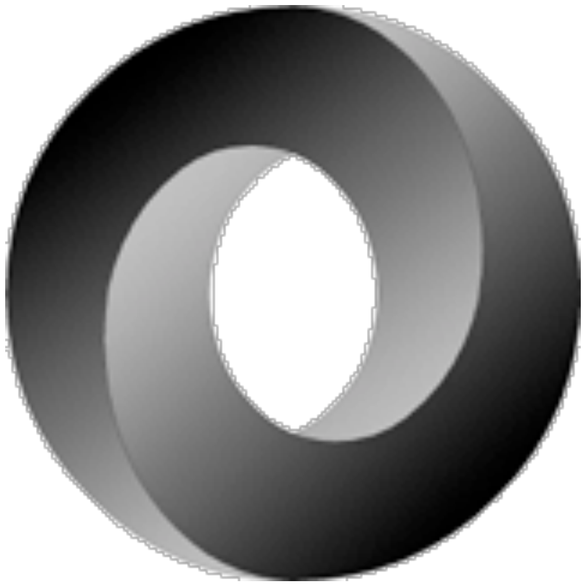
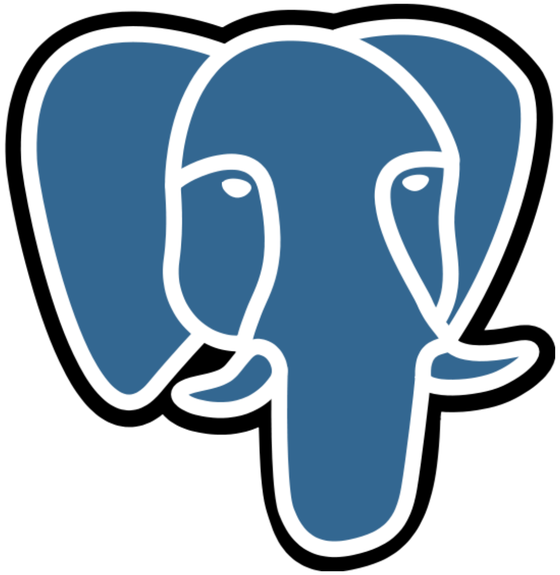
Future JSON support

looking ahead...

Daniel Gustafsson

Citus Team Meeting ⁕ 31/7 2023

https://flickr.com/photos/romankphoto/16256346839/

 +  = 

JSON in PostgreSQL

JSON type in 9.2

JSONB type in 9.4

New functions and optimizations in every release like FTS support in 10.0 and SQL/JSON PATH language in 12.0

Widely appreciated. Commonly reported as faster than MongoDB on JSON for data load and queries using an index

SQL/JSON support coming in 16.0

BIKESHED

ADVISORY

WARNING

```
[
  {
    "item": "kiwi",
    "inv": 10
  },
  {
    "item": "pear",
    "inv": 20
  }
]
```

```
CREATE TABLE inventory
(
  products jsonb
);

INSERT INTO inventory VALUES
(
  '[
    {"item":"kiwi","inv":10},
    {"item":"pear","inv":20}
  ]'
);
```

```
SELECT
  jsonb_array_elements(products)->>'item'
    AS item,
  jsonb_array_elements(products)->>'inv'
    AS inventory
FROM
  inventory;
```

```
 item | inventory
------+-----------
 kiwi | 10
 pear | 20
(2 rows)
```

```
{ "produce": [
  { "type": "fruits",
    "items": [
       { "item": "kiwi", "inv": 10 },
       { "item": "pear", "inv": 20 }
    ]
  },
  { "type": "vegetables",
    "items": [
      { "item": "lettuce", "inv": 100 },
      { "item": "celery",  "inv": 20 }
    ]
  }
]}
```

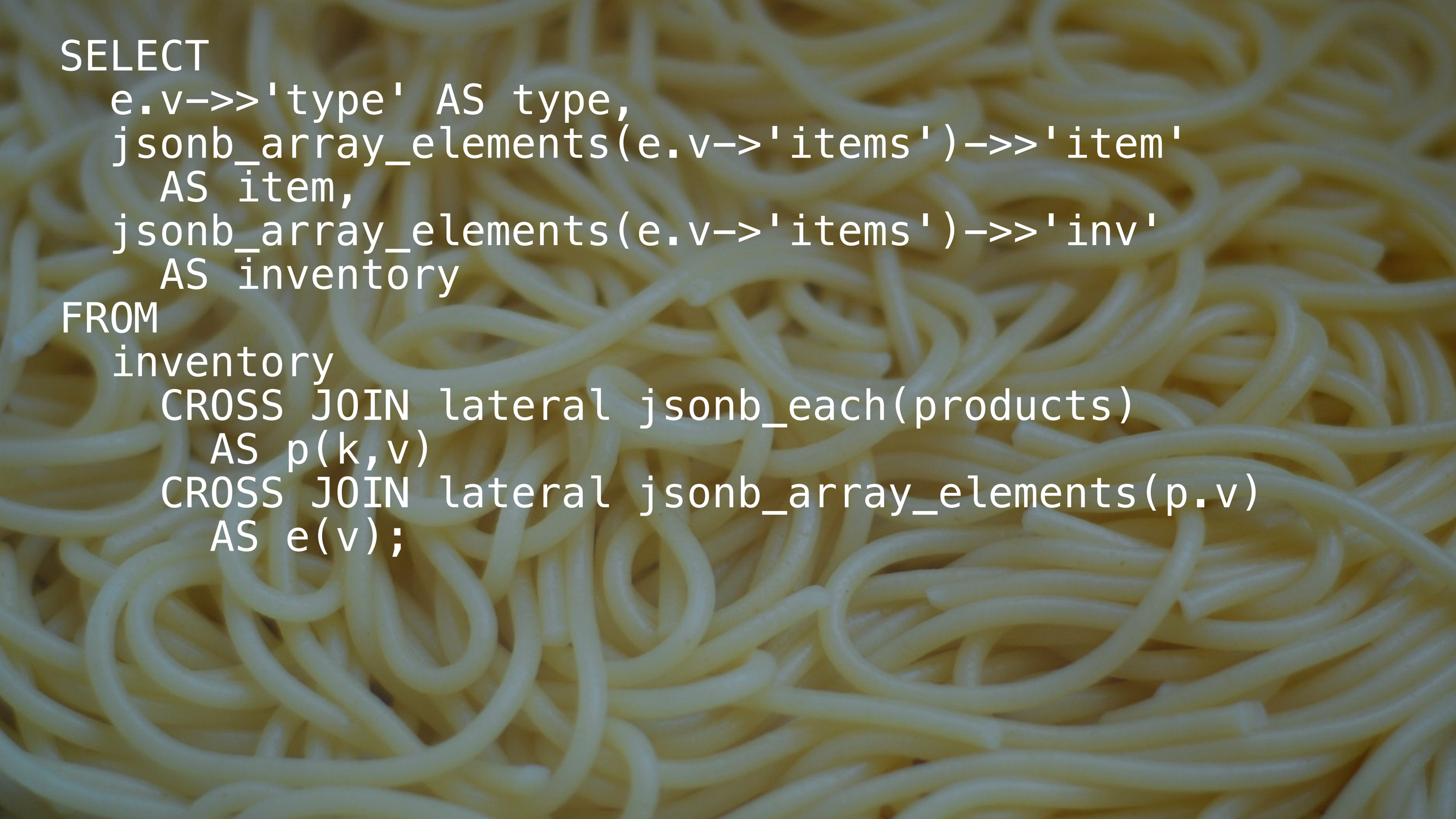

stackoverflow

```
SELECT
  e.v->>'type' AS type,
  jsonb_array_elements(e.v->'items')->>'item'
    AS item,
  jsonb_array_elements(e.v->'items')->>'inv'
    AS inventory
FROM
  inventory
    CROSS JOIN lateral jsonb_each(products)
      AS p(k,v)
    CROSS JOIN lateral jsonb_array_elements(p.v)
      AS e(v);
```

```
    type     |   item    | inventory
-------------+-----------+-----------
 fruits      | kiwi      | 10
 fruits      | pear      | 20
 vegetables  | lettuce   | 100
 vegetables  | celery    | 20
(4 rows)
```

```
                                QUERY PLAN
-------------------------------------------------------------------------
 Result
   ->  ProjectSet
         ->  Nested Loop
               ->  Nested Loop
                     ->  Seq Scan on inventory
                     ->  Function Scan on jsonb_each p
               ->  Memoize
                     Cache Key: p.v
                     Cache Mode: binary
                     ->  Function Scan on jsonb_array_elements e
(10 rows)
```

```
SELECT
  e.v->>'type' AS type,
  jsonb_array_elements(e.v->'items')->>'item'
    AS item,
  jsonb_array_elements(e.v->'items')->>'inv'
    AS inventory
FROM
  inventory
    CROSS JOIN lateral jsonb_each(products)
      AS p(k,v)
    CROSS JOIN lateral jsonb_array_elements(p.v)
      AS e(v);
```



```
SELECT
  e.v->>'type' AS type,
  jsonb_array_elements(e.v->'items')->>'item'
    AS item,
  jsonb_array_elements(e.v->'items')->>'inv'
    AS inventory
FROM
  inventory
    CROSS JOIN lateral jsonb_each(products)
      AS p(k,v)
    CROSS JOIN lateral jsonb_array_elements(p.v)
      AS e(v);
```

7.11 <JSON table>

Function

Query a JSON text and present it as a relational table.

Format

```
<JSON table> ::=
  JSON_TABLE <left paren>
      <JSON API common syntax>
      <JSON table columns clause>
      [ <JSON table plan clause> ]
      [ <JSON table error behavior> ON ERROR ]
  <right paren>

<JSON table columns clause> ::=
  COLUMNS <left paren>
      <JSON table column definition> [ { <comma> <JSON table column definition> }... ]
  <right paren>

<JSON table column definition> ::=
    <JSON table ordinality column definition>
```

JSON_TABLE

Standardized SQL

Supported by IBM DB2, Oracle and MySQL

Query JSONB structures and present as relational data

Resulting relation can be treated as ordinary relation content

```
SELECT
  produce,
  item,
  inv
FROM
  inventory,
  json_table(products, '$.produce[*]' COLUMNS (
      produce TEXT path '$.type',
      NESTED PATH '$.items[*]' COLUMNS (
          item TEXT path '$.item',
          inv integer path '$.inv'
      )
  ));
```

```
SELECT
  produce,
  item,
  inv
FROM
  inventory,
  json_table(products, '$.produce[*]' COLUMNS (
      produce TEXT path '$.type',
      NESTED PATH '$.items[*]' COLUMNS (
          item TEXT path '$.item',
          inv integer path '$.inv'
      )
  ));
```

```
{ "produce": [
  { "type": "fruits",
    "items": [
       { "item": "kiwi", "inv": 10 },
       { "item": "pear", "inv": 20 }
    ]
  },
  { "type": "vegetables",
    "items": [
      { "item": "lettuce", "inv": 100 },
      { "item": "celery",  "inv": 20 }
    ]
  }
]}
```

```
SELECT
   produce,
   item,
   inv
FROM
   inventory,
   json_table(products, '$.produce[*]' COLUMNS (
        produce TEXT path '$.type',
        NESTED PATH '$.items[*]' COLUMNS (
             item TEXT path '$.item',
             inv integer path '$.inv'
        )
   ));
```

```
{ "produce": [
  { "type": "fruits",
    "items": [
       { "item": "kiwi", "inv": 10 },
       { "item": "pear", "inv": 20 }
    ]
  },
  { "type": "vegetables",
    "items": [
      { "item": "lettuce", "inv": 100 },
      { "item": "celery",  "inv": 20 }
    ]
  }
]}
```

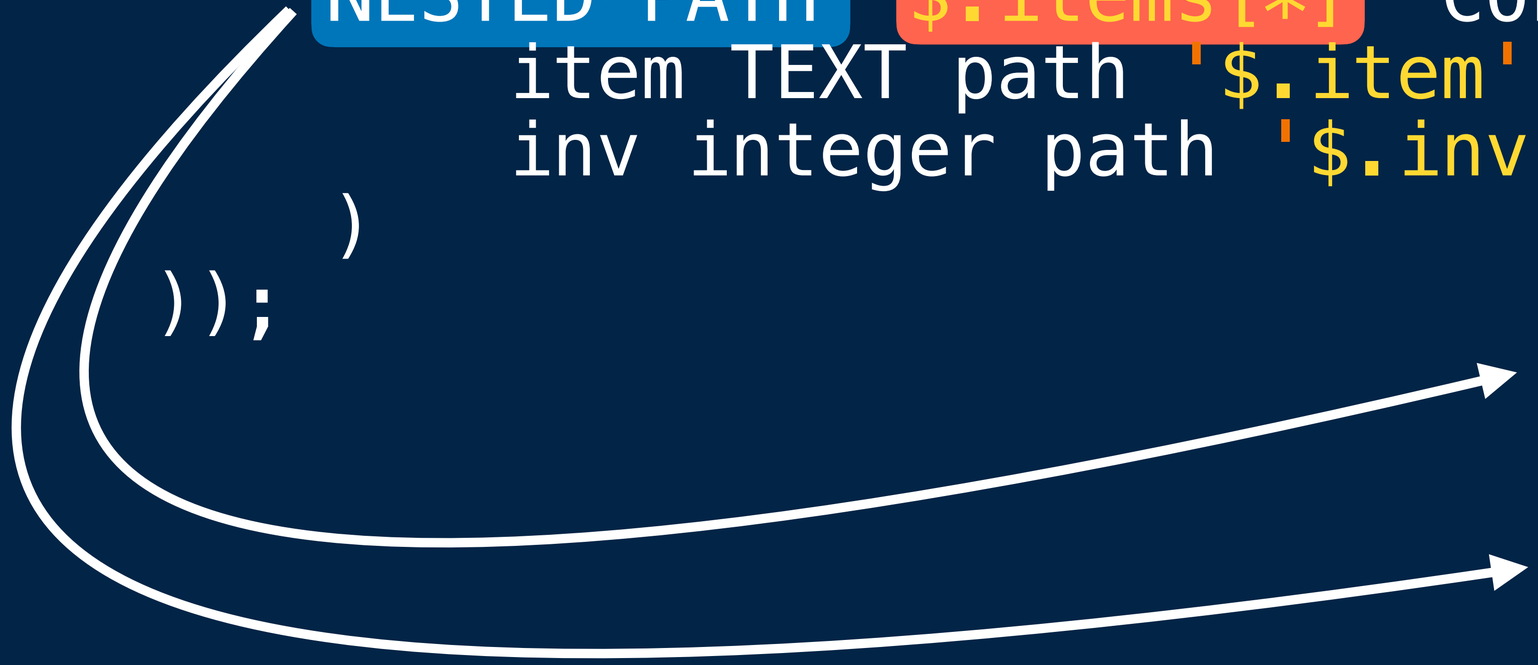
```
SELECT
   produce,
   item,
   inv
FROM
   inventory,
   json_table(products, '$.produce[*]' COLUMNS (
        produce TEXT path '$.type',
        NESTED PATH '$.items[*]' COLUMNS (
            item TEXT path '$.item',
            inv integer path '$.inv'
        )
   ));
```

```
{ "produce": [
  { "type": "fruits",
    "items": [
      { "item": "kiwi", "inv": 10 },
      { "item": "pear", "inv": 20 }
    ]
  },
  { "type": "vegetables",
    "items": [
      { "item": "lettuce", "inv": 100 },
      { "item": "celery",  "inv": 20 }
    ]
  }
]}
```

```
SELECT
   produce,
   item,
   inv
FROM
   inventory,
   json_table(products, '$.produce[*]' COLUMNS (
        produce TEXT path '$.type',
        NESTED PATH '$.items[*]' COLUMNS (
             item TEXT path '$.item',
             inv integer path '$.inv'
        )
   ));
```

```
{ "produce": [
  { "type": "fruits",
    "items": [
       { "item": "kiwi", "inv": 10 },
       { "item": "pear", "inv": 20 }
    ]
  },
  { "type": "vegetables",
    "items": [
      { "item": "lettuce", "inv": 100 },
      { "item": "celery",  "inv": 20 }
    ]
  }
]}
```

```
SELECT
   produce,
   item,
   inv
FROM
   inventory,
   json_table(products, '$.produce[*]' COLUMNS (
        produce TEXT path '$.type',
        NESTED PATH '$.items[*]' COLUMNS (
             item TEXT path '$.item',
             inv integer path '$.inv'
        )
   ));
```



```
{ "produce": [
  { "type": "fruits",
    "items": [
       { "item": "kiwi", "inv": 10 },
       { "item": "pear", "inv": 20 }
    ]
  },
  { "type": "vegetables",
    "items": [
      { "item": "lettuce", "inv": 100 },
      { "item": "celery",  "inv": 20 }
    ]
  }
]}
```

```
SELECT
   produce,
   item,
   inv
FROM
   inventory,
   json_table(products, '$.produce[*]' COLUMNS (
        produce TEXT path '$.type',
        NESTED PATH '$.items[*]' COLUMNS (
            item TEXT path '$.item',
            inv integer path '$.inv'
        )
   ));
```

```
{ "produce": [
  { "type": "fruits",
    "items": [
       { "item": "kiwi", "inv": 10 },
       { "item": "pear", "inv": 20 }
    ]
  },
  { "type": "vegetables",
    "items": [
      { "item": "lettuce", "inv": 100 },
      { "item": "celery",  "inv": 20 }
    ]
  }
]}
```

```
SELECT
   produce,
   item,
   inv
FROM
   inventory,
   json_table(products, '$.produce[*]' COLUMNS (
        produce TEXT path '$.type',
        NESTED PATH '$.items[*]' COLUMNS (
            item TEXT path '$.item',
            inv integer path '$.inv'
        )
   ));
```

```
{ "produce": [
  { "type": "fruits",
    "items": [
       { "item": "kiwi", "inv": 10 },
       { "item": "pear", "inv": 20 }
    ]
  },
  { "type": "vegetables",
    "items": [
      { "item": "lettuce", "inv": 100 },
      { "item": "celery",  "inv": 20 }
    ]
  }
]}
```

```
    type     |   item    | inventory
-------------+-----------+-----------
 fruits      | kiwi      | 10
 fruits      | pear      | 20
 vegetables  | lettuce   | 100
 vegetables  | celery    | 20
(4 rows)
```

```
                  QUERY PLAN
-----------------------------------------------
 Nested Loop
   ->  Seq Scan on inventory
   ->  Table Function Scan on "json_table"
(3 rows)
```

```
SELECT
  produce,
  sum(inv) AS total_inventory
FROM
  inventory,
  json_table(products, '$.produce[*]' COLUMNS (
      produce TEXT path '$.type',
      NESTED PATH '$.items[*]' COLUMNS (
          item TEXT path '$.item',
          inv integer path '$.inv'
      )
  ))
GROUP BY produce;
```

```
  produce    | total_inventory
-------------+-----------------
 vegetables |             120
 fruits     |              30
(2 rows)
```

```
SELECT
  id,
  produce,
  item,
  inv
FROM
  inventory,
  json_table(products, '$.produce[*]' COLUMNS (
      id FOR ORDINALITY,
      produce TEXT path '$.type',
      NESTED PATH '$.items[*]' COLUMNS (
          item TEXT path '$.item',
          inv integer path '$.inv'
      )
  ));
```

```
 id |   produce    |   item    | inv
----+--------------+-----------+-----
  1 | fruits       | kiwi      |  10
  1 | fruits       | pear      |  20
  2 | vegetables | lettuce | 100
  2 | vegetables | celery  |  20
(4 rows)
```

```
SELECT
  *
FROM
  json_table(
    '[{"item":"kiwi","inv":10},
     {"item":"pear","inv":20}]'::JSONB,
    '$[*]' COLUMNS(
      item TEXT path '$.item',
      inventory INTEGER path '$.inv'
    )
  );
```

TL;DR - JSON_TABLE

Returns a relation like any other

Advanced SQL analytics

Not limited to a 1:1 mapping

Transformative declaration

```
name type EXISTS ...
name type FORMAT ...
```

JSON_QUERY(*context, path ...*)

Returns the result of applying path expression to the context expression using a set of optional values.

Always returns a JSON string

Can return multiple objects

Expressive error semantics for controlling missing or incorrect data, ERROR, EMPTY, NULL, DEFAULT

```
  SELECT
    json_query(products, '$.produce[*].items[1].item'
                  WITH WRAPPER)
  FROM
    inventory;


     json_query
--------------------
 ["pear", "celery"]
(1 row)
```

JSON_VALUE(*context, path ...*)

Returns the result of applying path expression to the context expression using a set of optional values.

Can only return a single SQL/JSON scalar item

Expressive error semantics for controlling missing or incorrect data, ERROR, EMPTY, NULL, DEFAULT

```
SELECT
  json_value(jsonb '"03:04 2015-02-01"',
             '$.datetime("HH24:MI YYYY-MM-DD")'
            RETURNING date);

 json_value
------------
 2015-02-01
(1 row)
```

JSON_EXISTS(*context, path ...*)

Returns the boolean True if applying the path expression to the context expression with optional values yields any results.

```
SELECT
  json_exists(products, '$.produce[*].items')
FROM
  inventory;

 json_exists
-------------
 t
(1 row)
```

SQL/JSON Constructors

json_array()

json_arrayagg()

json_object()

json_objectagg()

```
SELECT
  json_array(SELECT relname FROM pg_class
             ORDER BY oid ASC limit 3);

                  json_array
-----------------------------------------------
 ["pg_foreign_data_wrapper_oid_index",
  "pg_foreign_server_oid_index",
  "pg_user_mapping_oid_index"]
(1 row)
```

```
SELECT
  json_object('fruit' value 'pear',
              'vegetable' : 'celery');


                  json_object
-----------------------------------------
 {"fruit" : "pear", "vegetable" : "celery"}
(1 row)
```

```
SELECT
  json_object('fruit' value 'pear',
              'vegetable' : 'celery');


                  json_object
-----------------------------------------------
 {"fruit" : "pear", "vegetable" : "celery"}
(1 row)
```

SQL/JSON object checks

expression `IS [NOT] JSON ...`

Tests if expression can be parsed as JSON, possibly of a specified type

Support for values, arrays, objects, scalars and unique keys

```
SELECT
  ' _' IS JSON AS underscore,
  '{}' IS JSON AS brackets;


 underscore | brackets
------------+----------
 f          | t
(1 row)
```

SQL/JSON Summary

Constructors and vtests coming in v16

JSON_TABLE and expression based extraction functionality hopefully coming in v17

Some functionality has already been committed

http://1.bp.blogspot.com/-FqLpbWurXrI/VDqOrGCgUXI/AAAAAAAAGxk/eiyxD4AuSDI/s1600/thats-all-folks-7172-1280x800.jpg