Binary storage for nested data structures and application to hstore data type

Oleg Bartunov, Teodor Sigaev

Moscow University

Hstore developers

- Teodor Sigaev, Oleg Bartunov
- Sternberg Astronomical Institute of Moscow University
- Major contributions:
  - PostgreSQL extendability: GiST, GIN, SP-GiST
  - Full-text search, ltree, pg_trgm, hstore, intarray,..

Agenda

- Introduction to hstore
- History of hstore development
- Hstore internals
- Limitations
- Hstore operators and functions
- Performance study
- Summary
- Development plans

Introduction to hstore

- Hstore — key/value storage (inspired by perl hash)

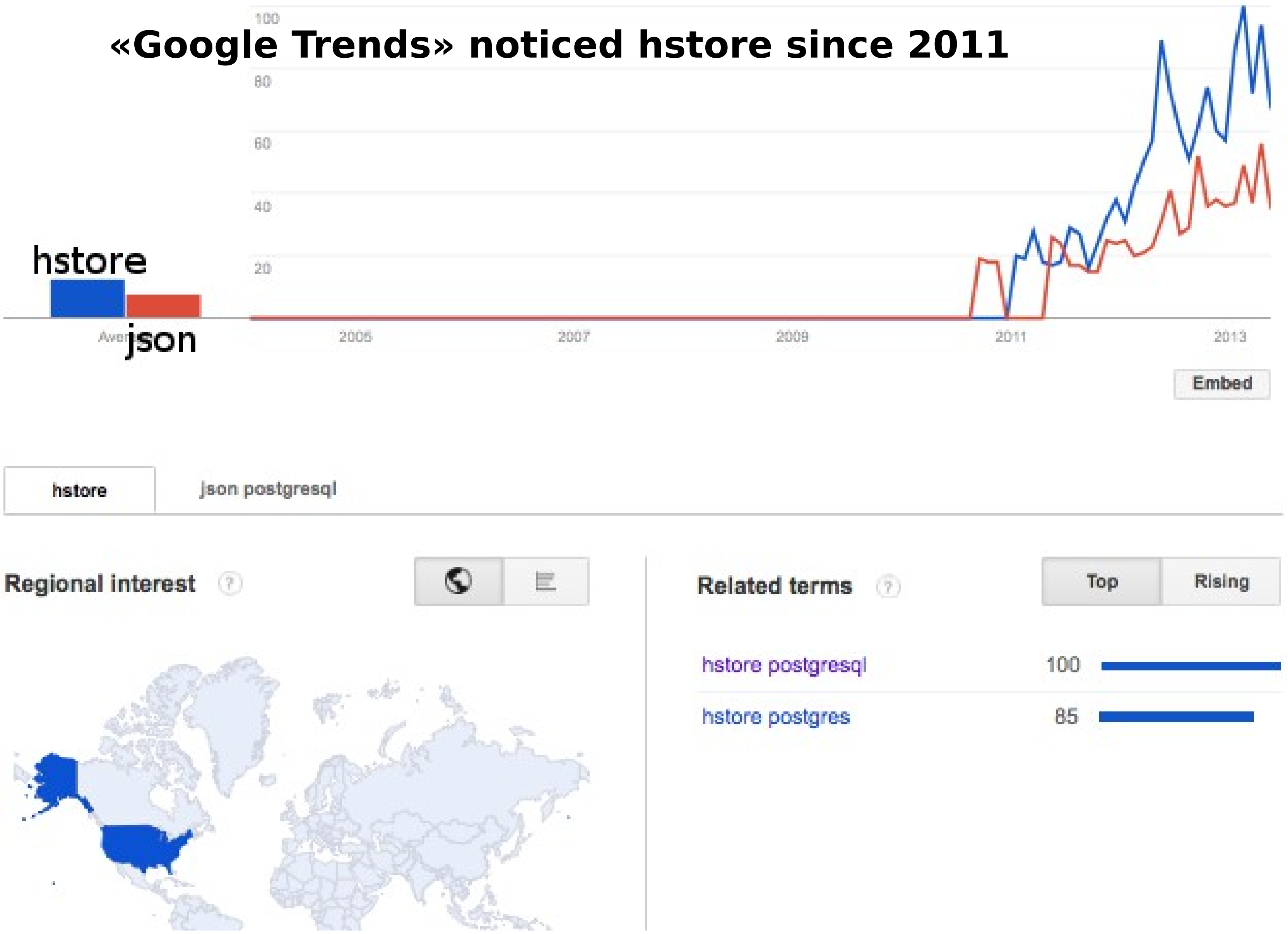
```
'a=>1, b=>2'::hstore
```

  - Key, value — strings
  - Get value for a key: hstore -> text
  - Operators with indexing support (GiST, GIN)

    Check for key: hstore ? text
    Contains: hstore @> hstore
    ……..check documentations for more ……..
  - Functions for hstore manipulations (akeys, avals, skeys, svals, each,......)

Introduction to hstore

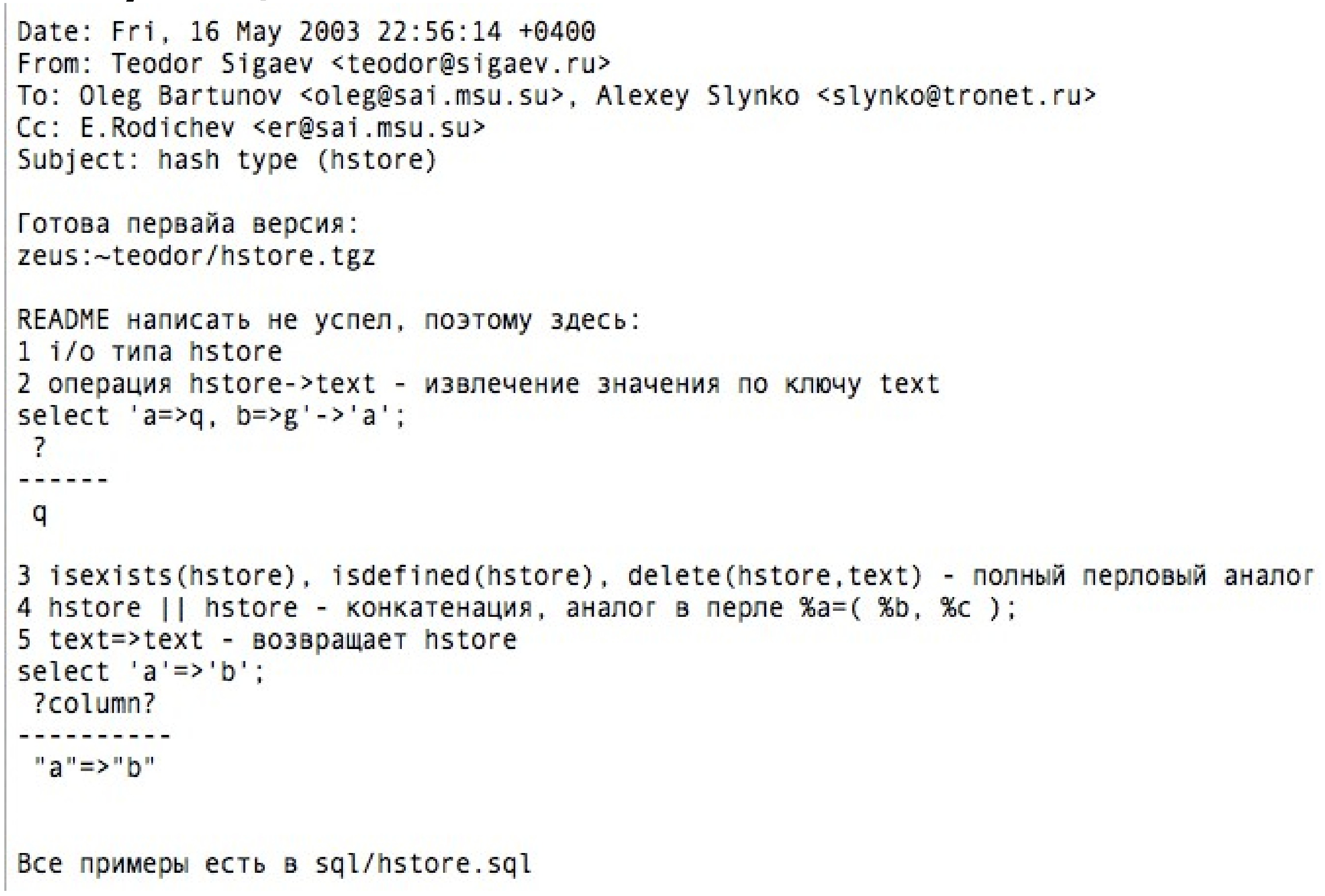
«Google Trends» noticed hstore since 2011



hstore postgresql	100
hstore postgres	85

History of hstore development

- May 16, 2003 — first version of hstore



```
Date: Fri, 16 May 2003 22:56:14 +0400
From: Teodor Sigaev <teodor@sigaev.ru>
To: Oleg Bartunov <oleg@sai.msu.su>, Alexey Slynko <slynko@tronet.ru>
Cc: E.Rodichev <er@sai.msu.su>
Subject: hash type (hstore)

Готова первайа версия:
zeus:~teodor/hstore.tgz

README написать не успел, поэтому здесь:
1 i/o типа hstore
2 операция hstore->text - извлечение значения по ключу text
select 'a=>q, b=>g'->'a';
 ?
------
 q

3 isexists(hstore), isdefined(hstore), delete(hstore,text) - полный перловый аналог
4 hstore || hstore - конкатенация, аналог в перле %a=( %b, %c );
5 text=>text - возвращает hstore
select 'a'=>'b';
 ?column?
----------
 "a"=>"b"


Все примеры есть в sql/hstore.sql
```

Introduction to hstore

- Hstore benefits
  - In provides a flexible model for storing a semi-structured data in relational database
- Hstore drawbacks
  - Too simple model !
    Hstore key-value model doesn't supports tree-like structures as json (introduced in 2006, 3 years after hstore)

hstore vs json

- PostgreSQL already has json since 9.0, which supports document-based model, but
  - It's slow, since it has no binary representation and needs to be parsed every time
  - Hstore is fast, thanks to binary representation and index support
  - It's possible to convert hstore to json and vice versa, but current hstore is limited to key-value
  - Need hstore with document-based model. Share it's binary representation with json !

History of hstore development

- May 16, 2003 - first (unpublished) version of hstore for PostgreSQL 7.3
- Dec, 05, 2006 - hstore is a part of PostgreSQL 8.2 (thanks, Hubert Depesz Lubaczewski!)
- May 23, 2007 - GIN index for hstore, PostgreSQL 8.3
- Sep, 20, 2010 - Andrew Gierth improved hstore, PostgreSQL 9.0
- May 24, 2013 - Nested hstore with array support, key->value model → document-based model PostgreSQL 9.4(?)

Hstore syntax

- Hash-like:

```
'a=>1'            '{a=>1}'
'a=>b, b=>c'   '{a=>b, b=>"c"}'
```

- Array-like:

```
'{a}'             '[a]'
'{a,b}'           '[a,b]'
```

- Scalar:
  'a'

Hstore types support

- Numeric

```
=# select 'a=>10.2'::hstore, '{1E6,2E-3,3.123456789}'::hstore;
  hstore    |              hstore
-----------+----------------------------------
 "a"=>10.2 | {1000000, 0.002, 3.123456789}

=# select pg_typeof('a=>1E6'::hstore -> 'a');
 pg_typeof
-----------
 text

=# select pg_typeof('a=>10.2'::hstore ^> 'a');
 pg_typeof
-----------
 numeric
```

Hstore types support

- Boolean

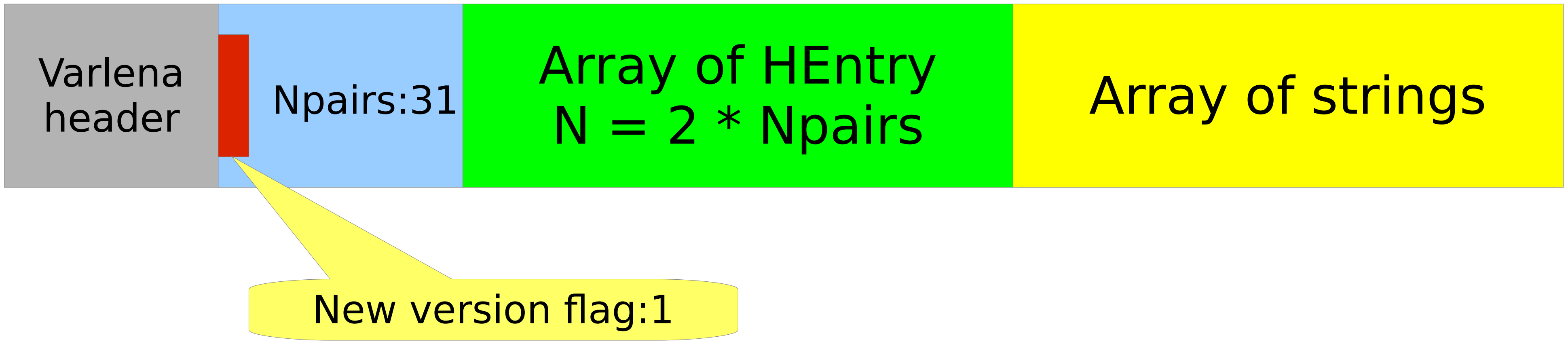
```
=# select 'a=>t'::hstore, '{TRUE,true,FALSE,f}'::hstore;
 hstore |     hstore
--------+---------------
 "a"=>t | {t, t, f, f}

=# select pg_typeof('a=>t'::hstore ?> 'a');
 pg_typeof
-----------
 boolean
```

Hstore types support

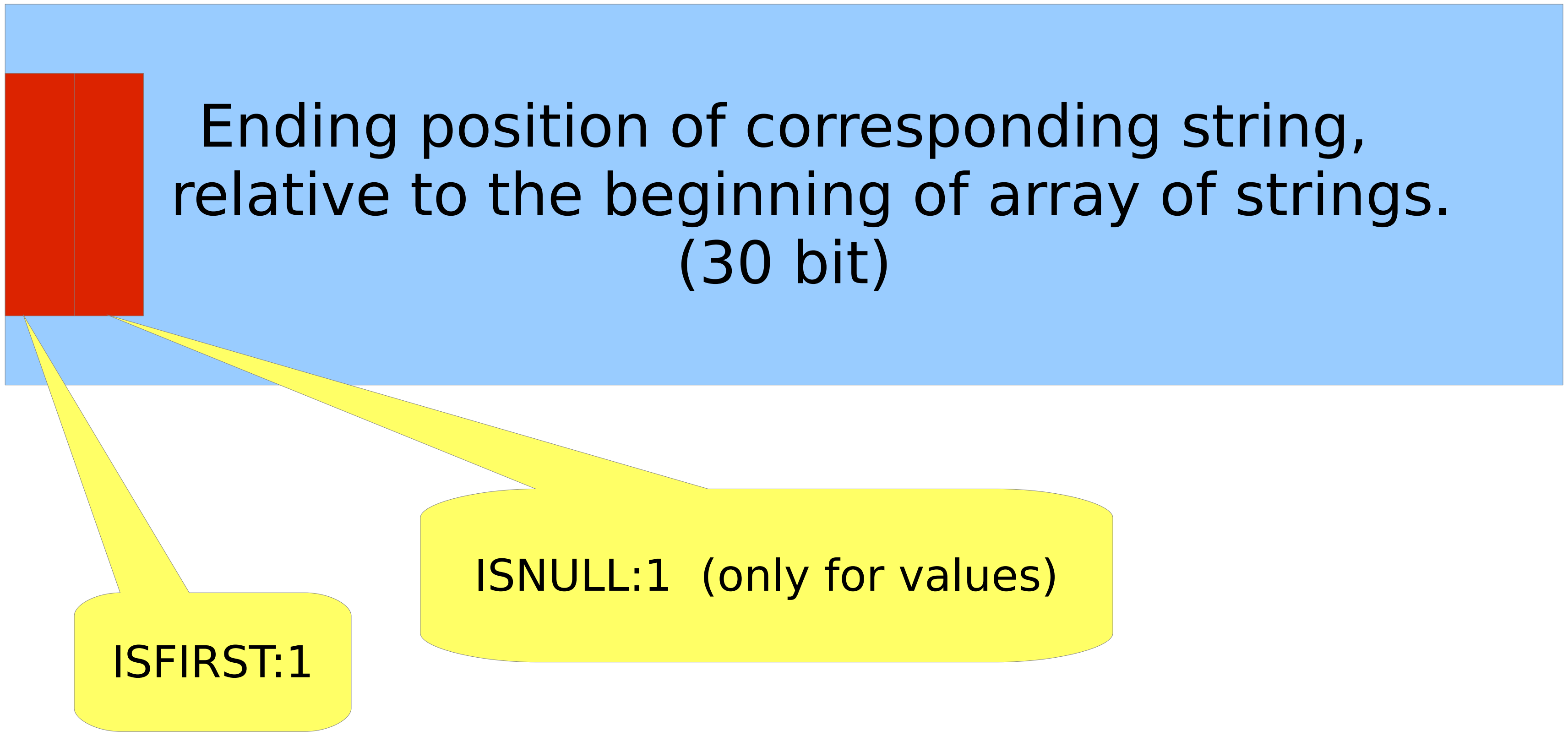
- NULL

```
=# select ('a=>NULL'::hstore -> 'a')  IS NULL;
 ?column?
----------
 t
```

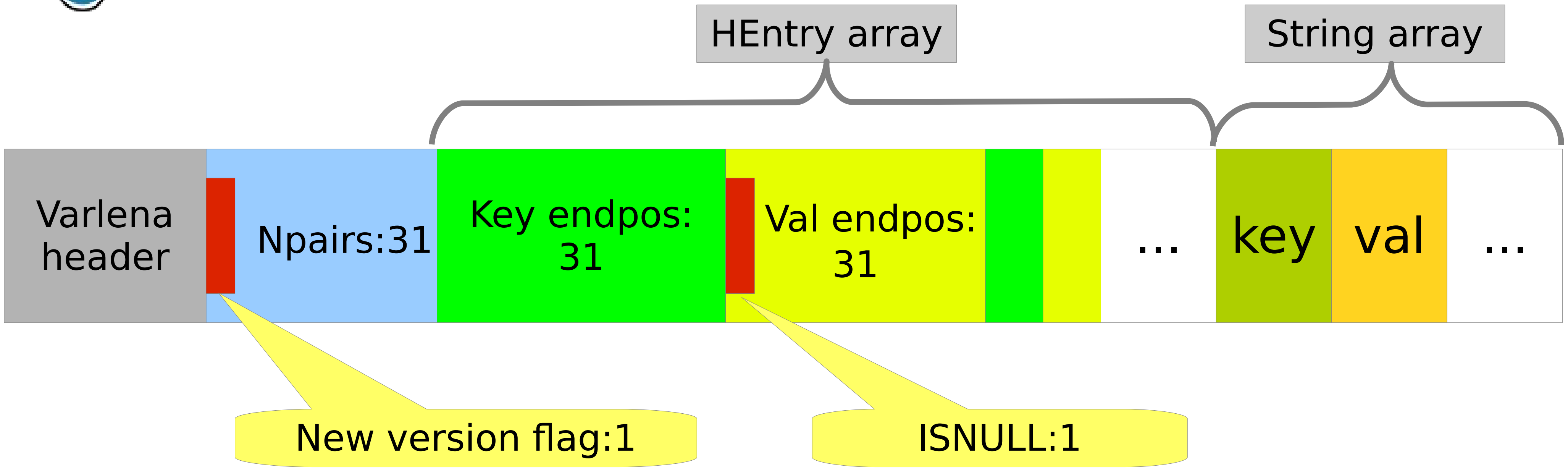
Current: HStore's internals



Current: HEntry


Ending position of corresponding string, relative to the beginning of array of strings. (30 bit)
ISNULL:1 (only for values)
ISFIRST:1

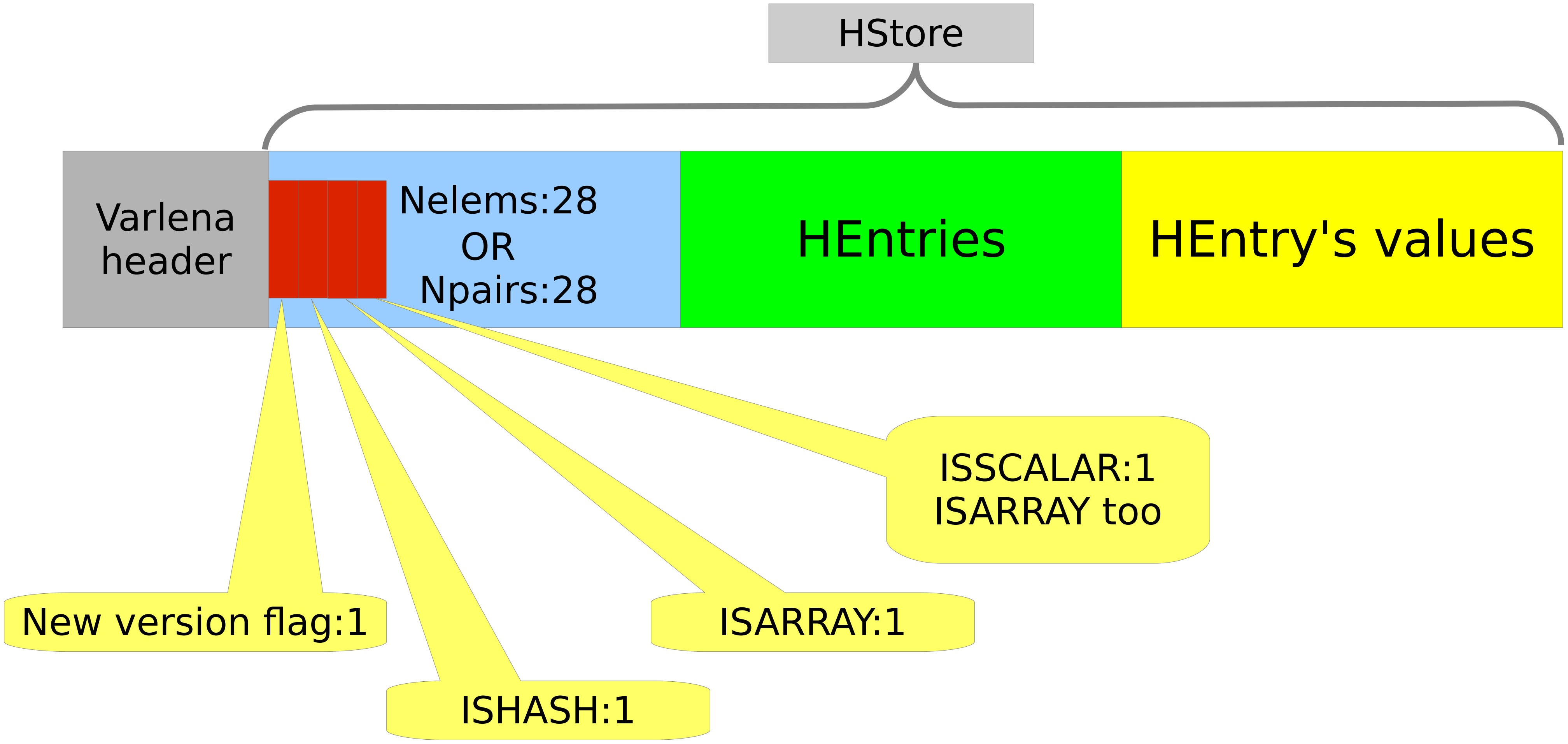
Current: Summary



	Start	End
First key	0	HEntry[0]
i-th key	HEntry[i*2 - 1]	HEntry[i*2]
i-th value	HEntry[i*2]	HEntry[i*2 + 1]

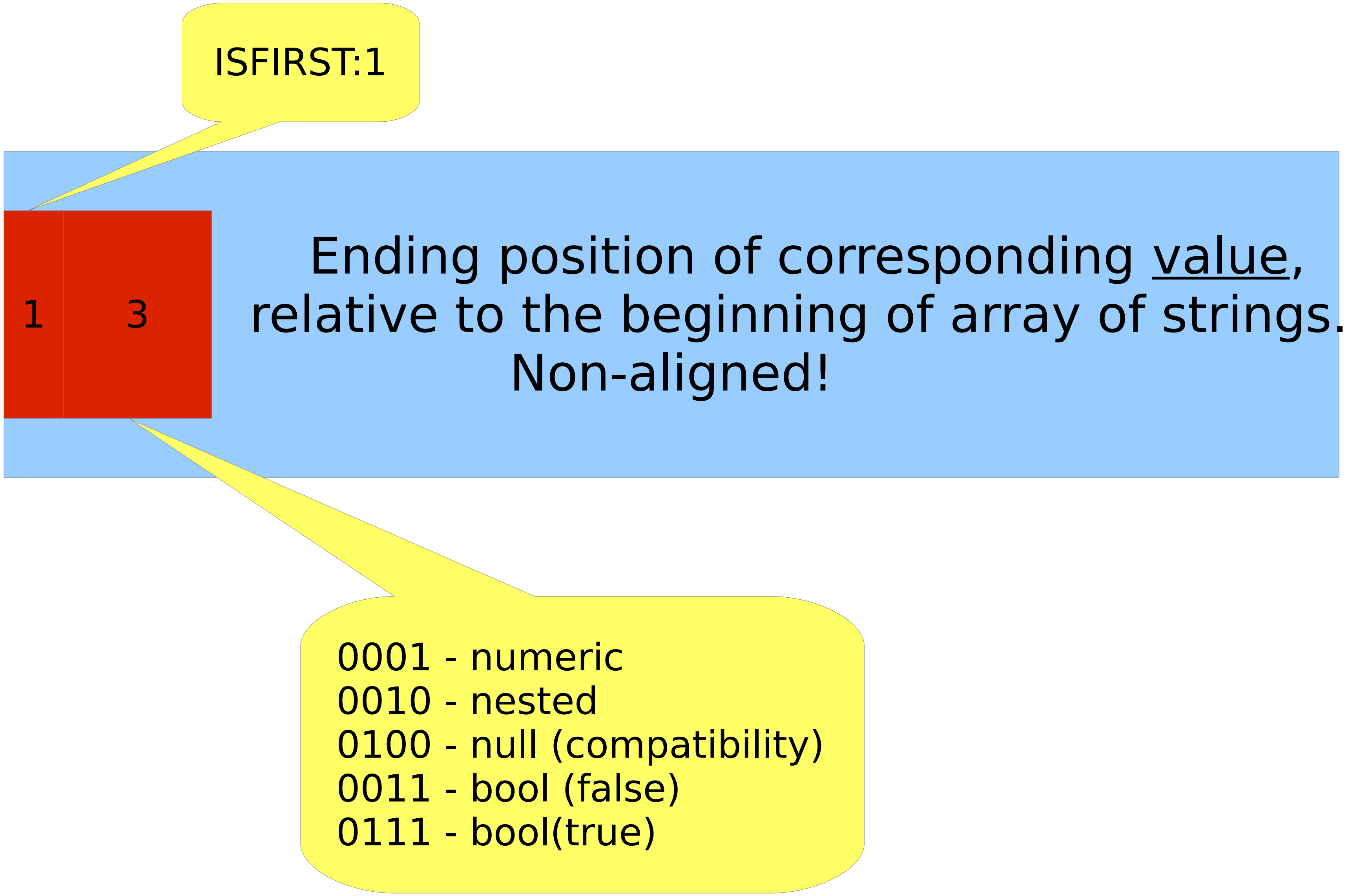
Pairs are lexicographically ordered by key

Nested: Layout



HEntry value could be an hstore itself
Scalar stored as a single-element array

Nested: HEntry

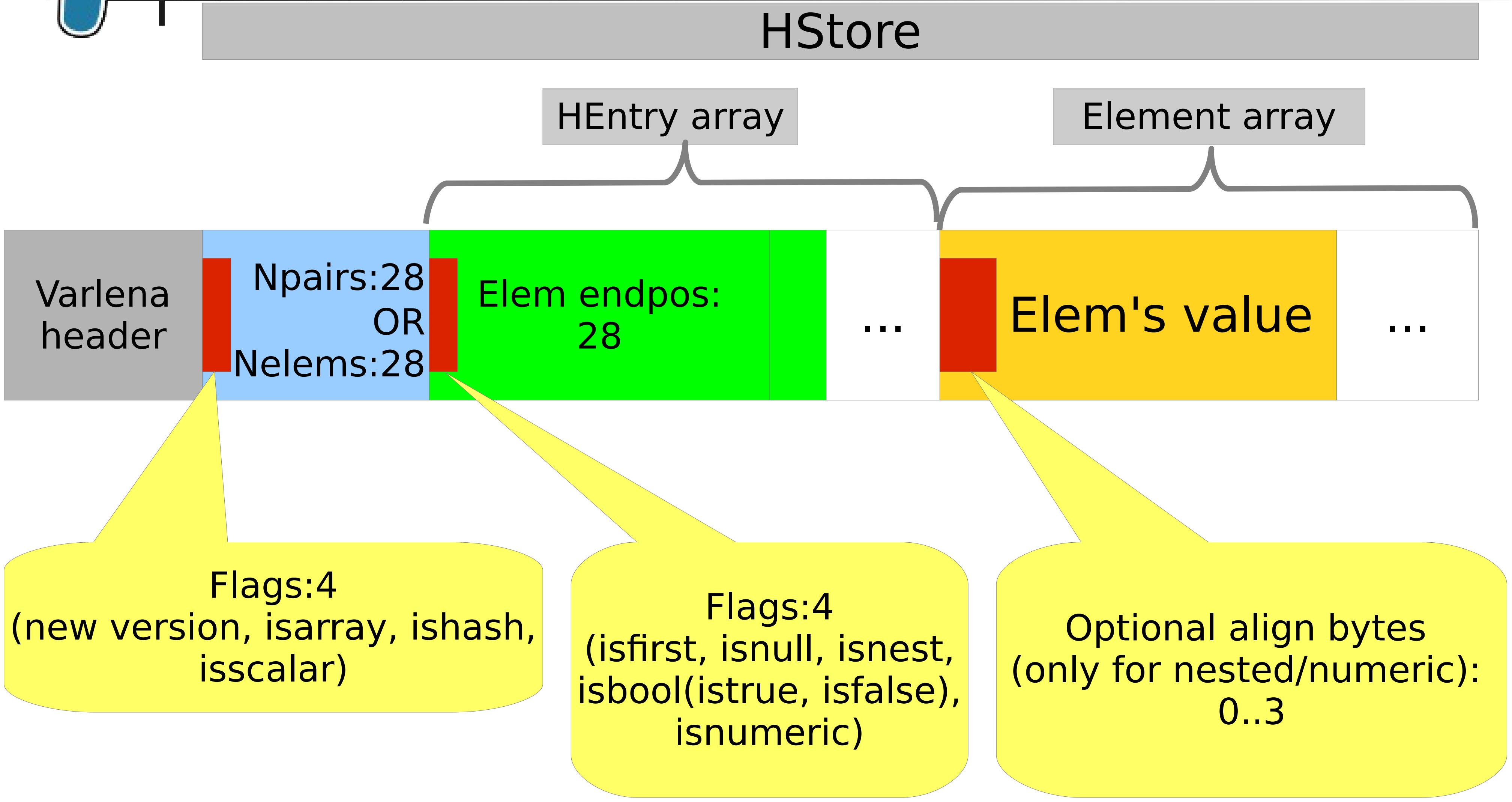


ISFIRST:1

Ending position of corresponding <u>value</u>, relative to the beginning of array of strings. Non-aligned!

0001 - numeric
0010 - nested
0100 - null (compatibility)
0011 - bool (false)
0111 - bool(true)

Nested: Summary


HStore
HEntry array
Element array
Varlena header
Npairs:28 OR Nelems:28
Elem endpos: 28
...
Elem's value
...
Flags:4 (new version, isarray, ishash, isscalar)
Flags:4 (isfirst, isnull, isnest, isbool(istrue, isfalse), isnumeric)
Optional align bytes (only for nested/numeric): 0..3

Nested: Access

For complex value start = INTALIGN(start)

HASH	Start	End
First key	0	HEntry[0]
i-th key	HEntry[i*2 - 1]	HEntry[i*2]
i-th value	align(HEntry[i*2])	HEntry[i*2 + 1]

Pairs are lexicographically ordered by key

ARRAY	Start	End
First elem	0	HEntry[0]
i-th elem	align(HEntry[i - 1])	HEntry[i]

Elements are not ordered

Hstore limitations

- Levels: unlimited
- Number of elements in array: 2^28
- Number of pairs in hash: 2^28
- Length of string: 2^28 bytes
- Length of nested hash or array: 2^28 bytes

2^28 bytes = 256 MB

Compatibility

- HStore as type is absolutely [pg_]upgrade-friendly
  (ISHASH bit could be set automatically,
  current version will always contains zeros)
- It's also true for GIN indexes: instead of KV notation it uses KVE
- It's not true for GiST: old version doesn't uses KV notation, now it uses KVE. Indexes should be rebuilded

Hstore syntax cont.

```
=# select '{{a=>1}, {1,2,3}, {c=>{d,f}}}'::hstore;
                    hstore
-----------------------------------------------------
 {{"a"=>1}, {1,2,3}, {"c"=>{"d", f}}}
```

- hstore.array_square_brackets [false],true

```
=# set hstore.array_square_brackets=true;
=# select '{{a=>1}, {1,2,3}, {c=>{d,f}}}'::hstore;
                    hstore
-----------------------------------------------------
 [{"a"=>1}, [1,2,3], {"c"=>["d", f]}]
```

Hstore syntax cont.

- hstore.root_hash_decorated true,[false]

```
=# select 'a=>1'::hstore;
 hstore
--------
 "a"=>1
=# set hstore.root_hash_decorated=true;
=# select 'a=>1'::hstore;
   hstore
------------
 {"a"=>1}
```

Hstore syntax cont.

```
=# set hstore.pretty_print=true;
=# select '{{a=>1}, {1,2,3}, {c=>{d,f}}}'::hstore;
      hstore
-------------------
 {                +
     {            +
         "a"=>1   +
     },           +
     {            +
         1,       +
         2,       +
         3        +
     },           +
     {            +
         "c"=>    +
         {        +
             "d", +
             f    +
         }        +
     }            +
 }
(1 row)
```

Operators and functions

- Get value by key
  - text hstore -> text

```
=# select 'a=>1,b=>{c=>3,d=>{4,5,6}},1=>f'::hstore -> 'b';
           ?column?
-----------------------------
 "c"=>3, "d"=>{4,5,6}
```

  - hstore hstore %> text

```
=# select 'a=>1,b=>{c=>3,d=>{4,5,6}},1=>f'::hstore %> 'b';
 ?column?
-----------------------------
 "c"=>3, "d"=>{4,5,6}
```

Operators and functions

- Get value by path
  - text hstore #> path

```
=# select 'a=>1,b=>{c=>3,d=>{4,5,6}},1=>f'::hstore #> '{b,d,0}';
?column?
--------
 4
```

  - hstore hstore #%> path

```
=# select 'a=>1,b=>{c=>3,d=>{4,5,6}},1=>f'::hstore #%>'{b,d}';
?column?
---------
 {4,5,6}
```

Operators and functions

- Get array element by index
  - text hstore->integer

```
=# select '{a,b,3,4,5}'::hstore->1;
 ?column?
----------
 b                                        – negative index starts from the end
=# select '{a,b,3,4,5}'::hstore-> -2;
 ?column?
----------
 4
```

Operators and functions

- Get array element by index
  - hstore hstore%>integer

```
=# select '{a,b,3,4,5}'::hstore%>1;
 ?column?
----------
  "b"                          – negative index starts from the end
=# select '{a,b,3,4,5}'::hstore%> -2;
 ?column?
----------
 4
```

Space is important :)

Operators and functions

- Chaining operators to go deep

```
=# select 'a=>1,b=>{c=>3,d=>{4,5,6}},1=>f'::hstore %> 'b'->'c';
 ?column?
----------
 3
=# select 'a=>1,b=>{c=>3,d=>{4,5,6}},1=>f'::hstore #%> '{b,d}'->0;
 ?column?
----------
 4
```

Operators and functions

- hstore hstore || hstore

```
=# select 'a=>1,b=>{c=>3,d=>{4,5,6}}'::hstore || 'b=>{c=>4}'::hstore;
         ?column?
----------------------------
 "a"=>1, "b"=>{"c"=>4}
```

- Concatenation with path

  hstore concat_path(hstore,text[],hstore)

```
=# select concat_path('a=>1,b=>{c=>3,d=>{4,5,6}}'::hstore,'{b,d}', '1');
                  concat_path
---------------------------------------------------
{"a"=>1, "b"=>{"c"=>3, "d"=>{4, 5, 6, 1}}}
```

Operators and functions

- Concatenation with path

  hstore concat_path(hstore,text[],hstore)

  With empty path it works exactly as old || operator

```
=# select concat_path('a=>1,b=>{c=>3,d=>{4,5,6}}'::hstore,'{}', 'a=>2');
              concat_path
---------------------------------------------------
{"a"=>2, "b"=>{"c"=>3, "d"=>{4, 5, 6}}}
```

Operators and functions

- Contains operators @>, <@ goes deep

```
=# SELECT 'a=>{1,2,{c=>3, x=>4}}, c=>b'::hstore @> 'a=>{{c=>3}}';
?column?
----------
 t
=# SELECT 'a=>{{c=>3}}' <@ 'a=>{1,2,{c=>3, x=>4}}, c=>b'::hstore;
?column?
----------
 t
```

Operators and functions

- setof hstore hvals(hstore)

```
=# SELECT *  FROM
      hvals('{{tags=>1, sh=>2}, {tags=>3, sh=>4}}'::hstore) AS q;
           q
------------------------
 "sh"=>2, "tags"=>1
 "sh"=>4, "tags"=>3
=# SELECT q->'tags'  FROM
      hvals('{{tags=>1, sh=>2}, {tags=>3, sh=>4}}'::hstore) AS q;
 ?column?
----------
   1
   3
```

Operators and functions

setof hstore hvals(hstore, text[])

```
=# SELECT *  FROM
      hvals('{{tags=>1, sh=>2,a=>{tags=>4}},
              {tags=>3,sh=>4}}'::hstore,'{null,tags}');
 hvals
-------
 1
 3
```

- setof text svals(hstore,text[])

Operators and functions

- Replace with path
  hstore replace(hstore,text[],hstore)

```
=# select replace('a=>1,b=>{c=>3,d=>{4,5,6}}'::hstore,'{b,d}', '1');
              replace
-----------------------------------------------
{"a"=>1, "b"=>{"c"=>3, "d"=>1}}
```

Operators and functions

- hstore <-> json conversion
  - json hstore_to_json(hstore)

```
=# select hstore_to_json('a=>1,b=>{c=>3,d=>{4,5,6}}'::hstore);
                hstore_to_json
-------------------------------------------------
 {"a": "1", "b": {"c": "3", "d": ["4", "5", "6"]}}
```

  - hstore json_to_hstore(json)

```
=# select json_to_hstore('{"a": "1", "b": {"c": "3", "d": ["4", "5",
"6"]}}'::json);
               json_to_hstore
-----------------------------------------------
{"a"=>1, "b"=>{"c"=>"3", "d"=>{"4", "5", "6"}}}
```

Operators and functions

- hstore <-> json cast
  - hstore::json

```
=# select 'a=>1'::hstore::json;
   json
------------
 {"a": 1}
```

  - json::hstore

```
=# select '{"a": 1}'::json::hstore;
 hstore
----------
 {"a"=>1}
```

Operators and functions

- hstore <-> json cast
  - Hstore **had** no types support as json, so :(

```
=# select '{"a":3.14}'::json::hstore::json;
     json
---------------
 {"a": "3.14"}

=# select '3.14'::json::hstore::json;
   json
----------
 ["3.14"]
```

Operators and functions

- hstore <-> json cast
  - Hstore **has** now types support and casting is fine !

```
=# select '{"a":3.14}'::json::hstore::json;
     json
---------------
 {"a": 3.14}

=# select '3.14'::json::hstore::json;
   json
----------
 3.14
```

Operators and functions

```
=# set hstore.pretty_print=true;
=# select hstore_to_json('{{a=>1}, {1,2,3}, {c=>{d,f}}}'::hstore);
  hstore_to_json
--------------------
 [                 +
     {             +
         "a": "1"  +
     },            +
     [             +
         "1",      +
         "2",      +
         "3"       +
     ],            +
     {             +
         "c":      +
         [         +
             "d",  +
             "f"   +
         ]         +
     }             +
 ]
(1 row)
```

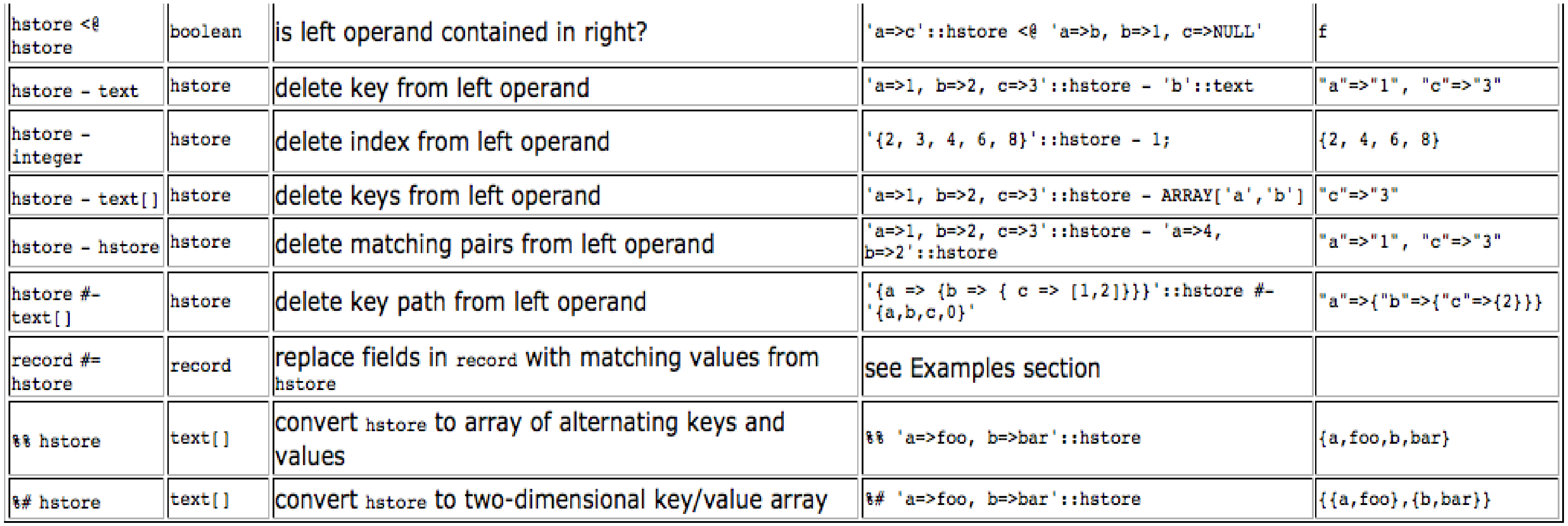
Operators matrix

```
-----------------+------+-----+---------------+---------------+----------+
right arg's type | text | int |text[](keys)   | text[](path)  |  hstore  |
-----------------+      |     |               |               |
return type      |      |     |               |               |          |
-----------------+------+-----+---------------+---------------+----------+
    text         |   -> |  -> |          ->   |      #>       |          |
-----------------+------+-----+---------------+---------------+----------+
    hstore       |%>, - |%>, -|           -   |      #%>, /   |   -,||   |
-----------------+------+-----+---------------+---------------+----------+
    bool         |   ?  |  ?  |       ?&, ?|  |      #?       |<@,@>,=,<>|
-----------------+------+-----+---------------+---------------+----------+
    numeric      |   ^> |  ^> |          ^>   |      #^>      |          |
-----------------+------+-----+---------------+---------------+----------+
```

Operators and functions

Operator	Returns	Description	Example	Result
`hstore -> text`	`text`	get value for key (`NULL` if not present)	`'a=>x, b=>y'::hstore -> 'a'`	`x`
`hstore -> integer`	`text`	get value for array index (`NULL` if not present)	`'{foo,bar,baz}'::hstore -> 1`	`bar`
`hstore ^> text`	`numeric`	get numeric value for key (`NULL` if not numeric or not present)	`'a=>42.0, b=>y'::hstore ^> 'a'`	`42.0`
`hstore ^> integer`	`numeric`	get numeric value for array index (`NULL` if not numeric or not present)	`'{foo,null,44}'::hstore ^> 2`	`44`
`hstore ?> text`	`numeric`	get boolean value for key (`NULL` if not boolean or not present)	`'a => 42.0, b => true'::hstore ?> 'b'`	`t`
`hstore ?> integer`	`numeric`	get boolean value for array index (`NULL` if not boolean or not present)	`'{false,null,44}'::hstore ?> 0`	`f`
`hstore #> text[]`	`text`	get value for key path (`NULL` if not present)	`'foo => {bar => yellow}'::hstore #> '{foo,bar}'`	`yellow`
`hstore #^> text[]`	`numeric`	get numeric value for key path (`NULL` if not numeric or not present)	`'foo => {bar => 99}'::hstore #^> '{foo,bar}'`	`99`
`hstore #?> text[]`	`boolean`	get boolean value for key path (`NULL` if not booelan or not present)	`'foo => {bar => true}'::hstore #?> '{foo,bar}'`	`t`
`hstore %> text`	`hstore`	get hstore value for key (`NULL` if not present)	`'foo => {bar => 99}'::hstore %> 'foo'`	`"bar"=>99`
`hstore %> integer`	`hstore`	get hstore value array index (`NULL` if not present)	`'[1, 2, {foo=>hi}]'::hstore %> 2'`	`"foo"=>"hi"`
`hstore #%> text[]`	`hstore`	get hstore value for key path (`NULL` if not present)	`'a => 1, b => {c => [44,44]}'::hstore #%> '{b,c}'`	`{44, 44}`
`hstore -> text[]`	`text[]`	get values for keys (`NULL` if not present)	`'a=>x, b=>y, c=>z'::hstore -> ARRAY['c','a']`	`{"z","x"}`
`hstore || hstore`	`hstore`	concatenate `hstore`s	`'a=>b, c=>d'::hstore || 'c=>x, d=>q'::hstore`	`"a"=>"b", "c"=>"x", "d"=>"q"`
`hstore ? text`	`boolean`	does `hstore` contain key?	`'a=>1'::hstore ? 'a'`	`t`
`hstore ? integer`	`boolean`	does `hstore` contain array index?	`'a,b,c'::hstore ? 2`	`t`
`hstore #? text[]`	`boolean`	does `hstore` contain key path?	`'[1, 2, {foo=>hi}]'::hstore #? '{2,foo}'`	`t`
`hstore ?& text[]`	`boolean`	does `hstore` contain all specified keys?	`'a=>1,b=>2'::hstore ?& ARRAY['a','b']`	`t`
`hstore ?| text[]`	`boolean`	does `hstore` contain any of the specified keys?	`'a=>1,b=>2'::hstore ?| ARRAY['b','c']`	`t`
`hstore @> hstore`	`boolean`	does left operand contain right?	`'a=>b, b=>1, c=>NULL'::hstore @> 'b=>1'`	`t`

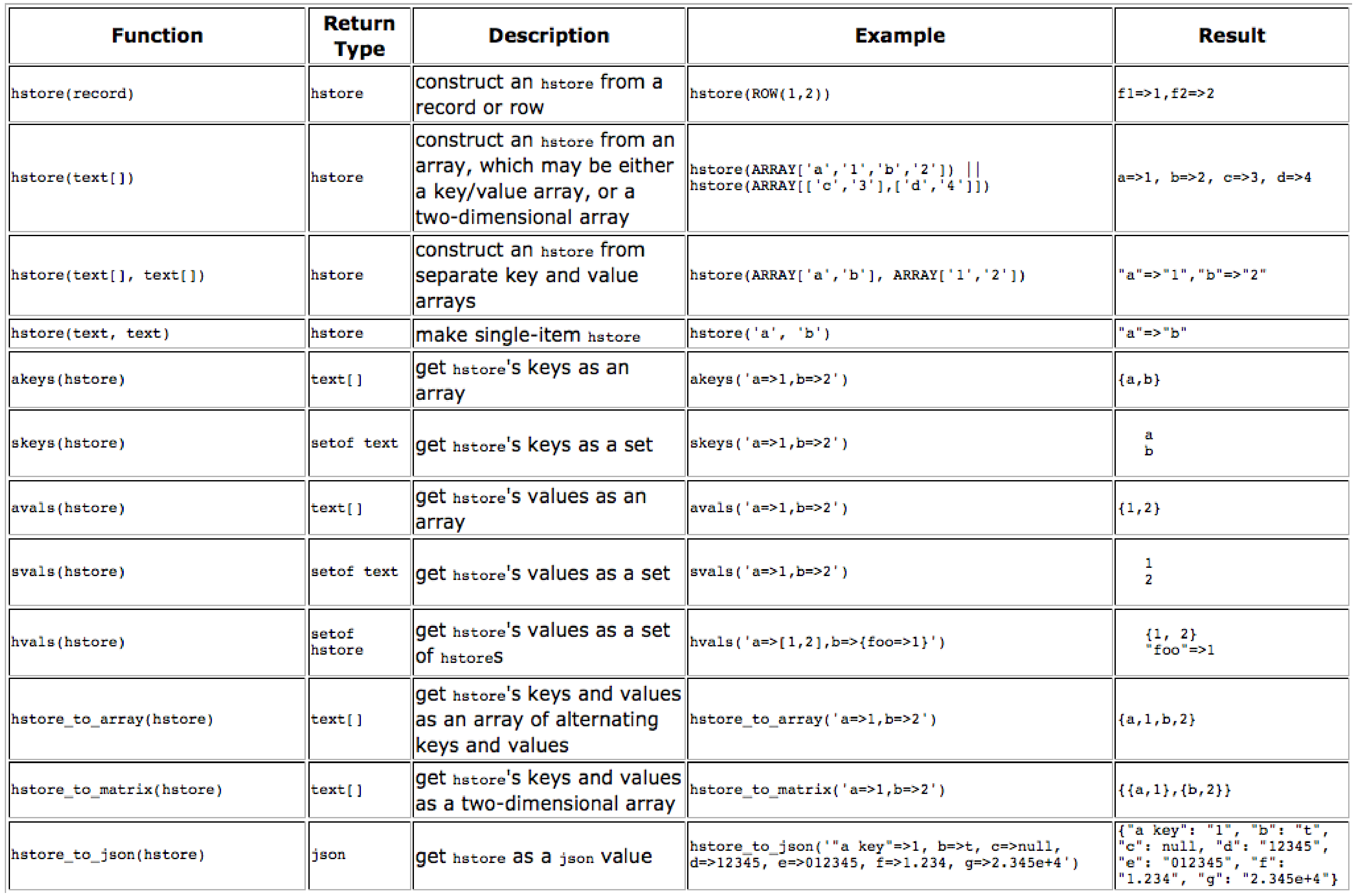
Operators and functions



hstore <@ hstore	boolean	is left operand contained in right?	'a=>c'::hstore <@ 'a=>b, b=>1, c=>NULL'	f
hstore - text	hstore	delete key from left operand	'a=>1, b=>2, c=>3'::hstore - 'b'::text	"a"=>"1", "c"=>"3"
hstore - integer	hstore	delete index from left operand	'{2, 3, 4, 6, 8}'::hstore - 1;	{2, 4, 6, 8}
hstore - text[]	hstore	delete keys from left operand	'a=>1, b=>2, c=>3'::hstore - ARRAY['a','b']	"c"=>"3"
hstore - hstore	hstore	delete matching pairs from left operand	'a=>1, b=>2, c=>3'::hstore - 'a=>4, b=>2'::hstore	"a"=>"1", "c"=>"3"
hstore #- text[]	hstore	delete key path from left operand	'{a => {b => { c => [1,2]}}}'::hstore #- '{a,b,c,0}'	"a"=>{"b"=>{"c"=>{2}}}
record #= hstore	record	replace fields in record with matching values from hstore	see Examples section	
%% hstore	text[]	convert hstore to array of alternating keys and values	%% 'a=>foo, b=>bar'::hstore	{a,foo,b,bar}
%# hstore	text[]	convert hstore to two-dimensional key/value array	%# 'a=>foo, b=>bar'::hstore	{{a,foo},{b,bar}}

Operators and functions

Table F-7. hstore Functions



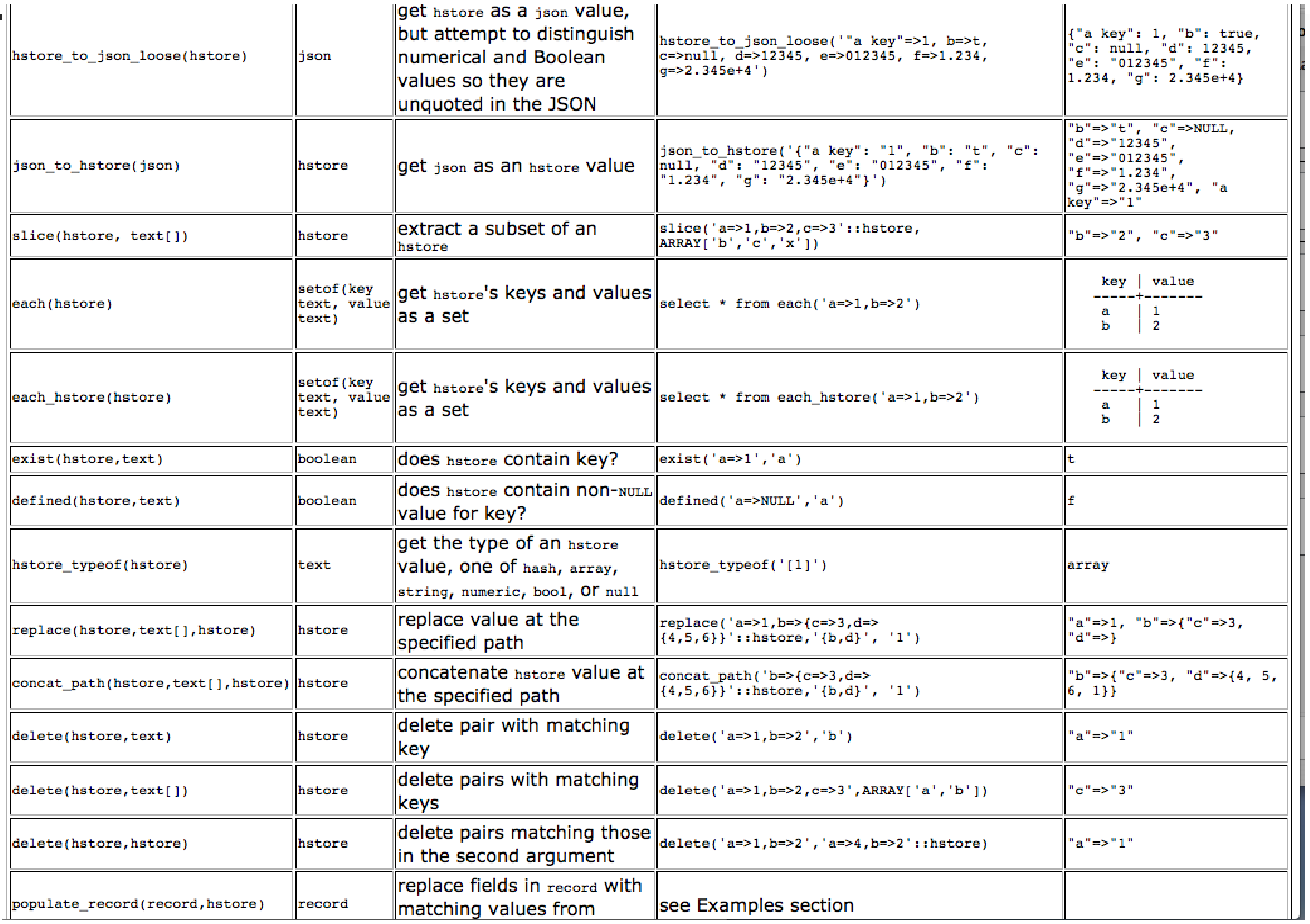
Function	Return Type	Description	Example	Result
`hstore(record)`	`hstore`	construct an `hstore` from a record or row	`hstore(ROW(1,2))`	`f1=>1,f2=>2`
`hstore(text[])`	`hstore`	construct an `hstore` from an array, which may be either a key/value array, or a two-dimensional array	`hstore(ARRAY['a','1','b','2']) || hstore(ARRAY[['c','3'],['d','4']])`	`a=>1, b=>2, c=>3, d=>4`
`hstore(text[], text[])`	`hstore`	construct an `hstore` from separate key and value arrays	`hstore(ARRAY['a','b'], ARRAY['1','2'])`	`"a"=>"1","b"=>"2"`
`hstore(text, text)`	`hstore`	make single-item `hstore`	`hstore('a', 'b')`	`"a"=>"b"`
`akeys(hstore)`	`text[]`	get `hstore`'s keys as an array	`akeys('a=>1,b=>2')`	`{a,b}`
`skeys(hstore)`	`setof text`	get `hstore`'s keys as a set	`skeys('a=>1,b=>2')`	`a` `b`
`avals(hstore)`	`text[]`	get `hstore`'s values as an array	`avals('a=>1,b=>2')`	`{1,2}`

`svals(hstore)`	`setof text`	get `hstore`'s values as a set	`svals('a=>1,b=>2')`	`1`
				`2`

`hvals(hstore)`	`setof hstore`	get `hstore`'s values as a set of `hstore`s	`hvals('a=>[1,2],b=>{foo=>1}')`	`{1, 2}`
				`"foo"=>1`

`hstore_to_array(hstore)`	`text[]`	get `hstore`'s keys and values as an array of alternating keys and values	`hstore_to_array('a=>1,b=>2')`	`{a,1,b,2}`
`hstore_to_matrix(hstore)`	`text[]`	get `hstore`'s keys and values as a two-dimensional array	`hstore_to_matrix('a=>1,b=>2')`	`{{a,1},{b,2}}`
`hstore_to_json(hstore)`	`json`	get `hstore` as a `json` value	`hstore_to_json('"a key"=>1, b=>t, c=>null, d=>12345, e=>012345, f=>1.234, g=>2.345e+4')`	`{"a key": "1", "b": "t", "c": null, "d": "12345", "e": "012345", "f": "1.234", "g": "2.345e+4"}`

Operators and functions



hstore_to_json_loose(hstore)	json	get hstore as a json value, but attempt to distinguish numerical and Boolean values so they are unquoted in the JSON	hstore_to_json_loose('"a key"=>1, b=>t, c=>null, d=>12345, e=>012345, f=>1.234, g=>2.345e+4')	{"a key": 1, "b": true, "c": null, "d": 12345, "e": "012345", "f": 1.234, "g": 2.345e+4}
json_to_hstore(json)	hstore	get json as an hstore value	json_to_hstore('{"a key": "1", "b": "t", "c": null, "d": "12345", "e": "012345", "f": "1.234", "g": "2.345e+4"}')	"b"=>"t", "c"=>NULL, "d"=>"12345", "e"=>"012345", "f"=>"1.234", "g"=>"2.345e+4", "a key"=>"1"
slice(hstore, text[])	hstore	extract a subset of an hstore	slice('a=>1,b=>2,c=>3'::hstore, ARRAY['b','c','x'])	"b"=>"2", "c"=>"3"
each(hstore)	setof(key text, value text)	get hstore's keys and values as a set	select * from each('a=>1,b=>2')	key | value -----+------- a | 1 b | 2
each_hstore(hstore)	setof(key text, value text)	get hstore's keys and values as a set	select * from each_hstore('a=>1,b=>2')	key | value -----+------- a | 1 b | 2
exist(hstore,text)	boolean	does hstore contain key?	exist('a=>1','a')	t
defined(hstore,text)	boolean	does hstore contain non-NULL value for key?	defined('a=>NULL','a')	f
hstore_typeof(hstore)	text	get the type of an hstore value, one of hash, array, string, numeric, bool, or null	hstore_typeof('[1]')	array
replace(hstore,text[],hstore)	hstore	replace value at the specified path	replace('a=>1,b=>{c=>3,d=>{4,5,6}}'::hstore,'{b,d}', '1')	"a"=>1, "b"=>{"c"=>3, "d"=>}
concat_path(hstore,text[],hstore)	hstore	concatenate hstore value at the specified path	concat_path('b=>{c=>3,d=>{4,5,6}}'::hstore,'{b,d}', '1')	"b"=>{"c"=>3, "d"=>{4, 5, 6, 1}}
delete(hstore,text)	hstore	delete pair with matching key	delete('a=>1,b=>2','b')	"a"=>"1"
delete(hstore,text[])	hstore	delete pairs with matching keys	delete('a=>1,b=>2,c=>3',ARRAY['a','b'])	"c"=>"3"
delete(hstore,hstore)	hstore	delete pairs matching those in the second argument	delete('a=>1,b=>2','a=>4,b=>2'::hstore)	"a"=>"1"
populate_record(record,hstore)	record	replace fields in record with matching values from	see Examples section	

Performance

- Data
  - 1,252,973 bookmarks from Delicious in json format
  - The same bookmarks in hstore format
  - The same bookmarks as text
- Server
  - desktop Linux, 8 GB RAM, 4-cores Xeon 3.2 GHz,
- Test
  - Input performance - copy data to table
  - Access performance - get value by key
  - Search performance contains @> operator

Performance

- Data
  - 1,252,973 bookmarks from Delicious in json format
  - The same bookmarks in hstore format
  - The same bookmarks as text

```
=# \dt+
                         List of relations
 Schema | Name | Type  |  Owner    |  Size    | Description
--------+------+-------+-----------+----------+-------------
 public | hs   | table | postgres | 1379 MB |
 public | js   | table | postgres | 1322 MB |
 public | tx   | table | postgres | 1322 MB |
```

Performance

```
=# select h  from hs limit 1;
                                              h
-----------------------------------------------------------------------------------------------
     "id"=>"http://delicious.com/url/b5b3cbf9a9176fe43c27d7b4af94a422#mcasas1",              +
     "link"=>"http://www.theatermania.com/broadway/",                                        +
     "tags"=>                                                                                +
     {                                                                                       +
         {                                                                                   +
             "term"=>"NYC",                                                                  +
             "label"=>NULL,                                                                  +
             "scheme"=>"http://delicious.com/mcasas1/"                                       +
         },                                                                                  +
         {                                                                                   +
             "term"=>"english",                                                              +
             "label"=>NULL,                                                                  +
             "scheme"=>"http://delicious.com/mcacas1/"                                       +
         },                                                                                  +
     },                                                                                      +
     "links"=>                                                                               +
     {                                                                                       +
         {                                                                                   +
             "rel"=>"alternate",                                                             +
             "href"=>"http://www.theatermania.com/broadway/",                                +
             "type"=>"text/html"                                                             +
         }                                                                                   +
     },                                                                                      +
     "title"=>"TheaterMania",                                                                +
     "author"=>"mcasas1",                                                                    +
     "source"=>NULL,                                                                         +
     "updated"=>"Tue, 08 Sep 2009 23:28:55 +0000",                                           +
     "comments"=>"http://delicious.com/url/b5b3cbf9a9176fe43c27d7b4af94a422",                +
     "guidislink"=>"false",                                                                  +
     "title_detail"=>                                                                        +
     {                                                                                       +
         "base"=>"http://feeds.delicious.com/v2/rss/recent?min=1&count=100",                 +
         "type"=>"text/plain",                                                               +
         "value"=>"TheaterMania",                                                            +
         "language"=>NULL                                                                    +
     },                                                                                      +
     "wfw_commentrss"=>"http://feeds.delicious.com/v2/rss/url/b5b3cbf9a9176fe43c27d7b4af94a422"+
```

Performance

- Input performance
  - Copy data (1,252,973 rows) as text, json,hstore

```
copy tt from  '/path/to/test.dump'
```

Text: 57 s

Json: 61 s

Hstore: 76 s – there is some room to speedup

Performance

- Access performance — get value by key
  - Base: select h from hs;
  - Hstore: select h->'updated' from hs;
  - Json: select j->>'updated' from js;
  - Regexp: select (regexp_matches(t, '"updated":"([^"]*)"'))[1] from tx;

Base:	0.3 s
hstore:	0.5 s
Json:	11. s
regexp:	18.8 s

Performance

- Access performance — get value by key

Base:	0.3 s
hstore:	0.5 s
Json:	11. s
regexp:	18.8 s

- Hstore is ~ 50x faster json
  thanks to binary representation !

Performance

- Search performance — contains @> operator
  - Hstore - seqscan, GiST, GIN

```
select count(*) from hs where h @> 'tags=>{{term=>NYC}}';
```

  - Json — estimation, GiST, GIN (functional indexes) exact time > estimation (there are may be many tags)

```
select count(*) from js where j#>>'{tags,0,term}' = 'NYC';
```

Performance

- Search performance — contains @> operator
  - Hstore - seqscan, GiST, GIN

seqscan	GiST	GIN	
	100s	400s	- create index
	64MB	815MB	
0.98s	0.3s	0.1s	
	3x	10x	

  - Json — estimation, GiST, GIN (functional indexes)

estimation	GiST	GIN	
	130s	500s	- create index
12s	2s	0.1s	
	6x	120x	

Recheck (GiST) calls json_to_hstore()

Summary

- Hstore is now nested and supports arrays
  Document-based model !
- Hstore access to specified field is fast (thanks to binary representation)
- Hstore operators can use GiST and GIN indexes
- Json users can use functional GIN index and get considerable speedup
- Hstore's binary representation can be used by json

Development plans

- Speedup hstore input
- Hstore query language - hpath, hquery ?
- Better indexing - SP-GiST-GIN hybrid index
- Statistics support (challenging task)
- Types support (+)
- Documentation (+), thanks David Wheeler !
- Submit patch for 9.4, David Wheeler review
- Add binary representation to json, Dunstan ?
- Add index support for json, Dunstan ?

GIN Fast-scan

- Observation
  - GIN indexes separately keys and values
  - Key 'tags' is very frequent -1138532,
    value '{{term=>NYC}}' is rare — 285
  - Current GIN: time (freq & rare) ~ time(freq)

    Fast-scan : time (freq & rare) ~ time(rare)

```
=# select count(*) from hs where h::hstore @>
'tags=>{{term=>NYC}}'::hstore;
count
-------
285
(1 row)

Time: 17.372 ms
```

Performance

- Search performance — contains @> operator
  - Hstore - seqscan, GiST, GIN GIN++

seqscan	GiST	GIN	GIN++	
	100s	400s		- create index
	64MB	815MB		
0.98s	0.3s	0.1s	0.017s	
	3x	10x	60x	

MongoDB 2.4.7

- Load data - ~8 min **SLOW !**

```
mongoimport --host localhost -c js --type json < delicious-rss-1250k
Mon Oct 28 19:16:47.025                         7400    2466/second
...
Mon Oct 28 19:24:38.030                         1250800 2638/second
Mon Oct 28 19:24:38.902 check 9 1252973
Mon Oct 28 19:24:38.902 imported 1252973 objects
```

- Search - ~ 1s (seqscan) **THE SAME**

```
db.js.find({tags: {$elemMatch:{ term: "NYC"}}}).count()
285
-- 980 ms
```

MongoDB 2.4.7

- Search — 1ms (index) **WOW !**

```
db.js.ensureIndex( {"tags.term" : 1} )
db.js.find({tags: {$elemMatch:{ term: "NYC"}}}).explain()
{
        "cursor" : "BtreeCursor tags.term_1",
        "isMultiKey" : true,
        "n" : 285,
        "nscannedObjects" : 285,
        "nscanned" : 285,
        "nscannedObjectsAllPlans" : 285,
  ..............................
        "millis" : 1,
        "indexBounds" : {
                "tags.term" : [
                        [
                                "NYC",
                                "NYC"
                        ]
                ]
```

GIN hstore hash index

- Idea: index hash(full paths to elements and values)

  {a=>{b=>{c=>1}}, d=>{1,2,3}}
  path-keys: a.b.c.1, d..1, d..2,d..3

  GIN: {hash(path-key)}

GIN hstore hash index

```
=# create index gin_hs_hash_idx on hs using gin(h gin_hstore_hash_ops);
CREATE INDEX
Time: 68777.418 ms
=# explain analyze select count(*) from hs
where h::hstore @> 'tags=>{{term=>NYC}}'::hstore;
                                                              QUERY PLAN
------------------------------------------------------------------------
  Aggregate  (cost=4733.21..4733.22 rows=1 width=0)
        (actual time=0.647..0.647 rows=1 loops=1)
   ->  Bitmap Heap Scan on hs  (cost=33.71..4730.08 rows=1253 width=0)
        (actual time=0.128..0.614 rows=285 loops=1)
         Recheck Cond: (h @> '"tags"=>{{"term"=>"NYC"}}'::hstore)
         ->  Bitmap Index Scan on gin_hs_hash_idx
                (cost=0.00..33.40 rows=1253 width=0)
                 (actual time=0.085..0.085 rows=285 loops=1)
               Index Cond: (h @> '"tags"=>{{"term"=>"NYC"}}'::hstore)
 Total runtime: 0.672 ms WOW++ !
(6 rows)
```

Performance

- Search performance — contains @> operator
  - MongoDB uses very «narrow» index
  - Hstore's indexes are general

```
Hstore - seqscan, GiST,   GIN    GIN++   GINhash   MongoDB
                  64MB    815MB          349MB     100MB
         0.98s    0.3s    0.1s   0.017s   0.0007s 0.001s
                  3x      10x     60x     1400x    1000x
```

Availability

- Patch to master branch is available

http://www.sigaev.ru/misc/nested_hstore-0.36.patch.gz

Thanks !

Engine Yard™ Sponsored