zalando

NOSQL
BEST PRACTICES

FOR POSTGRESQL

DMITRY DOLGOV

20-04-2018

Introduction

Less benchmarks

More opinionated best practices

Introduction

Application developers

DBAs

Extension developers

Application developers

When jsonb?

When jsonb?

➔ You have a distinct flexible model

When jsonb?

- ➔ You have a distinct flexible model
- ➔ You need to work with data provided in document oriented format

When jsonb?

- ➔ You have a distinct flexible model
- ➔ You need to work with data provided in document oriented format
- ➔ Workaround for technical issues (large number of tables or expensive alignment)

When not jsonb?

When not jsonb?

➔ Flexibility ”just in case”

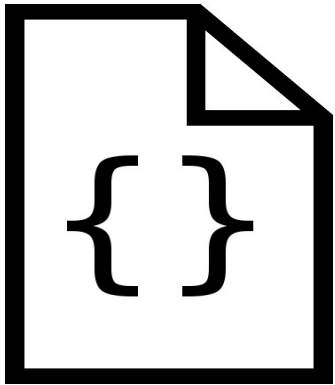
When not jsonb?

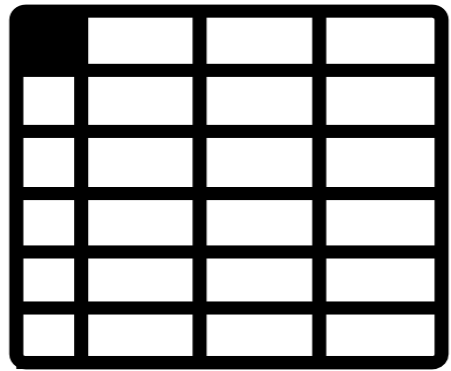
- ➔ Flexibility ”just in case”
- ➔ Reluctance to create a migration

When not jsonb?

- ➔ Flexibility ”just in case”
- ➔ Reluctance to create a migration
- ➔ Use jsonb column as a ”garbage can”

Jsonb -> Relation





zalando

When to move from jsonb to relation?

When to move from jsonb to relation?

➔ Queries rely significantly in information about internal structure of documents

When to move from jsonb to relation?

- ➔ Queries rely significantly in information about internal structure of documents
- ➔ There are too many constraints for documents

When to move from jsonb to relation?

- ➔ Queries rely significantly in information about internal structure of documents
- ➔ There are too many constraints for documents
- ➔ Some parts of document are used much more frequently than other

```
SELECT id, created FROM some_table
WHERE

(data->'name' = :a
AND (data @> ('{"items":[{"id":"' || :b || '"}]}'))
AND (data @> ('{"items":[{"elems":[{"name":"' || :c || '"}]}]}'))
AND (data @> ('{"items":[{"elems":[{"id":"' || :d || '"}]}]}'))
AND (data @> ('{"items":[{"name":"' || :e || '"}]}'))

ORDER BY created ASC, id ASC;
```

Complicated conditions

- ➔ jsquery
- ➔ SQL/JSON

Complicated conditions

```
SELECT id, created FROM some_table
WHERE

    data @@ 'items.#(id = ' || :a || ')'
    AND data @@ 'items.#.elems.#(name = ' || :b || ')'
    AND data @@ 'items.#.elems.#(id = ' || :c || ')'
    AND data @@ 'items.#(name = ' || :d || ')'

ORDER BY created ASC, id ASC;
```

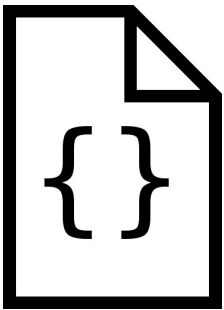
Complicated conditions

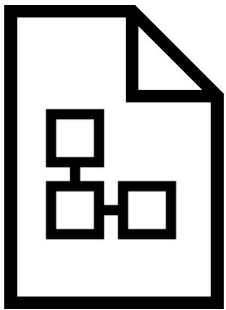
```
SELECT id, created FROM some_table
WHERE

    data @~ '$.items[*] ? (@id = ' || :a || ')'
    AND data @~ '$.items[*].elems[*] ? (@name = ' || :b || ')'
    AND data @~ '$.items[*].elems[*] ? (@id = ' || :c || ')'
    AND data @~ '$.items[*](@name = ' || :d || ')'

ORDER BY created ASC, id ASC;
```

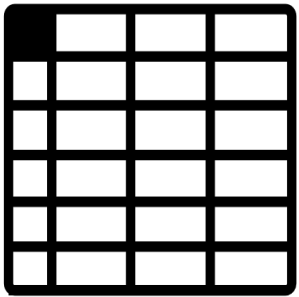
Complicated select

```
SELECT
    st.data #» '{item_a, another_item}' AS item_a,
    st.data #» '{item_c}'               AS item_c,
    jsonb_array_elements(
        data #> '{item_b, subitem_a, subitem_b}'
    ) -» 'some_key'                     AS item_e

    FROM some_table st LEFT JOIN another_table at
    ON (st.data #> '{item_b, key_a, key_b}') @>
        jsonb_build_array(jsonb_build_object(
            'key', 'some_key_name',
            'value', at.data #» '{item_b, another_item}'
        ));
```

Jsonb -> Relation







zalando

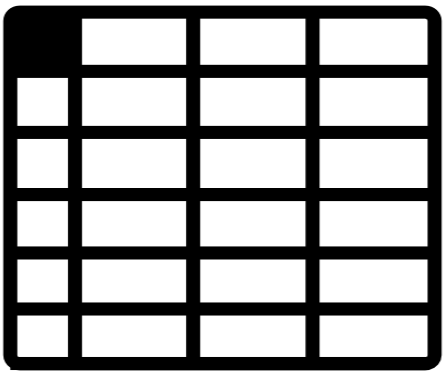
Constraints

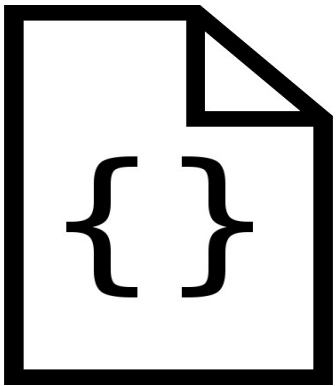
- ➔ Simple checks for value, type or size
- ➔ More convenient checks with jsquery
- ➔ Json schema

Constraints

```
CREATE TABLE test (
    data jsonb,
    CHECK (jsonb_typeof(data→'key') = 'array')
);
CREATE TABLE test (
    data jsonb,
    CHECK (data @@ 'key IS ARRAY OR key IS OBJECT')
);
CREATE TABLE test (
    data jsonb,
    CHECK (validate_json_schema('{"key": "array"}', data))
);
```

Relation -> Jsonb





zalando

```
SELECT jsonb_agg(query) FROM (
    SELECT id, data
    FROM jsonb_table
) query;
```

Seamless interaction between json and relation

zalando

```
[{
    "items": [
        {"id": 1, "value": "aaa"},
        {"id": 2, "value": "bbb"}
    ]
}, {
    "items": [
        {"id": 3, "value": "aaa"},
        {"id": 4, "value": "bbb"}
    ]
}]
```

```
WITH items AS (
    SELECT jsonb_array_elements(data->'items')
    AS item FROM test
)
SELECT * FROM items
WHERE item->>'value' = 'aaa';
```

```
item
--------
 {"id": 1, "value": "aaa"}
 {"id": 3, "value": "aaa"}
(2 rows)
```

```
WITH items AS (
    SELECT jsonb_array_elements(data->'items')
    AS item FROM test
)
SELECT * FROM items
WHERE item->>'value' = 'aaa';
```

```
item
--------
 {"id": 1, "value": "aaa"}
 {"id": 3, "value": "aaa"}
(2 rows)
```

```
{
    "items": {
        "item1": {"status": true},
        "item2": {"status": true},
        "item3": {"status": false}
    }
}
```

```
WITH items AS (
    SELECT jsonb_each(data->'items')
    AS item FROM test
)
SELECT (item).key FROM items
WHERE (item).value->>'status' = 'true';
```

```
 key
------
 item1
 item2
(2 rows)
```

```
WITH items AS (
    SELECT jsonb_each(data->'items')
    AS item FROM test
)
SELECT (item).key FROM items
WHERE (item).value->>'status' = 'true';
```

```
 key
------
 item1
 item2
(2 rows)
```

Multiple jsonb columns

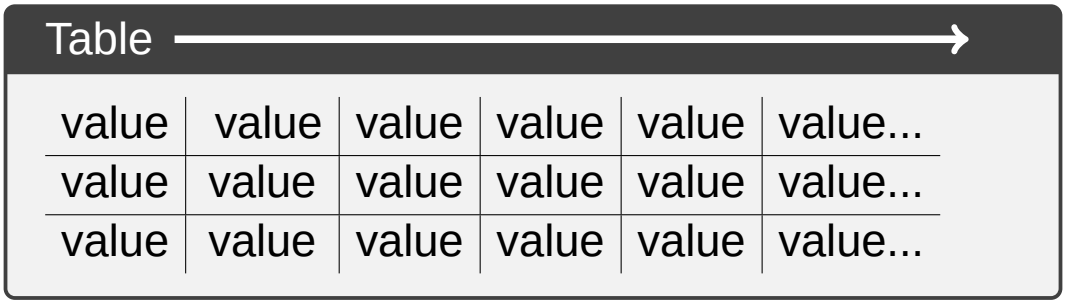
- ➔ Keep at the end for readability
- ➔ tuple_deform (PG11, JIT compilation)

Multiple jsonb columns

Table

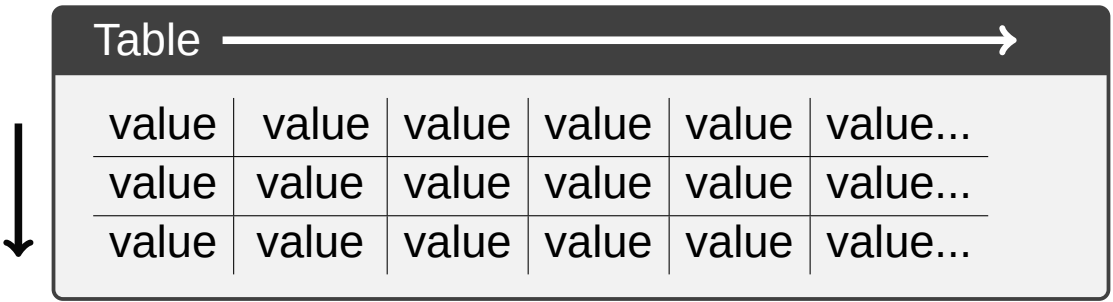
value	value	value	value	value	value...
value	value	value	value	value	value...
value	value	value	value	value	value...

Multiple jsonb columns



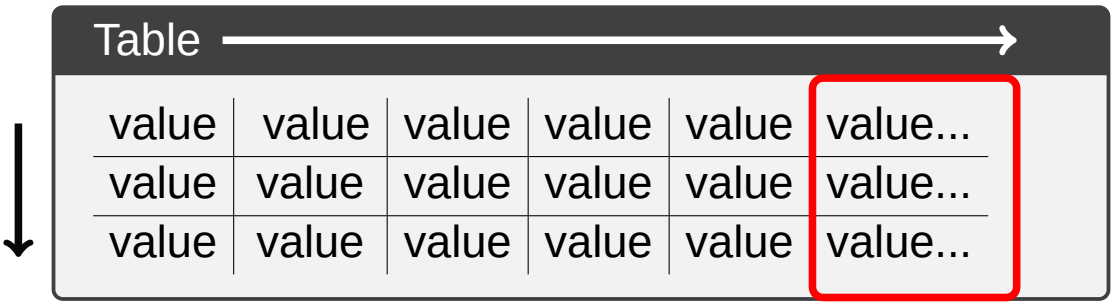
zalando

Multiple jsonb columns



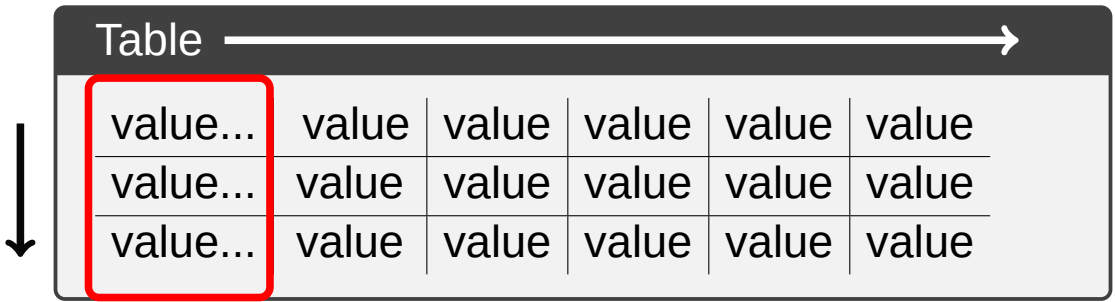
zalando

Multiple jsonb columns



zalando

Multiple jsonb columns



zalando

Document slice: in the DB or in the app?

- ➔ Amount of data passed from DB to application
- ➔ Performance hit in some cases
  (multiple detoasting)

```
select data->'key1'->'key2' from table;

select data->'key1', data->'key2' from table;
```

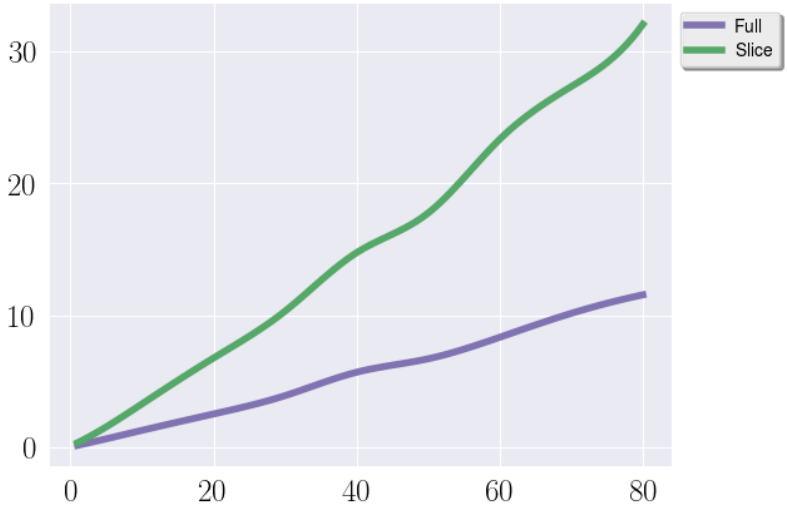
```
select data->‘key1‘->‘key2‘ from table;

select data->’key1’, data->’key2’ from table;
```


zalando

```
select data->'key1'->'key2' from table;

select data->‘key1‘, data->‘key2‘ from table;
```


zalando

Read latency, ms



zalando

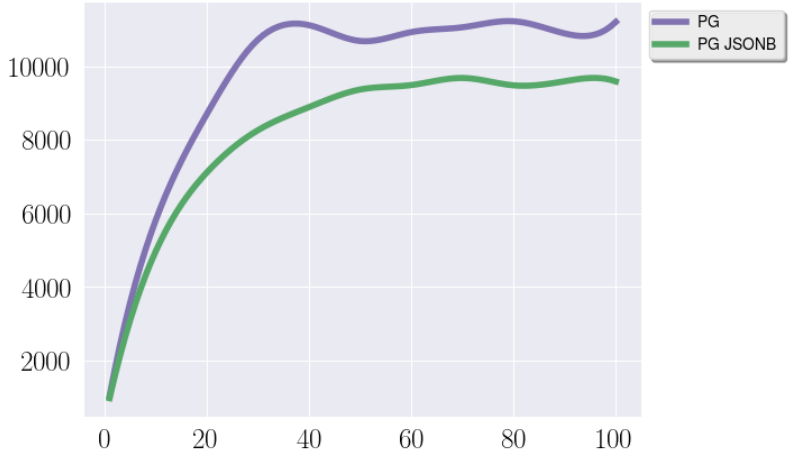
- ➔ Plain Json
- ➔ Binary Jsonb
- ➔ Relation

Insert workload

Throughput (ops/sec)

zalando

Throughput (ops/sec)

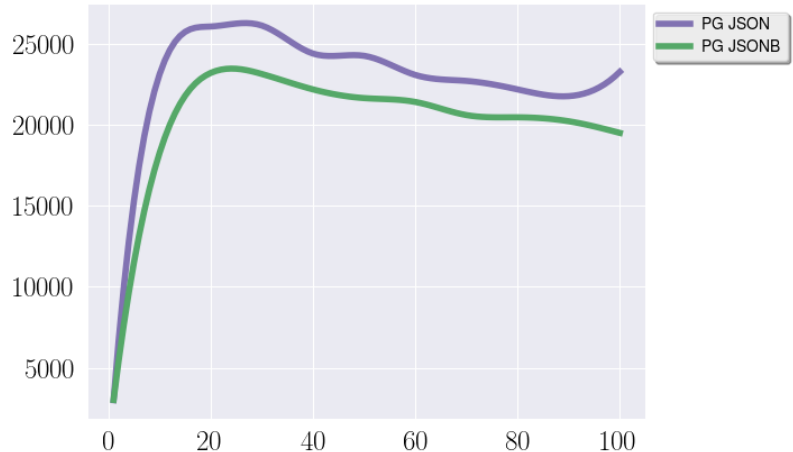


zalando

Read workload

zalando

Throughput (ops/sec)



zalando


Throughput (ops/sec)
25000
20000
15000
10000
5000
0
20
40
60
80
100
PG
PG JSONB

zalando

Jsonb array vs regular array

- ➔ Store elements of different type?
  Not really a ”single model” idea.
- ➔ Syntax is less natural (this may change)
- ➔ Updates are slower
- ➔ Arrays are 1-based, Jsonb 0-based

```
SELECT array[0] FROM some_table;
SELECT jsonb→0 FROM some_table;
– WIP
SELECT jsonb[0] FROM some_table;

UPDATE some_table SET array[0] = 'new_value';
UPDATE some_table
SET jsonb = jsonb_set(jsonb, '{0}', 'new_value');
– WIP
UPDATE some_table SET jsonb[0] = 'new_value';
```

Jsonb NULL != SQL NULL

```
SELECT jsonb_set(data, '{key}', NULL);
 jsonb_set
-----------
 NULL
(1 row)

SELECT jsonb_set(data, '{key}', 'null');
 jsonb_set
-----------
 {"key": null}
(1 row)
```

Some useful extensions

- ➔ jsquery
- ➔ postgres-json-schema
- ➔ is_jsonb_valid
- ➔ zson (custom compression methods WIP)
- ➔ jsonb_explorer

Types, please

@kickass-peanut

zalando

DBAs

Limitations

Size 256 MB

Depth - max_stack_depth

Stack depth is different for create & update

zalando

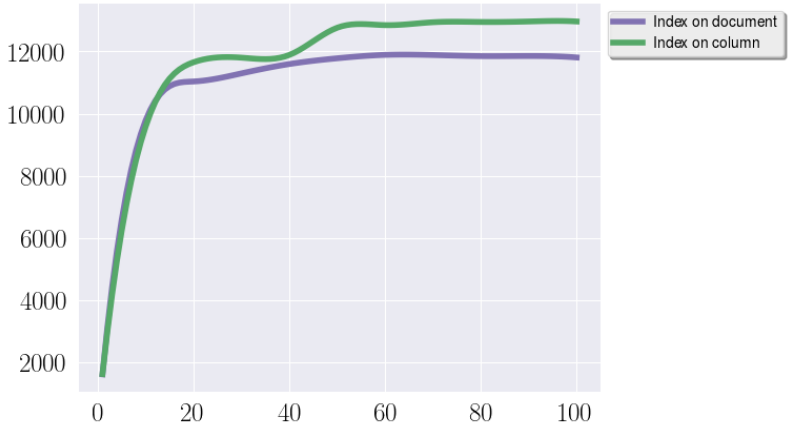
Indexing support

- ➔ GIN index (jsonb_ops, jsonb_path_ops)
- ➔ Functional BTree index
- ➔ jsquery strategies for GIN
- ➔ Partial indexes WIP

Place for ID

- ➔ Inside a document
- ➔ As as separate column

PG, Throughput (ops/sec)



zalando

Place for ID

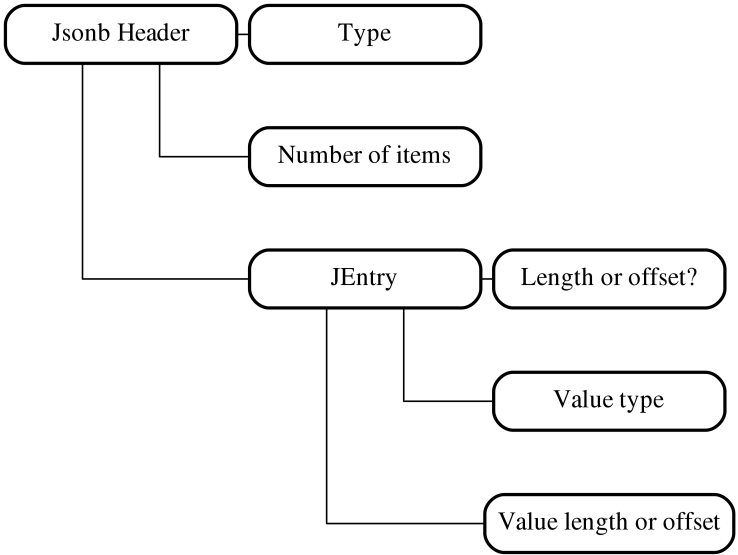
PostgreSQL 11 have HOT updates for some expression indexes, which will eliminate this problem.

Statistics

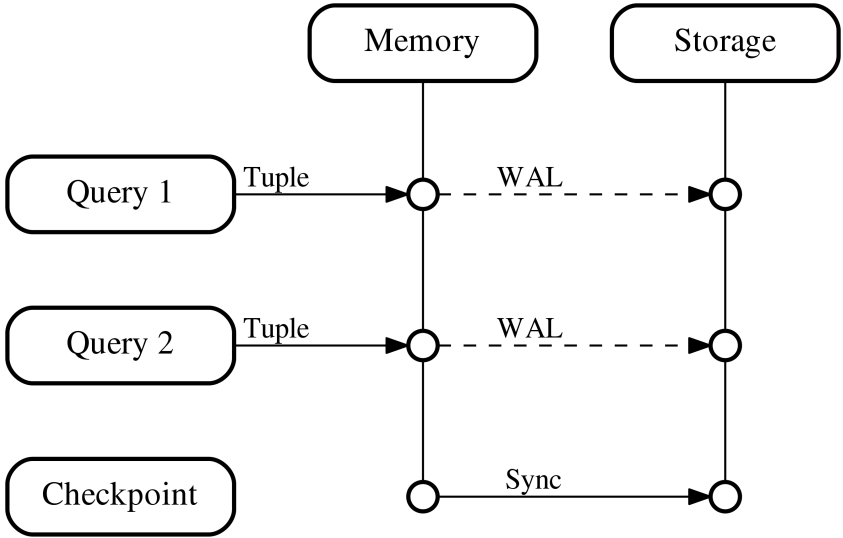
- ➔ There is no proper selectivity estimation for jsonb
- ➔ Optimizer can give wrong estimations for GIN and complex queries
- ➔ Functional indexes

How much to write?

Jsonb vs Json



zalando

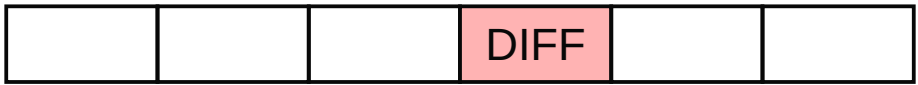

Memory
Storage
Query 1
Tuple
WAL
Query 2
Tuple
WAL
Checkpoint
Sync

zalando

How much to write?

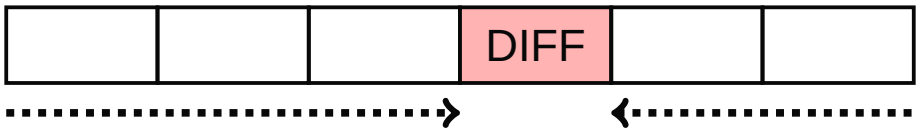
- ➔ Every update leads to update of an entire document (but it’s ok)
- ➔ WAL can have a full document or just a diff
- ➔ Old and new tuples fit into the same page - diff
- ➔ Old and new tuples fit into the same page - full
- ➔ If logical decoding is enabled - full

How much to write?



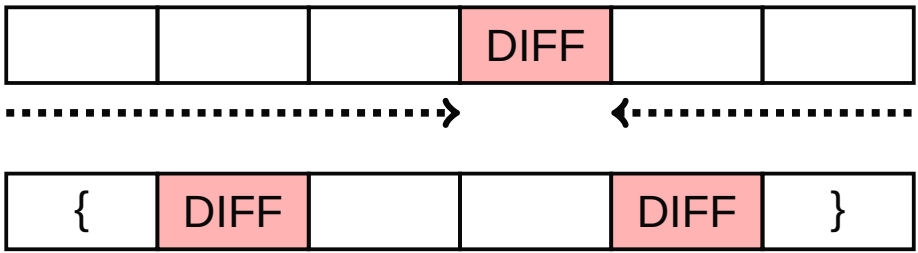
zalando

How much to write?



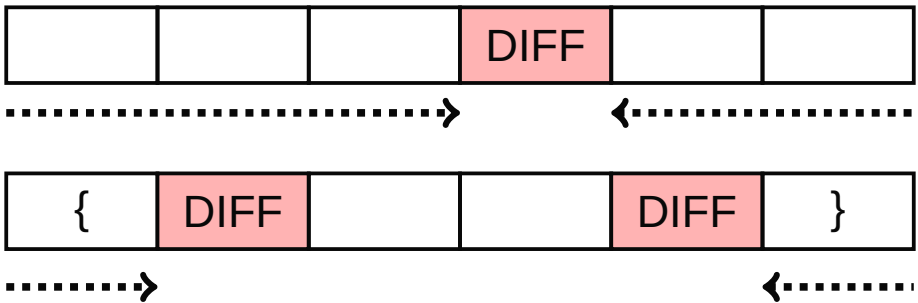
zalando

How much to write?



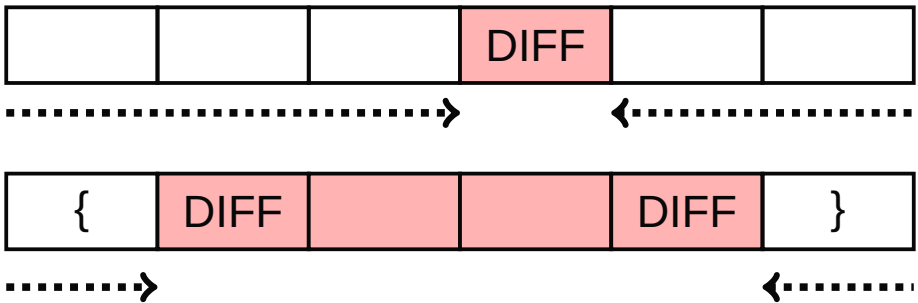
zalando

How much to write?



zalando

How much to write?

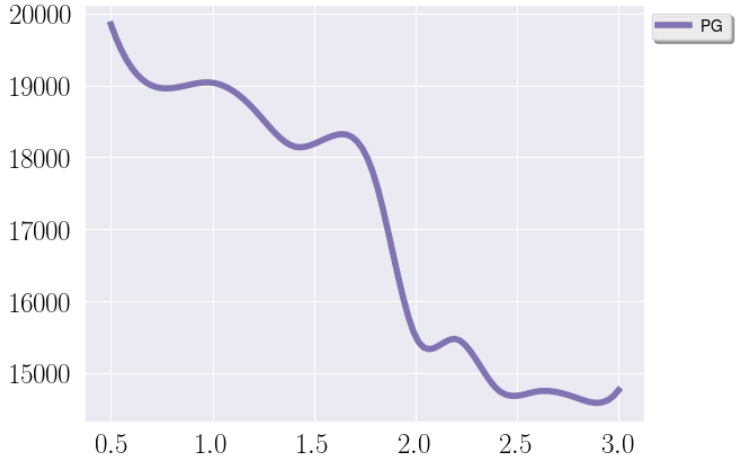


zalando

Huge documents

- ➔ TOAST has significant overhead (assemble, locks)
- ➔ Other than that linear degradation

Throughput, 40 clients



zalando

Alignment

Variable-length portion is aligned to a 4-byte

```
insert into test
values('{"a": "aa", "b": 1}');
```

```
abaa\x20\x00\x00\x00\x00\x80\x01\x00
```

```
insert into test
values('{"a": 1, "b": "aa"}');
```

```
\x00\x00ab\x00\x00\x20\x00\x00\x00\x00\x80\x01\x00aa
```

Extensions

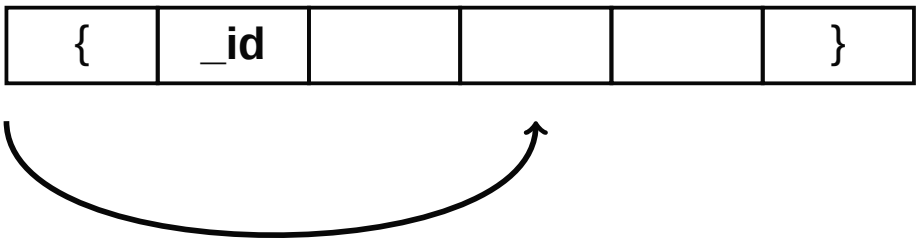
Why write an extension?

- ➔ Implement some convenient functionality (e.g. jsonb intersection)
- ➔ Create function optimized for your domain model

Why write an extension?

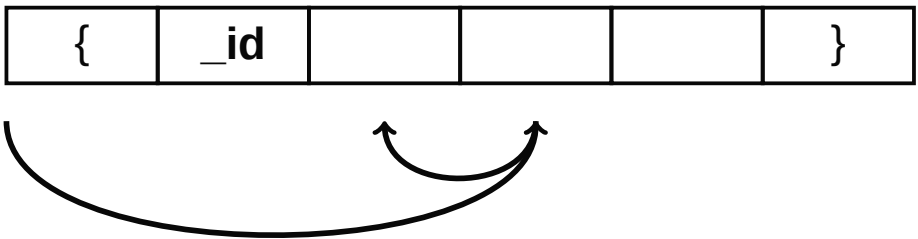
zalando

Why write an extension?



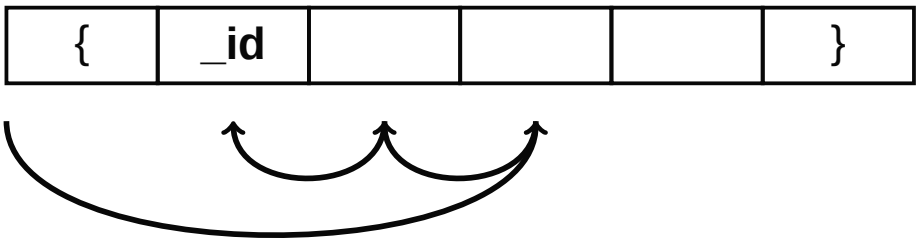
zalando

Why write an extension?



zalando

Why write an extension?



zalando

- ➔ Raw Jsonb container when search for an element
- ➔ Iterate through JsonbValue when update

Reuse infrastructure

- ➔ findJsonbValueFromContainer
- ➔ JsonbIterator
- ➔ addToParseState
- ➔ worker functions

Random tips

- ➔ Clone iterator
- ➔ String are not null-terminated

Questions?

- github.com/erthalion
- @erthalion
- dmitrii.dolgov at zalando dot de
- 9erthalion6 at gmail dot com

zalando