Что PostgreSQL 12 нам готовит?

Олег Бартунов
Александр Коротков
Postgres Professional

5 февраля 2019, МГУ

Generalized expression syntax for partition bounds (committed)

```
CREATE TABLE part (ts timestamp)
PARTITION BY RANGE(ts);
CREATE TABLE part1
PARTITION OF part FOR VALUES
FROM ('2018-01-01')
TO (current_timestamp + '1 day');
```

Support for expressions in partition bounds!

Run-time partition pruning for MergeAppend (committed)

```
# explain analyze  select * from news
  where category = (select category from hot_category)
  order by ts limit 10;

 Limit  (cost=36.79..37.26 rows=10 width=12) (actual time=0.0
   InitPlan 1 (returns $0)
     ->  Seq Scan on hot_category  (cost=0.00..35.50 rows=255
   ->  Merge Append  (cost=1.29..46833.10 rows=1000000 width=
         Sort Key: news_cat1.ts
         ->  Index Scan using news_cat1_ts_idx on news_cat1
             (cost=0.42..11302.75 rows=333333 width=12)
             (actual time=0.016..0.021 rows=10 loops=1)
               Filter: (category = $0)
         ->  Index Scan using news_cat2_ts_idx on news_cat2
             (cost=0.42..11302.77 rows=333334 width=12)
             (never executed)
               Filter: (category = $0)
         ->  Index Scan using news_cat3_ts_idx on news_cat3
             (cost=0.42..11302.75 rows=333333 width=12)
             (never executed)
               Filter: (category = $0)
```

Reduce partition tuple routing overheads (committed)

Inserts into 10k partitions table:

original: 96 TPS

patched: 17729 TPS

non-partitioned: 19121 TPS

KNN для SP-GiST (committed)

```
SELECT *
FROM knn_test
ORDER BY p <-> point(:x,:y) LIMIT :n;
```

	GiST		SP-GiST	
n	time, ms	buffers	time, ms	buffers
10	0,12	14	0,07	18
100	0,27	110	0,2	118
1000	1,58	1231	1,51	1264

KNN для B-tree (in-progress)

```
SELECT * FROM events
ORDER BY date <-> '2000-01-01'::date ASC
LIMIT 100;
```

	KNN B-tree		btree_gist		union		seq scan	
k	time, ms	buffers	time, ms	buffers	time, ms	buffers	time, ms	buffers
1	0.041	4	0.079	4	0.060	8	41.1	1824
10	0.048	7	0.091	9	0.097	17	41.8	1824
100	0.107	47	0.192	52	0.342	104	42.3	1824
1000	0.735	573	0.913	650	2.970	1160	43.5	1824
10000	5.070	5622	6.240	6760	36.300	11031	54.1	1824
100000	49.600	51608	61.900	64194	295.100	94980	115.0	1824

PostgreSQL version in log (committed)

```
2019-02-02 09:23:11.711 MSK [59708] LOG:  starting
PostgreSQL 12devel on x86_64-apple-darwin17.7.0, compiled
by Apple LLVM version 10.0.0 (clang-1000.11.45.5), 64-bit
2019-02-02 09:23:11.715 MSK [59708] LOG:  listening on
IPv6 address "::1", port 5434
2019-02-02 09:23:11.715 MSK [59708] LOG:  listening on
IPv6 address "fe80::1%lo0", port 5434
2019-02-02 09:23:11.715 MSK [59708] LOG:  listening on
IPv4 address "127.0.0.1", port 5434
2019-02-02 09:23:11.716 MSK [59708] LOG:  listening on
Unix socket "/tmp/.s.PGSQL.5434"


.....................................................
```

Locking B-tree leafs immediately in exclusive mode (committed)

test	original, TPS	patched, TPS
unordered inserts	409 591	412 765
ordered inserts	252 796	314 541
duplicate inserts	44 811	202 325

Improve behavior of to_timestamp() / to_date() functions (committed)

Before

```
# select to_timestamp('2019-13-01', 'YYYYMMDD');
      to_timestamp
------------------------
 2018-11-03 00:00:00+03

# select to_timestamp('2019 -01-01', 'YYYY-MM-DD');
      to_timestamp
------------------------
 2018-11-01 00:00:00+03
(1 row)
```

After

```
# select to_timestamp('2019-13-01', 'YYYYMMDD');
ERROR:  date/time field value out of range: "2019-13-01"

# select to_timestamp('2019 -01-01', 'YYYY-MM-DD');
      to_timestamp
------------------------
 2019-01-01 00:00:00+03
(1 row)
```

Now well documented!

Function to promote standby servers (committed)

How to promote a standby?

- Trigger file
- pg_ctl promote
- **SELECT pg_promote();**

Step towards managing cluster in pure SQL!

Speedup of relation deletes during recovery (committed)

Relation delete or truncate:

- Causes sequential scan of shared_buffers
- Slow with large shared_buffers
- Especially bad for standby, because of single-process recovery

Now, instead of

```
DELETE tab1; DELETE tab2; … DELETE tabN;
```

it's better to do

```
BEGIN;
DELETE tab1; DELETE tab2; … DELETE tabN;
COMMIT;
```

Single pass over shared_buffers instead of N.
Less replication lag!

Use the built-in float datatypes to implement geometric types (committed)

- Check for underflow, overflow and division by zero
- Consider NaN values to be equal
- Return NULL when the distance is NaN for all closest point operators
- Favour not-NaN over NaN where it makes sense

Before

```
# select point('NaN', 'NaN') ~= point('NaN', 'NaN');
 ?column?
----------
 f
```

After

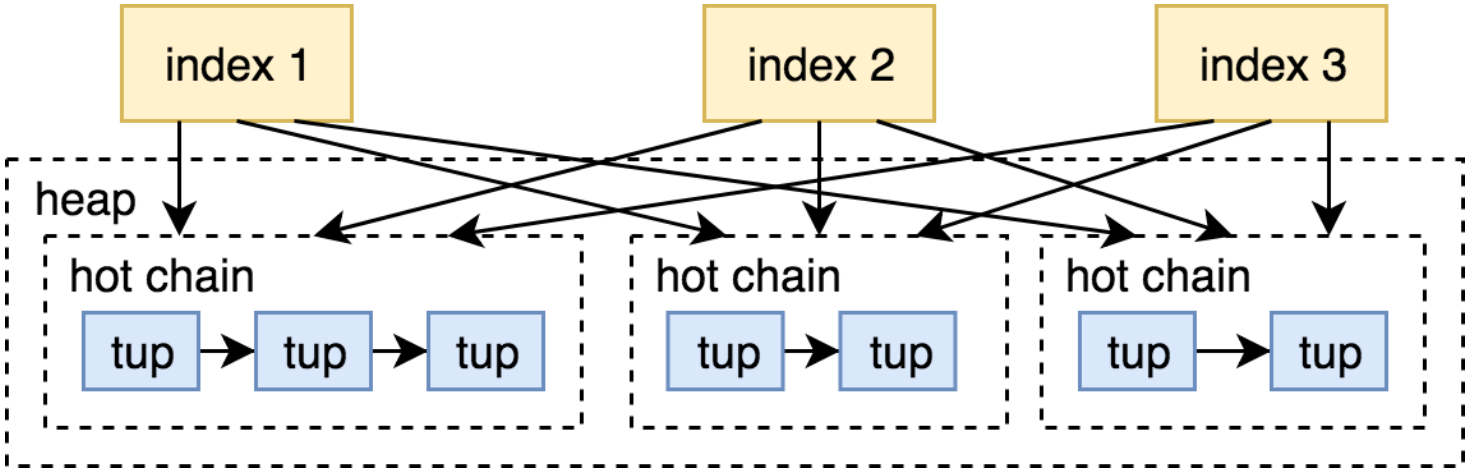
```
# select point('NaN', 'NaN') ~= point('NaN', 'NaN');
 ?column?
----------
 t
```

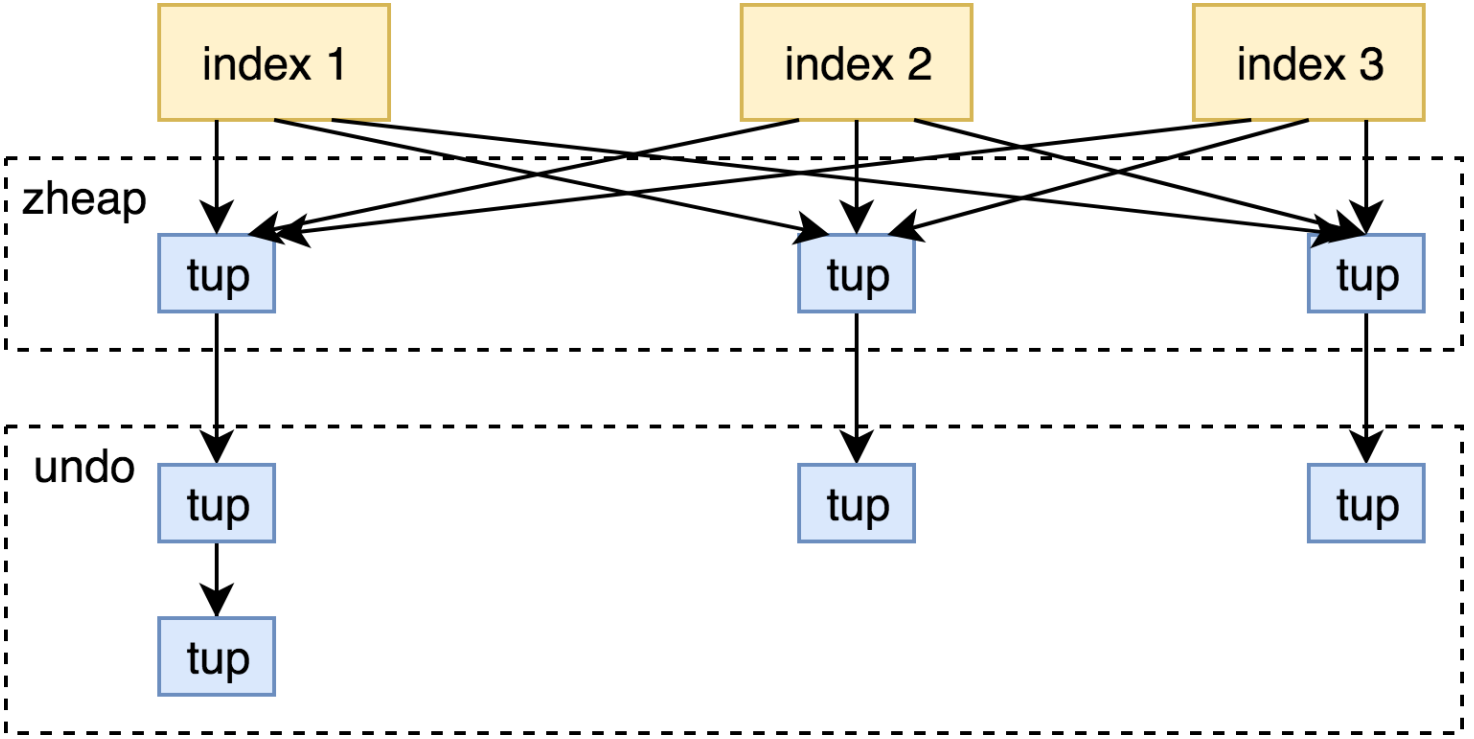
Add log_statement_sample_rate parameter (committed)

- Logging all the statements consumes much of resources
- Logging only long statements may distort your picture
- Sample logging is the solution!

```
log_statement_sample_rate = 1 ; log every statement
log_statement_sample_rate = 0 ; log no statements
log_statement_sample_rate = 0.5 ; log half of statement
log_statement_sample_rate = 0.1 ; log one tenth of
                                ; statement
```

zheap (in-progress) (1/2)





zheap (in-progress) (2/2)

- Undo chains instead of HOT chains.
- HOT chains are limited by free page space, undo chains are not.
- Less heap bloat.
- Less index bloat, because HOT-like updates are more frequent (no page overflow).
- Index update situation isn't any better yet. It has been told that there would be delete-marked indexes, but not yet shown.

Pluggable storage (in-progress)

- Support for INSERT/UPDATE/DELETE, triggers etc.
- Support for custom maintenance (own vacuum).
- Support for table rewrite.
- Support for custom tuple format.
- Support for custom tuple storage.
- Index-heap relationship must be the same. Only HOT-like update OR insertion to EVERY index.
- Row must be identified by 6-byte TID.
- System catalog must be heap.

MERGE SQL statement (in-progress) (1/2)

```
MERGE INTO target AS t
USING source AS s
ON t.tid = s.sid
WHEN MATCHED AND t.balance > s.delta THEN
  UPDATE SET balance = t.balance - s.delta
WHEN MATCHED THEN
  DELETE
WHEN NOT MATCHED AND s.delta > 0 THEN
  INSERT VALUES (s.sid, s.delta)
WHEN NOT MATCHED THEN
  DO NOTHING;
```

MERGE SQL statement (in-progress) (2/2)

- Conforming SQL Standard (unlike INSERT ON CONFLICT)
- Doesn't prevent unique constraint violation in current version (unlike INSERT ON CONFLICT)
- A lot of criticism to current patch including:
  - Parser does normalization, which it shouldn't do
  - Imperative optimizations
  - Incapsulation violations
  - …
- Doesn't seem feasible for PG 12 :(

Connection pooler (in-progress)

- Builtin connection pooler is a very big feature, which is impossible to fit at single release cycle.
- We decided to extract connection proxy as separate patch and submit it for PG 12. It looks similar to external pooler (like pgbouncer), but has number of differences.
  - It is embedded and requires no extra steps for installation and configurations.
  - It is not single threaded (no bottleneck)
  - It supports all clients (if client needs session semantic, then it will be implicitly given dedicated backend)

SQL/JSON (in-progress)

The SQL/JSON **construction** functions:

- JSON_OBJECT - serialization of an JSON object.
  - json[b]_build_object()
- JSON_ARRAY - serialization of an JSON array.
  - json[b]_build_array()
- JSON_ARRAYAGG - serialization of an JSON object from aggregation of SQL data
  - json[b]_agg()
- JSON_OBJECTAGG - serialization of an JSON array from aggregation of SQL data
  - json[b]_object_agg()

SQL/JSON (in-progress)

Jsonpath provides an ability to operate (in standard specified way) with json structure at SQL-language level

- Dot notation – $.a.b.c
  $ - the current context element
- Array - [*]
- Filter ? - $.a.b.c ? (@.x > 10)
  @ - current context in filter expression
- Methods - $.a.b.c.x.type()
  type(), size(), double(), ceiling(), floor(), abs(), datetime(), keyvalue()

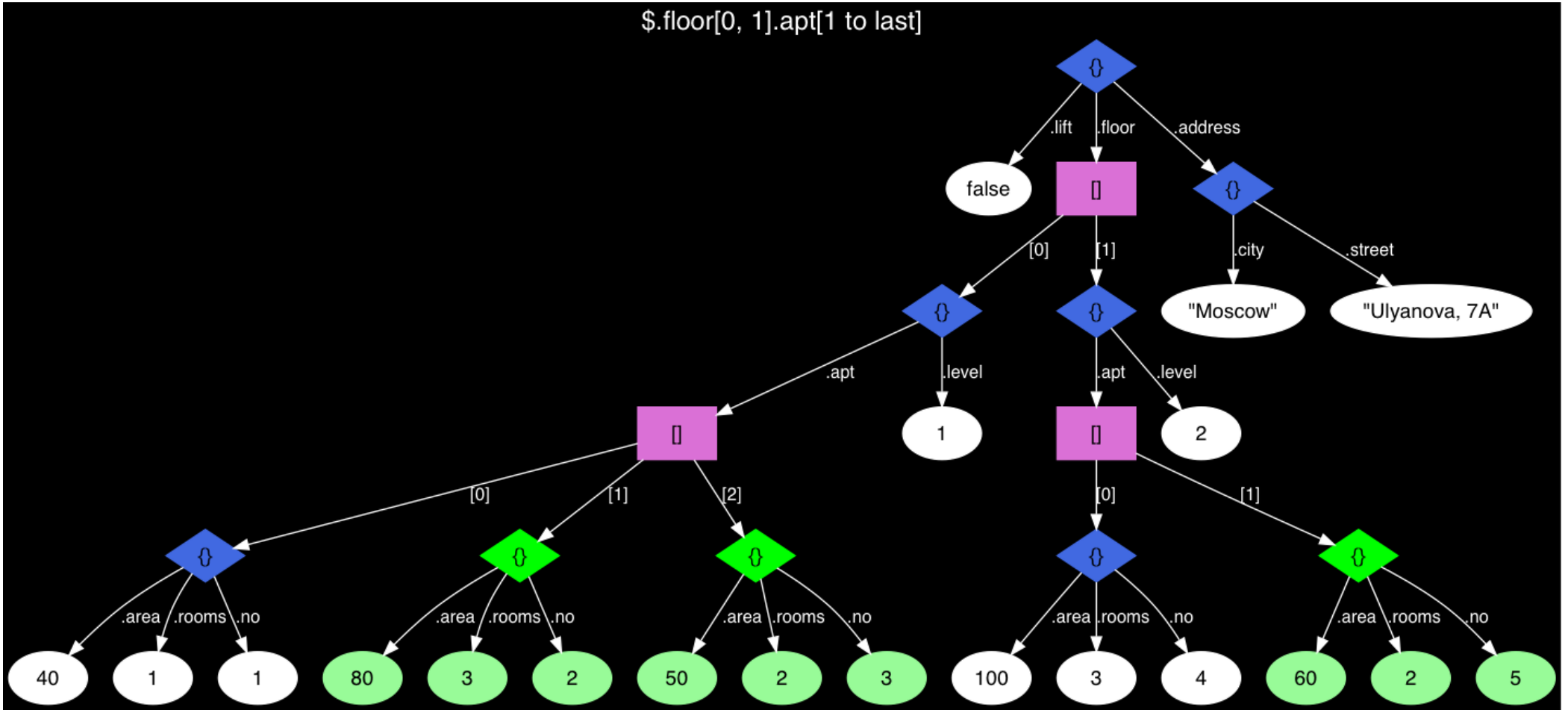
```
'$.floor[*].apt[*] ? (@.area > 40 && @.area < 90)'
```

Why JSON path is a type ?

- Standard permits only string literals in JSON path specification.
- WHY a data type ?
- To accelerate JSON path queries using existing indexes for jsonb we need boolean operators for json[b] and jsonpath.
- Implementation as a type is much easier than integration of JSON path processing with executor (complication of grammar and executor).
- In simple cases, expressions with operators can be more concise than with SQL/JSON functions.
- It is Postgres-way to use operators with custom query types (tsquery for FTS, lquery for ltree, jsquery for jsonb,...)

```
{
        "address": {
                "city": "Moscow",
                "street": "Ulyanova, 7A"
        },
        "lift": false,
        "floor": [
                {
                        "level": 1,
                        "apt": [
                                {"no": 1, "area": 40, "rooms": 1},
                                {"no": 2, "area": 80, "rooms": 3},
                                {"no": 3, "area": 50, "rooms": 2}
                        ]
                },
                {
                        "level": 2,
                        "apt": [
                                {"no": 4, "area": 100, "rooms": 3},
                                {"no": 5, "area": 60, "rooms": 2}
                        ]
                }
        ]
}
```

$.floor[0,1].apt[1 to last]

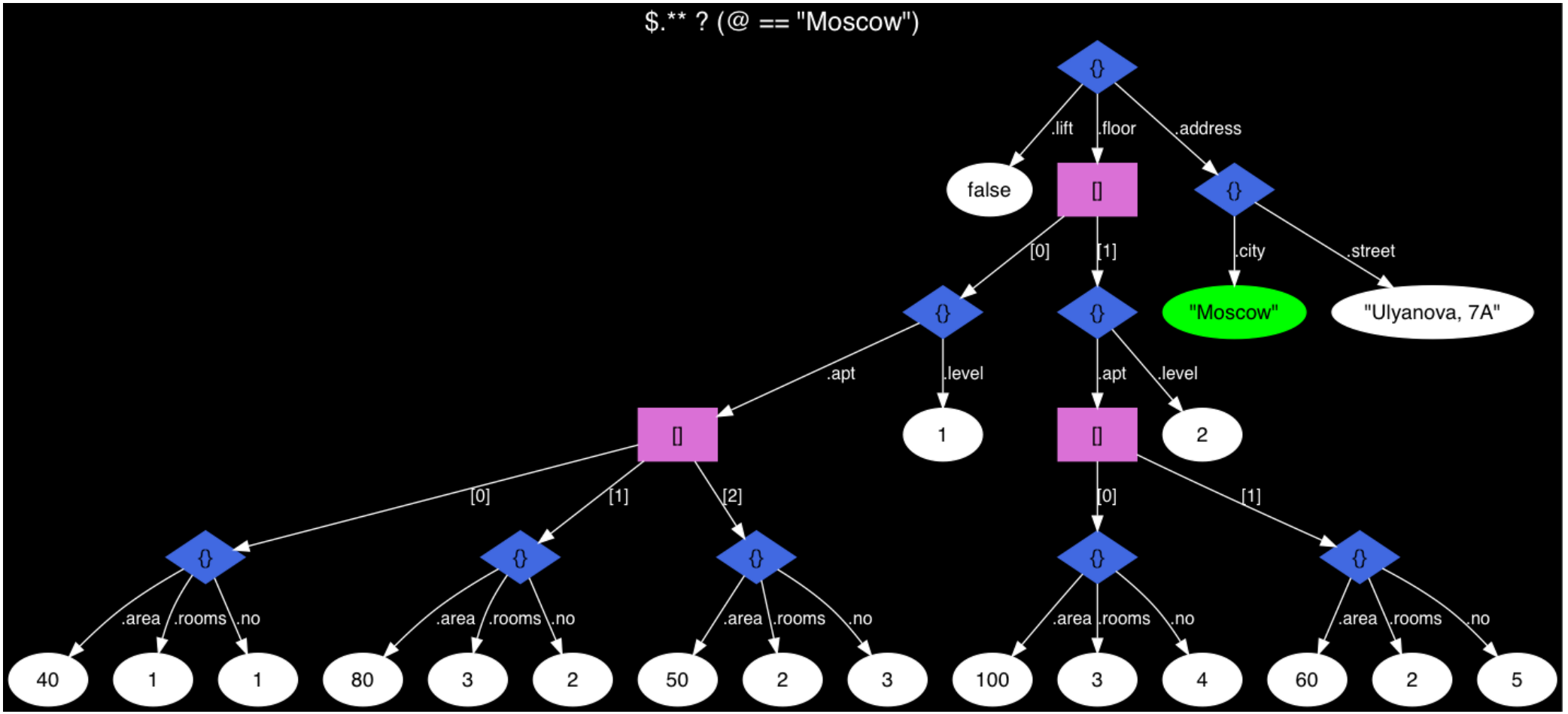


$.floor[0, 1].apt[1 to last]

```
SELECT JSON_QUERY(js, '$.floor[0, 1].apt[1 to last]' WITH WRAPPER)
FROM house;


SELECT jsonb_agg(apt)
FROM (SELECT apt->generate_series(1, jsonb_array_length(apt) - 1)
FROM (SELECT js->'floor'->unnest(array[0, 1])->'apt'
FROM house) apts(apt)) apts(apt);
```

Extension: wildcard search



$.** ? (@ == "Moscow")

```
SELECT JSON_EXISTS(js, '$.** ? (@ == "Moscow")') FROM house;

WITH RECURSIVE t(value) AS
(SELECT * FROM house
 UNION ALL
    ( SELECT
        COALESCE(kv.value, e.value) AS value
      FROM
        t
      LEFT JOIN LATERAL jsonb_each(CASE WHEN jsonb_typeof(t.value) = 'object'
THEN t.value ELSE NULL END) kv ON true
      LEFT JOIN LATERAL jsonb_array_elements(CASE WHEN jsonb_typeof(t.value) = 'array'
THEN t.value ELSE NULL END) e ON true
      WHERE
        kv.value IS NOT NULL OR e.value IS NOT NULL)
)
SELECT EXISTS (SELECT 1 FROM t WHERE value = '"Moscow"');
```

SQL/JSON (in-progress)

The SQL/JSON **retrieval** functions:

- JSON_VALUE - Extract an SQL value of a predefined type from a JSON value.
- JSON_QUERY - Extract a JSON text from a JSON text using an SQL/JSON path expression.
- JSON_TABLE - Query a JSON text and present it as a relational table.
- IS [NOT] JSON - test whether a string value is a JSON text.
- JSON_EXISTS - test whether a JSON path expression returns any SQL/JSON items

SQL/JSON examples: JSON_VALUE

```
SELECT x, JSON_VALUE(jsonb '{"a": 1, "b": 2}','$.* ? (@ > $x)'
                     PASSING x AS x
                     RETURNING int
                     DEFAULT -1 ON EMPTY
                     DEFAULT -2 ON ERROR
                    ) y
FROM
generate_series(0, 2) x;
 x | y
---+----
 0 | -2
 1 |  2
 2 | -1
(3 rows)
```

SQL/JSON examples: JSON_TABLE

```
SELECT
  apt.*
FROM
  house,
  JSON_TABLE(
    js, '$.floor[*]' COLUMNS (
      level int,
      NESTED PATH '$.apt[*]' COLUMNS (
        no int,
        area int,
        num_rooms int PATH '$.rooms'
      )
    )
  ) apt;
```

```
 level | no | area | num_rooms
-------+----+------+-----------
     1 |  1 |   40 |         1
     1 |  2 |   80 |         3
     1 |  3 |   50 |         2
     2 |  4 |  100 |         3
     2 |  5 |   60 |         2
(5 rows)
```

SQL/JSON standard conformance

SQL/JSON feature	PostgreSQL 12	Oracle 18c	MySQL 8.0.4	MS SQL Server 2017
JSON_OBJECT: 7	6	5	1	0
JSON_ARRAY: 4	4	4	1	0
JSON_OBJECTAG JSON_ARRAYAGG	2	2	2	0
RETURNING: 3	3	2	2	0
FORMAT JSON: 1	1	1	0	0
IS JSON: 2	2	1	0	0
JSON_EXISTS: 5	5	5	0	1
JSON_QUERY: 3	3	3	0	1

SQL/JSON standard conformance

SQL/JSON feature	PostgreSQL 12	Oracle 18c	MySQL 8.0.4	MS SQL Server 2017
JSON PATH: 15	15	11	5	2
TOTAL: 42	**41/42**	**34/42**	**11/42**	**4/42**

PostgreSQL 12 could have **the best implementation** of SQL/JSON standard !

SQL/JSON (in-progress)

- Very **BIG** patchset. First version was posted at February 2017.
- It's hard to commit all the patches to single PostgreSQL release. Let's try do at least jsonpath for PG 12.
- Error handling appears to be pain. PG_TRY()/PG_CATCH() are unacceptable by community even if safe for now. Have to reimplement it in invasive way...

SQL/JSON (доп.материалы)

- Презентация по SQL/JSON
  http://www.sai.msu.su/~megera/postgres/talks/sqljson-china-2018.pdf
- Введение в SQL/JSON
  https://github.com/obartunov/sqljsondoc/blob/master/README.jsonpath.md
- Посты про SQL/JSON
  https://obartunov.livejournal.com/tag/sqljson

Postgres PROFESSIONAL