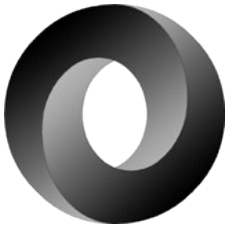
NoSQL as Not Only SQL

Backend Berlin
Dezember 2015
Stefanie Janine Stölting
@sjstoelting

JSON Datatypes



JSON

Available since 9.2

BSON

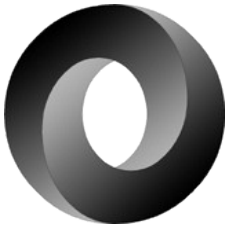
Available as extension on GitHub since 2013

JSONB

Available since 9.4

Crompessed JSON

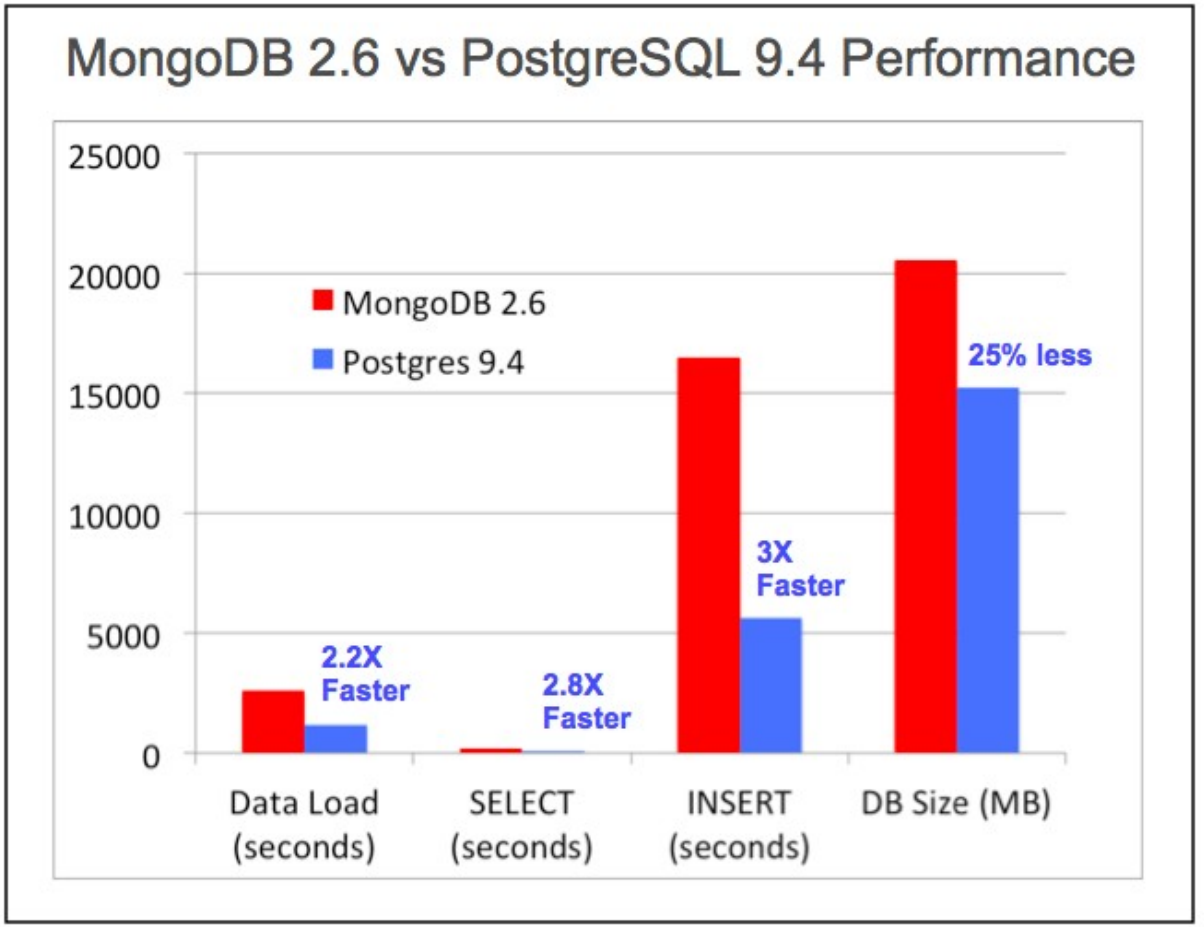
ACID



Atomicity, Consistency, Isolation, Durability is a set of properties that guarantee that database transactions are processed reliably. [1]

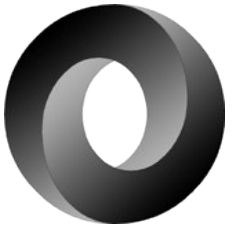
1 See https://en.wikipedia.org/wiki/ACID

Performance



Test done by EnterpriseDB, see the articleby Marc Linster

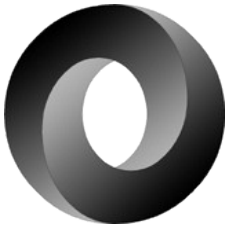
JSON Functions



row_to_json({row})
Returns the row as JSON

array_to_json({array})
Returns the array as JSON

JSON Opertators



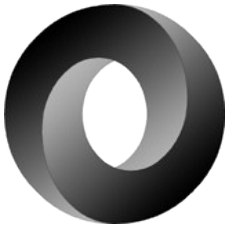
Array element
->{int}

Array element by name
->{text}

Object element
->> {text}

Value at path
#> {text}

Index on JSON



Index JSONB content for faster access

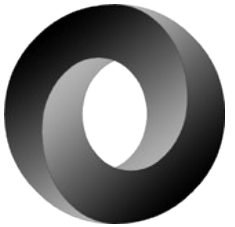
- GIN index overall

```
CREATE INDEX idx_1 ON jsonb.actor USING
GIN (jsondata);
```

- Even unique B-Tree indexes are possible

```
CREATE UNIQUE INDEX actor_id_2 ON
jsonb.actor((CAST(jsondata→'actor_id'::int)));
```

New JSON functions

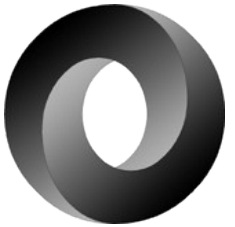


With the next version, 9.5, there will be new functions available, for example:

- jsonb_pretty
- jsonb_set

If you can't wait using them, you might be interested in the jsonbx extentsion available at PGXN

Data sources



The Chinook database is available at chinookdatabase.codeplex.com

Amazon book reviews of 1998 are available at examples.citusdata.com/customer_reviews_nested_1998.json.gz

CTE

Common Table Expressions will be used in examples

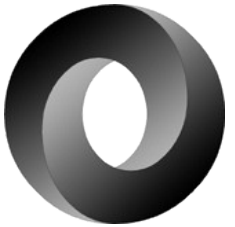
Example:

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n), min(n), max(n) FROM t;
```

Result:

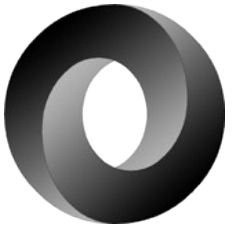
	sum bigint	min integer	max integer
1	5050	1	100

Live Examples



Let's see, how it does work.

Live with Chinook data

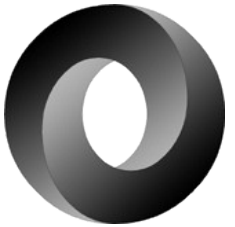


```
-- Step 1: Albums with tracks as JSON
WITH albums AS
    (
        SELECT a."ArtistId" AS artist_id
            , a."Title" AS album_title
            , array_agg(t."Name") AS album_tracks
        FROM "Album" AS a
        INNER JOIN "Track" AS t
            ON a."AlbumId" = t."AlbumId"
        GROUP BY a."ArtistId"
            , a."Title"
    )
SELECT row_to_json(albums) AS album_tracks
FROM albums
;
```

	album_tracks
1	{"artist_id":133,"album_title":"In Step","album_tracks":["The House Is Rockin'","Crossfire","Tightrope","Let Me Love You Baby","Leave My Girl Alone","T
2	{"artist_id":83,"album_title":"Deixa Entrar","album_tracks":["Deixa Entrar","Falamansa Song","Xote Dos Milagres","Zeca Violeiro","Avisa","Principiando/I
3	{"artist_id":15,"album_title":"The Best Of Buddy Guy - The Millenium Collection","album_tracks":["First Time I Met The Blues","Let Me Love You Baby",""
4	{"artist_id":78,"album_title":"Vault: Def Leppard's Greatest Hits","album_tracks":["Pour Some Sugar On Me","Photograph","Love Bites","Let's Get Rock
5	{"artist_id":205,"album_title":"Carry On","album_tracks":["No Such Thing","Poison Eye","Arms Around Your Love","Safe and Sound","She'll Never Be You
6	{"artist_id":58,"album_title":"Come Taste The Band","album_tracks":["Comin' Home","Lady Luck","Gettin' Tighter","Dealer","I Need Love","Drifter","Lov
7	{"artist_id":110,"album_title":"Nevermind","album_tracks":["Smells Like Teen Spirit","In Bloom","Come As You Are","Breed","Lithium","Polly","Territoria

200 row(s) fetched - 14ms Grid

Live with Chinook data

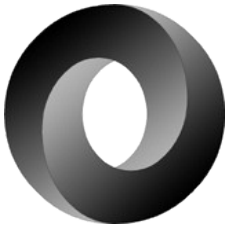


```
-- Step 2 Abums including tracks with aritsts
WITH albums AS
    (
        SELECT a."ArtistId" AS artist_id
            , a."Title" AS album_title
            , array_agg(t."Name") AS album_tracks
        FROM "Album" AS a
            INNER JOIN "Track" AS t
                ON a."AlbumId" = t."AlbumId"
        GROUP BY a."ArtistId"
            , a."Title"
    )
, js_albums AS
    (
        SELECT row_to_json(albums) AS album_tracks
        FROM albums
    )
SELECT a."Name" AS artist
    , al.album_tracks AS albums_tracks
FROM sqlite_artist AS a
    INNER JOIN js_albums AS al
        ON a."ArtistId" = CAST(al.album_tracks->>'artist_id' AS INT)
;
```

	artist	albums_tracks
1	AC/DC	{"artist_id":1,"album_title":"Let There Be Rock","album_tracks":["Go Down","Dog Eat Dog","Let There Be Rock","Bad Boy Boogie",
2	AC/DC	{"artist_id":1,"album_title":"For Those About To Rock We Salute You","album_tracks":["For Those About To Rock (We Salute You)
3	Accept	{"artist_id":2,"album_title":"Restless and Wild","album_tracks":["Fast As a Shark","Restless and Wild","Princess of the Dawn"]}
4	Accept	{"artist_id":2,"album_title":"Balls to the Wall","album_tracks":["Balls to the Wall"]}
5	Aerosmith	{"artist_id":3,"album_title":"Big Ones","album_tracks":["Walk On Water","Love In An Elevator","Rag Doll","What It Takes","Dude (
6	Alanis Morissette	{"artist_id":4,"album_title":"Jagged Little Pill","album_tracks":["All I Really Want","You Oughta Know","Perfect","Hand In My Pock
7	Alice In Chains	{"artist_id":5,"album_title":"Facelift","album_tracks":["We Die Young","Man In The Box","Sea Of Sorrow","Bleed The Freak","I Can
8	Apocalyptica	{"artist_id":7,"album_title":"Plays Metallica By Four Cellos","album_tracks":["Enter Sandman","Master Of Puppets","Harvester Of

200 row(s) fetched - 48ms Grid

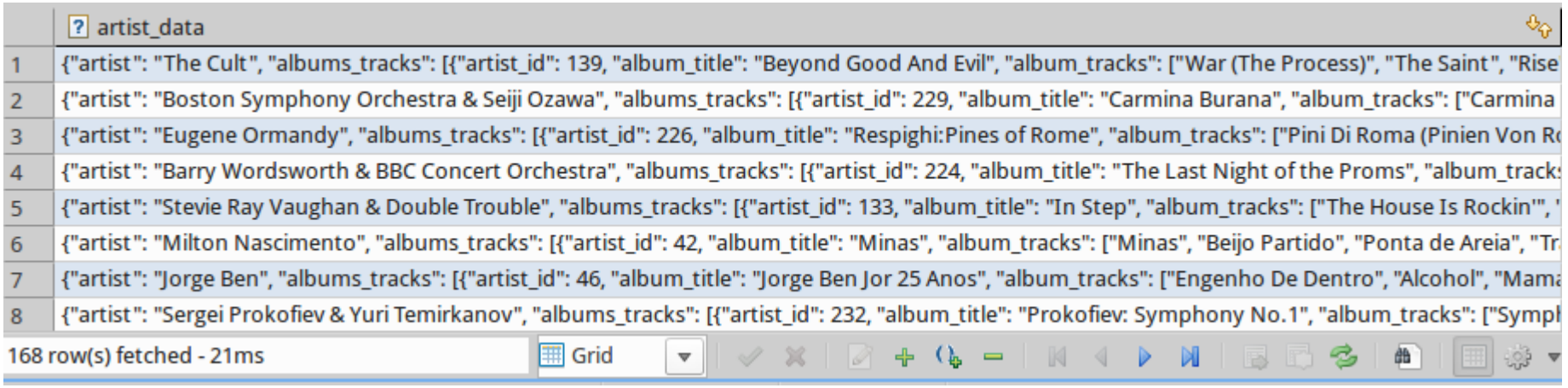
Live with Chinook data



```
-- Step 3 Return one row for an artist with all albums as VIEW
CREATE OR REPLACE VIEW v_artist_data AS
WITH albums AS
    (
        SELECT a."ArtistId" AS artist_id
            , a."Title" AS album_title
            , array_agg(t."Name") AS album_tracks
        FROM "Album" AS a
            INNER JOIN "Track" AS t
                ON a."AlbumId" = t."AlbumId"
        GROUP BY a."ArtistId"
            , a."Title"
    )
, js_albums AS
    (
        SELECT row_to_json(albums) AS album_tracks
        FROM albums
    )
, artist_albums AS
    (
        SELECT a."Name" AS artist
            , array_agg(al.album_tracks) AS albums_tracks
        FROM "Artist" AS a
            INNER JOIN js_albums AS al
                ON a."ArtistId" = CAST(al.album_tracks->>'artist_id' AS INT)
        GROUP BY a."Name"
    )
SELECT CAST(row_to_json(artist_albums) AS JSONB) AS artist_data
FROM artist_albums
;
```

Live with Chinook data

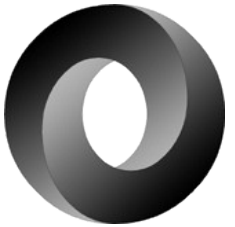
```
-- Select data from the view
SELECT *
FROM v_artist_data
;
```



	artist_data
1	{"artist": "The Cult", "albums_tracks": [{"artist_id": 139, "album_title": "Beyond Good And Evil", "album_tracks": ["War (The Process)", "The Saint", "Rise
2	{"artist": "Boston Symphony Orchestra & Seiji Ozawa", "albums_tracks": [{"artist_id": 229, "album_title": "Carmina Burana", "album_tracks": ["Carmina
3	{"artist": "Eugene Ormandy", "albums_tracks": [{"artist_id": 226, "album_title": "Respighi:Pines of Rome", "album_tracks": ["Pini Di Roma (Pinien Von R
4	{"artist": "Barry Wordsworth & BBC Concert Orchestra", "albums_tracks": [{"artist_id": 224, "album_title": "The Last Night of the Proms", "album_tracks
5	{"artist": "Stevie Ray Vaughan & Double Trouble", "albums_tracks": [{"artist_id": 133, "album_title": "In Step", "album_tracks": ["The House Is Rockin'",
6	{"artist": "Milton Nascimento", "albums_tracks": [{"artist_id": 42, "album_title": "Minas", "album_tracks": ["Minas", "Beijo Partido", "Ponta de Areia", "Tr
7	{"artist": "Jorge Ben", "albums_tracks": [{"artist_id": 46, "album_title": "Jorge Ben Jor 25 Anos", "album_tracks": ["Engenho De Dentro", "Alcohol", "Mama
8	{"artist": "Sergei Prokofiev & Yuri Temirkanov", "albums_tracks": [{"artist_id": 232, "album_title": "Prokofiev: Symphony No.1", "album_tracks": ["Symph

168 row(s) fetched - 21ms Grid

Live with Chinook data



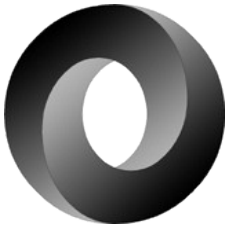
```
-- SELECT data from that VIEW, that does querying
SELECT jsonb_pretty(artist_data) pretty_artistdata
FROM v_artist_data
WHERE artist_data->>'artist' IN ('Miles Davis', 'AC/DC')
;
```

	pretty_artistdata
1	{¶ "artist": "Miles Davis",¶ "albums_tracks": ¶ [¶ {¶ "artist_id": 68,¶ "album_title": "The Essential Miles Davis [Disc 2]",¶ "albun
2	{¶ "artist": "AC/DC",¶ "albums_tracks": ¶ [¶ {¶ "artist_id": 1,¶ "album_title": "For Those About To Rock We Salute You",¶ "albu

2 row(s) fetched - 18ms

Grid

Live with Chinook data



```
-- SELECT some data that VIEW using JSON methods
SELECT jsonb_pretty(artist_data#>'{albums_tracks}') AS all_albums
    , jsonb_pretty(artist_data#>'{albums_tracks, 0}') AS tracks_0
    , artist_data#>'{albums_tracks, 0, album_title}' AS title
    , artist_data#>'{albums_tracks, 0, artist_id}' AS artist_id
    , artist_data->>'artist' AS artist
FROM v_artist_data
WHERE artist_data->'albums_tracks' @> '[{"album_title":"Miles Ahead"}]'
;
```

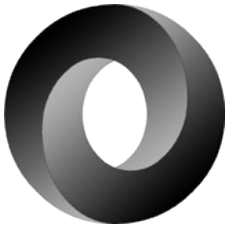
	all_albums	tracks_0	title	artist_id	artist
1	[¶ {¶ "artist_id": 68,¶ "albu	{¶ "artist_id": 68,¶ "album_title": "The Essenti	"The Essential Miles Davis [Disc 2]"	68	Miles Davis


```
{
  "artist_id": 68,
  "album_title": "The Essential Miles Davis [Disc 2]",
  "album_tracks":
  [
    "My Funny Valentine (Live)",
    "E.S.P.",
    "Nefertiti",
    "Petits Machins (Little Stuff)",
```


1 row(s) fetched - 19ms

Grid

Live with Chinook data

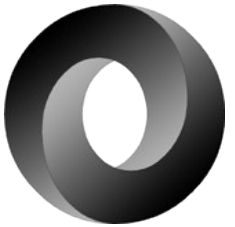


```
-- Array to records
SELECT jsonb_array_elements(artist_data#>'{albums_tracks}')->>'artist_id' AS artist_id
    , artist_data->>'artist' AS artist
    , jsonb_array_elements(artist_data#>'{albums_tracks}')->>'album_title' AS ablum_title
    , jsonb_array_elements(jsonb_array_elements(artist_data#>'{albums_tracks}')#>'{album_tracks}') AS song_titles
FROM v_artist_data
WHERE artist_data->'albums_tracks' @> '[{"artist_id":139}]'
ORDER BY 3, 4;
```

	artist_id	artist	ablum_title	song_titles
1	139	The Cult	Beyond Good And Evil	"American Gothic"
2	139	The Cult	Beyond Good And Evil	"Breathe"
3	139	The Cult	Beyond Good And Evil	"Heart Of Soul"
4	139	The Cult	Beyond Good And Evil	"Lil' Evil"
5	139	The Cult	Beyond Good And Evil	"Rain"
6	139	The Cult	Beyond Good And Evil	"Resurrection Joe"
7	139	The Cult	Beyond Good And Evil	"Rise"
8	139	The Cult	Beyond Good And Evil	"Shape The Sky"

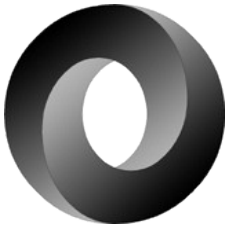
30 row(s) fetched - 17ms Grid

Live with Chinook data



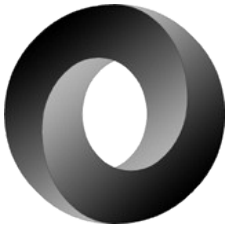
```
-- Create a function, which will be used for UPDATE on the view v_artrist_data
CREATE OR REPLACE FUNCTION trigger_v_artist_data_update()
    RETURNS trigger AS
$BODY$
    -- Data variables
    DECLARE rec               RECORD;
    -- Error variables
    DECLARE v_state           TEXT;
    DECLARE v_msg       TEXT;
    DECLARE v_detail    TEXT;
    DECLARE v_hint            TEXT;
    DECLARE v_context   TEXT;
BEGIN
    -- Update table Artist
    IF OLD.artist_data->>'artist' <> NEW.artist_data->>'artist' THEN
        UPDATE "Artist"
        SET "Name" = NEW.artist_data->>'artist'
        WHERE "ArtistId" = artist_data#>'{albums_tracks, 0, artist_id}';
    END IF;

    -- Update table Album in a foreach
    -- Update table Track in a foreach

    RETURN NEW;
    EXCEPTION WHEN unique_violation THEN
        RAISE NOTICE 'Sorry, but the something went wrong while trying to update artist data';
    RETURN OLD;
    WHEN others THEN
        GET STACKED DIAGNOSTICS
            v_state = RETURNED_SQLSTATE,
            v_msg = MESSAGE_TEXT,
            v_detail = PG_EXCEPTION_DETAIL,
            v_hint = PG_EXCEPTION_HINT,
            v_context = PG_EXCEPTION_CONTEXT;
        RAISE NOTICE '%', v_msg;
        RETURN OLD;
END;
$BODY$
    LANGUAGE plpgsql;
```

Live with Chinook data



```
-- The trigger will be fired instead of an UPDATE statemen to save data
CREATE TRIGGER v_artist_data_instead_update INSTEAD OF UPDATE
    ON v_artist_data
    FOR EACH ROW
    EXECUTE PROCEDURE trigger_v_artist_data_update()
;
```

Live Amazon reviews

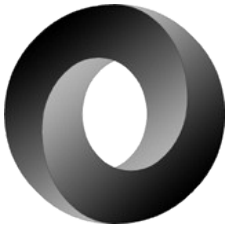


```
-- Create a table for JSON data with 1998 Amazon reviews
CREATE TABLE reviews(review_jsonb jsonb);
```

Name	Value
Query	CREATE TABLE reviews(review_jsonb jsonb)
Updated Rows	0

1 row(s) fetched - 32ms

Live Amazon reviews



```
-- Import customer reviews from a file
COPY reviews FROM '/var/tmp/customer_reviews_nested_1998.json'
```

Name	Value
Query	-- Import customer reviews from a file COPY reviews FROM '/var/tmp/customer_reviews_nested_1998.json'
Updated Rows	0

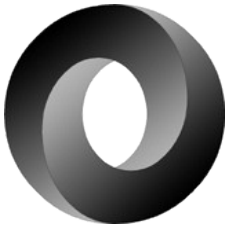
1 row(s) fetched - 10730ms

```
-- Maintenance the filled table
VACUUM ANALYZE reviews;
```

Name	Value
Query	-- Maintenance the filled table VACUUM ANALYZE reviews
Updated Rows	0

1 row(s) fetched - 2371ms

Live Amazon reviews

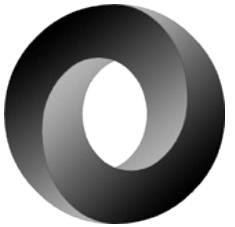


```
-- There should be 589.859 records imported into the table
SELECT count(*) FROM reviews;
```

	count
1	589.859

1 row(s) fetched - 104ms

Live Amazon reviews



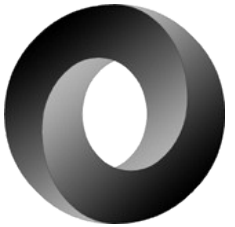
```
-- Select data with JSON
SELECT
    review_jsonb#>> '{product,title}' AS title
    , avg((review_jsonb#>> '{review,rating}')::int) AS average_rating
FROM reviews
WHERE review_jsonb@>'{"product": {"category": "Sheet Music & Scores"}}'
GROUP BY 1
ORDER BY 2 DESC
;
```

Without an Index: 248ms

	title	average_rating
1	Complete Works for Solo Keyboard	5
2	The Magic Flute (Die Zauberflote in Full Score)	5
3	Requiem in Full Score	5
4	The Four Seasons and Other Violin Concertos in Full Score	5
5	Symphony No. 3 (Dover Miniature Scores)	5

12 row(s) fetched - 248ms Grid

Live Amazon reviews



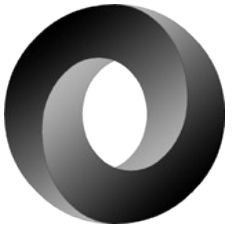
```
-- Create a GIN index
CREATE INDEX review_review_jsonb ON reviews USING GIN (review_jsonb);
```

Name	Value
Query	-- Create a GIN index CREATE INDEX review_review_jsonb ON reviews USING GIN (review_jsonb)
Updated Rows	0

1 row(s) fetched - 21079ms

Grid

Live Amazon reviews



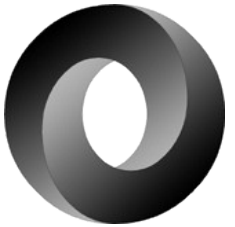
```
-- Select data with JSON
SELECT
    review_jsonb#>> '{product,title}' AS title
    , avg((review_jsonb#>> '{review,rating}')::int) AS average_rating
FROM reviews
WHERE review_jsonb@>'{"product": {"category": "Sheet Music & Scores"}}'
GROUP BY 1
ORDER BY 2 DESC
;
```

The same query as before with the previously created GIN Index: 7ms

	title	average_rating
1	Complete Works for Solo Keyboard	5
2	The Magic Flute (Die Zauberflote in Full Score)	5
3	Requiem in Full Score	5
4	The Four Seasons and Other Violin Concertos in Full Score	5
5	Symphony No. 3 (Dover Miniature Scores)	5

12 row(s) fetched - 7ms Grid

Live Amazon reviews



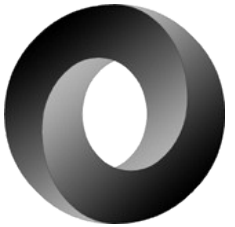
```
-- SELECT some statistics from the JSON data
SELECT  review_jsonb#>>'{product,category}' AS category
    , avg((review_jsonb#>>'{review,rating}')::int) AS average_rating
    , count((review_jsonb#>>'{review,rating}')::int) AS count_rating
FROM reviews
GROUP BY 1
;
```

Without an Index: 9747ms

	category	average_rating	count_rating
1		4,487	1.521
2	Accessories	4,703	37
3	Action & Adventure	4,261	3.938
4	African American Cinema	4,694	36
5	Alternative Rock	4,522	15.508

84 row(s) fetched - 9747ms Grid

Live Amazon reviews



```
-- Create a B-Tree index on a JSON expression
CREATE INDEX reviews_product_category ON reviews ((review_jsonb#>>'{product,category}'));
```

Name	Value
Query	-- Create a B-Tree index on a JSON expression CREATE INDEX reviews_product_category ON reviews ((review_jsonb#>>'{product,category}'))
Updated Rows	0

1 row(s) fetched - 11875ms

Grid

Live Amazon reviews

```
-- SELECT some statistics from the JSON data
SELECT  review_jsonb#>>'{product,category}' AS category
    , avg((review_jsonb#>>'{review,rating}')::int) AS average_rating
    , count((review_jsonb#>>'{review,rating}')::int) AS count_rating
FROM reviews
GROUP BY 1
;
```

The same query as before with the previously created BTREE Index: 1605ms

	category	average_rating	count_rating
1		4,487	1.521
2	Accessories	4,703	37
3	Action & Adventure	4,261	3.938
4	African American Cinema	4,694	36
5	Alternative Rock	4,522	15.508

84 row(s) fetched - 1605ms Grid

User Group / Conferences

PostgreSQL User Group Berlin

PGDay FOSDEM
Brussels, January 29 2016

FOSDEM Dev Room
Brussels, January 31 2016

Link List

Extended JSONB functions for 9.5

Extended JSONB functions as extension for 9.4 from PGXN (jsonbx)

Slide and source on Github: https://github.com/sjstoelting/talks

JSON by the other Elephant