

PGConf Europe 2026

JSONB IN POSTGRESQL

Evil data type or just needs to be tamed?

Boriss Mejías

Global Solutions Architect
Air Guitar Player

PGConf Europe 2026

JSONB IN POSTGRESQL

Evil ~~data type~~ beast or just needs to be tamed?

Boriss Mejías

Global Solutions Architect
Air Guitar Player



WHY DOES SOMEONE WANT JSON?



SCHEMA FREEDOM

SCHEMA FREEDOM

ONE COLUMN

Every document carries the attributes it needs.

IMAGE

```
{  
  "height": 800,  
  "width": 600  
}
```

2 attributes

ONE COLUMN

Every document carries the attributes it needs.

IMAGE

```
{  
  "height": 800,  
  "width": 600  
}
```

2 attributes

GEOTAGGED IMAGE

```
{  
  "height": 2304,  
  "width": 4096,  
  "latitude": -33.43108,  
  "longitude": -70.65935  
}
```

4 attributes

ONE COLUMN, THREE SHAPES

Every document carries the attributes it needs.

IMAGE

```
{  
  "height": 800,  
  "width": 600  
}
```

2 attributes

GEOTAGGED IMAGE

```
{  
  "height": 2304,  
  "width": 4096,  
  "latitude": -33.43108,  
  "longitude": -70.65935  
}
```

4 attributes

AUDIO CLIP

```
{  
  "container": "ogg",  
  "duration": "00:05:12",  
  "tags": [  
    "interview",  
    "slonik",  
    "postgres"  
  ]  
}
```

3 attributes, one of them an array



METAL MUSIC USE CASE

METAL BAND

One band, one JSON document.

```
{  
  "id": 46015,  
  "name": "Saratoga",  
  "genre": ["Heavy Metal", "Power Metal",  
            "Speed Metal"],  
  "theme": ["Social issues", "Personal issues",  
            "Anti-totalitarianism"],  
  "active": "1992-2013, 2014-present",  
  "status": "Active",  
  "country": "Spain",  
  "formed_in": "1992"  
}
```

METAL BAND

One band, one JSON document.

```
{  
  "id": 46015,  
  "name": "Saratoga",  
  "genre": ["Heavy Metal", "Power Metal",  
            "Speed Metal"],  
  "theme": ["Social issues", "Personal issues",  
            "Anti-totalitarianism"],  
  "active": "1992-2013, 2014-present",  
  "status": "Active",  
  "country": "Spain",  
  "formed_in": "1992"  
}
```



BANDS	
band	jsonb

One row per band

METAL BAND AS A TABLE

```
CREATE TABLE metal.bands (  
    band jsonb  
);
```



BANDS	
band	jsonb

METAL BAND AS A TABLE

A unique id, enforced with an expression index.

```
CREATE TABLE metal.bands (  
    band jsonb  
);  
  
CREATE UNIQUE INDEX metal_band_idx  
ON metal.bands ((band->>'id'));
```



BANDS	
band	jsonb

Unique index on band->>'id'

METAL ALBUMS

Each album points to its band. But how?

```
{ "id": 26714,  
  "band": 46015,  
  "title": "XXX",  
  "year": 2021 }  
  
{ "id": 26715,  
  "band": 46015,  
  "title": "22/10/22... La historia continúa",  
  "year": 2023 }  
  
{ "id": 26716,  
  "band": 46015,  
  "title": "En estado puro",  
  "year": 2026 }
```

METAL ALBUMS

Each album points to its band. But how?

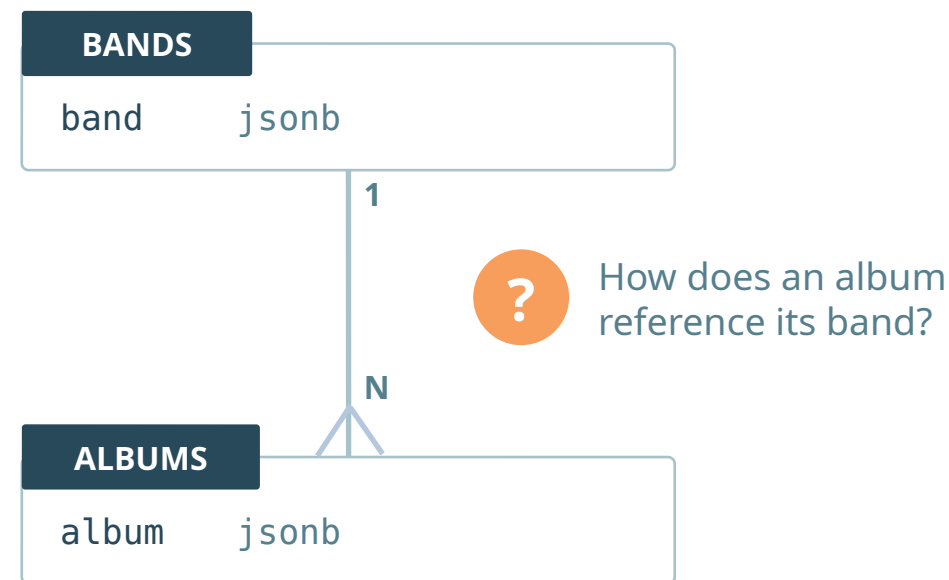
```
{ "id": 26714,  
  "band": 46015,  
  "title": "XXX",  
  "year": 2021 }  
  
{ "id": 26715,  
  "band": 46015,  
  "title": "22/10/22... La historia continúa",  
  "year": 2023 }  
  
{ "id": 26716,  
  "band": 46015,  
  "title": "En estado puro",  
  "year": 2026 }
```



METAL ALBUMS

Each album points to its band. But how?

```
{ "id": 26714,  
  "band": 46015,  
  "title": "XXX",  
  "year": 2021 }  
  
{ "id": 26715,  
  "band": 46015,  
  "title": "22/10/22... La historia continúa",  
  "year": 2023 }  
  
{ "id": 26716,  
  "band": 46015,  
  "title": "En estado puro",  
  "year": 2026 }
```



METAL BAND AS A TABLE

A unique id, enforced with an expression index.

```
CREATE TABLE metal.bands (  
    band jsonb  
);
```

```
CREATE UNIQUE INDEX metal_band_idx  
ON metal.bands ((band->>'id'));
```



BANDS	
band	jsonb

Unique index on band->>'id'

METAL BAND WITH PRIMARY KEY

Keys and relations become real columns.

```
CREATE TABLE metal.bands (  
    id      bigint PRIMARY KEY  
    , band  jsonb  
);
```

METAL BAND WITH PRIMARY KEY

Keys and relations become real columns.

```
CREATE TABLE metal.bands (  
    id      bigint PRIMARY KEY  
    , band  jsonb  
);  
  
CREATE TABLE metal.albums (  
    id      bigint PRIMARY KEY  
    , band  bigint REFERENCES metal.bands(id)  
    , title text  
    , year  integer  
);
```

METAL BAND WITH PRIMARY KEY

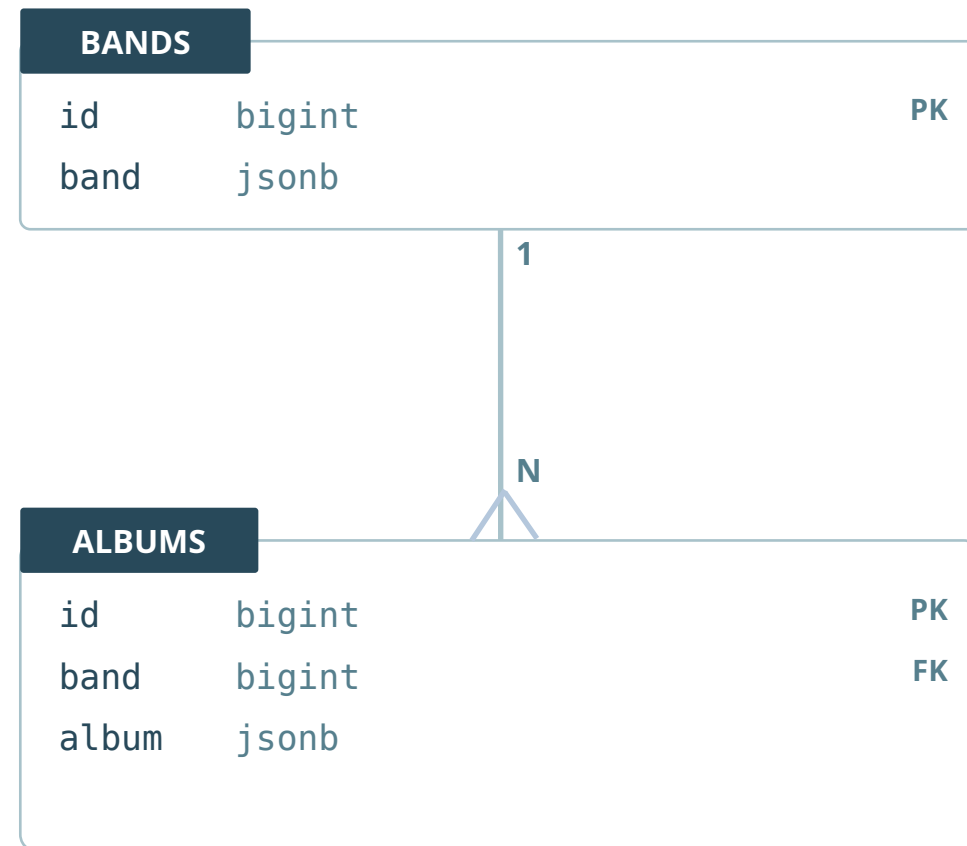
Keys and relations become real columns.

```
CREATE TABLE metal.bands (  
    id      bigint PRIMARY KEY  
    , band  jsonb  
);  
  
CREATE TABLE metal.albums (  
    id      bigint PRIMARY KEY  
    , band  bigint REFERENCES metal.bands(id)  
    , album jsonb  
);
```

METAL BAND WITH PRIMARY KEY

Keys and relations become real columns.

```
CREATE TABLE metal.bands (  
    id      bigint PRIMARY KEY  
    , band  jsonb  
);  
  
CREATE TABLE metal.albums (  
    id      bigint PRIMARY KEY  
    , band  bigint REFERENCES metal.bands(id)  
    , album jsonb  
);
```



METAL MUSIC USE CASE

METAL ALBUM REVIEWS

METAL ALBUM REVIEWS

DOCUMENT

```
{  
  "id": 1595042,  
  "album": 26715,  
  "title": "Saratoga is alive and kicking",  
  "score": 0.9,  
  "content": "This live performance of Saratoga  
shows their experience as a band and new energy  
with El Estepario Siberiano on drums..."  
}
```

METAL ALBUM REVIEWS

Key and foreign key as columns; the rest stays in jsonb.

DOCUMENT

```
{
  "id": 1595042,
  "album": 26715,
  "title": "Saratoga is alive and kicking",
  "score": 0.9,
  "content": "This live performance of Saratoga
shows their experience as a band and new energy
with El Estepario Siberiano on drums..."
}
```

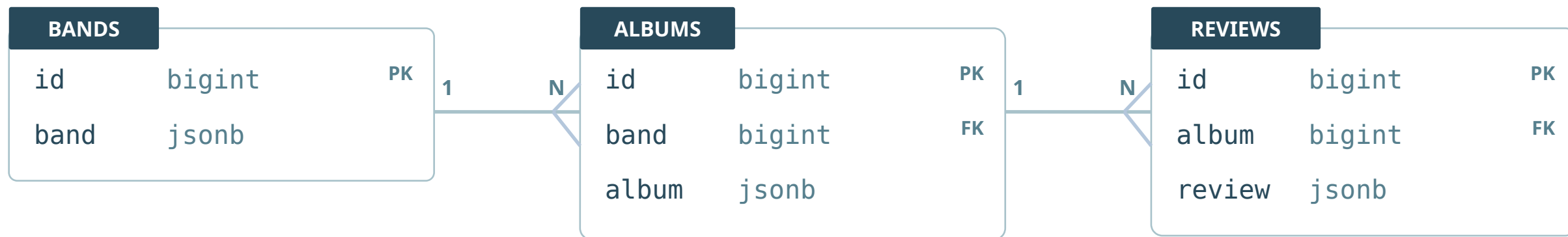
TABLE

```
CREATE TABLE metal.reviews (
  id      bigint PRIMARY KEY
, album  bigint REFERENCES metal.albums(id)
, review jsonb
);
```

id and **album** become relational columns. **title**, **score** and **content** stay inside **review** jsonb.

BARE METAL

The relational skeleton, with jsonb where the shape varies.



INDEXING JSON DATA

WHAT WILL THE APP QUERIES BE ABOUT?

Composite values (JSONB, ARRAY) are indexed with GIN.

A ROW

id	46015
band	<pre>{ "name": "Saratoga", "genre": ["Heavy Metal", "Power Metal", "Speed Metal"], "theme": ["Social issues", "Personal issues", "Anti-totalitarianism"], "active": "1992-2013, 2014-present", "status": "Active", "country": "Spain", "formed_in": "1992" }</pre>

INDEX THE PART YOU QUERY

GIN for the genre array, B-tree for the name.

GIN · GENRE

```
"genre": ["Heavy Metal", "Power Metal", "Speed Metal"]
```

```
CREATE INDEX idx_bands_genres_gin  
ON metal.bands  
USING GIN ((band->'genre'));
```

An array is a composite value, so it gets a GIN index.

INDEX THE PART YOU QUERY

GIN for the genre array, B-tree for the name.

GIN · GENRE

"genre": ["Heavy Metal", "Power Metal", "Speed Metal"]

```
CREATE INDEX idx_bands_genres_gin
ON metal.bands
USING GIN ((band->'genre'));
```

An array is a composite value, so it gets a GIN index.

B-TREE · NAME

"name": "Saratoga"

```
CREATE INDEX idx_bands_name_btree
ON metal.bands ((band->>'name'));
```

A single text value is fine with a B-tree.

FOR TEXT USE PG_TRGM

A trigram index on the review content.

A REVIEW ROW

id	1595042
album	26715
review	<pre>{ "title": "Saratoga is alive and kicking", "score": 0.9, "content": "This live performance of Saratoga shows their experience as a band and new energy with El Estepario Siberiano on drums..." }</pre>

FOR TEXT USE PG_TRGM

A trigram index on the review content.

A REVIEW ROW

id	1595042
album	26715
review	<pre>{ "title": "Saratoga is alive and kicking", "score": 0.9, "content": "This live performance of Saratoga shows their experience as a band and new energy with El Estepario Siberiano on drums..." }</pre>

FOR TEXT USE PG_TRGM

A trigram index on the review content.

A REVIEW ROW

id	1595042
album	26715
review	<pre>{ "title": "Saratoga is alive and kicking", "score": 0.9, "content": "This live performance of Saratoga shows their experience as a band and new energy with El Estepario Siberiano on drums..." }</pre>

TRIGRAM INDEX

```
CREATE EXTENSION pg_trgm  
WITH SCHEMA metal;
```

```
CREATE INDEX idx_album_review_trgm  
ON metal.reviews  
USING GIN  
((review->>'content') gin_trgm_ops);
```

I'M MISSING STRONG TYPING

STRONG TYPING

CASTS AND CONSTRAINTS

Typing the score field: an index with a cast

```
"score": 0.9
```

CASTS AND CONSTRAINTS

Typing the score field: an index with a cast

INDEX WITH A TYPE CAST

"score": 0.9

```
CREATE INDEX idx_review_score_double  
ON metal.reviews  
(((review->>'score')::double precision));
```

CASTS AND CONSTRAINTS

Typing the score field: an index with a cast, or just a constraint.

INDEX WITH A TYPE CAST

"score": 0.9

```
CREATE INDEX idx_review_score_double  
ON metal.reviews  
(((review->>'score')::double precision));
```

JUST ADD A CONSTRAINT

"score": 0.9

```
ALTER TABLE metal.reviews  
ADD CONSTRAINT numeric_score  
CHECK (pg_input_is_valid(review->>'score',  
                        'double precision'));
```

CASTS AND CONSTRAINTS

Typing the score field: an index with a cast, or just a constraint.

INDEX WITH A TYPE CAST

"score": 0.9

```
CREATE INDEX idx_review_score_double  
ON metal.reviews  
(((review->>'score')::double precision));
```

JUST ADD A CONSTRAINT

"score": 0.9

```
ALTER TABLE metal.reviews  
ADD CONSTRAINT numeric_score  
CHECK (pg_input_is_valid(review->>'score',  
                        'double precision'));
```

Table metal.reviews: 1.5M rows, 167MB

CASTS AND CONSTRAINTS

Typing the score field: an index with a cast, or just a constraint.

INDEX WITH A TYPE CAST

"score": 0.9

```
CREATE INDEX idx_review_score_double  
ON metal.reviews  
(((review->>'score')::double precision));
```

JUST ADD A CONSTRAINT

"score": 0.9

```
ALTER TABLE metal.reviews  
ADD CONSTRAINT numeric_score  
CHECK (pg_input_is_valid(review->>'score',  
                        'double precision'));
```

Table metal.reviews: 1.5M rows, 167MB

Creation time:	1206.056 ms
Index Size:	32 MB

CASTS AND CONSTRAINTS

Typing the score field: an index with a cast, or just a constraint.

INDEX WITH A TYPE CAST

"score": 0.9

```
CREATE INDEX idx_review_score_double  
ON metal.reviews  
(((review->>'score')::double precision));
```

JUST ADD A CONSTRAINT

"score": 0.9

```
ALTER TABLE metal.reviews  
ADD CONSTRAINT numeric_score  
CHECK (pg_input_is_valid(review->>'score',  
                        'double precision'));
```

Table metal.reviews: 1.5M rows, 167MB

Creation time:	1206.056 ms	273.785 ms
Index Size:	32 MB	\m/

STRONG TYPING

USING SQL/JSON

IS JSON predicates as CHECK constraints.

GIN · GENRE

"genre": ["Heavy Metal", "Power Metal", "Speed Metal"]

```
CREATE INDEX idx_bands_genres_gin
ON metal.bands
USING GIN ((band->'genre'));
```

An array is a composite value, so it gets a GIN index.

USING SQL/JSON

IS JSON predicates as CHECK constraints.

GIN · GENRE

"genre": ["Heavy Metal", "Power Metal", "Speed Metal"]

```
CREATE INDEX idx_bands_genres_gin
ON metal.bands
USING GIN ((band->'genre'));
```

An array is a composite value, so it gets a GIN index.

IS JSON ARRAY

```
ALTER TABLE metal.bands
ADD CONSTRAINT array_of_genres
CHECK (band->'genre' IS JSON ARRAY);
```

SCHEMA FREEDOM

...and what it costs

LET'S ADD SOME FIELDS

Each UPDATE writes a complete new version of the row.

```
UPDATE metal.reviews  
SET review = review || '{"likes": 25214}'  
WHERE id = 1595042;
```

LET'S ADD SOME FIELDS

Each UPDATE writes a complete new version of the row.

```
UPDATE metal.reviews
SET review = review || '{"likes": 25214}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"reposts": 134}'
WHERE id = 1595042;
```

LET'S ADD SOME FIELDS

Each UPDATE writes a complete new version of the row.

```
UPDATE metal.reviews
SET review = review || '{"likes": 25214}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"reposts": 134}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"shares": 1717}'
WHERE id = 1595042;
```

LET'S ADD SOME FIELDS

Each UPDATE writes a complete new version of the row.

```
UPDATE metal.reviews
SET review = review || '{"likes": 25214}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"reposts": 134}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"shares": 1717}'
WHERE id = 1595042;
```

start

TUPLE

1 MB

MVCC (ALL VERSIONS)

1 MB

LET'S ADD SOME FIELDS

Each UPDATE writes a complete new version of the row.

```
UPDATE metal.reviews
SET review = review || '{"likes": 25214}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"reposts": 134}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"shares": 1717}'
WHERE id = 1595042;
```

	TUPLE	MVCC (ALL VERSIONS)
start	1 MB	1 MB
+ likes	1 MB	2 MB

LET'S ADD SOME FIELDS

Each UPDATE writes a complete new version of the row.

```
UPDATE metal.reviews
SET review = review || '{"likes": 25214}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"reposts": 134}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"shares": 1717}'
WHERE id = 1595042;
```

	TUPLE	MVCC (ALL VERSIONS)
start	1 MB	1 MB
+ likes	1 MB	2 MB
+ reposts	1 MB	3 MB

LET'S ADD SOME FIELDS

Each UPDATE writes a complete new version of the row.

```
UPDATE metal.reviews
SET review = review || '{"likes": 25214}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"reposts": 134}'
WHERE id = 1595042;
```

```
UPDATE metal.reviews
SET review = review || '{"shares": 1717}'
WHERE id = 1595042;
```

	TUPLE	MVCC (ALL VERSIONS)
start	1 MB	1 MB
+ likes	1 MB	2 MB
+ reposts	1 MB	3 MB
+ shares	1 MB	4 MB

Three UPDATES leave four versions of the same 1 MB value.

ONE UPDATE INSTEAD OF THREE

Set all the new fields in a single statement.

```
UPDATE metal.reviews
SET review = review || '{
    "likes": 25214,
    "reposts": 134,
    "shares": 1717
}'
WHERE id = 1595042;
```

start

TUPLE

1 MB

MVCC (ALL VERSIONS)

1 MB

ONE UPDATE INSTEAD OF THREE

Set all the new fields in a single statement.

```
UPDATE metal.reviews
SET review = review || '{
    "likes": 25214,
    "reposts": 134,
    "shares": 1717
}'
WHERE id = 1595042;
```

	TUPLE	MVCC (ALL VERSIONS)
start	1 MB	1 MB
+ 3 fields	1 MB	2 MB

TAKEAWAY Three UPDATES leave 4 MB of versions. One UPDATE leaves 2 MB.



FREQUENTLY UPDATED FIELDS

FUF = Candidate for Columns

FUF = CANDIDATE FOR COLUMNS

METAL ALBUM REVIEWS

Move likes, reposts and shares out of the document.

id	1585042
album	26715
likes	25214
reposts	134
shares	1717
review	<pre>{ "title": "Saratoga is alive and kicking", "score": 0.9, "content": "This live performance of Saratoga shows their experience as a band and new energy with El Estepario Siberiano on drums..." }</pre>

FUF = CANDIDATE FOR COLUMNS

METAL ALBUM REVIEWS

Move likes, reposts and shares out of the document.

id	1585042
album	26715
likes	25214
reposts	134
shares	1717
review	{ "title": "Saratoga is alive and kicking", "score": 0.9, "content": "This live performance of Saratoga shows their experience as a band and new energy with El Esteparío Siberiano on drums..." }

```
UPDATE metal.reviews
SET likes = likes + 1,
    reposts = reposts + 1,
    shares = shares + 1
WHERE id = 18956;
```

FUF = CANDIDATE FOR COLUMNS

METAL ALBUM REVIEWS

Move likes, reposts and shares out of the document.

id	1585042
album	26715
likes	25214
reposts	134
shares	1717
review	{ "title": "Saratoga is alive and kicking", "score": 0.9, "content": "This live performance of Saratoga shows their experience as a band and new energy with El Esteparío Siberiano on drums..." }

```
UPDATE metal.reviews
SET likes = likes + 1,
    reposts = reposts + 1,
    shares = shares + 1
WHERE id = 18956;
```

mvcc: 1 MB + 50 bytes

instead of 2 to 4 MB when the fields lived in the jsonb

FUF = CANDIDATE FOR COLUMNS

METAL ALBUM REVIEWS

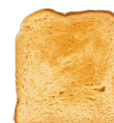
Move likes, reposts and shares out of the document.

id	1585042
album	26715
likes	25214
reposts	134
shares	1717
review	{ "title": "Saratoga is alive and kicking", "score": 0.9, "content": "This live performance of Saratoga shows their experience as a band and new energy with El Esteparío Siberiano on drums..." }

```
UPDATE metal.reviews  
SET likes = likes + 1,  
    reposts = reposts + 1,  
    shares = shares + 1  
WHERE id = 18956;
```

mvcc: 1 MB + 50 bytes

instead of 2 to 4 MB when the fields lived in the jsonb



TOAST: the large review stays out of line. The new row version only carries a pointer to it.

FUF = CANDIDATE FOR COLUMNS

METAL ALBU

FUF = CANDIDATE FOR COLUMNS

METAL ALBU

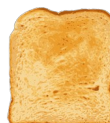
Move likes, reposts and shares out of the document.

id	1585042
album	26715
likes	25214
reposts	134
shares	1717
review	<pre>{ "title": "Saratoga and kicking", "score": 0, "content": "The performance of Saratoga shows their confidence and new energy with El Estepa on drums..." }</pre>

```
UPDATE metal.reviews
SET likes = likes + 1,
    reposts = reposts + 1,
    shares = shares + 1
WHERE id = 18956;
```

mvcc: 1 MB + 50 bytes
instead of 2 to 4 MB when the fields lived in the jsonb

mvcc: 1 MB + 50 bytes
instead of 2 to 4 MB when the fields lived in the jsonb



TOAST: the large review stays out of line. The new row version only carries a pointer to it.

CLOSING WORDS

CLOSING WORDS

TAMING THE JSONBEAST

Evil beast, or does it just need to be tamed?

GREAT FOR SCHEMA FREEDOM

Documents can differ
from row to row.

THE ANSWER JSONB isn't evil. It just needs to be tamed.

TAMING THE JSONBEAST

Evil beast, or does it just need to be tamed?

GREAT FOR SCHEMA FREEDOM

Documents can differ
from row to row.

GREAT MIX WITH RELATIONAL BEST PRACTICES

Primary keys and
foreign keys stay as
columns.

THE ANSWER JSONB isn't evil. It just needs to be tamed.

TAMING THE JSONBEAST

Evil beast, or does it just need to be tamed?

GREAT FOR SCHEMA FREEDOM

Documents can differ
from row to row.

GREAT MIX WITH RELATIONAL BEST PRACTICES

Primary keys and
foreign keys stay as
columns.

EXTRACT USEFUL FIELDS AS COLUMNS

Frequently updated
fields are the first
candidates.

THE ANSWER JSONB isn't evil. It just needs to be tamed.

TAMING THE JSONBEAST

Evil beast, or does it just need to be tamed?

GREAT FOR SCHEMA FREEDOM

Documents can differ
from row to row.

GREAT MIX WITH RELATIONAL BEST PRACTICES

Primary keys and
foreign keys stay as
columns.

EXTRACT USEFUL FIELDS AS COLUMNS

Frequently updated
fields are the first
candidates.

USE THE INDEX TYPE THAT IS REQUIRED

GIN for composites,
B-tree for scalars,
pg_trgm for text.

THE ANSWER JSONB isn't evil. It just needs to be tamed.

TAMING THE JSONBEAST

Evil beast, or does it just need to be tamed?

GREAT FOR SCHEMA FREEDOM

Documents can differ
from row to row.

GREAT MIX WITH RELATIONAL BEST PRACTICES

Primary keys and
foreign keys stay as
columns.

EXTRACT USEFUL FIELDS AS COLUMNS

Frequently updated
fields are the first
candidates.

USE THE INDEX TYPE THAT IS REQUIRED

GIN for composites,
B-tree for scalars,
pg_trgm for text.

ADD CONSTRAINTS ON TYPES

Casts and CHECK
constraints bring strong
typing back.

THE ANSWER JSONB isn't evil. It just needs to be tamed.

BORISS MEJÍAS

THANK YOU

Global Solutions Architect · Air Guitar Player



Feedback



MASTODON

@tchorix@mastodon.world

GITHUB

github.com/bmejias

LINKEDIN

linkedin.com/in/boriss-mejías-4637401

PGConf Europe 2026