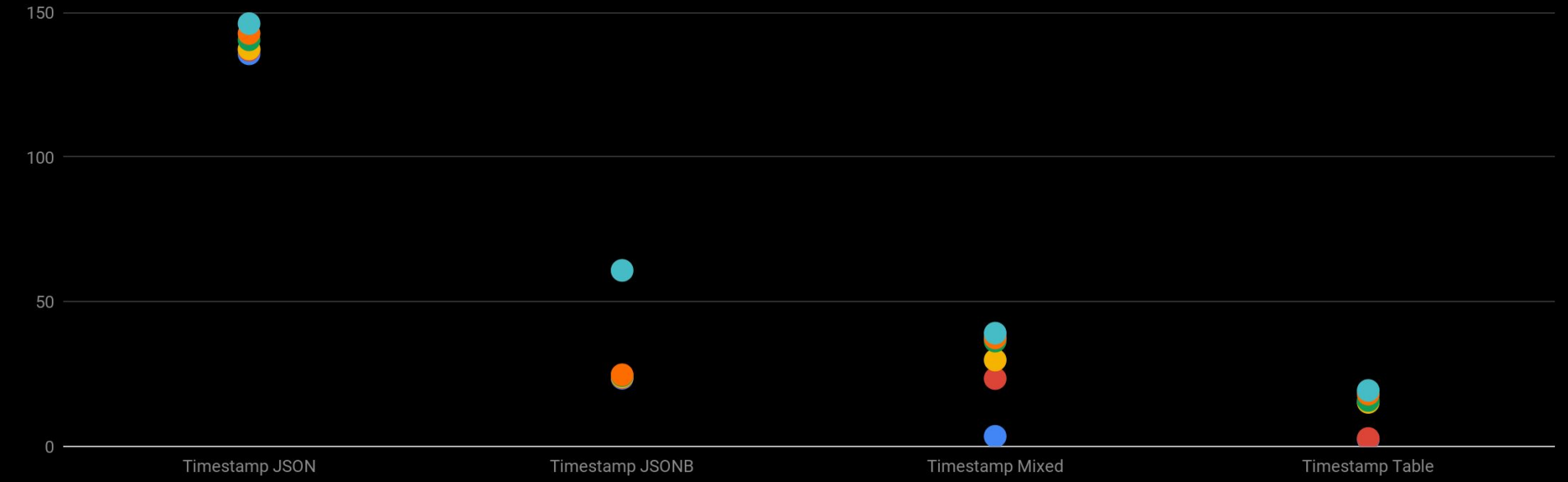
JSON vs JSONB vs Tables

Creating Indexes

$ _

JSON vs JSONB vs Tables

Creating Indexes



$ _

JSON vs JSONB vs Tables

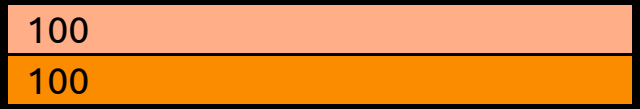
Real life example: Last 10 Tweets (indexed)

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;
```

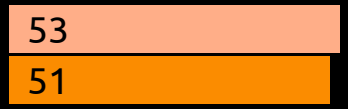
Time: 3.053 ms



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



3.

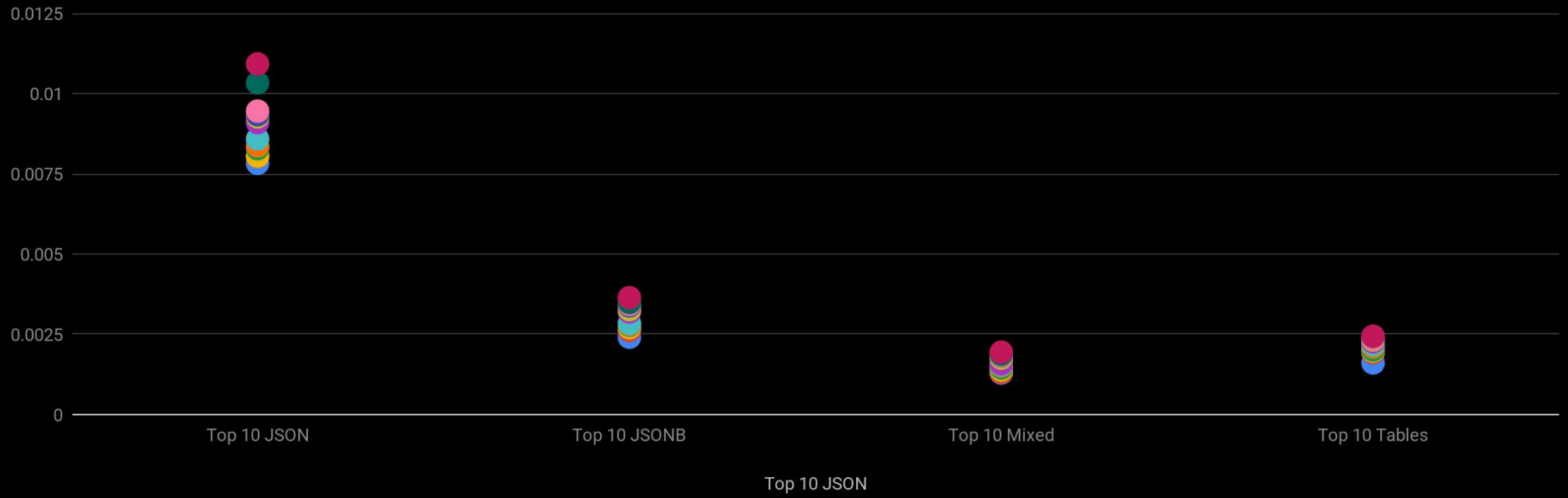
```
SELECT
    tweet.id,
    tweet.text,
    tweet.created_at,
    tweet.source,
    tweet.retweet_count,
    tweet.favorite_count,
    "user".name,
    "user".screen_name,
    "user".verified,
    "user".profile_image_url_https
FROM tweet
JOIN "user" ON tweet.user_id = "user".id
ORDER BY timestamp_ms::BIGINT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

Time: 2.198 ms

$ _

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

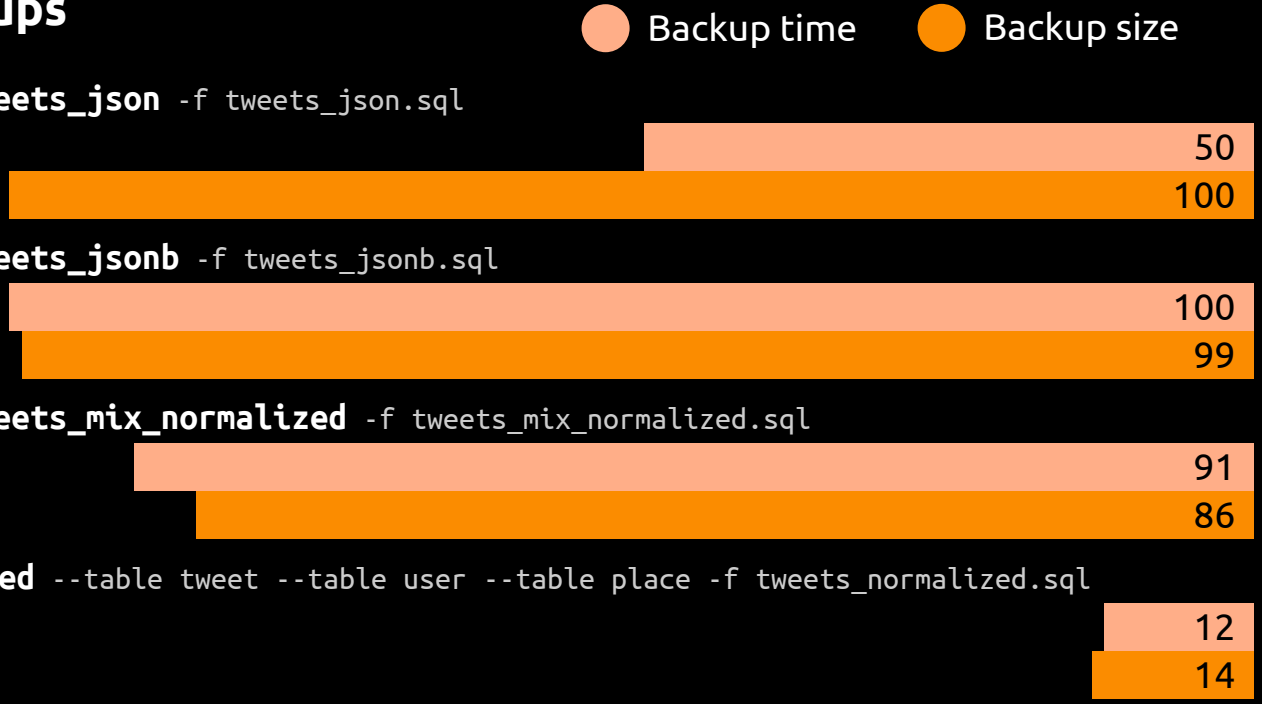
Name	Table	Size
idx_json_timestamp_ms	tweets_json	87 MB
idx_json_user_name	tweets_json	**132 MB**
idx_jsonb_timestamp_ms	tweets_jsonb	87 MB
idx_jsonb_user_name	tweets_jsonb	**132 MB**
idx_mixed_timestamp_ms	tweets_mix_normalized	87 MB
idx_mixed_user_name	tweets_mix_normalized	**132 MB**
idx_tweet_timestamp_ms	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



$ _

JSON vs JSONB vs Tables

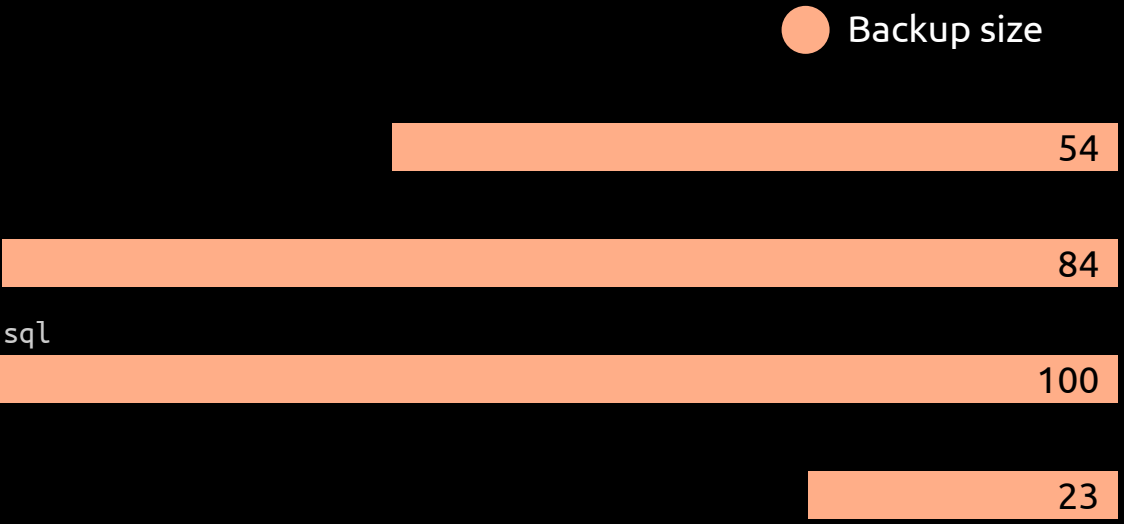
Real life example: Restores

```
$ 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)
```

$ _

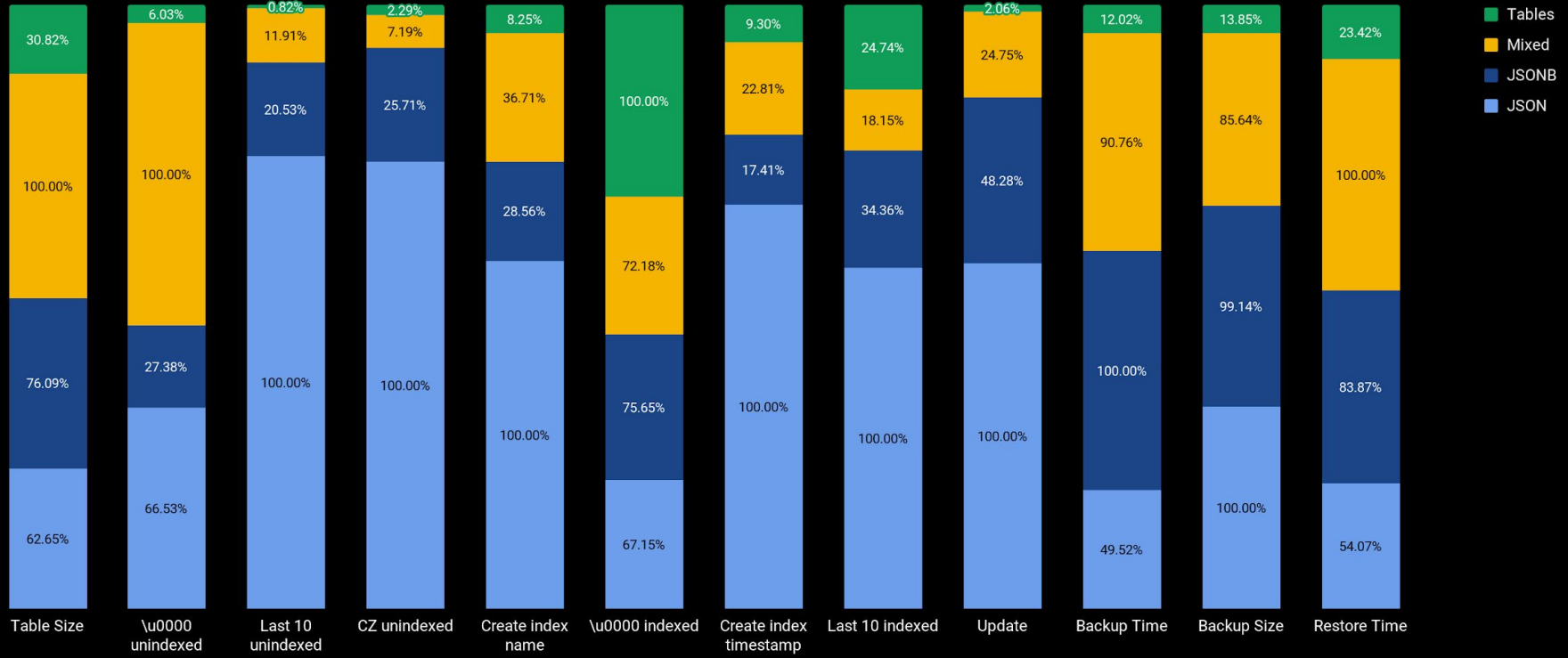
Conclusion Prelude

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

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

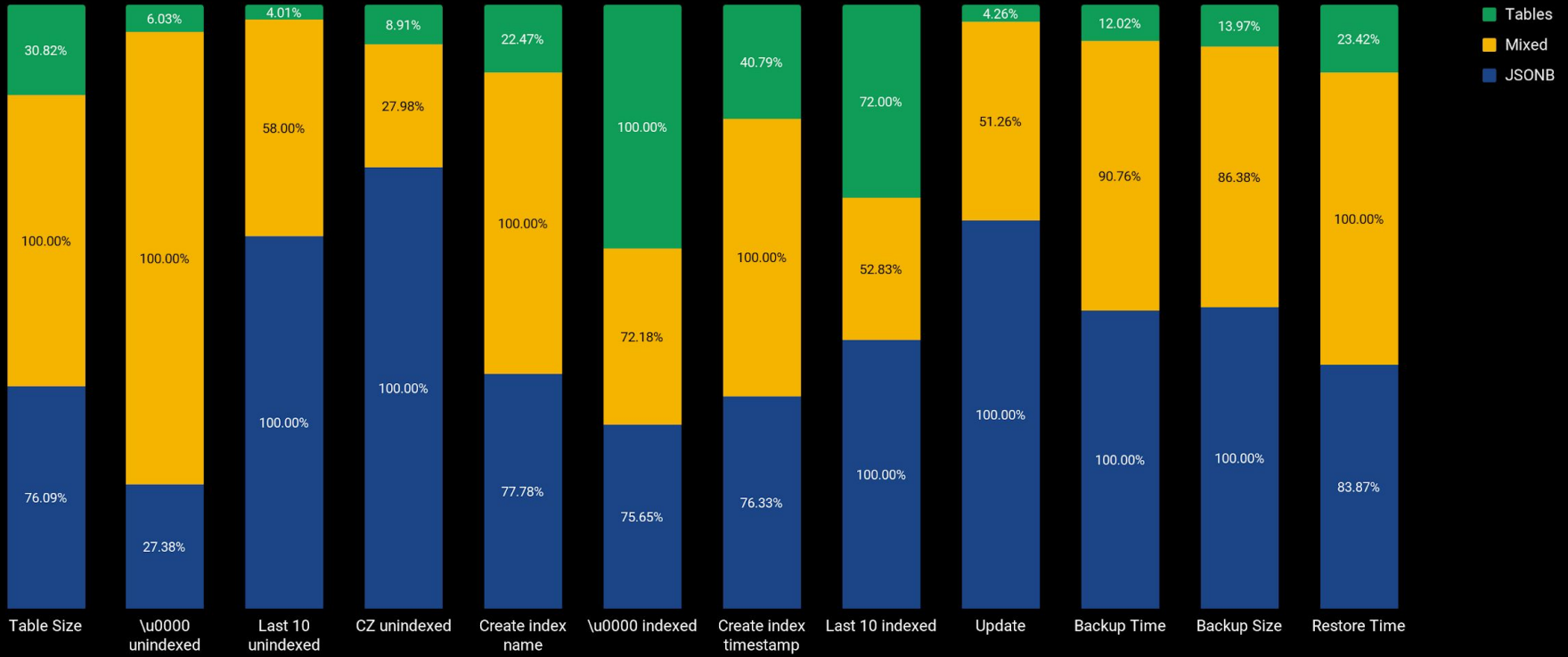
$ _

Conclusion


Tables
Mixed
JSONB
JSON
30.82%
100.00%
76.09%
62.65%
Table Size
6.03%
100.00%
27.38%
66.53%
\u0000 unindexed
0.82%
11.91%
20.53%
100.00%
Last 10 unindexed
2.29%
7.19%
25.71%
100.00%
CZ unindexed
8.25%
36.71%
28.56%
100.00%
Create index name
100.00%
72.18%
75.65%
67.15%
\u0000 indexed
9.30%
22.81%
17.41%
100.00%
Create index timestamp
24.74%
18.15%
34.36%
100.00%
Last 10 indexed
2.06%
24.75%
48.28%
100.00%
Update
12.02%
90.76%
100.00%
49.52%
Backup Time
13.85%
85.64%
99.14%
100.00%
Backup Size
23.42%
100.00%
83.87%
54.07%
Restore Time

$ _

Conclusion


Tables
Mixed
JSONB
30.82%
100.00%
76.09%
6.03%
100.00%
27.38%
4.01%
58.00%
100.00%
8.91%
27.98%
100.00%
22.47%
100.00%
77.78%
100.00%
72.18%
75.65%
40.79%
100.00%
76.33%
72.00%
52.83%
100.00%
4.26%
51.26%
100.00%
12.02%
90.76%
100.00%
13.97%
86.38%
100.00%
23.42%
100.00%
83.87%
Table Size
\u0000 unindexed
Last 10 unindexed
CZ unindexed
Create index name
\u0000 indexed
Create index timestamp
Last 10 indexed
Update
Backup Time
Backup Size
Restore Time