XML, HSTORE, JSON, JSONB—OH MY!

David E. Wheeler
@theory
iovation

PGCon
2014-05-23

Unstructured Data Types

- Several supported
- XML, HSTORE, JSON, JSONB
- Which should use use?
- When should you use it?
- Why?
- A brief tour

“It has been said that XML is like violence; if a little doesn’t solve the problem, use more.”

— Chris Maden

XML

XML

- Added in 8.2

XML

- Added in 8.2
- Data Type

XML

- Added in 8.2
- Data Type
- Publishing

XML

- Added in 8.2
- Data Type
- Publishing
- Export

XML

- Added in 8.2
- Data Type
- Publishing
- Export
- SQL:2003 Conformance

XML

- Added in 8.2
- Data Type
- Publishing
- Export
- SQL:2003 Conformance
- XPath

XML Implementation

XML Implementation

- Input validation

XML Implementation

- Input validation
- Text storage

XML Implementation

- Input validation
- Text storage
- No comparison operators

XML Implementation

- Input validation
- Text storage
- No comparison operators
- So no indexing

XML Implementation

- Input validation
- Text storage
- No comparison operators
- So no indexing
- Can cast to text

XML Implementation

- Input validation
- Text storage
- No comparison operators
- So no indexing
- Can cast to text
- Better: Use XPath

XML Generation

XML Generation

- xmlelement()

XML Generation

- xmlelement()
- xmlattributes()

XML Generation

- xmlelement()
- xmlattributes()
- xmlcomment()
- xmlconcat()

XML Generation

- xmlelement()
- xmlcomment()
- xmlattributes()
- xmlconcat()

```
% SELECT xmlelement(name foo,
      xmlattributes('xyz' as bar),
      xmlelement(name abc),
      xmlcomment('ow'),
      xmlelement(name xyz,
          xmlelement(name yack, 'barber')
      )
  );
                                    xmlelement
--------------------------------------------------------------------
 <foo bar="xyz"><abc/><!--ow--><xyz><yack>barber</yack></xyz></foo>
```

XML Predicates

XML Predicates

- IS DOCUMENT

XML Predicates

- IS DOCUMENT
- xml_is_well_formed()

XML Predicates

- IS DOCUMENT
- xml_is_well_formed()
- XMLEXISTS()

XML Predicates

- IS DOCUMENT
- xml_is_well_formed()
- XMLEXISTS()

XPath!

```
% SELECT xmlexists(
      $$//town[text() = 'Ottawa']$$
      PASSING '<towns><town>Portland</town><town>Ottawa</town></towns>'
  );
 xmlexists
-----------
 t
```

XPath

XPath

```
% SELECT xpath(
      '/p/a/text()',
      '<p><a>test</a><a>me</a></p>'
  );
   xpath
-----------
 {test,me}

%
```

XPath

```
% SELECT xpath(
      '/p/a/text()',
      '<p><a>test</a><a>me</a></p>'
  );
   xpath
-----------
 {test,me}

% SELECT xpath_exists(
      '/p/a/text()',
      '<p><a>test</a><a>me</a></p>'
  );
 xpath_exists
--------------
 t
```

XPath

```
% SELECT xpath(
      '/p/a/text()',
      '<p><a>test</a><a>me</a></p>'
  );
   xpath
-----------
 {test,me}

% SELECT xpath_exists(
      '/p/a/text()',
      '<p><a>test</a><a>me</a></p>'
  );
 xpath_exists
--------------
 t
```

Supports namespacing.

XML Table Mapping

XML Table Mapping

```
% SELECT query_to_xml($$
       SELECT n.nspname, c.relname
         FROM pg_class c JOIN pg_namespace n
           ON c.relnamespace = n.oid
        WHERE n.nspname NOT IN ('pg_catalog', 'pg_toast',
'information_schema')
  $$, true, false, '');

                         query_to_xml
---------------------------------------------------------------
 <table xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">+
 <row>                                                        +
   <nspname>public</nspname>                                  +
   <relname>stuff</relname>                                   +
 </row>                                                       +
 <row>                                                        +
   <nspname>public</nspname>                                  +
   <relname>stuff_pkey</relname>                              +
 </row>                                                       +
 <row>                                                        +
   <nspname>public</nspname>                                  +
   <relname>idx_stuff_somekey</relname>                       +
 </row>                                                       +
 </table>                                                     +
```

Querying

Querying

```
% \set xpath '\'/us-patent-grant/us-bibliographic-data-grant/examiners/*/last-name/text()\''
```

Querying

```
% \set xpath '\'/us-patent-grant/us-bibliographic-data-grant/examiners/*/last-name/text()\''
```

```
% SELECT COUNT(*)
    FROM grants
   WHERE '{Brooks,Mroczka}' = xpath(:xpath, xmldoc)::text[];
 count
-------
     4
(1 row)

Time: 37311.163 ms
```

Querying

```
% \set xpath '\'/us-patent-grant/us-bibliographic-data-grant/examiners/*/last-name/text()\''

% SELECT COUNT(*)
    FROM grants
   WHERE '{Brooks,Mroczka}' = xpath(:xpath, xmldoc)::text[];
 count
-------
     4
(1 row)

Time: 37311.163 ms
```

A bit slow.

Index Query

Index Query

```
% CREATE INDEX idx_grant_examiners
      ON grants(CAST(xpath(:xpath, xmldoc) AS text[]));
CREATE INDEX

%
```

Index Query

```
% CREATE INDEX idx_grant_examiners
      ON grants(CAST(xpath(:xpath, xmldoc) AS text[]));
CREATE INDEX

% SELECT COUNT(*)
    FROM grants
   WHERE '{Brooks,Mroczka}' = xpath(:xpath, xmldoc)::text[];
 count
-------
     4
(1 row)

Time: 0.716 ms
```

Nice.

Query Index Values

Query Index Values

```
% SELECT COUNT(*)
    FROM grants
   WHERE 'Brooks' = ANY(xpath(:xpath, xmldoc)::text[]);
 count
-------
    34
(1 row)

Time: 39348.995 ms
```

Table scan

Query Index Values

```
% SELECT COUNT(*)
    FROM grants
   WHERE 'Brooks' = ANY(xpath(:xpath, xmldoc)::text[]);
 count
-------
    34
(1 row)

Time: 39348.995 ms
```

Yikes!

Querying

Querying

```
% \set xpath '\'/us-patent-grant/us-bibliographic-data-grant/examiners/primary-examiner/last-name/text()\''
```

Querying

```
% \set xpath '\'/us-patent-grant/us-bibliographic-data-grant/examiners/primary-examiner/last-name/text()\''

% SELECT COUNT(*)
    FROM grants
   WHERE CAST((xpath(:xpath, xmldoc))[1] AS text) = 'Brooks';
 count
-------
    12
(1 row)

Time: 37187.900 ms
```

Index Query

Index Query

```
% CREATE INDEX idx_grant_primary_examiner
      ON grants(CAST((xpath(:xpath, xmldoc))[1] AS text));
CREATE INDEX

%
```

Index Query

```
% CREATE INDEX idx_grant_primary_examiner
      ON grants(CAST((xpath(:xpath, xmldoc))[1] AS text));
CREATE INDEX

% SELECT count(*)
    FROM grants
   WHERE CAST((xpath(:xpath, xmldoc))[1] AS text) = 'Brooks';
 count
-------
    12
(1 row)

Time: 1.046 ms
```

Index Query

```
% CREATE INDEX idx_grant_primary_examiner
      ON grants(CAST((xpath(:xpath, xmldoc))[1] AS text));
CREATE INDEX

% SELECT count(*)
    FROM grants
   WHERE CAST((xpath(:xpath, xmldoc))[1] AS text) = 'Brooks';
 count
-------
    12
(1 row)

Time: 1.046 ms
```

Scalar

Index Query

```
% CREATE INDEX idx_grant_primary_examiner
      ON grants(CAST((xpath(:xpath, xmldoc))[1] AS text));
CREATE INDEX

% SELECT count(*)
    FROM grants
   WHERE CAST((xpath(:xpath, xmldoc))[1] AS text) = 'Brooks';
 count
-------
    12
(1 row)

Time: 1.046 ms
```

Mo betta.

When to use XML

When to use XML

- When XML required:

When to use XML

- When XML required:
  - Existing documents

When to use XML

- When XML required:
  - Existing documents
  - SOAP API

When to use XML

- When XML required:
  - Existing documents
  - SOAP API
  - XHTML

When to use XML

- When XML required:
  - Existing documents
  - SOAP API
  - XHTML
- Good document storage

When to use XML

- When XML required:
  - Existing documents
  - SOAP API
  - XHTML
- Good document storage
  - Stored as text

When to use XML

- When XML required:
  - Existing documents
  - SOAP API
  - XHTML
- Good document storage
  - Stored as text
- Fast I/O

When to use XML

- When XML required:
  - Existing documents
  - SOAP API
  - XHTML
- Good document storage
  - Stored as text
  - Fast I/O
- XPath is Awesome

When to use XML

- When XML required:
  - Existing documents
  - SOAP API
  - XHTML
- Good document storage
  - Stored as text
  - Fast I/O
- XPath is Awesome
  - Best on indexed scalars

When to use XML

- When XML required:
  - Existing documents
  - SOAP API
  - XHTML
- Good document storage
  - Stored as text
  - Fast I/O
- XPath is Awesome
  - Best on indexed scalars
  - When table scans okay

When to use XML

- When XML required:
  - Existing documents
  - SOAP API
  - XHTML
- Good document storage
  - Stored as text
  - Fast I/O
- XPath is Awesome
  - Best on indexed scalars
  - When table scans okay
  - Encumbers per-row parsing overhead

“HSTORE: Like Perl hashes, but flatter, less typed, and incompatible.”

—Theory

HSTORE

HSTORE

- Also added in 8.2

HSTORE

- Also added in 8.2
- Simple key/value pairs

HSTORE

- Also added in 8.2
- Simple key/value pairs
- Strings only

HSTORE

- Also added in 8.2
- Simple key/value pairs
- Strings only
- No nested values

HSTORE

- Also added in 8.2
- Simple key/value pairs
- Strings only
- No nested values
- Lots of useful operators

HSTORE

- Also added in 8.2
- Simple key/value pairs
- Strings only
- No nested values
- Lots of useful operators
- GiST and GIN indexing

HSTORE Improvements

HSTORE Improvements

- Improved in 9.0:

HSTORE Improvements

- Improved in 9.0:
  - Many new operators

HSTORE Improvements

- Improved in 9.0:
  - Many new operators
  - BTree and Hash indexing

HSTORE Improvements

- Improved in 9.0:
  - Many new operators
  - BTree and Hash indexing
  - Increased capacity

HSTORE Syntax

HSTORE Syntax

```
% CREATE EXTENSION hstore;
CREATE EXTENSION

%
```

HSTORE Syntax

```
% CREATE EXTENSION hstore;
CREATE EXTENSION

% SELECT 'a => 1, a => 2'::hstore;
  hstore
----------
 "a"=>"1"

%
```

HSTORE Syntax

```
% CREATE EXTENSION hstore;
CREATE EXTENSION

% SELECT 'a => 1, a => 2'::hstore;
  hstore
----------
 "a"=>"1"

% SELECT '"hey there" => NULL, "salad" => "super"'::hstore;
               hstore
-------------------------------------
 "salad"=>"super", "hey there"=>NULL

%
```

HSTORE Operators

HSTORE Operators

```
% SELECT 'user => "fred", id => 1'::hstore -> 'user' AS user;
 user
------
 fred

%
```

Value for key.

HSTORE Operators

```
% SELECT 'user => "fred", id => 1'::hstore -> 'user' AS user;
 user
------
 fred

% SELECT 'user => "fred", id => 1'::hstore
         -> ARRAY['user','id'] AS vals;
   vals
----------
 {fred,1}

%
```

Value for keys.

HSTORE Operators

```
% SELECT 'user => "fred", id => 1'::hstore -> 'user' AS user;
 user
------
 fred

% SELECT 'user => "fred", id => 1'::hstore
         -> ARRAY['user','id'] AS vals;
   vals
----------
 {fred,1}

% SELECT 'user => "fred", id => 1'::hstore @> 'id=>1' AS one;
 one
-----
 t
```

Does left contain right?

More HSTORE Operators

More HSTORE Operators

- ||	Concatenation

More HSTORE Operators

- ||	Concatenation
- ?	Key exists

More HSTORE Operators

- || Concatenation
- ?   Key exists
- ?&   Keys exist

More HSTORE Operators

- `||` Concatenation
- `?` Key exists
- `?&` Keys exist
- `?|` Do any keys exist

More HSTORE Operators

- || Concatenation
- ? Key exists
- ?& Keys exist
- ?| Do any keys exist
- – Delete key or keys

More HSTORE Operators

- || Concatenation
- ? Key exists
- ?& Keys exist
- ?| Do any keys exist
- – Delete key or keys
- %% Convert to array

HSTORE Functions

HSTORE Functions

```
% SELECT hstore('a', 'b');
  hstore
----------
 "a"=>"b"

%
```

HSTORE constructor.

HSTORE Functions

```
% SELECT hstore('a', 'b');
  hstore
----------
 "a"=>"b"

% SELECT hstore(ROW(1, 2));
        hstore
----------------------
 "f1"=>"1", "f2"=>"2"

%
```

Convert row to HSTORE.

HSTORE Functions

```
% SELECT hstore('a', 'b');
  hstore
----------
 "a"=>"b"

% SELECT hstore(ROW(1, 2));
        hstore
----------------------
 "f1"=>"1", "f2"=>"2"

% SELECT hstore(ARRAY['a','1','b','2']);
       hstore
--------------------
 "a"=>"1", "b"=>"2"

%
```

Convert array to HSTORE.

HSTORE Functions

HSTORE Functions

```
% SELECT akeys('a => 1, b => 2');
 akeys
-------
 {a,b}

%
```

HSTORE Functions

```
% SELECT akeys('a => 1, b => 2');
 akeys
-------
 {a,b}

% SELECT avals('a => 1, b => 2');
 avals
-------
 {1,2}

%
```

HSTORE Functions

```
% SELECT akeys('a => 1, b => 2');
 akeys
-------
 {a,b}

% SELECT avals('a => 1, b => 2');
 avals
-------
 {1,2}

% SELECT hstore_to_json('"a key" => 1, b => t, c => null');
            hstore_to_json
-------------------------------------
 {"b": "t", "c": null, "a key": "1"}

%
```

HSTORE Sets

HSTORE Sets

```
% SELECT skeys('"a key" => 1, b => t');
 skeys
--------
 b
 a key

%
```

HSTORE Sets

```
% SELECT skeys('"a key" => 1, b => t');
 skeys
-------
 b
 a key

% SELECT svals('"a key" => 1, b => t');
 svals
-------
 t
 1

%
```

HSTORE Sets

```
% SELECT skeys('"a key" => 1, b => t');
 skeys
-------
 b
 a key

% SELECT svals('"a key" => 1, b => t');
 svals
-------
 t
 1

% SELECT * FROM each('"a key" => 1, b => t, c => null');
  key  | value
-------+--------
 b     | t
 c     |
 a key | 1
```

HSTORE Performance

HSTORE Performance

- Grabbed 1998 Amazon review data

HSTORE Performance

- Grabbed 1998 Amazon review data
  - Thanks CitusDB!

HSTORE Performance

- Grabbed 1998 Amazon review data
    - Thanks CitusDB!
- Converted from nested JSON

HSTORE Performance

- Grabbed 1998 Amazon review data
  - Thanks CitusDB!
- Converted from nested JSON
- To flattened HSTORE

HSTORE Load

HSTORE Load

```
> createdb hreviews
> psql -d hreviews -c '
    CREATE EXTENSION HSTORE;
    CREATE TABLE reviews(review hstore);
'
CREATE TABLE
>
```

HSTORE Load

```
> createdb hreviews
> psql -d hreviews -c '
    CREATE EXTENSION HSTORE;
    CREATE TABLE reviews(review hstore);
'
CREATE TABLE
> time psql hreviews -c "COPY reviews FROM 'reviews.hstore'"
COPY 589859
     0.00s user 0.00s system 0% cpu 8.595 total
```

HSTORE Load

```
> createdb hreviews
> psql -d hreviews -c '
    CREATE EXTENSION HSTORE;
    CREATE TABLE reviews(review hstore);
'
CREATE TABLE
> time psql hreviews -c "COPY reviews FROM 'reviews.hstore'"
COPY 589859
     0.00s user 0.00s system 0% cpu 8.595 total
```

- 68,628 records/second

HSTORE Load

```
> createdb hreviews
> psql -d hreviews -c '
    CREATE EXTENSION HSTORE;
    CREATE TABLE reviews(review hstore);
'
CREATE TABLE
> time psql hreviews -c "COPY reviews FROM 'reviews.hstore'"
COPY 589859
     0.00s user 0.00s system 0% cpu 8.595 total
```

- 68,628 records/second
- 233 MB COPY file

HSTORE Load

```
> createdb hreviews
> psql -d hreviews -c '
    CREATE EXTENSION HSTORE;
    CREATE TABLE reviews(review hstore);
'
CREATE TABLE
> time psql hreviews -c "COPY reviews FROM 'reviews.hstore'"
COPY 589859
     0.00s user 0.00s system 0% cpu 8.595 total
```

- 68,628 records/second
- 256 MB Database
- 233 MB COPY file

HSTORE Load

```
> createdb hreviews
> psql -d hreviews -c '
    CREATE EXTENSION HSTORE;
    CREATE TABLE reviews(review hstore);
'
CREATE TABLE
> time psql hreviews -c "COPY reviews FROM 'reviews.hstore'"
COPY 589859
     0.00s user 0.00s system 0% cpu 8.595 total
```

- 68,628 records/second
- 256 MB Database
- 233 MB COPY file
- 9.9% storage overhead

Bucket 'O Books

Bucket 'O Books

```
% SELECT width_bucket(length(review->'product_title'), 1, 50, 5) title_bkt,
       round(avg((review->'review_rating')::numeric), 2) review_avg,
       COUNT(*)
  FROM reviews
 WHERE review->'product_group' = 'DVD'
 GROUP BY title_bkt
 ORDER BY title_bkt;
```

Bucket 'O Books

```
% SELECT width_bucket(length(review->'product_title'), 1, 50, 5) title_bkt,
         round(avg((review->'review_rating')::numeric), 2) review_avg,
         COUNT(*)
    FROM reviews
   WHERE review->'product_group' = 'DVD'
   GROUP BY title_bkt
   ORDER BY title_bkt;

 title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 207.665 ms
```

Bucket 'O Books

```
% SELECT width_bucket(length(review->'product_title'), 1, 50, 5) title_bkt,
         round(avg((review->'review_rating')::numeric), 2) review_avg,
         COUNT(*)
    FROM reviews
   WHERE review->'product_group' = 'DVD'
   GROUP BY title_bkt
   ORDER BY title_bkt;

 title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 207.665 ms
```

Could be better.

HSTORE GIN Indexing

HSTORE GIN Indexing

```
% CREATE INDEX idx_reviews_gin ON reviews USING gin(review);
CREATE INDEX
Time: 227250.133 ms
%
```

Get coffee.

HSTORE GIN Indexing

```
% CREATE INDEX idx_reviews_gin ON reviews USING gin(review);
CREATE INDEX
Time: 227250.133 ms
% SELECT width_bucket(length(review->'product_title'), 1, 50, 5) title_bkt,
       round(avg((review->'review_rating')::numeric), 2) review_avg,
       COUNT(*)
  FROM reviews
 WHERE review @> '"product_group" => "DVD"'
 GROUP BY title_bkt
 ORDER BY title_bkt;
```

Containment.

HSTORE GIN Indexing

```
% CREATE INDEX idx_reviews_gin ON reviews USING gin(review);
CREATE INDEX
Time: 227250.133 ms
% SELECT width_bucket(length(review->'product_title'), 1, 50, 5) title_bkt,
       round(avg((review->'review_rating')::numeric), 2) review_avg,
       COUNT(*)
  FROM reviews
 WHERE review @> '"product_group" => "DVD"'
 GROUP BY title_bkt
 ORDER BY title_bkt;
 title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 28.509 ms
```

Better.

HSTORE Index Sizing

HSTORE Index Sizing

- Database now 301 MB

HSTORE Index Sizing

- Database now 301 MB
- 17.6% Overhead for GIN index

HSTORE Index Sizing

- Database now 301 MB
- 17.6% Overhead for GIN index
- Use expression index for scalar values

HSTORE Index Sizing

- Database now 301 MB
- 17.6% Overhead for GIN index
- Use expression index for scalar values

```
% DROP INDEX idx_reviews_gin;
DROP INDEX;

% CREATE INDEX idx_dvd_reviews
      ON reviews ((review -> 'product_group'));
CREATE INDEX
Time: 8081.842 ms
```

No coffee.

HSTORE Index Sizing

HSTORE Index Sizing

- Database now 268 MB

HSTORE Index Sizing

- Database now 268 MB
- 5% Overhead for expression index

HSTORE Index Sizing

- Database now 268 MB
- 5% Overhead for expression index
- And performance?

Back to the Books

Back to the Books

```
% SELECT width_bucket(length(review->'product_title'), 1, 50, 5) title_bkt,
         round(avg((review->'review_rating')::numeric), 2) review_avg,
         COUNT(*)
    FROM reviews
   WHERE review->'product_group' = 'DVD'
   GROUP BY title_bkt
   ORDER BY title_bkt;

 title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 20.547 ms
```

Fetch scalar again.

Back to the Books

```
% SELECT width_bucket(length(review->'product_title'), 1, 50, 5) title_bkt,
         round(avg((review->'review_rating')::numeric), 2) review_avg,
         COUNT(*)
    FROM reviews
   WHERE review->'product_group' = 'DVD'
   GROUP BY title_bkt
   ORDER BY title_bkt;

 title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 20.547 ms
```

Very nice.

Dumping HSTORE

Dumping HSTORE

- Binary representation

Dumping HSTORE

- Binary representation
- Must be parsed and formatted

Dumping HSTORE

- Binary representation
- Must be parsed and formatted
- Quite fast:

Dumping HSTORE

- Binary representation
- Must be parsed and formatted
- Quite fast:

```
% time pg_dump hreviews > /dev/null
  0.30s user 0.23s system 37% cpu 1.379 total
```

When to use HSTORE

When to use HSTORE

- Fast key/value store

When to use HSTORE

- Fast key/value store
  - Binary representation

When to use HSTORE

- Fast key/value store
  - Binary representation
  - Slower I/O

When to use HSTORE

- Fast key/value store
  - Binary representation
  - Slower I/O
  - Faster operations

When to use HSTORE

- Fast key/value store
  - Binary representation
  - Slower I/O
  - Faster operations
  - GIN index support

When to use HSTORE

- Fast key/value store
  - Binary representation
  - Slower I/O
  - Faster operations
  - GIN index support
- Limited utility

When to use HSTORE

- Fast key/value store
  - Binary representation
  - Slower I/O
  - Faster operations
  - GIN index support
- Limited utility
  - No nesting

When to use HSTORE

- Fast key/value store
  - Binary representation
  - Slower I/O
  - Faster operations
  - GIN index support
- Limited utility
  - No nesting
  - Strings only

When to use HSTORE

- Fast key/value store
  - Binary representation
  - Slower I/O
  - Faster operations
  - GIN index support
- Limited utility
  - No nesting
  - Strings only
  - Custom format

When to use HSTORE

- Fast key/value store
  - Binary representation
  - Slower I/O
  - Faster operations
  - GIN index support
- Limited utility
  - No nesting
  - Strings only
  - Custom format
  - Requires parsing

“I discovered JSON. I do not claim to have invented JSON, because it already existed in nature.”

—Douglas Crockford

JSON

JSON

- Added in 9.2

JSON

- Added in 9.2
- Simple validation on input

JSON

- Added in 9.2
- Simple validation on input
- Stored as text (like XML)

JSON

- Added in 9.2
- Simple validation on input
- Stored as text (like XML)
- Uses server encoding

JSON

- Added in 9.2
- Simple validation on input
- Stored as text (like XML)
- Uses server encoding
- Preserves key order and duplicates

JSON

- Added in 9.2
- Simple validation on input
- Stored as text (like XML)
- Uses server encoding
- Preserves key order and duplicates
- Operators & Functions added in 9.3

JSON

- Added in 9.2
- Simple validation on input
- Stored as text (like XML)
- Uses server encoding
- Preserves key order and duplicates
- Operators & Functions added in 9.3
- Building functions in 9.4

JSON Operators

JSON Operators

```
% SELECT '[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2;
  ?column?
-------------
 {"c":"baz"}

%
```

Array lookup.

JSON Operators

```
% SELECT '[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2;
  ?column?
-------------
 {"c":"baz"}

% SELECT '{"a": {"b":"foo"}}'::json->'a';
  ?column?
-------------
 {"b":"foo"}

%
```

Key lookup.

JSON Operators

```
% SELECT '[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2;
  ?column?
-------------
 {"c":"baz"}

% SELECT '{"a": {"b":"foo"}}'::json->'a';
  ?column?
-------------
 {"b":"foo"}

% SELECT '{"a":1,"b":2}'::json->>'b';
 ?column?
----------
 2
%
```

Returns text.

JSON Path Operators

JSON Path Operators

```
% SELECT '[{"a": {"b":{"c": "foo"}}}]'::json#>'{a,b}';
  ?column?
-------------
 {"c": "foo"}

%
```

Path lookup.

JSON Path Operators

```
% SELECT '[{"a": {"b":{"c": "foo"}}}'::json#>'{a,b}';
  ?column?
-------------
 {"c": "foo"}

% SELECT '{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}';
 ?column?
----------
 3
%
```

Returns text.

JSON Constructors

JSON Constructors

```
% SELECT to_json('Tom says "Hi"'::text);
      to_json
-------------------
 "Tom says \"Hi\""

%
```

Scalars okay.

JSON Constructors

```
% SELECT to_json('Tom says "Hi"'::text);
      to_json
-------------------
 "Tom says \"Hi\""

% SELECT to_json(ROW(1,2));
     to_json
-----------------
 {"f1":1,"f2":2}

%
```

Composites objectified.

JSON Constructors

```
% SELECT to_json('Tom says "Hi"'::text);
      to_json
-------------------
 "Tom says \"Hi\""

% SELECT to_json(ROW(1,2));
     to_json
-----------------
 {"f1":1,"f2":2}

% SELECT to_json(ROW(1,true,NULL,'foo'));
                 to_json
-----------------------------------------
 {"f1":1,"f2":true,"f3":null,"f4":"foo"}

%
```

Data types respected.

JSON Constructors

JSON Constructors

```
% SELECT json_build_array(1,2,'three');
 json_build_array
------------------
 [1, 2, "three"]

%
```

Heterogeneity okay.

JSON Constructors

```
% SELECT json_build_array(1,2,'three');
 json_build_array
------------------
 [1, 2, "three"]

% SELECT json_build_object('foo',1,'bar',true);
     json_build_object
---------------------------
 {"foo" : 1, "bar" : true}

%
```

JSON Constructors

```
% SELECT json_build_array(1,2,'three');
 json_build_array
------------------
 [1, 2, "three"]

% SELECT json_build_object('foo',1,'bar',true);
     json_build_object
---------------------------
 {"foo" : 1, "bar" : true}

% SELECT json_build_object(
    'foo', 1,
    'bar', json_build_array(1,2,'three')
);
          json_build_object
------------------------------------
 {"foo" : 1, "bar" : [1, 2, "three"]}
%
```

Nesting!

JSON Sets

JSON Sets

```
% SELECT * FROM json_each('{"a":"foo", "b":"bar"}');
 key | value
-----+-------
 a   | "foo"
 b   | "bar"

%
```

JSON Values

JSON Sets

```
% SELECT * FROM json_each('{"a":"foo", "b":"bar"}');
 key | value
-----+-------
 a   | "foo"
 b   | "bar"

% SELECT * FROM json_each_text('{"a":"foo", "b":"bar"}');
 key | value
-----+-------
 a   | foo
 b   | bar

%
```

Text values.

Other JSON Functions

Other JSON Functions

- json_array_length()

Other JSON Functions

- json_array_length()
- json_object_keys()

Other JSON Functions

- json_array_length()
- json_object_keys()
- json_array_elements()

Other JSON Functions

- json_array_length()
- json_object_keys()
- json_array_elements()
- json_array_elements_text()

Other JSON Functions

- json_array_length()
- json_object_keys()
- json_array_elements()
- json_array_elements_text()
- json_typeof()
- json_to_record()

Other JSON Functions

- json_array_length()
- json_object_keys()
- json_array_elements()
- json_array_elements_text()
- json_typeof()
- json_to_record()

And more!

JSON Performance

JSON Performance

```
> createdb jreviews
> psql -d jreviews -c 'CREATE TABLE reviews(review json);'
CREATE TABLE
>
```

JSON Performance

```
> createdb jreviews
> psql -d jreviews -c 'CREATE TABLE reviews(review json);'
CREATE TABLE
> time psql jreviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 6.767 total
```

JSON Performance

```
> createdb jreviews
> psql -d jreviews -c 'CREATE TABLE reviews(review json);'
CREATE TABLE
> time psql jreviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 6.767 total
```

- 86,413 records/second

JSON Performance

```
> createdb jreviews
> psql -d jreviews -c 'CREATE TABLE reviews(review json);'
CREATE TABLE
> time psql jreviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 6.767 total
```

- 86,413 records/second
- 208 MB COPY file

JSON Performance

```
> createdb jreviews
> psql -d jreviews -c 'CREATE TABLE reviews(review json);'
CREATE TABLE
> time psql jreviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 6.767 total
```

- 86,413 records/second
- 208 MB COPY file
- 240 MB Database

JSON Performance

```
> createdb jreviews
> psql -d jreviews -c 'CREATE TABLE reviews(review json);'
CREATE TABLE
> time psql jreviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 6.767 total
```

- 86,413 records/second
- 208 MB COPY file
- 240 MB Database
- 15% storage overhead

JSON Performance

```
> createdb jreviews
> psql -d jreviews -c 'CREATE TABLE reviews(review json);'
CREATE TABLE
> time psql jreviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 6.767 total
```

- 86,413 records/second
- 208 MB COPY file
- 240 MB Database
- 15% storage overhead
- Faster than HSTORE

JSON Performance

```
> createdb jreviews
> psql -d jreviews -c 'CREATE TABLE reviews(review json);'
CREATE TABLE
> time psql jreviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 6.767 total
```

- 86,413 records/second
- 208 MB COPY file
- 240 MB Database
- 15% storage overhead
- Faster than HSTORE
- Slightly more overhead

Bucket ’O Books 2

Bucket 'O Books 2

```
% SELECT width_bucket(length(review#>>'{product,title}'), 1, 50, 5) title_bkt,
       round(avg((review#>>'{review,rating}')::numeric), 2) review_avg,
       COUNT(*)
  FROM reviews
 WHERE review#>>'{product,group}' = 'DVD'
 GROUP BY title_bkt
 ORDER BY title_bkt;
```

Path lookup.

Bucket 'O Books 2

```
% SELECT width_bucket(length(review#>>'{product,title}'), 1, 50, 5) title_bkt,
         round(avg((review#>>'{review,rating}')::numeric), 2) review_avg,
         COUNT(*)
    FROM reviews
   WHERE review#>>'{product,group}' = 'DVD'
   GROUP BY title_bkt
   ORDER BY title_bkt;

 title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 1765.824 ms
```

Yow!

JSON Indexing

JSON Indexing

- Operations slower than HSTORE

JSON Indexing

- Operations slower than HSTORE
- Text parsed per operation

JSON Indexing

- Operations slower than HSTORE
- Text parsed per operation
- No GIN or GiST

JSON Indexing

- Operations slower than HSTORE
- Text parsed per operation
- No GIN or GiST
- Can use expression index

JSON Indexing

- Operations slower than HSTORE
- Text parsed per operation
- No GIN or GiST
- Can use expression index

```
% CREATE INDEX idx_dvd_reviews
      ON reviews ((review#>>'{product,group}'));
CREATE INDEX
Time: 10222.241 ms
```

Back to the Books 3

Back to the Books 3

```
% SELECT width_bucket(length(review#>>'{product,title}'), 1, 50, 5) title_bkt,
         round(avg((review#>>'{review,rating}')::numeric), 2) review_avg,
         COUNT(*)
    FROM reviews
   WHERE review#>>'{product,group}' = 'DVD'
   GROUP BY title_bkt
   ORDER BY title_bkt;

 title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 91.863 ms
```

Pretty good.

Dumping JSON

Dumping JSON

- Stored as text

Dumping JSON

- Stored as text
- No parsing on output

Dumping JSON

- Stored as text
- No parsing on output
- Dumping 50-60% faster than HSTORE

Dumping JSON

- Stored as text
- No parsing on output
- Dumping 50-60% faster than HSTORE

```
% time pg_dump jreviews > /dev/null
   0.22s user 0.17s system 65% cpu 0.59 total
```

When to use JSON

When to use JSON

- Document storage

When to use JSON

- Document storage
- Duplicate preservation

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation
- Good storage

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation
- Good storage
    - Stored as text

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation
- Good storage
  - Stored as text
  - Fast I/O

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation
- Good storage
  - Stored as text
  - Fast I/O
- Operations are Awesome

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation
- Good storage
  - Stored as text
  - Fast I/O
- Operations are Awesome
  - Fetch keys, paths

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation
- Good storage
  - Stored as text
  - Fast I/O
- Operations are Awesome
  - Fetch keys, paths
  - Best on indexed scalars

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation
- Good storage
  - Stored as text
  - Fast I/O
- Operations are Awesome
  - Fetch keys, paths
  - Best on indexed scalars
  - When table scans okay

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation
- Good storage
  - Stored as text
  - Fast I/O
- Operations are Awesome
  - Fetch keys, paths
  - Best on indexed scalars
  - When table scans okay
  - Encumbers per-row parsing overhead

When to use JSON

- Document storage
- Duplicate preservation
- Key order preservation
- Good storage
  - Stored as text
  - Fast I/O
- Operations are Awesome
  - Fetch keys, paths
  - Best on indexed scalars
  - When table scans okay
  - Encumbers per-row parsing overhead

Sound familiar?

“Does this [JSONB] mean I can do Mongo-style queries, retrieving a set of documents which match particular key: value criteria, using PostgreSQL?”

— Mike MacCana via HN

JSONB

JSONB

- New in 9.4

JSONB

- New in 9.4
- Full JSON implementation

JSONB

- New in 9.4
- Full JSON implementation
- Uses server encoding

JSONB

- New in 9.4
- Full JSON implementation
- Uses server encoding
- Inspired by HSTORE 2

JSONB

- New in 9.4
- Full JSON implementation
- Uses server encoding
- Inspired by HSTORE 2
- Binary storage

JSONB

- New in 9.4
- Full JSON implementation
- Uses server encoding
- Inspired by HSTORE 2
- Binary storage
- HSTORE-style query operators

JSONB

- New in 9.4
- Full JSON implementation
- Uses server encoding
- Inspired by HSTORE 2
- Binary storage
- HSTORE-style query operators
- No key order or duplicate preservation

JSONB

- New in 9.4
- Full JSON implementation
- Uses server encoding
- Inspired by HSTORE 2
- Binary storage
- HSTORE-style query operators
- No key order or duplicate preservation
- Fast access operations

JSONB

- New in 9.4
- Full JSON implementation
- Uses server encoding
- Inspired by HSTORE 2
- Binary storage
- HSTORE-style query operators
- No key order or duplicate preservation
- Fast access operations
- GIN indexing

JSONB Operators

JSONB Operators

- All the JSON operators, plus…

JSONB Operators

- All the JSON operators, plus…

```
% SELECT '{"a": 1, "b": 2}'::jsonb = '{"b": 2, "a": 1}'::jsonb;
 ?column?
----------
 t

%
```

Equality!

JSONB Operators

- All the JSON operators, plus…

```
% SELECT '{"a": 1, "b": 2}'::jsonb = '{"b": 2, "a": 1}'::jsonb;
 ?column?
----------
 t

% SELECT '{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb;
 ?column?
----------
 t

%
```

Containment!

JSONB Operators

- All the JSON operators, plus…

```
% SELECT '{"a": 1, "b": 2}'::jsonb = '{"b": 2, "a": 1}'::jsonb;
 ?column?
----------
 t

% SELECT '{"a":1, "b":2}'::jsonb @> '{"b":2}'::jsonb;
 ?column?
----------
 t

% SELECT '{"a":1, "b":2}'::json ? 'b';
 ?column?
----------
 t
%
```

Existence!

Nested JSONB Operators

Nested JSONB Operators

```
% SELECT '{"a": [1,2]}'::jsonb = '{"a": [1,2]}'::jsonb;
 ?column?
----------
 t

%
```

Nested array.

Nested JSONB Operators

```
% SELECT '{"a": [1,2]}'::jsonb = '{"a": [1,2]}'::jsonb;
 ?column?
----------
 t

% SELECT '{"a": {"b": 2, "c": 3}}'::jsonb
         @> '{"a": {"c": 3}}'::jsonb;
 ?column?
----------
 t

%
```

Paths including values..

Nested JSONB Operators

```
% SELECT '{"a": [1,2]}'::jsonb = '{"a": [1,2]}'::jsonb;
 ?column?
----------
 t

% SELECT '{"a": {"b": 2, "c": 3}}'::jsonb
         @> '{"a": {"c": 3}}'::jsonb;
 ?column?
----------
 t

% SELECT '[1, 2, 3]'::jsonb @> '[3, 1]'::jsonb;
 ?column?
----------
 t
%
```

Ordering ignored.

JSONB Existence

JSONB Existence

```
% SELECT '{"a":1, "b":2, "c":3}'::jsonb ?| ARRAY['b', 'd'];
  ?column?
─────────────
 t

%
```

Any exist?

JSONB Existence

```
% SELECT '{"a":1, "b":2, "c":3}'::jsonb ?| ARRAY['b', 'd'];
  ?column?
-------------
 t

% SELECT '["a", "b", "c"]'::jsonb ?& ARRAY['a', 'b'];
  ?column?
-------------
 t

%
```

All exist?

JSONB Performance

JSONB Performance

```
> createdb breviews
> psql -d breviews -c 'CREATE TABLE reviews(review jsonb);'
CREATE TABLE
>
```

JSONB Performance

```
> createdb breviews
> psql -d breviews -c 'CREATE TABLE reviews(review jsonb);'
CREATE TABLE
> time psql breviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 9.841 total
```

JSONB Performance

```
> createdb breviews
> psql -d breviews -c 'CREATE TABLE reviews(review jsonb);'
CREATE TABLE
> time psql breviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 9.841 total
```

- 59,939 records/second

JSONB Performance

```
> createdb breviews
> psql -d breviews -c 'CREATE TABLE reviews(review jsonb);'
CREATE TABLE
> time psql breviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 9.841 total
```

- **59,939 records/second**
- **208 MB COPY file**

JSONB Performance

```
> createdb breviews
> psql -d breviews -c 'CREATE TABLE reviews(review jsonb);'
CREATE TABLE
> time psql breviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 9.841 total
```

- **59,939 records/second**
- **208 MB COPY file**
- **277 MB Database**

JSONB Performance

```
> createdb breviews
> psql -d breviews -c 'CREATE TABLE reviews(review jsonb);'
CREATE TABLE
> time psql breviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 9.841 total
```

- 59,939 records/second
- 208 MB COPY file
- 277 MB Database
- 32.9% storage overhead

JSONB Performance

```
> createdb breviews
> psql -d breviews -c 'CREATE TABLE reviews(review jsonb);'
CREATE TABLE
> time psql breviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 9.841 total
```

- 59,939 records/second
- 208 MB COPY file
- 277 MB Database
- 32.9% storage overhead
- Slower than JSON (86,413 r/s)

JSONB Performance

```
> createdb breviews
> psql -d breviews -c 'CREATE TABLE reviews(review jsonb);'
CREATE TABLE
> time psql breviews -c "COPY reviews FROM 'reviews.json'"
COPY 589859
     0.00s user 0.00s system 0% cpu 9.841 total
```

- 59,939 records/second
- 208 MB COPY file
- 277 MB Database
- 32.9% storage overhead
- Slower than JSON (86,413 r/s)
- Bigger than JSON (15% overhead)

Bucket ’O Books Redux

Bucket 'O Books Redux

```
% SELECT width_bucket(length(review#>>'{product,title}'), 1, 50, 5) title_bkt,
         round(avg((review#>>'{review,rating}')::numeric), 2) review_avg,
         COUNT(*)
    FROM reviews
   WHERE review#>>'{product,group}' = 'DVD'
   GROUP BY title_bkt
   ORDER BY title_bkt;

 title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 381.158 ms
```

JSON: 1765.824

HSTORE: 207.665

JSONB GIN Indexing

JSONB GIN Indexing

- Supports GIN Index

JSONB GIN Indexing

- Supports GIN Index
- Supports @>, ?, ?& and ?| operators

JSONB GIN Indexing

- Supports GIN Index
- Supports @>, ?, ?& and ?| operators

```
% CREATE INDEX idx_reviews_gin ON reviews USING gin(review);
CREATE INDEX
Time: 20296.090 ms
%
```

10x faster than HSTORE.

JSONB GIN Indexing

- Supports GIN Index
- Supports @>, ?, ?& and ?| operators

```
% CREATE INDEX idx_reviews_gin ON reviews USING gin(review);
CREATE INDEX
Time: 20296.090 ms
%
```

- DB Size: 341 MB

JSONB GIN Indexing

- Supports GIN Index
- Supports @>, ?, ?& and ?| operators

```
% CREATE INDEX idx_reviews_gin ON reviews USING gin(review);
CREATE INDEX
Time: 20296.090 ms
%
```

- DB Size: 341 MB
- 23.14% overhead

JSONB GIN Indexing

- Supports GIN Index
- Supports @>, ?, ?& and ?| operators

```
% CREATE INDEX idx_reviews_gin ON reviews USING gin(review);
CREATE INDEX
Time: 20296.090 ms
%
```

- DB Size: 341 MB
- 23.14% overhead
- Was 17.6% for HSTORE

JSONB GIN Performance

JSONB GIN Performance

```
% SELECT width_bucket(length(review#>>'{product,title}'), 1, 50, 5) title_bkt,
         round(avg((review#>>'{review,rating}')::numeric), 2) review_avg,
         COUNT(*)
    FROM reviews
   WHERE review @> '{"product": {"group": "DVD"}}'
   GROUP BY title_bkt
   ORDER BY title_bkt;
```

Containment.

JSONB GIN Performance

```
% SELECT width_bucket(length(review#>>'{product,title}'), 1, 50, 5) title_bkt,
       round(avg((review#>>'{review,rating}')::numeric), 2) review_avg,
       COUNT(*)
  FROM reviews
 WHERE review @> '{"product": {"group": "DVD"}}'
 GROUP BY title_bkt
 ORDER BY title_bkt;
 title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 35.633 ms
```

Pretty good.

JSONB GIN `json_path_ops`

JSONB GIN `json_path_ops`

- `jsonb_path_ops` for @> more efficient

JSONB GIN `json_path_ops`

- `jsonb_path_ops` for @> more efficient
- One index entry per path

JSONB GIN json_path_ops

- jsonb_path_ops for @> more efficient
- One index entry per path

```
% DROP INDEX idx_reviews_gin;
DROP INDEX
CREATE INDEX idx_reviews_gin
      ON reviews USING gin(review jsonb_path_ops);
CREATE INDEX
Time: 9086.793 ms
%
```

2x faster.

JSONB GIN json_path_ops

- jsonb_path_ops for @> more efficient
- One index entry per path

```
% DROP INDEX idx_reviews_gin;
DROP INDEX
CREATE INDEX idx_reviews_gin
      ON reviews USING gin(review jsonb_path_ops);
CREATE INDEX
Time: 9086.793 ms
%
```

- DB Size: 323 MB

JSONB GIN `json_path_ops`

- `jsonb_path_ops` for @> more efficient
- One index entry per path

```
% DROP INDEX idx_reviews_gin;
DROP INDEX
CREATE INDEX idx_reviews_gin
      ON reviews USING gin(review jsonb_path_ops);
CREATE INDEX
Time: 9086.793 ms
%
```

- DB Size: 323 MB
- 16.6% overhead

`json_path_ops` Performance

json_path_ops Performance

```
% SELECT width_bucket(length(review#>>'{product,title}'), 1, 50, 5) title_bkt,
         round(avg((review#>>'{review,rating}')::numeric), 2) review_avg,
         COUNT(*)
    FROM reviews
   WHERE review @> '{"product": {"group": "DVD"}}'
   GROUP BY title_bkt
   ORDER BY title_bkt;
title_bkt | review_avg | count
-----------+------------+-------
         1 |       4.27 |  2646
         2 |       4.44 |  4180
         3 |       4.53 |  1996
         4 |       4.38 |  2294
         5 |       4.48 |   943
         6 |       4.42 |   738
(6 rows)

Time: 29.817 ms
```

About the same.

Dumping JSONB

Dumping JSONB

- Binary representation

Dumping JSONB

- Binary representation
- Must be parsed on output

Dumping JSONB

- Binary representation
- Must be parsed on output
- Dumping 41% slower than HSTORE

Dumping JSONB

- Binary representation
- Must be parsed on output
- Dumping 41% slower than HSTORE

```
% time pg_dump breviews > /dev/null
   0.19s user 0.16s system 17% cpu 1.95 total
```

When to use JSONB

When to use JSONB

- When on 9.4

When to use JSONB

- When on 9.4
- No duplicate preservation

When to use JSONB

- When on 9.4
- No duplicate preservation
- No key order preservation

When to use JSONB

- When on 9.4
- No duplicate preservation
- No key order preservation
- Great object storage

When to use JSONB

- When on 9.4
- No duplicate preservation
- No key order preservation
- Great object storage
  - Binary representation

When to use JSONB

- When on 9.4
- No duplicate preservation
- No key order preservation
- Great object storage
  - Binary representation
  - Efficient operators

When to use JSONB

- When on 9.4
- No duplicate preservation
- No key order preservation
- Great object storage
  - Binary representation
  - Efficient operators
- Operations are Awesome

When to use JSONB

- When on 9.4
- No duplicate preservation
- No key order preservation
- Great object storage
  - Binary representation
  - Efficient operators
- Operations are Awesome
  - Fetch keys, paths

When to use JSONB

- When on 9.4
- No duplicate preservation
- No key order preservation
- Great object storage
  - Binary representation
  - Efficient operators
- Operations are Awesome
  - Fetch keys, paths
  - GIN index-aware

When to use JSONB

- When on 9.4
- No duplicate preservation
- No key order preservation
- Great object storage
  - Binary representation
  - Efficient operators
- Operations are Awesome
  - Fetch keys, paths
  - GIN index-aware
  - Expression indexes with GIN

When to use JSONB

- When on 9.4
- No duplicate preservation
- No key order preservation
- Great object storage
  - Binary representation
  - Efficient operators
- Operations are Awesome
  - Fetch keys, paths
  - GIN index-aware
  - Expression indexes with GIN
- Slower I/O

Review

Review

- XML

Review

- XML
  - Only when necessary

Review

- XML
  - Only when necessary
- HSTORE

Review

- XML
  - Only when necessary
- HSTORE
  - Flat okay

Review

- XML
  - Only when necessary
- HSTORE
  - Flat okay
  - Strings okay

Review

- XML
  - Only when necessary
- HSTORE
  - Flat okay
  - Strings okay
  - Fast operations

Review

- XML
  - Only when necessary
- HSTORE
  - Flat okay
  - Strings okay
  - Fast operations
- JSON

Review

- XML
  - Only when necessary
- HSTORE
  - Flat okay
  - Strings okay
  - Fast operations
- JSON
  - Document storage

Review

- XML
  - Only when necessary
- HSTORE
  - Flat okay
  - Strings okay
  - Fast operations
- JSON
  - Document storage
  - Key preservation

Review

- XML
  - Only when necessary
- HSTORE
  - Flat okay
  - Strings okay
  - Fast operations
- JSON
  - Document storage
  - Key preservation
- JSONB

Review

- XML
  - Only when necessary
- HSTORE
  - Flat okay
  - Strings okay
  - Fast operations
- JSON
  - Document storage
  - Key preservation
- JSONB
  - Everything else

"Use JSONB unless you have a very good reason not to."

—Theory

XML, HSTORE, JSON, JSONB—OH MY!

David E. Wheeler
@theory
theory.so

PGCon
2014-05-23