JSON by example

FOSDEM PostgreSQL Devevoper Room
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
Compressed JSON
Fully transactional
Up to 1 GB (uses TOAST)
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

jsonb_to_recordset
Returns a recordset from JSONB
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

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

Even unique **B-Tree** indexes are possible

```sql
CREATE UNIQUE INDEX actor_id_2 ON jsonb.actor((CAST(jsondata->>'actor_id' AS INTEGER)));
```
New JSON functions

PostgreSQL 9.5 new JSONB functions:
  jsonb_pretty: Formats JSONB human readable
  jsonb_set: Update or add values
PostgreSQL 9.5 new JSONB operators:
  ||: Concatenate two JSONB
  -: Delete key

Available as extions for 9.4 at PGXN: jsonbx
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

<table>
<thead>
<tr>
<th>Table</th>
<th>Column</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artist</td>
<td>ArtistId</td>
<td>integer</td>
</tr>
<tr>
<td>Artist</td>
<td>Name</td>
<td>character varying (120)</td>
</tr>
<tr>
<td>Album</td>
<td>AlbumId</td>
<td>integer</td>
</tr>
<tr>
<td>Album</td>
<td>Title</td>
<td>character varying (160)</td>
</tr>
<tr>
<td>Album</td>
<td>ArtistId</td>
<td>integer</td>
</tr>
<tr>
<td>Track</td>
<td>TrackId</td>
<td>integer</td>
</tr>
<tr>
<td>Track</td>
<td>Name</td>
<td>character varying (200)</td>
</tr>
<tr>
<td>Track</td>
<td>AlbumId</td>
<td>integer</td>
</tr>
<tr>
<td>Track</td>
<td>MediaTypeId</td>
<td>integer</td>
</tr>
<tr>
<td>Track</td>
<td>GenreId</td>
<td>integer</td>
</tr>
<tr>
<td>Track</td>
<td>Composer</td>
<td>character varying (220)</td>
</tr>
<tr>
<td>Track</td>
<td>Milliseconds</td>
<td>integer</td>
</tr>
<tr>
<td>Track</td>
<td>Bytes</td>
<td>integer</td>
</tr>
<tr>
<td>Track</td>
<td>UnitPrice</td>
<td>numeric</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th></th>
<th>sum bigint</th>
<th>min integer</th>
<th>max integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5050</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>album_id</th>
<th>track_id</th>
<th>track_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>&quot;For Those About To Rock (We Salute You)&quot;</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>&quot;Balls to the Wall&quot;</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>&quot;Fast As a Shark&quot;</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>&quot;Restless and Wild&quot;</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>&quot;Princess of the Dawn&quot;</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>&quot;Put The Finger On You&quot;</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>&quot;Let's Get It Up&quot;</td>
</tr>
</tbody>
</table>

200 row(s) fetched - 7ms
-- 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

```sql
<table>
<thead>
<tr>
<th>artist_id</th>
<th>album_title</th>
</tr>
</thead>
<tbody>
<tr>
<td>251</td>
<td>&quot;Armada: Music from the Courts of England and Spain&quot;</td>
</tr>
<tr>
<td>120</td>
<td>&quot;Dark Side Of The Moon&quot;</td>
</tr>
<tr>
<td>227</td>
<td>&quot;Pavarotti's Opera Made Easy&quot;</td>
</tr>
<tr>
<td>8</td>
<td>&quot;Revelations&quot;</td>
</tr>
<tr>
<td>247</td>
<td>&quot;English Renaissance&quot;</td>
</tr>
<tr>
<td>138</td>
<td>&quot;The Singles&quot;</td>
</tr>
<tr>
<td>242</td>
<td>&quot;Adams, John: The Chairman Dances&quot;</td>
</tr>
</tbody>
</table>
```

168 row(s) fetched - 38ms
-- 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,
           "MediaTypeId" AS media_type_id,
           "Milliseconds" AS milliseconds,
           "UnitPrice" AS unit_price
    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
-- Step 3 Return one row for an artist with all albums as VIEW

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
;
```
Live with Chinook data

-- 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'
);
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"}]';
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)
    , jsonb_array_elements(jsonb_array_elements(artist_data->>'{albums}'->>'{album_tracks}'->>'track_id' AS song_id)
FROM v_json_artist_data
WHERE artist_data->>'artist' = 'Metallica'
ORDER BY album_title,
    song_id
;
-- Convert albums to a recordset
SELECT *
FROM jsonb_to_recordset(
    SELECT (artist_data->>'albums')::jsonb
    FROM v_json_artist_data
    WHERE (artist_data->>'artist_id')::int = 50
) AS x(album_id int, artist_id int, album_title text, album_tracks jsonb)
;
-- Convert the tracks to a recordset

```sql
SELECT album_id,
       track_id,
       track_name,
       media_type_id,
       milliseconds,
       unit_price
FROM jsonb_to_recordset(
    (
        SELECT artist_data#>'{albums, 1, album_tracks}'
        FROM v_json_artist_data
        WHERE (artist_data->>'artist_id')::int = 50
    )
) AS x(album_id int, track_id int, track_name text, media_type_id int, milliseconds int, unit_price float)
;
```

<table>
<thead>
<tr>
<th>album_id</th>
<th>track_id</th>
<th>track_name</th>
<th>media_type_id</th>
<th>milliseconds</th>
<th>unit_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35</td>
<td>Free Speech For The Dumb</td>
<td>1</td>
<td>155.428</td>
<td>0.99</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>It's Electric</td>
<td>1</td>
<td>213.995</td>
<td>0.99</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>Sabbra Cadabra</td>
<td>1</td>
<td>380.342</td>
<td>0.99</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>Turn The Page</td>
<td>1</td>
<td>366.524</td>
<td>0.99</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>Die Die My Darling</td>
<td>1</td>
<td>149.315</td>
<td>0.99</td>
</tr>
<tr>
<td>6</td>
<td>35</td>
<td>Loverman</td>
<td>1</td>
<td>472.764</td>
<td>0.99</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>Mercyful Fate</td>
<td>1</td>
<td>671.712</td>
<td>0.99</td>
</tr>
<tr>
<td>8</td>
<td>35</td>
<td>Astronomy</td>
<td>1</td>
<td>397.531</td>
<td>0.99</td>
</tr>
<tr>
<td>9</td>
<td>35</td>
<td>Whiskey In The Jar</td>
<td>1</td>
<td>305.005</td>
<td>0.99</td>
</tr>
<tr>
<td>10</td>
<td>35</td>
<td>Tuesday's Gone</td>
<td>1</td>
<td>545.750</td>
<td>0.99</td>
</tr>
<tr>
<td>11</td>
<td>35</td>
<td>The More I See</td>
<td>1</td>
<td>287.973</td>
<td>0.99</td>
</tr>
</tbody>
</table>

11 row(s) fetched - 44ms
-- 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 with an UPSERT
-- Update table Track with an UPSERT
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

```
-- Create a function, which will be used for UPDATE on the view v_artist_data
CREATE OR REPLACE FUNCTION trigger_v_json_artist_data_update()
RETURNS trigger AS
$$BODY$
    -- Data variables
    DECLARE rec RECORD;
    -- Error variables
    DECLARE v_state TEXT;
    DECLARE v_msg TEXT;
    DECLARE v_detail TEXT;
$$BODY$
```

1 row(s) fetched - 8ms
-- 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();
-- 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
;
-- 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;

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>-- Update a JSONB column with a jsonb_set result</td>
</tr>
<tr>
<td></td>
<td>UPDATE json_artist_data</td>
</tr>
<tr>
<td></td>
<td>SET artist_data = jsonb_set(artist_data, '{artist}', '&quot;NEW Metallica':'::jsonb)</td>
</tr>
<tr>
<td></td>
<td>WHERE (artist_data-&gt;&gt;'artist_id')::int = 50</td>
</tr>
<tr>
<td>Updated Rows</td>
<td>1</td>
</tr>
</tbody>
</table>

1 row(s) fetched - 20ms
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;
Live with Chinook data

-- Lets have a view on the explain plans
– SELECT the data from the view

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Entity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subquery Scan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTE Scan</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seq Scan</td>
<td>Track</td>
<td>0.00 - 68.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.00 - 64.87</td>
</tr>
<tr>
<td></td>
<td>Aggregate</td>
<td>146.83 - 150.65</td>
</tr>
<tr>
<td></td>
<td>Hash Join</td>
<td>9.89 - 118.00</td>
</tr>
<tr>
<td></td>
<td>CTE Scan</td>
<td>0.00 - 57.66</td>
</tr>
<tr>
<td></td>
<td>Hash</td>
<td>6.06 - 6.06</td>
</tr>
<tr>
<td></td>
<td>Seq Scan</td>
<td>0.00 - 6.06</td>
</tr>
<tr>
<td></td>
<td>Aggregate</td>
<td>8.42 - 10.92</td>
</tr>
<tr>
<td></td>
<td>CTE Scan</td>
<td>0.00 - 6.12</td>
</tr>
<tr>
<td></td>
<td>Hash Join</td>
<td>7.49 - 14.24</td>
</tr>
<tr>
<td></td>
<td>CTE Scan</td>
<td>0.00 - 4.00</td>
</tr>
<tr>
<td></td>
<td>Hash</td>
<td>4.44 - 4.44</td>
</tr>
<tr>
<td></td>
<td>Seq Scan</td>
<td>0.00 - 4.44</td>
</tr>
<tr>
<td></td>
<td>Artist as a_1</td>
<td></td>
</tr>
</tbody>
</table>
Live with Chinook data

-- View the changes in the table instead of the JSONB view
-- The result should be the same, only the column name differ

```sql
SELECT * 
FROM "Artist"
WHERE "ArtistId" = 50
;
```

<table>
<thead>
<tr>
<th>ArtistId</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>NEW Metallica</td>
</tr>
</tbody>
</table>

1 row(s) fetched - 3ms
-- Let's have a view on the explain plans

- SELECT the data from table Artist

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Entity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq Scan</td>
<td>Artist</td>
<td>0.00 - 5.05</td>
</tr>
</tbody>
</table>
-- 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
;
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
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query</td>
<td>-- Revert the name change of Metallica with in a different way: With the replace operator</td>
</tr>
<tr>
<td></td>
<td>UPDATE json_artist_data</td>
</tr>
<tr>
<td></td>
<td>SET artist_data = artist_data</td>
</tr>
<tr>
<td></td>
<td>WHERE (artist_data-&gt;&gt;'artist_id')::int = 50</td>
</tr>
<tr>
<td>Updated Rows</td>
<td>1</td>
</tr>
</tbody>
</table>
Live with Chinook data

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

```sql
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
;
```

<table>
<thead>
<tr>
<th>artist_id</th>
<th>artist</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>Metallica</td>
</tr>
</tbody>
</table>

1 row(s) fetched - 5ms
-- Remove some data with the `-` operator

```sql
SELECT jsonb_pretty(artist_data) AS complete,
       jsonb_pretty(artist_data - 'albums') AS minus_albums,
       jsonb_pretty(artist_data) = jsonb_pretty(artist_data - 'albums') AS is_different
FROM v_json_artist_data
WHERE artist_data->>'artist' IN ('Miles Davis', 'AC/DC');
```

<table>
<thead>
<tr>
<th>complete</th>
<th>minus_albums</th>
<th>is_different</th>
</tr>
</thead>
<tbody>
<tr>
<td>[null]</td>
<td>[null]</td>
<td>false</td>
</tr>
<tr>
<td>[null]</td>
<td>[null]</td>
<td>false</td>
</tr>
</tbody>
</table>
CREATE TABLE reviews(review_jsonb jsonb);
Live Amazon reviews

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
</table>
| Query         | -- Import customer reviews from a file
                COPY reviews FROM '/var/tmp/customer_reviews_nested_1998.json' |
| Updated Rows  | 0                                                                     |
-- There should be 589.859 records imported into the table

```sql
SELECT count(*)
FROM reviews
;
```

<table>
<thead>
<tr>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>589.859</td>
</tr>
</tbody>
</table>

1 row(s) fetched - 104ms
SELECT jsonb_pretty(review_jsonb) FROM reviews LIMIT 1
Live Amazon reviews

-- Select data with JSON

```sql
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 title
ORDER BY average_rating DESC;
```

Without an Index: 248ms
CREATE INDEX review_review_jsonb ON reviews USING GIN (review_jsonb);
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 title
ORDER BY average_rating DESC
;

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

Without an Index: 9747ms
Live Amazon reviews

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

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