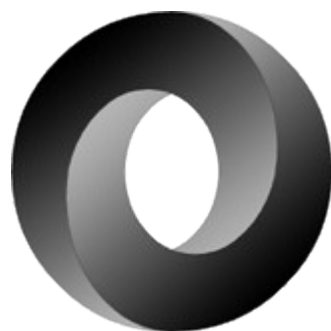


JSON by the other elephant

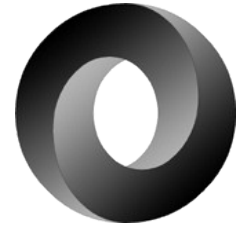


PostgreSQL

FOSDEM PGDay
January 2016
Stefanie Janine Stölting
[@sjstoelting](#)



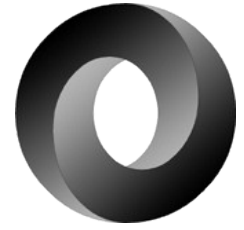
JSON



- JavaScript Object Notation
- Don't have to care about encoding, it is always Unicode, most implementations use UTF8
- Used for data exchange in web application
- Currently two standards [RFC 7159](#) by Douglas Crockford und ECMA-404
- PostgreSQL implementation is RFC 7159



JSON Datatypes



JSON

Available since 9.2

BSON

Available as extension on GitHub since 2013

JSONB

Available since 9.4

Crompressed JSON



ACID



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

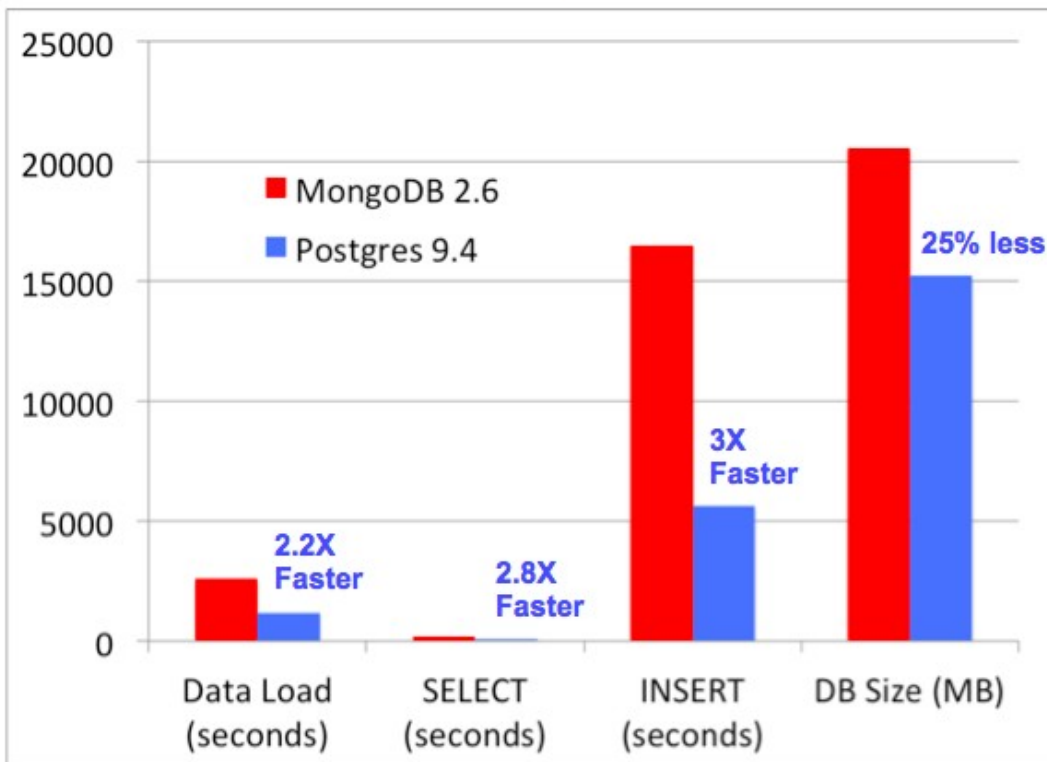
¹ See <https://en.wikipedia.org/wiki/ACID>



Performance



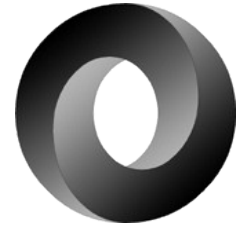
MongoDB 2.6 vs PostgreSQL 9.4 Performance



Test done by
[EnterpriseDB](#),
see the [article](#) by
[Marc Linster](#)



JSON Functions



`row_to_json({row})`

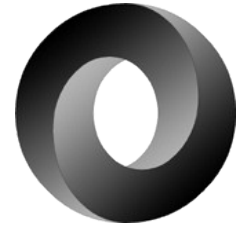
Returns the row as JSON

`array_to_json({array})`

Returns the array as JSON



JSON Operators



Array element

->{int}

Array element by name

->{text}

Object element

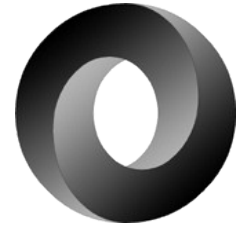
->> {text}

Value at path

#> {text}



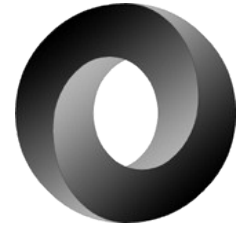
Index on JSON



- Index JSONB content for faster access with indexes
 - **GIN** index overall
 - CREATE INDEX idx_1 ON jsonb.actor USING GIN (jsontdata);
 - Even unique **B-Tree** indexes are possible
 - CREATE UNIQUE INDEX actor_id_2 ON jsonb.actor((CAST(jsontdata->>'actor_id' AS INTEGER)));



New JSON functions



PostgreSQL 9.5 new JSONB functions:

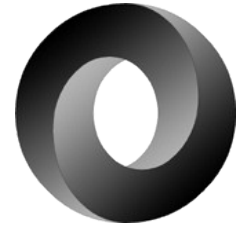
`jsonb_pretty`

`jsonb_set`

Available as extensions for 9.4 at PGXN: [jsonb_x](#)



Data sources



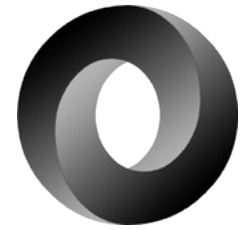
The Chinook database is available at chinookdatabase.codeplex.com

Amazon book reviews of 1998 are available at

examples.citusdata.com/customer_reviews_nested_1998.json.gz



Chinook Tables



	T tablename
1	Artist
2	Invoice
3	Employee
4	Customer
5	Playlist
6	InvoiceLine
7	Album
8	Genre
9	PlaylistTrack
10	MediaType
11	Track

	T table_name	T column_name	T data_type
1	Artist	ArtistId	integer
2	Artist	Name	character varying (120)

	T table_name	T column_name	T data_type
1	Album	AlbumId	integer
2	Album	Title	character varying (160)
3	Album	ArtistId	integer

	T table_name	T column_name	T data_type
1	Track	TrackId	integer
2	Track	Name	character varying (200)
3	Track	AlbumId	integer
4	Track	MediaTypeId	integer
5	Track	GenreId	integer
6	Track	Composer	character varying (220)
7	Track	Milliseconds	integer
8	Track	Bytes	integer
9	Track	UnitPrice	numeric



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



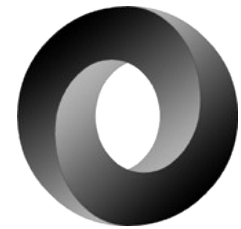
Live Examples



Let's see, how it does work.



Live with Chinook data



-- Step 1: Tracks as JSON with the album identifier

```
WITH tracks AS
(
    SELECT "AlbumId" AS album_id
        , "TrackId" AS track_id
        , "Name" AS track_name
    FROM "Track"
)
SELECT row_to_json(tracks) AS tracks
FROM tracks
;
```

tracks	
1	{"album_id":1,"track_id":1,"track_name":"For Those About To Rock (We Salute You)"}
2	{"album_id":2,"track_id":2,"track_name":"Balls to the Wall"}
3	{"album_id":3,"track_id":3,"track_name":"Fast As a Shark"}
4	{"album_id":3,"track_id":4,"track_name":"Restless and Wild"}
5	{"album_id":3,"track_id":5,"track_name":"Princess of the Dawn"}
6	{"album_id":1,"track_id":6,"track_name":"Put The Finger On You"}
7	{"album_id":1,"track_id":7,"track_name":"Let's Get It Up"}

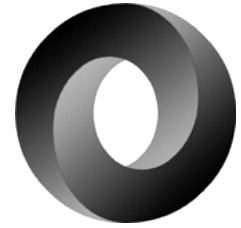
200 row(s) fetched - 7ms

Grid

✓ ✗ ✎ + (- ⏮ ⏪ ⏩ ⏭ ↺ ↻ 📄 ⚙



Live with Chinook data



```
-- Step 2 Albums including tracks with artist identifier
WITH tracks AS
(
    SELECT "AlbumId" AS album_id
      , "TrackId" AS track_id
      , "Name" AS track_name
    FROM "Track"
)
, json_tracks AS
(
    SELECT row_to_json(tracks) AS tracks
    FROM tracks
)
, albums AS
(
    SELECT a."ArtistId" AS artist_id
      , a."AlbumId" AS album_id
      , a."Title" AS album_title
      , array_agg(t.tracks) AS album_tracks
    FROM "Album" AS a
      INNER JOIN json_tracks AS t
      ON a."AlbumId" = (t.tracks->>'album_id')::int
    GROUP BY a."ArtistId"
      , a."AlbumId"
      , a."Title"
)
SELECT artist_id
      , array_agg(row_to_json(albums)) AS album
FROM albums
GROUP BY artist_id
;
```



Live with Chinook data



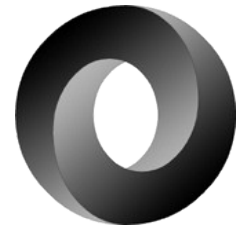
	artist_id	album
1	251	{"artist_id":251,"album_id":319,"album_title":"Armada: Music from the Courts of England and Spain","album_tracks":[{"album_id":319,"track_id":3389,"track_title":"The Armada Song"}]}
2	120	{"artist_id":120,"album_id":183,"album_title":"Dark Side Of The Moon","album_tracks":[{"album_id":183,"track_id":2591,"track_title":"Breathe (Afterglow)"}]}
3	227	{"artist_id":227,"album_id":293,"album_title":"Pavarotti's Opera Made Easy","album_tracks":[{"album_id":293,"track_id":3389,"track_title":"The Armada Song"}]}
4	8	{"artist_id":8,"album_id":271,"album_title":"Revelations","album_tracks":[{"album_id":271,"track_id":3389,"track_title":"The Armada Song"}]}
5	247	{"artist_id":247,"album_id":314,"album_title":"English Renaissance","album_tracks":[{"album_id":314,"track_id":2591,"track_title":"Breathe (Afterglow)"}]}
6	138	{"artist_id":138,"album_id":211,"album_title":"The Singles","album_tracks":[{"album_id":211,"track_id":2591,"track_title":"Breathe (Afterglow)"}]}
7	242	{"artist_id":242,"album_id":307,"album_title":"Adams, John: The Chairman Dances","album_tracks":[{"album_id":307,"track_id":3389,"track_title":"The Armada Song"}]}

168 row(s) fetched - 38ms

Grid



Live with Chinook data



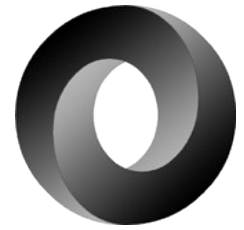
-- Step 3 Return one row for an artist with all albums as VIEW

```
CREATE OR REPLACE VIEW v_json_artist_data AS
WITH tracks AS
(
    SELECT "AlbumId" AS album_id
        , "TrackId" AS track_id
        , "Name" AS track_name
    FROM "Track"
)
, json_tracks AS
(
    SELECT row_to_json(tracks) AS tracks
    FROM tracks
)
, albums AS
(
    SELECT a."ArtistId" AS artist_id
        , a."AlbumId" AS album_id
        , a."Title" AS album_title
        , array_agg(t.tracks) AS album_tracks
    FROM "Album" AS a
        INNER JOIN json_tracks AS t
        ON a."AlbumId" = (t.tracks->>'album_id')::int
    GROUP BY a."ArtistId"
        , a."AlbumId"
        , a."Title"
)
, json_albums AS
(
    SELECT artist_id
        , array_agg(row_to_json(albums)) AS album
    FROM albums
    GROUP BY artist_id
)
```

-- -> Next page



Live with Chinook data



```
-- Step 3 Return one row for an artist with all albums as VIEW  
, artists AS
```

```
(  
    SELECT a."ArtistId" AS artist_id  
        , a."Name" AS artist  
        , jsa.album AS albums  
    FROM "Artist" AS a  
        INNER JOIN json_albums AS jsa  
        ON a."ArtistId" = jsa.artist_id  
)  
SELECT (row_to_json(artists))::jsonb AS artist_data  
FROM artists  
;
```



Live with Chinook data



-- Select data from the view

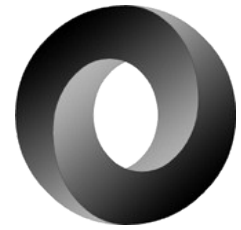
```
SELECT *  
FROM v_json_artist_data  
;
```

	? artist_data
1	{"albums": [{"album_id": 319, "artist_id": 251, "album_title": "Armada: Music from the Courts of England and Spain", "album_tracks": [{"album_id": 319, "track_id": 3389, "track_name": "The Armada Song"}]}
2	{"albums": [{"album_id": 183, "artist_id": 120, "album_title": "Dark Side Of The Moon", "album_tracks": [{"album_id": 183, "track_id": 3390, "track_name": "Breathe (Afterglow)"}]}
3	{"albums": [{"album_id": 293, "artist_id": 227, "album_title": "Pavarotti's Opera Made Easy", "album_tracks": [{"album_id": 293, "track_id": 3391, "track_name": "Nessun Dorma"}]}
4	{"albums": [{"album_id": 271, "artist_id": 8, "album_title": "Revelations", "album_tracks": [{"album_id": 271, "track_id": 3389, "track_name": "The Armada Song"}]}
5	{"albums": [{"album_id": 314, "artist_id": 247, "album_title": "English Renaissance", "album_tracks": [{"album_id": 314, "track_id": 3392, "track_name": "The English Renaissance"}]}
6	{"albums": [{"album_id": 211, "artist_id": 138, "album_title": "The Singles", "album_tracks": [{"album_id": 211, "track_id": 2591, "track_name": "The Singles"}]}
7	{"albums": [{"album_id": 307, "artist_id": 242, "album_title": "Adams, John: The Chairman Dances", "album_tracks": [{"album_id": 307, "track_id": 3393, "track_name": "The Chairman Dances"}]}

168 row(s) fetched - 29ms

Grid

✓ ✗ ↺ + ↻ ⏮ ⏪ ⏩ ⏭ 📄 📁 🔄 🗑️ ⚙️ ▼



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

jsonb_pretty

1	{ "albums": [{ "album_id": 4, "artist_id": 1, "album_title": "Let There Be Rock", "album_tracks": [{ "album_id": 4, "track_id": 15, "track_name": "Go Down" }, { "album_id": 4, "track_id": 16, "track_name": "Dog Eat Dog" }, { "album_id": 4,
2	{ "albums": [{ "album_id": 48, "artist_id": 68, "album_title": "The Essential Miles Davis [Disc 1]"

2 row(s) fetched - 25ms



Live with Chinook data



```
-- SELECT some data from that VIEW using JSON methods
SELECT artist_data->>'artist' AS artist
      , artist_data#>'{albums, 1, album_title}' AS album_title
      , jsonb_pretty(artist_data#>'{albums, 1, album_tracks}') AS album_tracks
FROM v_json_artist_data
WHERE artist_data->'albums' @> '["album_title":"Miles Ahead"]'
;
```

	T artist	? album_title	T album_tracks
1	Miles Davis	"Miles Ahead"	[{"album_id": 157, "track_id": 1902

1 row(s) fetched - 27ms

Grid



Live with Chinook data



-- Array to records

```
SELECT artist_data->>'artist_id' AS artist_id
      , artist_data->>'artist' AS artist
      , jsonb_array_elements(artist_data#>'{albums}')->>'album_title' AS album_title
      , jsonb_array_elements(jsonb_array_elements(artist_data#>'{albums}')#>'{album_tracks}')->>'track_name' AS song_titles
FROM v_json_artist_data
WHERE artist_data->>'artist' = 'Metallica'
ORDER BY 3
;
```

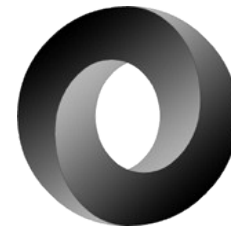
	T artist_id 📄	T artist 📄	T album_title 📄	T song_titles 📄
1	50	Metallica	...And Justice For All	Last Caress/Green Hell
2	50	Metallica	...And Justice For All	Hit The Lights
3	50	Metallica	...And Justice For All	The Prince
4	50	Metallica	...And Justice For All	Bleeding Me
5	50	Metallica	...And Justice For All	Low Man's Lyric
6	50	Metallica	...And Justice For All	The Frayed Ends Of Sanity
7	50	Metallica	...And Justice For All	Sad But True
8	50	Metallica	...And Justice For All	Fight Fire With Fire
9	50	Metallica	...And Justice For All	The Wait
10	50	Metallica	...And Justice For All	Ronnie
11	50	Metallica	...And Justice For All	Trapped Under Ice
12	50	Metallica	...And Justice For All	Purify
13	50	Metallica	...And Justice For All	(Anesthesia) Pulling Teeth
14	50	Metallica	...And Justice For All	Devil's Dance
15	50	Metallica	...And Justice For All	Turn The Page
16	50	Metallica	...And Justice For All	St. Anger
17	50	Metallica	...And Justice For All	Stone Dead Forever
18	50	Metallica	...And Justice For All	Astronomy
19	50	Metallica	...And Justice For All	The Unforgiven
20	50	Metallica	...And Justice For All	Poor Twisted Me

200 row(s) fetched - 26ms

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_json_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')::varchar(120) <> (NEW.artist_data->>'artist')::varchar(120) THEN
        UPDATE "Artist"
        SET "Name" = (NEW.artist_data->>'artist')::varchar(120)
        WHERE "ArtistId" = (OLD.artist_data->>'artist_id')::int;
    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



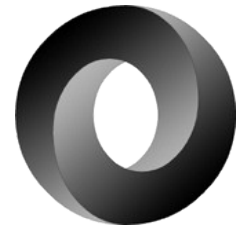
Name	Value
	<pre>-- Create a function, which will be used for UPDATE on the view v_artrist_data CREATE OR REPLACE FUNCTION trigger_v_json_artist_data_update() RETURNS trigger AS \$BODY\$ -- Data variables DECLARE rec RECORD; -- Error variables DECLARE v_state TEXT; DECLARE v_msg TEXT; DECLARE v_detail TEXT;</pre>

1 row(s) fetched - 8ms

Grid



Live with Chinook data



```
-- The trigger will be fired instead of an UPDATE statement to save data
CREATE TRIGGER v_json_artist_data_instead_update INSTEAD OF UPDATE
ON v_json_artist_data
FOR EACH ROW
EXECUTE PROCEDURE trigger_v_json_artist_data_update();
```

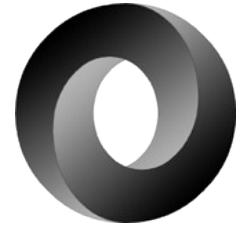
Name	Value
Query	-- The trigger will be fired instead of an UPDATE statement to save data CREATE TRIGGER v_json_artist_data_instead_update INSTEAD OF UPDATE ON v_json_artist_data FOR EACH ROW EXECUTE PROCEDURE trigger_v_json_artist_data_update()
Updated Rows	0

1 row(s) fetched - 13ms

Grid



Live with Chinook data



```
-- Manipulate data with jsonb_set
SELECT artist_data->>'artist_id' AS artist_id
      , artist_data->>'artist' AS artist
      , jsonb_set(artist_data, '{artist}', '"Whatever we want, it is just text"'::jsonb)->>'artist' AS new_artist
FROM v_json_artist_data
WHERE (artist_data->>'artist_id')::int = 50
;
```

	T artist_id	T artist	T new_artist
1	50	Metallica	Whatever we want, it is just text

1 row(s) fetched - 5ms

Grid



Live with Chinook data



```
-- Update a JSONB column with a jsonb_set result
UPDATE v_json_artist_data
SET artist_data= jsonb_set(artist_data, '{artist}', '"NEW Metallica"'::jsonb)
WHERE (artist_data->>'artist_id')::int = 50
;
```

Name	Value
Query	-- Update a JSONB column with a jsonb_set result UPDATE json_artist_data SET artist_data= jsonb_set(artist_data, '{artist}', '"NEW Metallica"'::jsonb) WHERE (artist_data->>'artist_id')::int = 50
Updated Rows	1
1 row(s) fetched - 20ms	
Grid	



Live with Chinook data



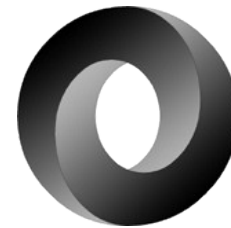
```
-- View the changes done by the UPDATE statement
SELECT artist_data->>'artist_id' AS artist_id
      , artist_data->>'artist' AS artist
FROM v_json_artist_data
WHERE (artist_data->>'artist_id')::int = 50
;
```

	T artist_id	T artist
1	50	NEW Metallica

1 row(s) fetched - 1ms Grid



Live with Chinook data



-- Manipulate data with the concatenating / overwrite operator

```
SELECT artist_data->>'artist_id' AS artist_id
, artist_data->>'artist' AS artist
, jsonb_set(artist_data, '{artist}', '"Whatever we want, it is just text"'::jsonb)->>'artist' AS new_artist
, artist_data || '{"artist":"Metallica"}'::jsonb->>'artist' AS correct_name
FROM v_json_artist_data
WHERE (artist_data->>'artist_id')::int = 50
;
```

	T artist_id	T artist	T new_artist	T correct_name
1	50	NEW Metallica	Whatever we want, it is just text	Metallica

1 row(s) fetched - 6ms

Grid



Live with Chinook data

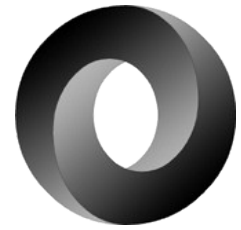


```
-- Revert the name change of Metallica with in a different way: With the replace operator
UPDATE v_json_artist_data
SET artist_data = artist_data || '{"artist":"Metallica"}'::jsonb
WHERE (artist_data->>'artist_id')::int = 50
;
```

Name	Value
Query	-- Revert the name change of Metallica with in a different way: With the replace operator UPDATE json_artist_data SET artist_data = artist_data '{"artist":"Metallica"}'::jsonb WHERE (artist_data->>'artist_id')::int = 50
Updated Rows	1



Live with Chinook data



-- View the changes done by the UPDATE statement with the replace operator

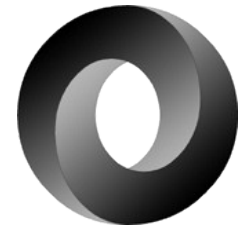
```
SELECT artist_data->>'artist_id' AS artist_id  
      , artist_data->>'artist' AS artist  
FROM v_json_artist_data  
WHERE (artist_data->>'artist_id')::int = 50  
;
```

	T artist_id	T artist
1	50	Metallica
1 row(s) fetched - 5ms		

Grid



Live Amazon reviews

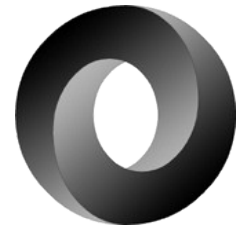


```
-- Create a table for JSON data with 1998 Amazon reviews  
CREATE TABLE reviews(review_jsonb jsonb);
```

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

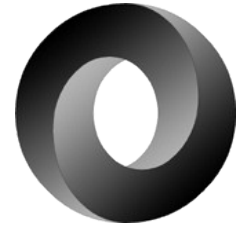


-- There should be 589.859 records imported into the table
SELECT count(*) FROM reviews;

	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



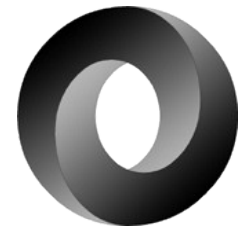
-- Create a GIN index

```
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	



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

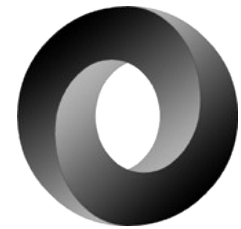
	title	average_rating
1	Complete Works for Solo Keyboard	5
2	The Magic Flute (Die Zauberflote in Full Score)	5
3	Requiem in Full Score	5
4	The Four Seasons and Other Violin Concertos in Full Score	5
5	Symphony No. 3 (Dover Miniature Scores)	5

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

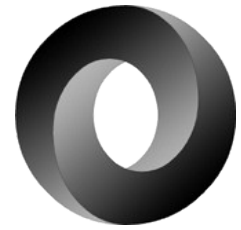
	category	average_rating	count_rating
1		4,487	1.521
2	Accessories	4,703	37
3	Action & Adventure	4,261	3.938
4	African American Cinema	4,694	36
5	Alternative Rock	4,522	15.508

84 row(s) fetched - 9747ms

Grid



Live Amazon reviews



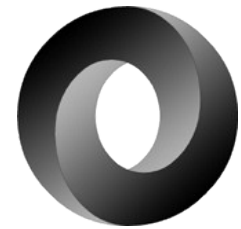
-- Create a B-Tree index on a JSON expression

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	



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

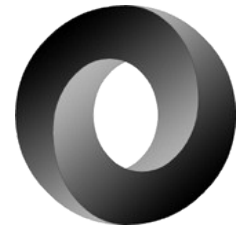
	category	average_rating	count_rating
1		4,487	1.521
2	Accessories	4,703	37
3	Action & Adventure	4,261	3.938
4	African American Cinema	4,694	36
5	Alternative Rock	4,522	15.508

84 row(s) fetched - 1605ms

Grid



JSON by example



This document by [Stefanie Janine Stölting](#) is covered by the [Creative Commons Attribution 4.0 International](#)