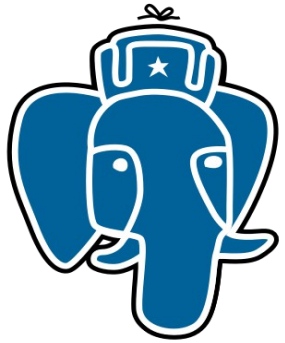


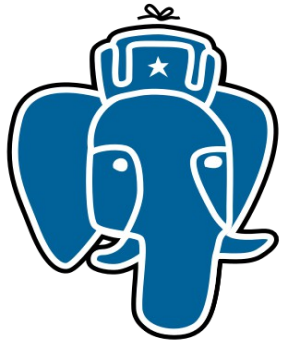
# Расширяемость PostgreSQL

Олег Бартунов,
ГАИШ МГУ, PostgreSQL Major Contributor

# Олег Бартунов::Teodor Sigaev

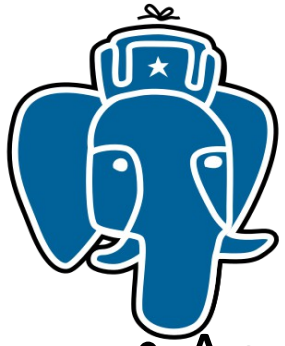
- Locale support
- Extensions:
  - intarray
  - pg_trgm
  - ltree
  - hstore, hstore v2.0 → jsonb
  - plantuner
- Full Text Search (FTS)
- Extendability (indexing)
  - GiST, GIN, SP-GiST


https://www.facebook.com/oleg.bartunov
obartunov@gmail.com



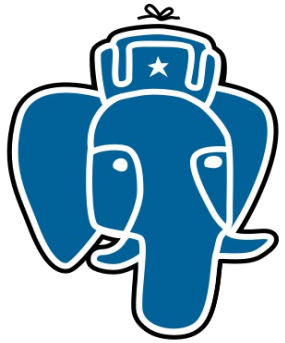
# IT-цикл проекта

- Проектирование архитектуры
- Разработка
- Поддержка
- Развитие
  - Увеличение нагрузки, новая функциональность

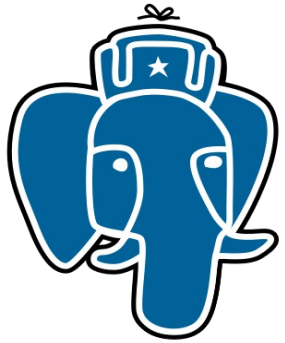


# IT-цикл проекта

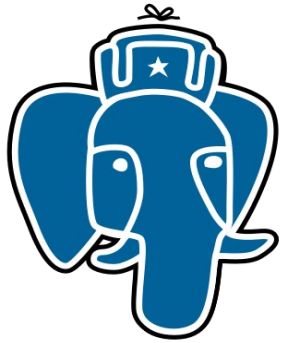
- Архитектура системы должна быть расширяемой
- Расширяемость компонентов системы
- Костыли — средство расширяемости



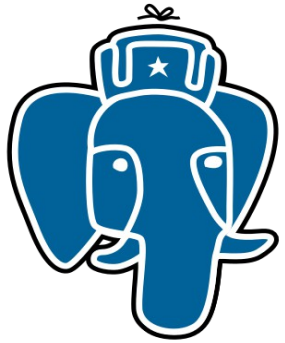
- Костыль — средство добавления недостающей функциональности или исправления серьёзных дыр без должного редизайна системы
- Систем без «костылей» не бывает



# Костыли имеют тенденцию накапливаться

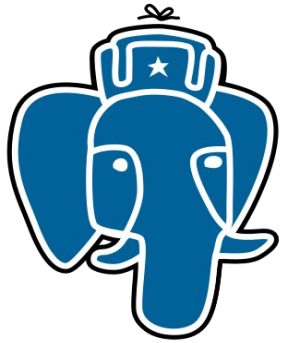


# Система все-таки работает ...



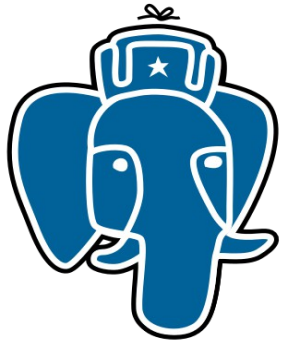
# Поменять СУБД очень тяжело !

- Frontend-cache-backend-application-cache-файловое хранилище-база данных
- Смена СУБД — мучительный процесс, никакой автомагии
- Выбор СУБД на стадии разработки — важнейшая тема ! 99.99% проектов не требует разработки своей СУБД.



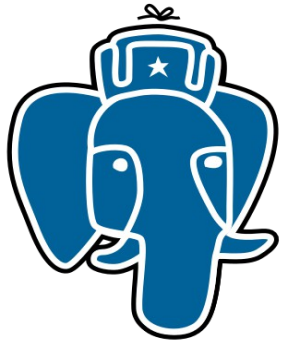
# Выбор  СУБД

- Архитектор — функциональность, производительность, доступность
- Хакер-дба — удобство, привычка, религия
- Бизнес-маркетинг —  сторонние факторы



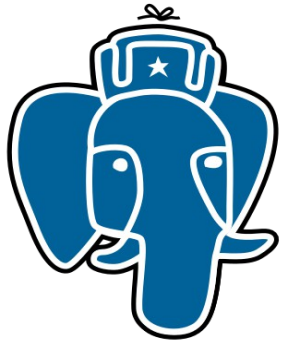
# Выбор СУБД

- **Расширяемость** СУБД — важнейший фактор вашего развития и независимости !
- Расширяемость:
  - Новая функциональность (типы данных, операции)
  - Своими руками
  - Без остановки сервера
  - Без ущерба производительности и надежности



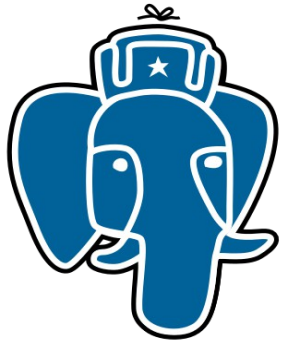
# PostgreSQL

- Абсолютно доступная (BSD, нет компании-владельца)
- Хороший научный бэкграунд и история
- Много успешных коммерческих форков (Greenplum, AsterData, JustOneDB, Hadapt...)
- Отличная функциональность и производительность
- Хорошее соответствие стандартам ISO/ANSI SQL 92,99,2003
- Большое и дружественное сообщество разработчиков и пользователей
- Российские разработчики (расширяемость)



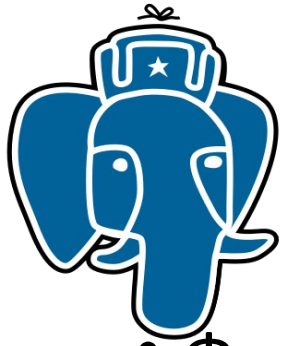
# PostgreSQL

- Skype  - шкалируется до миллиарда польз.
- Hi5.com — 60 млн. пользователей, #8 Alexa traffic rank
- MyYearBook.com — 18,000 req/sec, 300 Gb database
- NASA — обработка спутниковых данных (MODIS)
- Instagram — x100 млн картинок
- Sony (Free Realms) — 10 млн игроков
- Heroku , EngineYard, Saleforce ?



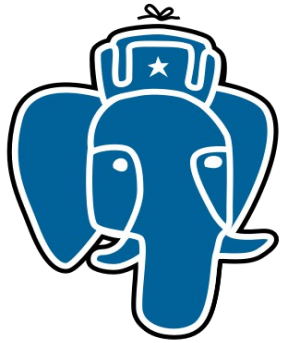
# PostgreSQL

- Рамблер
- 1С:Предприятие
- MirTesen, MoiKrug.ru (Yandex)
- Avito.ru — 2000 req/sec
- IRR.ru ( «Из рук в руки»)
- rabota.ru, price.ru, РБК, МастерХост
- Военные — версия 7.X входит в МСВС, 9.0 - Заря
- Национальная СУБД ?



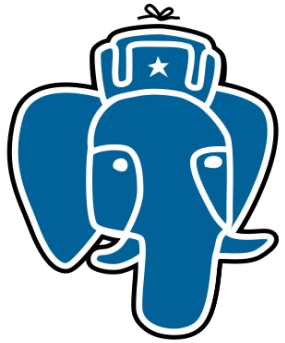
# Расширяемость PostgreSQL

- Функции, операторы, типы данных
- Языки (sql,pl/pgsql,pl/perl,pl/python, pl/tcl, pl/R, pl/java …,pl/v8). Pl/psql от EnterpriseDB!
- Индексный доступ (Btree, Hash, GiST, GIN, SP-GiST)
- Внешние таблицы (oracle, mysql, informix, couchdb, redis, mongodb, twitter, www ...)



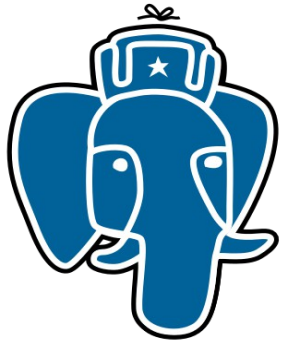
# Индексные методы доступа

- Индекс — это поисковое дерево, в листьях  которого содержатся указатели на записи в таблице
- Индекс не содержит информации о видимости записи (MVCC !)
- Индексы только ускоряют выполнение запроса (операторы,операнды)
- Результаты выборки с использованием индекса должны совпадать с последовательным сканом  и фильтрацией
- Индексы могут быть **partial** (where price > 0.0), **functional** (to_tsvector(text)), **multicolumn** (timestamp, tsvector)
- Индексные методы доступа (pg_catalog.pg_am)  - btree,hash, gist, gin, spgist



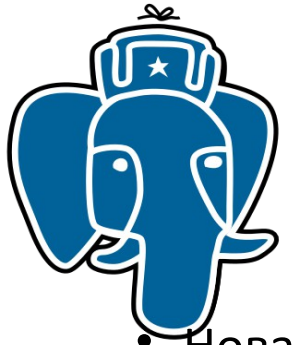
# Индексные методы доступа

- Btree — операции сравнения
- Hash — равенство
- GiST, GIN, SP-GiST — произвольные операции (похожесть, пересечение множеств, включение,....)



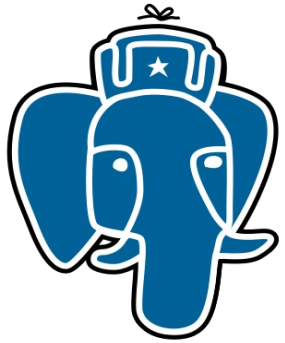
# Индексные методы доступа

- +Методы навигации по дереву
- +Добавление и вставка в дерево
- +Конкурентность, восстанавливаемость
- ?Интерфейсы для описания работы с данными (поиск и вставка)



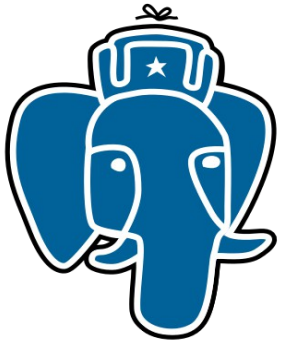
# Расширяемость PostgreSQL

- Новая функциональность добавляется **без остановки сервера**
- Не требуются «магические» знания core-девелоперов
- Новые типы данных - «first class citizens». Все свойства встроенных типов данных — конкурентность, восстанавливаемость, индексный доступ, версионность
- Примеры: ltree — иерархические данные, hstore — слабо-структурированные данные, pg_trgm — очепятки, postgis — GIS, полнотекстовый поиск, …..

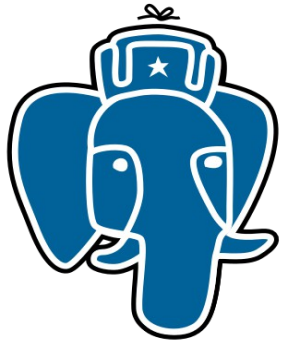


# Расширяемость PostgreSQL

**Как это работает ?**

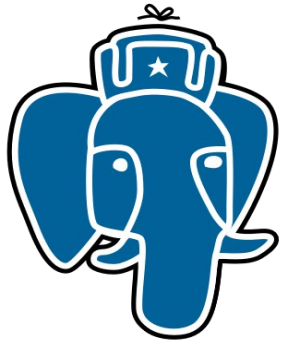


# Разоблачение магии



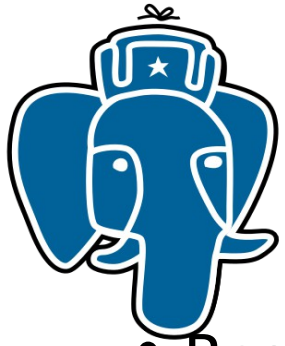
# План

1. Системный каталог
2. Бинарное представление типа на примере HSTORE и JSON
3. Функции
4. Операторы



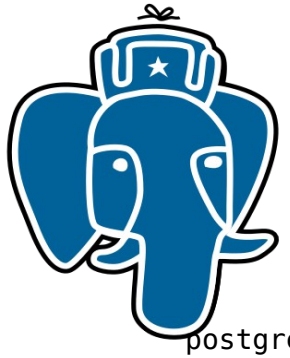
# План

5. Cast’ы
6. Оценки селективности
7. Индексы
8. Extension'ы



# Системный каталог

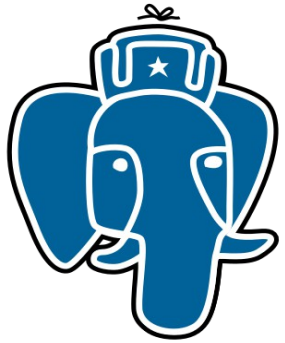
- Все метаданные объектов базы (таблицы, типы данных, индексы, операторы, функции, расширения...) хранятся в системном каталоге (pg_catalog.*)
- 2 интерфейса - SQL (psql -E), функциональный
- Syscache — кэш каталога, используется executor, planner, optimizer
- Bootstrap — специальный режим initdb, создает системный каталог из *.bki файлов



# Системный каталог

```
postgres=# \dt pg_catalog.*
                  List of relations
   Schema   |          Name           | Type  |  Owner
------------+-------------------------+-------+----------
 pg_catalog | pg_aggregate            | table | postgres
 pg_catalog | pg_am                   | table | postgres
 pg_catalog | pg_amop                 | table | postgres
 pg_catalog | pg_amproc               | table | postgres
 pg_catalog | pg_attrdef              | table | postgres
 pg_catalog | pg_attribute            | table | postgres
 pg_catalog | pg_auth_members         | table | postgres
 pg_catalog | pg_authid               | table | postgres
 pg_catalog | pg_cast                 | table | postgres
 pg_catalog | pg_class                | table | postgres
 pg_catalog | pg_collation            | table | postgres
 pg_catalog | pg_constraint           | table | postgres
 pg_catalog | pg_conversion           | table | postgres
 pg_catalog | pg_database             | table | postgres
 pg_catalog | pg_db_role_setting      | table | postgres
 ..........................................................
 pg_catalog | pg_user_mapping         | table | postgres
(51 rows)
```

```
postgres=# \d pg_cast
    Table "pg_catalog.pg_cast"
   Column    |  Type  | Modifiers
-------------+--------+-----------
 castsource  | oid    | not null
 casttarget  | oid    | not null
 castfunc    | oid    | not null
 castcontext | "char" | not null
 castmethod  | "char" | not null
Indexes:
    "pg_cast_oid_index" UNIQUE, btree (oid)
    "pg_cast_source_target_index" UNIQUE,
btree (castsource, casttarget)
```

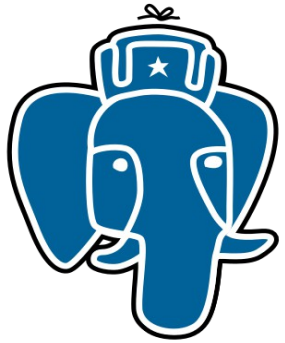


# Системный каталог

позволяет добавлять новые объекты:

- типы данных
- операторы
- классы операторов
- …

онлайн



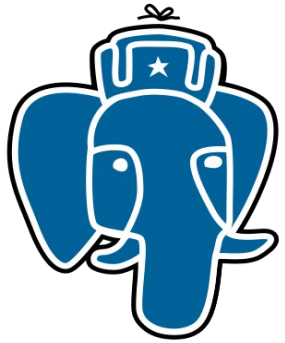
# JSON

- 9.2: native -> json

  array_to_json(), row_to_json()

- 9.3: json -> native

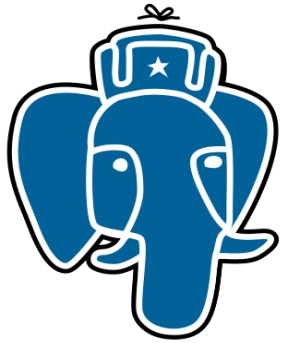
  ->, ->>, #>, #>>, etc



# Бинарное представление JSON

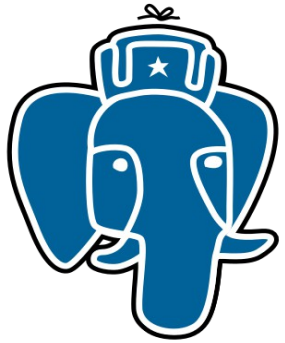
```
SELECT '{"a":1,"b":2}'::json;

0000: 7B 22 61 22 | 3A 31 2C 22 {"a":1,"
0008: 62 22 3A 32 | 7D            b":2}
```

Бинарное представление = текстовое!



# HSTORE

- Эффективное бинарное представление
- Вложенность, поддержка массивов (WIP)
- JSON – как представление: hstore_to_json()

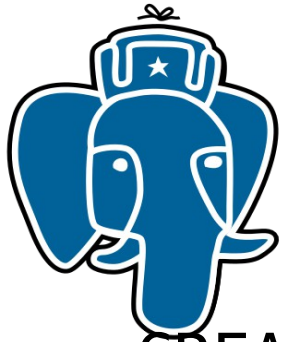


# Бинарное представление HSTORE

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

```
0000: 02 00 00 80 | 01 00 00 80 ☻  A☺  A
0008: 02 00 00 00 | 03 00 00 00 ☻   ♥
0010: 04 00 00 00 | 61 31 62 32 ♦   a1b2
```

Эффективное бинарное представление!



# Сравнение

```
CREATE TABLE hstore_test AS (SELECT
'a=>1, b=>2, c=>3, d=>4, e=>5'::hstore AS v
FROM generate_series(1,1000000));

CREATE TABLE json_test AS (SELECT
'{"a":1, "b":2, "c":3, "d":4, "e":5}'::json AS v FROM
generate_series(1,1000000));
```

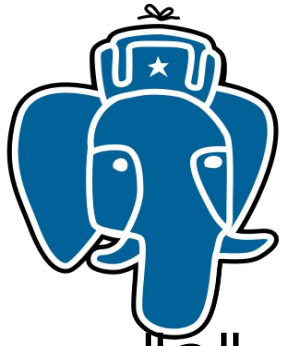
# Сравнение

```
SELECT sum((v->'a')::text::int)
FROM json_test;
```

1291,060 ms

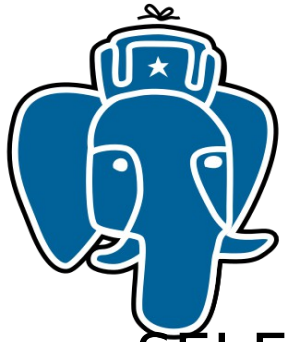
```
SELECT sum((v->'a')::int)
FROM hstore_test;
```

303,267 ms



# Сравнение

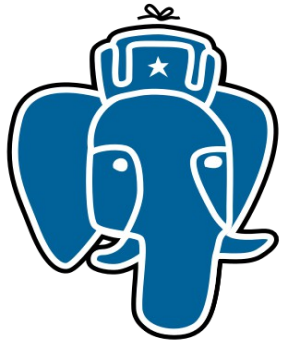
```
"a"=>{"9840", "8818", "1613", "2793", "6633", "1356",
"3578", "6939", "8876", "8848", "8150", "9889", "5905",
"1023", "6154", "6708", "1430", "9521", "4566", "6001",
"7807", "9140"}
```



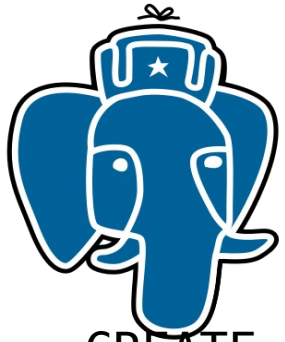
# Сравнение

```
SELECT SUM((h%>'a'->0)::int)
FROM aa;
```
457.706 ms

```
SELECT SUM((j->'a'->>0)::int)
FROM jj;
```
7462.802 ms



# Как это работает?



# CREATE TYPE

```
CREATE TYPE hstore (
    INTERNALLENGTH = -1,
    INPUT = hstore_in,
    OUTPUT = hstore_out,
    RECEIVE = hstore_recv,
    SEND = hstore_send,
    STORAGE = extended
);
```

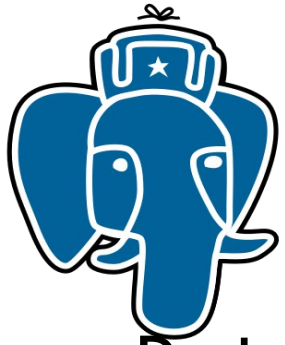
```
          pg_type
---------------+------------
typname        | hstore
typnamespace   | 2200
typowner       | 16384
typlen         | -1
typbyval       | f
typinput       | hstore_in
typoutput      | hstore_out
typreceive     | hstore_recv
typsend        | hstore_send
typstorage     | x
......
```

(extended)

# hstore_in

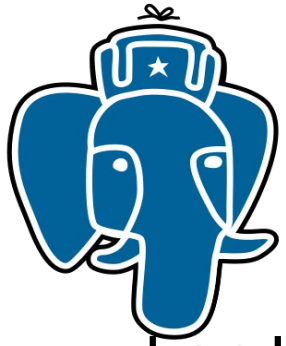
```
CREATE FUNCTION
hstore_in(cstring)
RETURNS hstore
AS '$libdir/hstore',
'hstore_in'
LANGUAGE C STRICT IMMUTABLE;
```

```
                pg_proc
----------------+--------------
proname         | hstore_in
pronamespace    | 2200
proowner        | 16384
prolang         | 13
procost         | 1
prorows         | 0
provariadic     | 0
prorettype      | 16385
proargtypes     | 2275
prosrc          | hstore_in
probin          | $libdir/hstore(hstore)
......                          (cstring)
```



# Type input

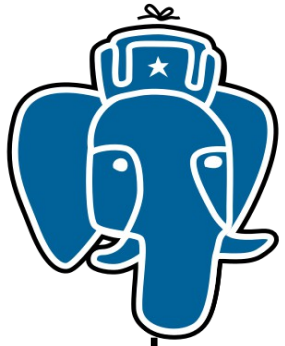
```
Datum
hstore_in(PG_FUNCTION_ARGS)
{
    HSParser     state;
    int4         buflen;
    HStore      *out;
......
```



# Type input

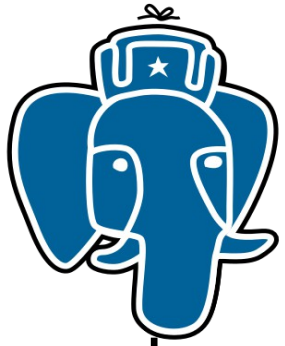
```
test=# SELECT 'a => 1, b => 2'::hstore;
 hstore
----------------------
 "a"=>"1", "b"=>"2"
```

Как это работает?



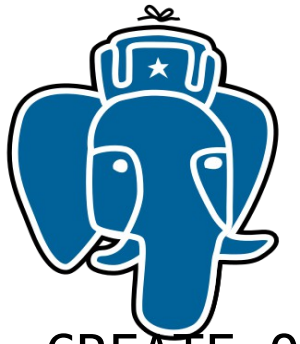
# Type input

```
test=# SELECT typinput FROM pg_type
       WHERE typname = 'hstore';
 typinput
-----------
 hstore_in
```



# Type input

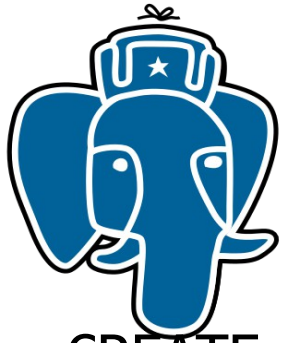
```
test=# SELECT prolang, prosrc, probin
       FROM pg_proc
       WHERE proname = 'hstore_in' AND
             Proargtypes = '2275';

                                    (cstring)
 prolang |  prosrc   |     probin
---------+-----------+-----------------
      13 | hstore_in | $libdir/hstore
(1 row)
```



# CREATE OPERATOR

```
CREATE OPERATOR -> (
    LEFTARG = hstore,
    RIGHTARG = text,
    PROCEDURE = fetchval
);
```

```
                pg_operator
-------------+------------
oprname      | ->
oprnamespace | 2200
oprowner     | 16384
oprkind      | b
oprleft      | 16385
oprright     | 25
oprresult    | 25           (hstore)
oprcom       | 0            (text)
oprnegate    | 0
oprcode      | fetchval
......
```

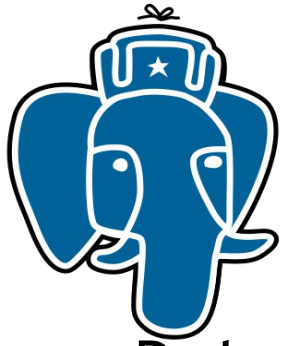


# CREATE FUNCTION

```
CREATE FUNCTION
fetchval(hstore,text)
RETURNS text
AS '$libdir/hstore',
'hstore_fetchval'
LANGUAGE C STRICT IMMUTABLE;
```

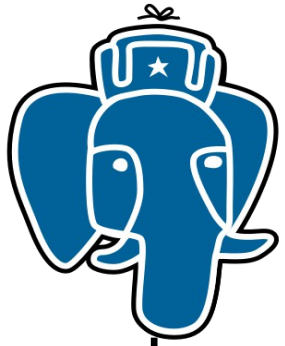
pg_proc

```
---------------+---------------
proname        | fetchval
pronamespace   | 2200
proowner       | 16384
prolang        | 13
procost        | 1
prosrc         | hstore_fetchval
probin         | $libdir/hstore
proconfig      |
proacl         |
......
```



# Operator

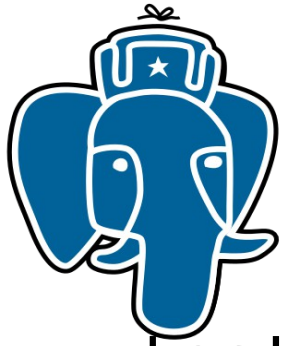
```
Datum
hstore_fetchval(PG_FUNCTION_ARGS)
{
    HStore  *hs = PG_GETARG_HS(0);
    text    *key = PG_GETARG_TEXT_PP(1);
......
```



# Operator

```
test=# SELECT ('a => 1, b => 2'::hstore)
        ->'a';
 ?column?
----------
 1
(1 row)
```

Как это работает?

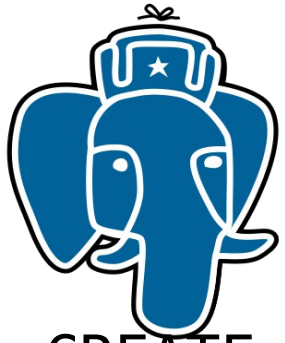


# Operator

```
test # SELECT oprcode FROM pg_operator WHERE oprname =
'->' AND oprleft = 16385 AND oprright = 25 ;
 oprcode
----------
 fetchval
(1 row)
```

(hstore)

(text)



# CREATE CAST

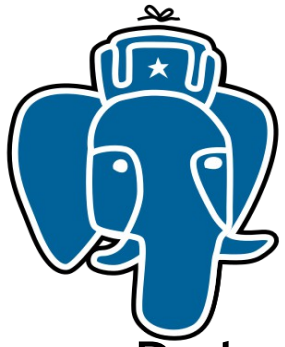
```
CREATE CAST
(hstore AS json)
WITH FUNCTION
hstore_to_json(hstore);
```

```
          pg_cast
------------+------
castsource  | 16385
casttarget  | 114
castfunc    | 16425
castcontext | e
castmethod  | f
```

(hstore)

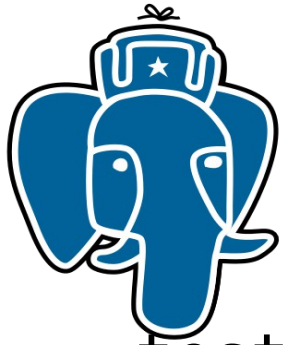
(json)

(hstore_to_json)



# Cast

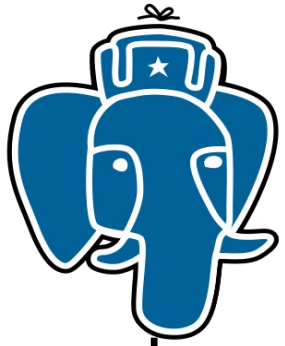
```
Datum
hstore_to_json(PG_FUNCTION_ARGS)
{
    HStore          *in = PG_GETARG_HS(0);
    int               buflen,
                      i;
    int               count = HS_COUNT(in);
......
```



# Cast

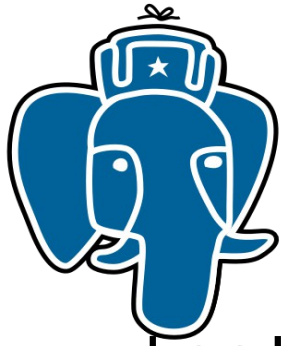
```
test=# SELECT 'a=>1, b=>2'::hstore::json;
          json
-----------------------
 {"a": "1", "b": "2"}
```

Как это работает?



# Cast

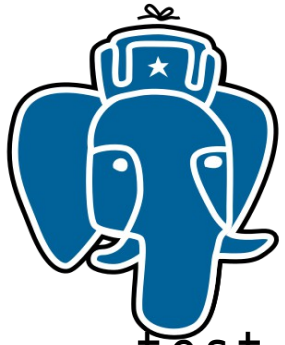
```
test=# SELECT castfunc
       FROM pg_cast
       WHERE castsource = 16385
       AND casttarget = 114;                    (hstore)
 castfunc                                 (json)
----------
    16425
(1 row)
```



# Cast

```
test=# SELECT prolang, prosrc, probin
       FROM pg_proc
       WHERE oid = 16425;
 prolang |     prosrc      |     probin
---------+------------------+-----------------
      13 | hstore_to_json | $libdir/hstore
(1 row)
```

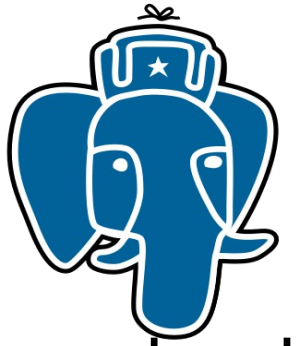
(castfunc)



# Оценки селективности

```
test= # EXPLAIN SELECT * FROM hstore_test
        WHERE v @> 'a => 1'::hstore;

                            QUERY PLAN
-------------------------------------------------------------------
Seq Scan on hstore_test  (cost=0.00..22810.00 rows=1000 width=55)
   Filter: (v @> '"a"=>"1"'::hstore)
```

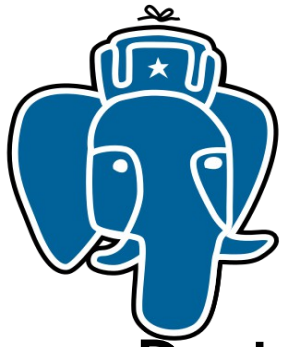
Откуда это взялось?



# Оценки селективности

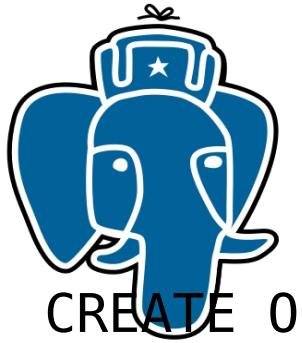
```
test=# SELECT oprrest FROM pg_operator WHERE
oprname = '@>' AND oprleft = 16385 AND oprright =
16385;
 oprrest
---------
 contsel
(1 row)
```

(hstore) (hstore)



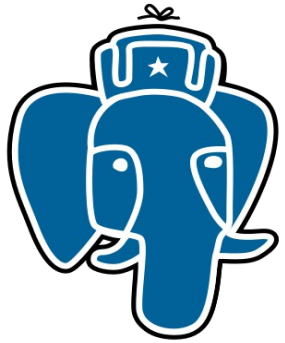
# contsel

```
Datum
contsel(PG_FUNCTION_ARGS)
{
    PG_RETURN_FLOAT8(0.001);
}
```



# CREATE OPERATOR CLASS

```
CREATE OPERATOR CLASS gin_hstore_ops
DEFAULT FOR TYPE hstore USING gin AS
    OPERATOR        7       @>,
    OPERATOR        9       ?(hstore,text),
    OPERATOR        10      ?|(hstore,text[]),
    OPERATOR        11      ?&(hstore,text[]),
    FUNCTION        1       bttextcmp(text,text),
    FUNCTION        2       gin_extract_hstore(internal, internal),
    FUNCTION       3      gin_extract_hstore_query(internal, internal,
int2, internal, internal),
    FUNCTION       4      gin_consistent_hstore(internal, int2,
internal, int4, internal, internal),
    STORAGE         text;
```



# CREATE OPERATOR CLASS

## pg_opfamily

```
-------------+----------------
oid          | 16494
opfmethod    | 2742
opfname      | gin_hstore_ops
opfnamespace | 2200
opfowner     | 16384
```

(gin)

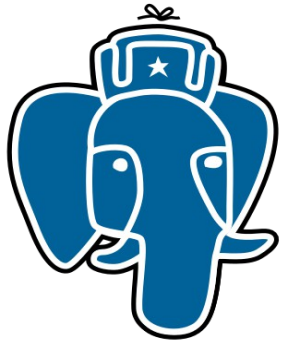
## pg_opclass

```
-------------+----------------
oid          | 16495
opcmethod    | 2742
opcname      | gin_hstore_ops
opcnamespace | 2200
opcowner     | 10
opcfamily    | 16494
opcintype    | 16385
opcdefault   | t
opckeytype   | 25
```

(gin)

(hstore)

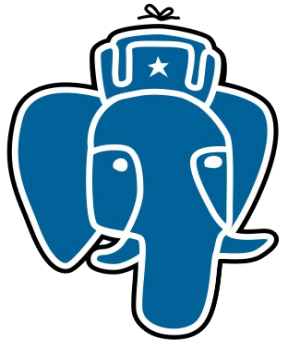
(text)



# CREATE OPERATOR CLASS

## pg_amproc

```
 amproclefttype | amprocrighttype | amprocnum |          amproc
----------------+-----------------+-----------+--------------------------
          16385 |           16385 |         1 | bttextcmp
          16385 |           16385 |         2 | gin_extract_hstore
          16385 |           16385 |         3 | gin_extract_hstore_query
          16385 |           16385 |         4 | gin_consistent_hstore
```

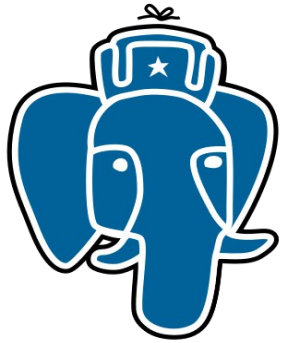


# CREATE OPERATOR CLASS

pg_amop

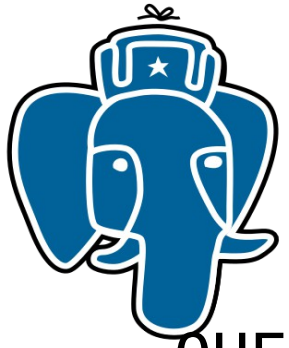
| | | | | |
|---|---|---|---|---|
| amopfamily | 16494 | 16494 | 16494 | 16494 |
| amoplefttype | 16494 | 16494 | 16494 | 16494 |
| amoprighttype | 25 | 1009 | 1009 | 16494 |
| amopstrategy | 9 | 10 | 11 | 7 |
| amoppurpose | s | s | s | s |
| amopopr | 16417 | 16399 | 16401 | 16403 |
| amopmethod | 2742 | 2742 | 2742 | 2742 |
| amopsortfamily | 0 | 0 | 0 | 0 |

(@>) (?) (?|) (?&)



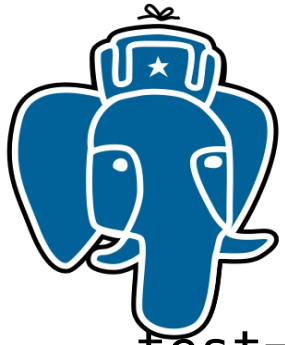
# HSTORE index

```
test=# CREATE INDEX hstore_idx ON hstore_test
       USING gin (v);

test=# SELECT * FROM hstore_test
       WHERE v @> 'a=>10'::hstore;
```



# HSTORE index

```
 QUERY PLAN
-----------------------------------------------------------------------
Bitmap Heap Scan on hstore_test  (cost=39.75..3049.43
rows=1000 width=57) (actual time=81.436..81.459 rows=100
loops=1)
   Recheck Cond: (v @> '"a"=>"10"'::hstore)
   ->  Bitmap Index Scan on hstore_idx  (cost=0.00..39.50
rows=1000 width=0) (actual time=81.414..81.414 rows=100
loops=1)
         Index Cond: (v @> '"a"=>"10"'::hstore)
 Total runtime: 81.522 ms
```



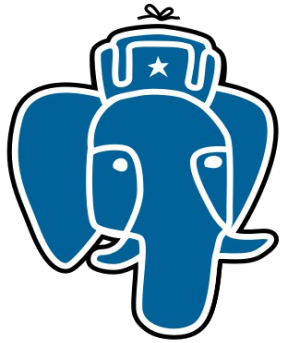
# HSTORE index

```
test=# SELECT i.indexrelid
FROM pg_index I
JOIN pg_opclass opc ON opc.oid = i.indclass[0]
JOIN pg_amop amop ON opc.opcfamily = amop.amopfamily
JOIN pg_operator op ON amop.amopopr = op.oid
WHERE indrelid = 16530 AND op.oprname = '@>'

      AND oprleft = 16385 AND oprright = 16385;
indexrelid
------------
      16536
```

(hstore_test)

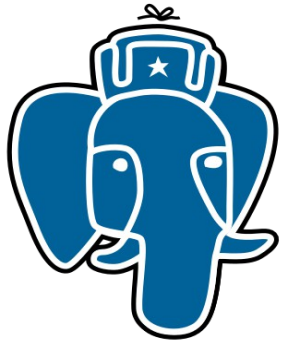
(hstore)

(hstore)



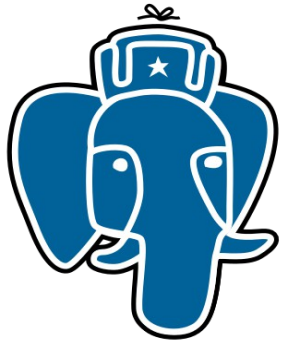
# Expression index

```
test=# CREATE INDEX hstore_idx ON hstore_test
       USING gin (v);

test=# SELECT * FROM hstore_test WHERE (v->'a')::int = 10;

-----------------------------------------------------Seq Scan on
hstore_test  (cost=0.00..31354.00 rows=5000 width=57) (actual
time=1.284..327.407 rows=100 loops=1)
   Filter: (((v -> 'a'::text))::integer = 10)
   Rows Removed by Filter: 999900
 Total runtime: 327.470 ms
```



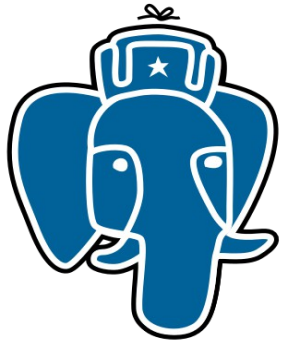
# Expression index

```
test=# CREATE INDEX hstore_a_idx
       ON hstore_test (((v->'a')::int));

test=# SELECT * FROM hstore_test WHERE (v->'a')::int = 10;

-------------------------------------------------------------
Index Scan using hstore_a_idx on hstore_test  (cost=0.43..16535.93 rows=5000
width=57) (actual time=0.018..0.033 rows=100 loops=1)
   Index Cond: (((v -> 'a'::text))::integer = 10)
 Total runtime: 0.057 ms
```



# Expression index

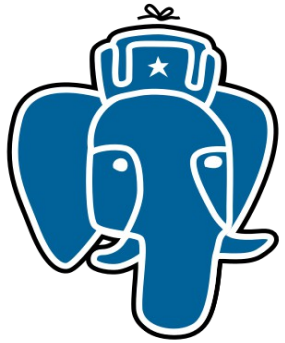
```
test=# CREATE INDEX hstore_a_idx
       ON hstore_test (((v->'a')::int));

test=# SELECT * FROM hstore_test WHERE v->'a' = '10';

-------------------------------------------------------------
Seq Scan on hstore_test  (cost=0.00..26354.00 rows=5000 width=57) (actual
time=0.747..231.017 rows=100 loops=1)
   Filter: ((v -> 'a'::text) = '10'::text)
   Rows Removed by Filter: 999900
 Total runtime: 231.077 ms
(4 rows)
```



# Extension

Решает

- Проблему pg_dump/pg_restore
- Проблему версий

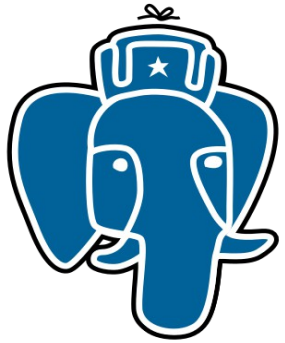


# CREATE EXTENSION

```
CREATE EXTENSION
hstore;
```

pg_extension

```
---------------+--------
oid            | 316474
extname        | hstore
extowner       | 16384
extnamespace   | 2200
extrelocatable | t
extversion     | 1.0
extconfig      |
extcondition   |
```



# CREATE EXTENSION

pg_depend

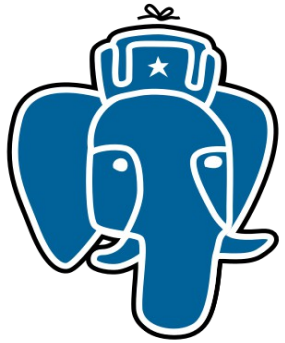
| classid | objid | objsubid | refclassid | refobjid | refobjsubid | deptype |
|---|---|---|---|---|---|---|
| 1247 | 316475 | 0 | 3079 | 316474 | 0 | e |
| 1255 | 316476 | 0 | 3079 | 316474 | 0 | e |
| 1255 | 316477 | 0 | 3079 | 316474 | 0 | e |
| 1255 | 316478 | 0 | 3079 | 316474 | 0 | e |

(pg_type) (hstore) (pg_extension) (hstore) (extension)

(pg_proc) (hstore_in) (pg_extension) (hstore) (extension)

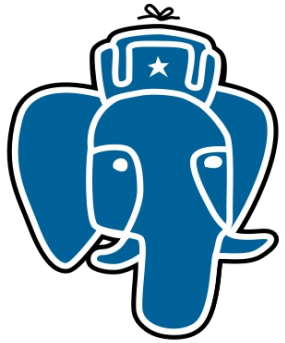
(pg_proc) (hstore_out) (pg_extension) (hstore) (extension)

(pg_proc) (hstore_recv) (pg_extension) (hstore) (extension)



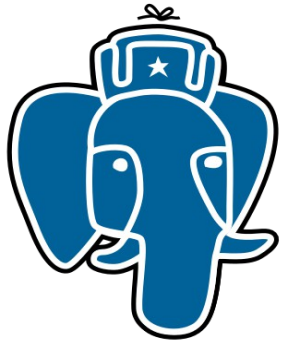
# hstore.control

```
# hstore extension
comment = 'data type for storing sets of (key, value) pairs'
default_version = '1.1'
module_pathname = '$libdir/hstore'
relocatable = true
```



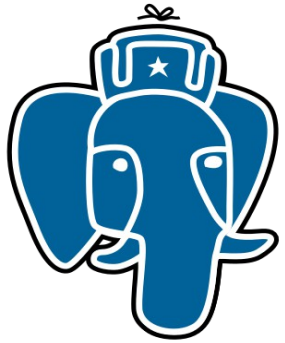
# hstore--1.1.sql

```
/* contrib/hstore/hstore--1.1.sql */

-- complain if script is sourced in psql, rather than via CREATE EXTENSION
\echo Use "CREATE EXTENSION hstore" to load this file. \quit

CREATE TYPE hstore;

CREATE FUNCTION hstore_in(cstring)
RETURNS hstore
AS 'MODULE_PATHNAME'
LANGUAGE C STRICT IMMUTABLE;
.....................................................................
```



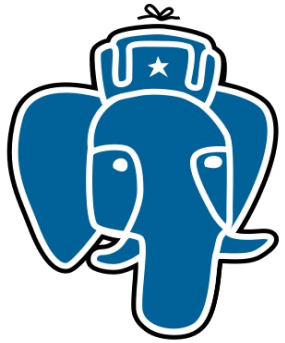
# Agenda

- Расширяемость БД
- Что такое PostgreSQL
- Расширяемость PostgreSQL
  - Btree
  - GiST — обобщенное поисковое дерево
  - GIN — обобщенный обратный индекс
  - *SP-GiST — небалансированные деревья*
- Приложения
  - Полнотекстовый поиск в PostgreSQL
- High-performance in-memory database



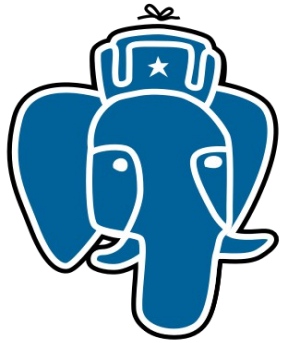
# Расширяемость БД

- IT-цикл жизнедеятельности проекта:
  - Проектирование
  - Разработка
  - Поддержка
    - Следить за производительностью системы — правило нескольких секунд (конкуренты !)
    - Автоматические агенты
    - Увеличение информации, посещаемости
  - Развитие
    - Новые идеи, новые проекты, увеличение нагрузки



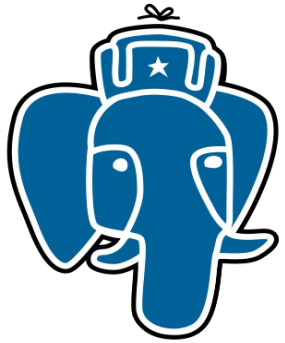
# Расширяемость БД

- Расширяемость БД (без остановки сервера)  — важнейший фактор !
- **Новая функциональность:**
  - Новые типы данных  (IP, ISSN...)
  - Новые запросы (похожие изображения)
  - Сохранение производительности и надежности
- **Масштабируемость**
  - Вертикальная, горизонтальная
- **Новая функциональность** и **масштабируемость** связаны



# Расширяемость БД

- Пример:
  {Фото}, {Пользователь}, {Тэги}
  - Соединения таблиц  не масштабируются горизонтально
  - Используют шарды (shards)
  - Можно денормализовать схему и  использовать массивы
    - Требуется индексная поддержка для эффективной работы
  - И все это надо сделать **«НА ХОДУ»**!



# Что такое PostgreSQL ?

**PostgreSQL** - это свободно распространяемая объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

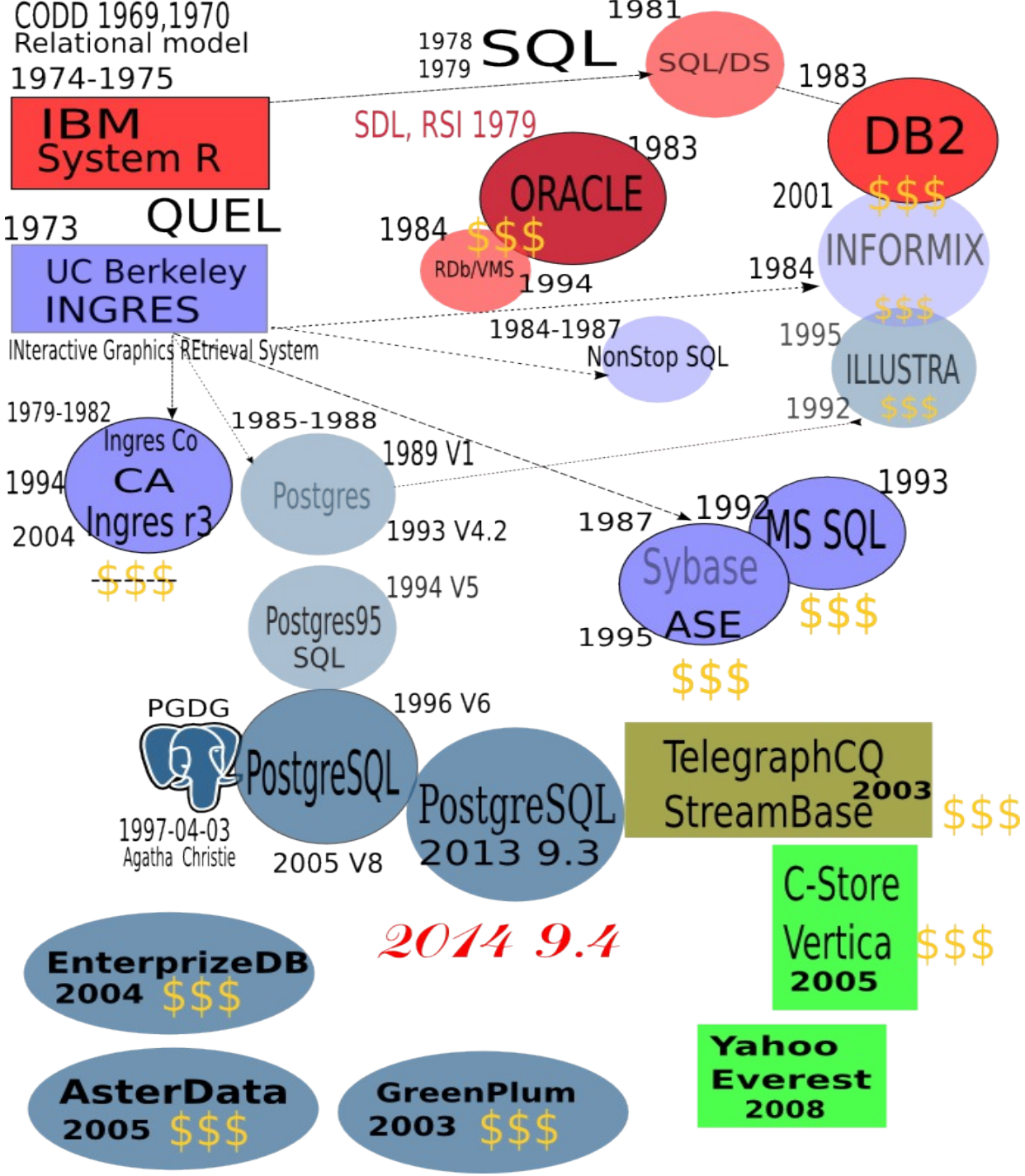
Произношение: **post-gress-Q-L**, **post-gres**, **пост-гресс**, **pgsql** (**пэ-жэ-эс-ку-эль**)

Web: **http://www.postgresql.org**

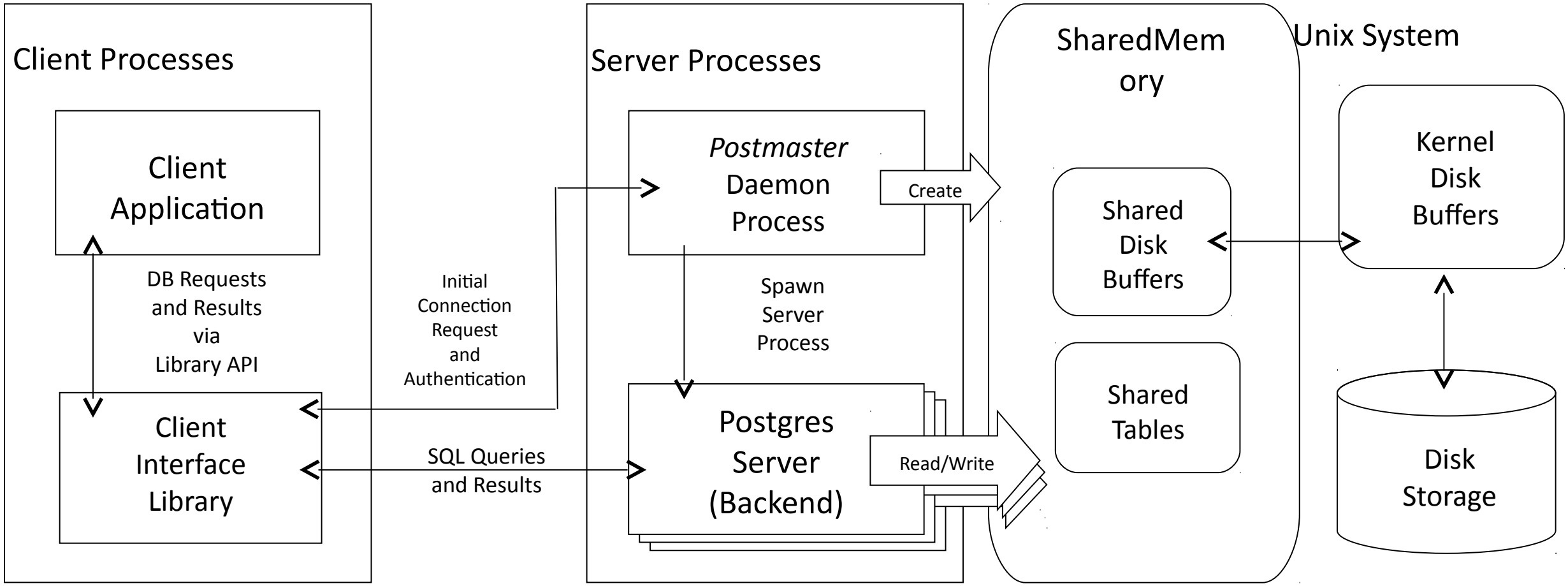
Лицензия: **BSD**

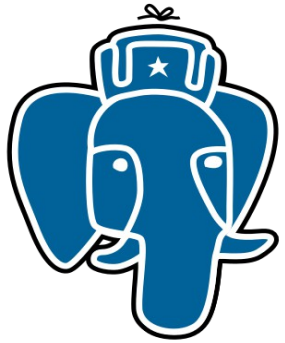
Some forks:

PostgresPlus (EnterpriseDB)
Bisgres (GreenPlum)
Everest (Yahoo)
AsterData (Teradata)
JustOneDB,
HadoopDB (Hadapt)



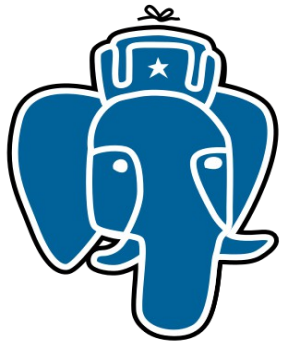
# Что такое PostgreSQL: Architecture





# Что такое PostgreSQL: Особенности

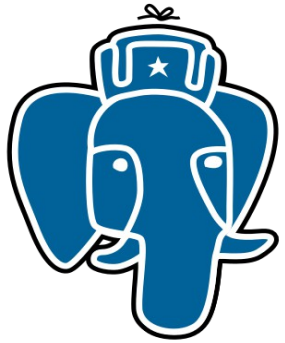
- Высокая степень параллелизма - MVCC
- Расширяемость **на ходу** (без мод. ядра) !
  - Типы данных, функции, агрегаты, операторы
  - Языки (sql,pl/pgsql,pl/perl,pl/tcl, pl/R, pl/java, pl/python, pl/v8, ...)
  - Индексы ( Btree, GiST, GIN, SP-GiST)
- Cost-based оптимизатор
- Хорошее соответствие ISO/ANSI SQL 92,99,2003
- Открытый код (BSD), открытая модель развития — нет владельца !



| Название | ASE | DB2 | FireBird | InterBase | MS SQL | MySQL | Oracle | PostgreSQL |
|---|---|---|---|---|---|---|---|---|
| Лицензия | $$$ | $$$ | IPL[2] | $$$ | $$$ | GPL/$$$ | $$$ | BSD |
| ACID | Yes | Yes | Yes | Yes | Yes | Depends[1] | Yes | Yes |
| Referential integrity | Yes | Yes | Yes | Yes | Yes | Depends[1] | Yes | Yes |
| Transaction | Yes | Yes | Yes | Yes | Yes | Depends[1] | Yes | Yes |
| Unicode | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Schema | Yes | Yes | Yes | Yes | No[5] | No | Yes | Yes |
| Temporary table | No | Yes | No | Yes | Yes | Yes | Yes | Yes |
| View | Yes | Yes | Yes | Yes | Yes | No | Yes | Yes |
| Materialized view | No | Yes | No | No | No | No | Yes | No[3] |
| Expression index | No | No | No | No | No | No | Yes | Yes |
| Partial index | No | No | No | No | No | No | Yes | Yes |
| Inverted index | No | No | No | No | No | Yes | Yes | Yes[6] |
| Bitmap index | No | Yes | No | No | No | No | Yes | No |
| Domain | No | No | Yes | Yes | No | No | Yes | Yes |
| Cursor | Yes | Yes | Yes | Yes | Yes | No | Yes | Yes |
| User Defined Functions | Yes | Yes | Yes | Yes | Yes | No[4] | Yes | Yes |
| Trigger | Yes | Yes | Yes | Yes | Yes | No[4] | Yes | Yes |
| Stored procedure | Yes | Yes | Yes | Yes | Yes | No[4] | Yes | Yes |
| Tablespace | Yes | Yes | No | ? | No[5] | No[1] | Yes | Yes |
| Название | ASE | DB2 | FireBird | InterBase | MS SQL | MySQL | Oracle | PostgreSQL |

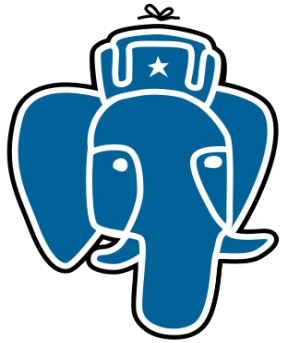
чания:

- **1** - для поддержки транзакций и ссылочной целостности требуется InnoDB (не является типом таблицы по умолчанию)
- **2** - Interbase Public License
- **3** - Materialized view (обновляемые представления) могут быть эмулированы на PL/pgSQL
- **4** - только в MySQL 5.0, которая является экспериментальной версией
- **5** - только в MS SQL Server 2005 (Yukon)
- **6** - GIN (Generalized Inverted Index) с версии 8.2



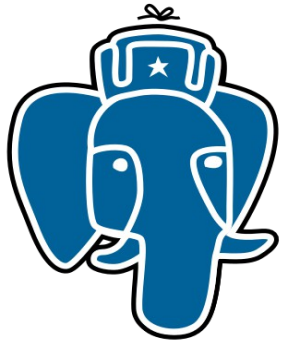
# Что такое PostgreSQL: Limitations

- Максимальный размер БД – unlimited
- Максимальный размер  таблицы – 32Тб
- Максимальная длина записи –  1.6 Tb
- Максимальная длина атрибута – 1 Gb
- Максимальное кол-во записей – unlimited
- Максимальное кол-во атрибутов – 250-1600
- Максимальное кол-во индексов - unlimited



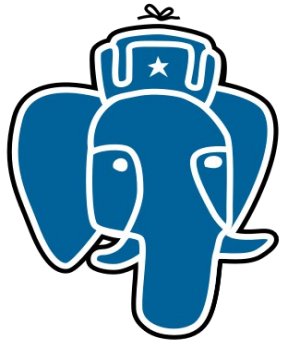
# Что такое PostgreSQL

- Поддержка:
  - Сообщество — мэйлинг лист
  - EnterpriseDB
  - 2ndQuadrant
  - Много мелких компаний
- Более подробно о PostgreSQL можно прочитать в http://www.sai.msu.su/~megera/postgres/talks/what_is_postgresql.html



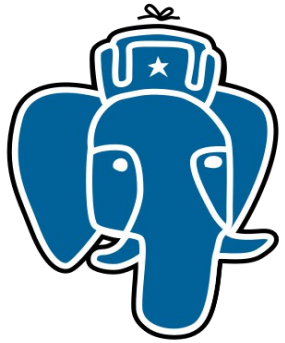
# Что такое PostgreSQL: Пользователи

- Skype  - шкалируется до миллиарда польз.
- Hi5.com — 60 млн. пользователей, #8 Alexa traffic rank
- NyYearBook.com — 18,000 req/sec, 300 Gb database
- NASA — обработка спутниковых данных (MODIS)
- Instagram — x100 млн картинок
- Sony (Free Realms) — 10 млн игроков



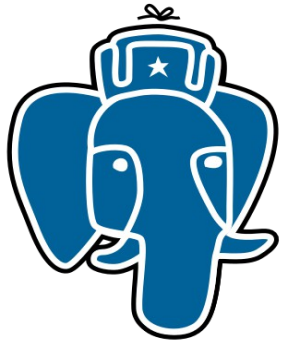
# Что такое PostgreSQL: Пользователи

- Рамблер
- 1С:Предприятие
- MirTesen, MoiKrug.ru (Yandex)
- Avito.ru — 2000 req/sec
- IRR.ru ( «Из рук в руки»)
- rabota.ru, price.ru, РБК, МастерХост
- Военные — версия 7.X входит в МСВС
- Национальная СУБД в составе НПП !?
- Астрономы — много-терабайт



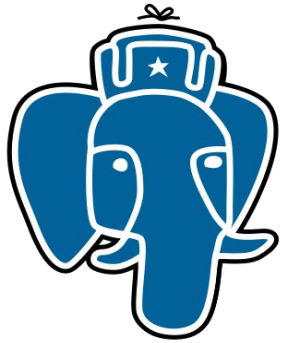
# Расширяемость PostgreSQL

- Новые типы данных требует индексные AM (access methods). Разработка новых AM, тестирование — трудно и утомительно
- Использовать Btree, Rtree  вместо разработки новых AM (access methods)
  Michael Stonebraker, «Inclusion of new types in relational database systems», 1985
  - Повысить уровень абстракции процедур доступа и обновления записей
  - Только фиксированный набор операций (операции сравнения для Btree)



# Расширяемость PostgreSQL

- Создание нового типа данных
  - Определить тип ( CREATE TYPE)
  - Написать функции ввода/вывода
  - Создать операторы (CREATE OPERATOR)
  - Написать функции сравнения
  - Оператор по-умолчанию для индекса по primary key (CREATE OPERATOR CLASS)
- для индексации новых типов использовать **GiST,GIN)** или написать новый → **SP-GiST in 9.2**

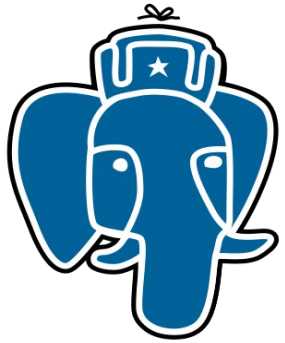


# GiST

# Обобщенное поисковое дерево

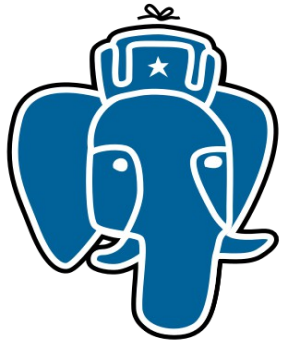
# Generalized Search Tree ( GiST)

J. M. Hellerstein, J. F. Naughton, and Avi Pfeffer. "Generalized search trees for database systems.", VLDB 21, 1995



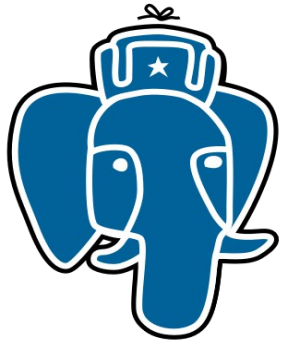
# Расширяемость PostgreSQL: GiST

- Generalized Search Tree ( GiST)
  - AM рассматривается как иерархия предикатов, в которой каждый предикат выполняется для всех подузлов этой иерархии
  - Шаблон (template) для реализации новых AM
- GiST предоставляет методы
  - навигации по дереву, 9.1 — эффективный knn (поиск ближ. соседей)
  - Обновления дерева



# Расширяемость PostgreSQL: GiST

- Конкурентность и восстановление после сбоев
- Поддерживает расширяемый набор запросов ( в отличие от Btree)
- GiST позволяет реализовать новый AM эксперту в области данных
- Новые  типы данных обладают производительностью (индексный доступ, конкурентность) и надежностью (протокол логирования), как и встроенные типы

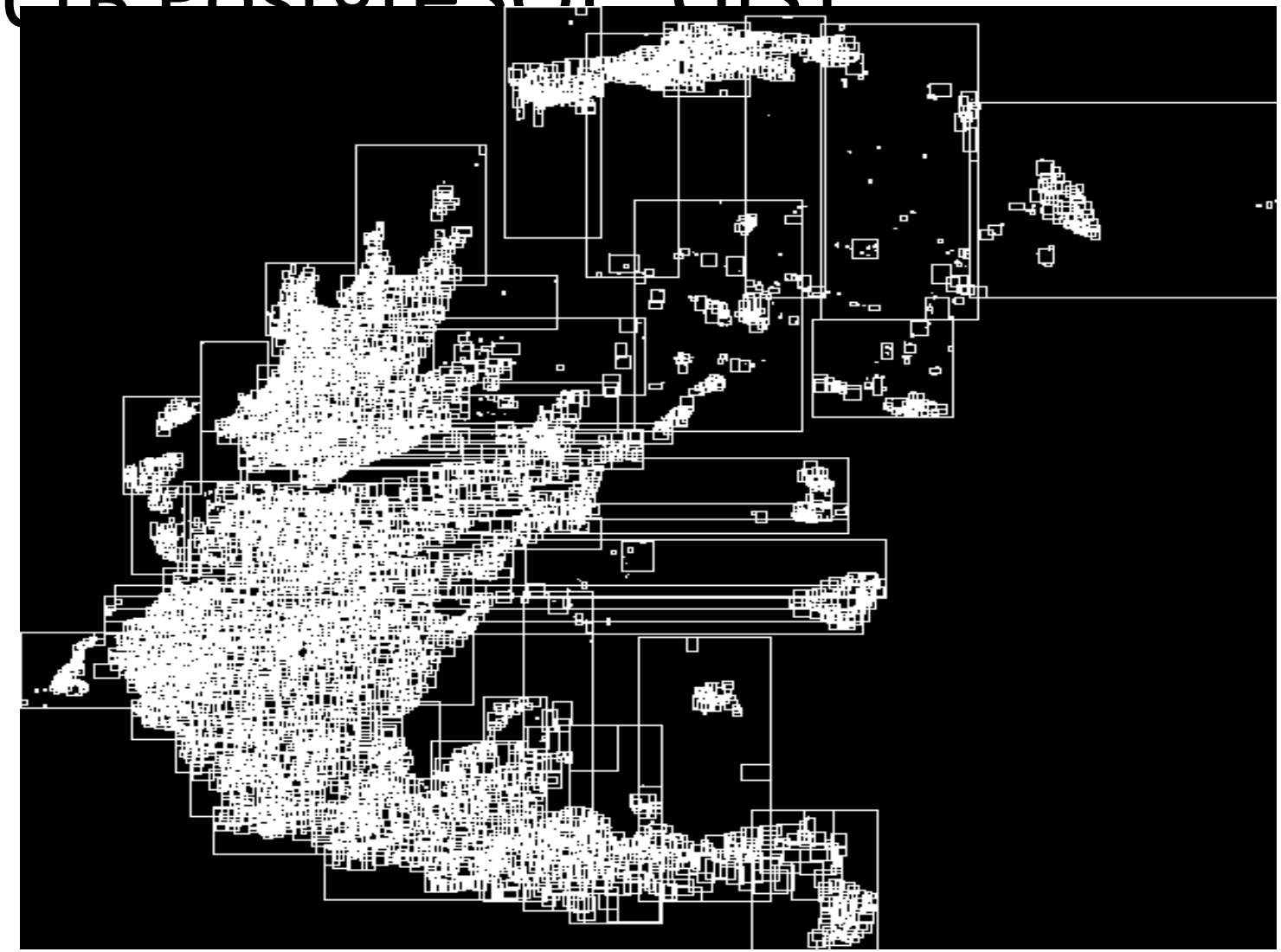


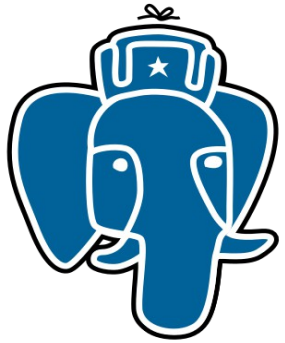
# Расширяемость PostgreSQL: GiST

- Программный интерфейс GiST:
  - GISTENTRY * compress( GISTENTRY * in )
  - GISTENTRY * decompress( GISTENTRY * in )
  - bool equal( Datum a, Datum b)
  - float * penalty( GISTENTRY *origentry, GISTENTRY *newentry, float *result)
  - Datum union(GistEntryVector *entryvec, int *size)
  - bool consistent( GISTENTRY *entry, Datum query, StrategyNumber strategy )
  - GIST_SPLITVEC * split(GistEntryVector *entryvec, GIST_SPLITVEC *v)
- http://www.sai.msu.su/~megera/postgres/talks/gist_tutorial.html

# Расширяемость PostgreSQL: GiST

- Пример — Rtree (GiST) для населенных пунктов Греции
  - Маленькие прямоугольники — исходные данные (MBR населенных пунктов)
  - Большие прямоугольники — 1-й уровень дерева
  - Подробности: http://www.sai.msu.su/~megera/wiki/Rtree_Index





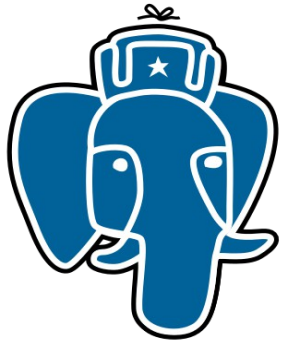
# Расширяемость PostgreSQL: GiST

- Intarray - AM для целочисленных массивов
  - Операторы overlap, contains

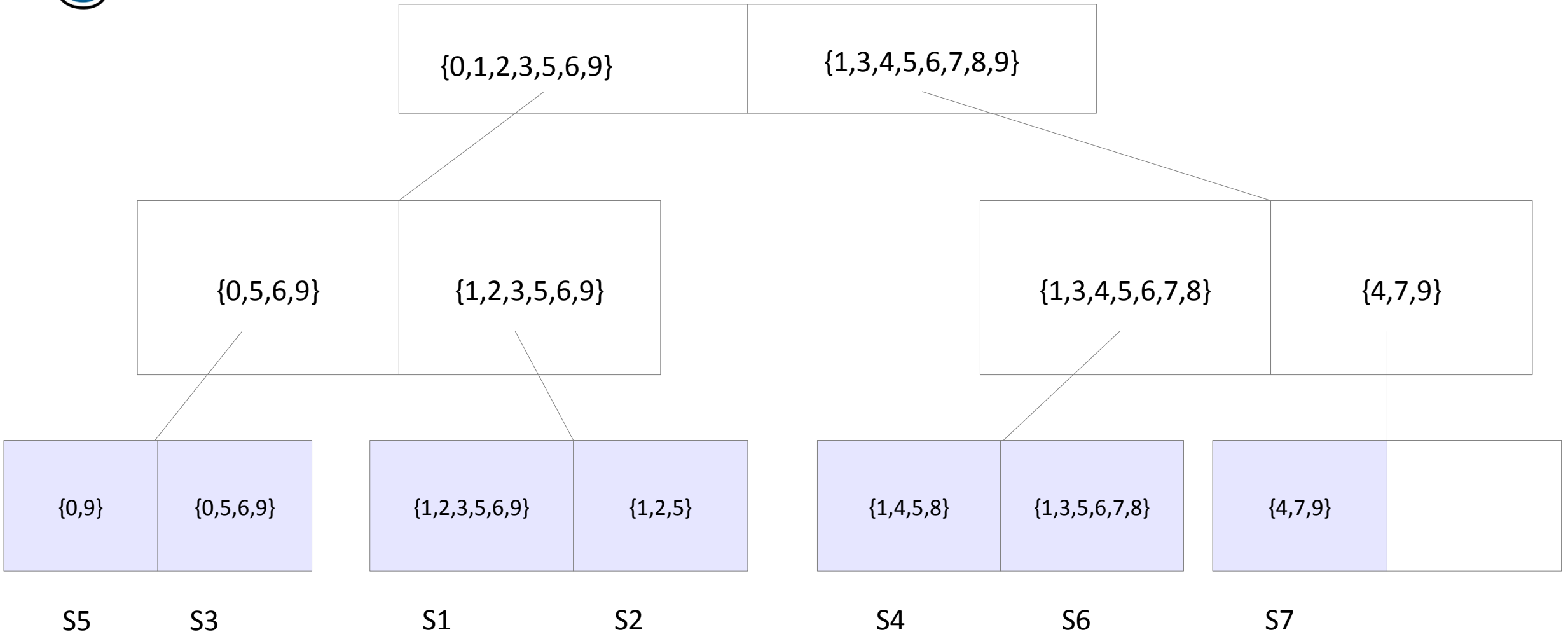
S1 = {1,**2**,3,5,6,**9**}
S2 = {1,**2**,5}
S3 = {0,5,6,**9**}
S4 = {1,4,5,8}
S5 = {0,9}
S6 = {3,5,6,7,8}
S7 = {4,7,**9**}

Q = {**2,9**}

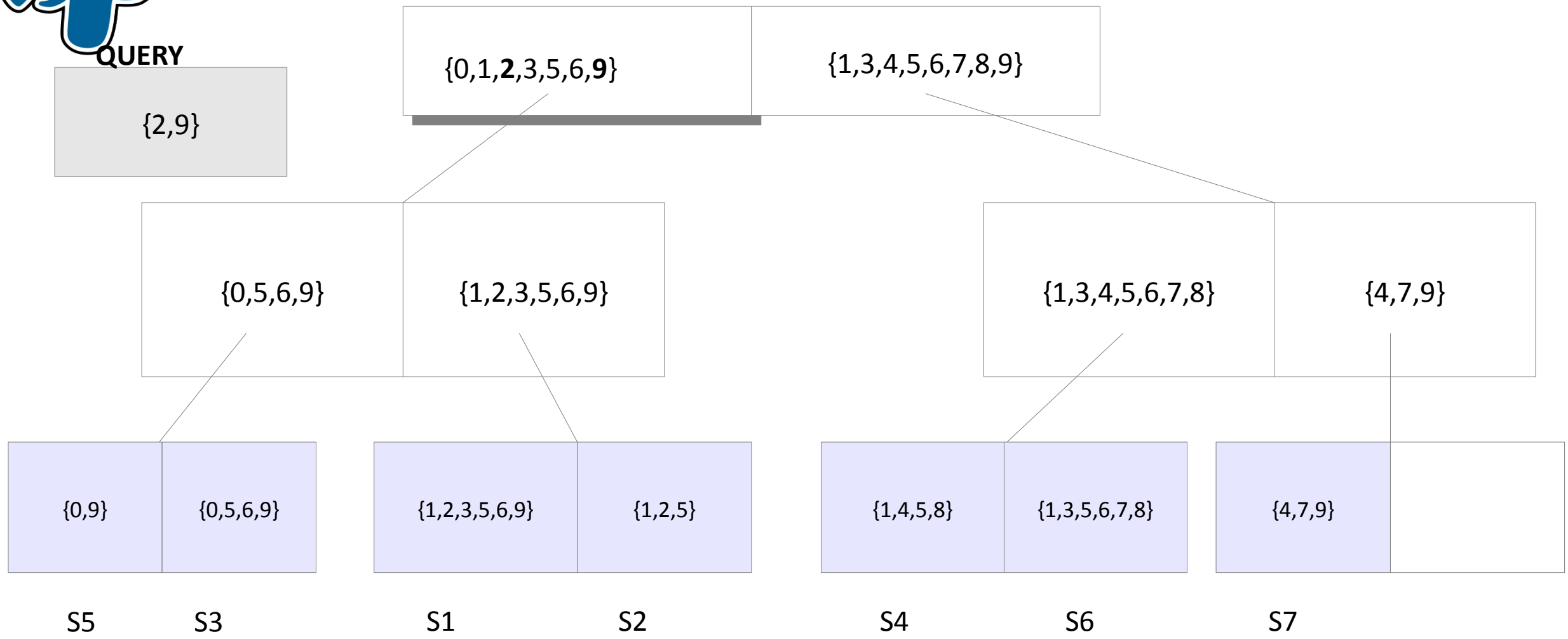
"THE RD-TREE: AN INDEX STRUCTURE FOR SETS", Joseph M. Hellerstein



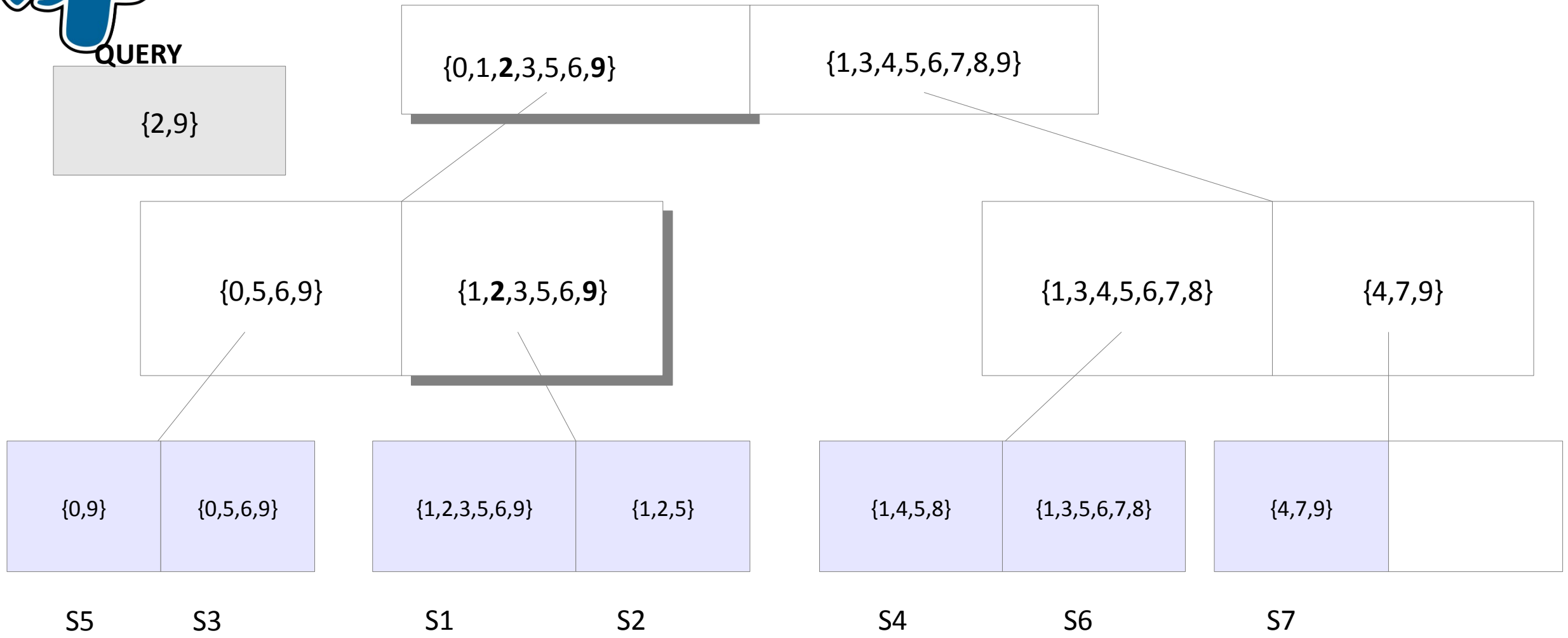
# Расширяемость PostgreSQL: GiST



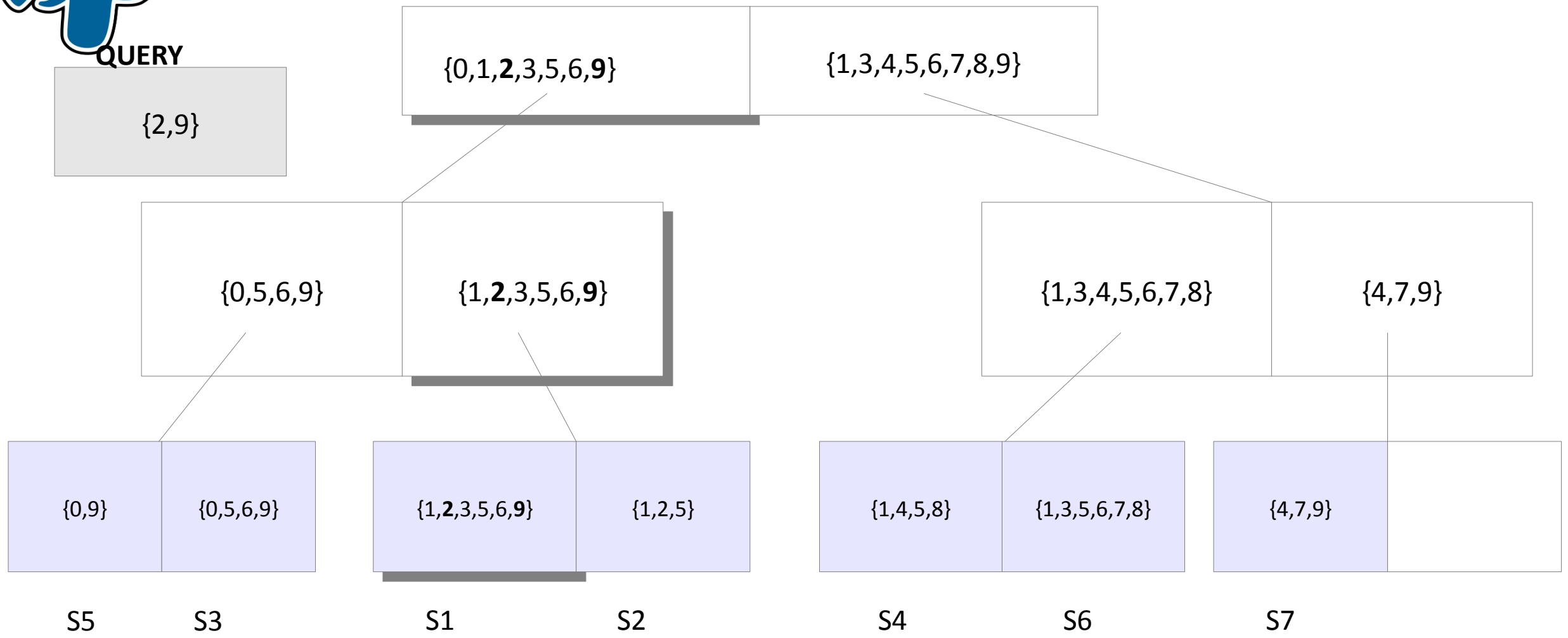
# RD-Tree


QUERY
{2,9}
{0,1,**2**,3,5,6,**9**}
{1,3,4,5,6,7,8,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,3,4,5,6,7,8}
{4,7,9}
{0,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,2,5}
{1,4,5,8}
{1,3,5,6,7,8}
{4,7,9}
S5
S3
S1
S2
S4
S6
S7

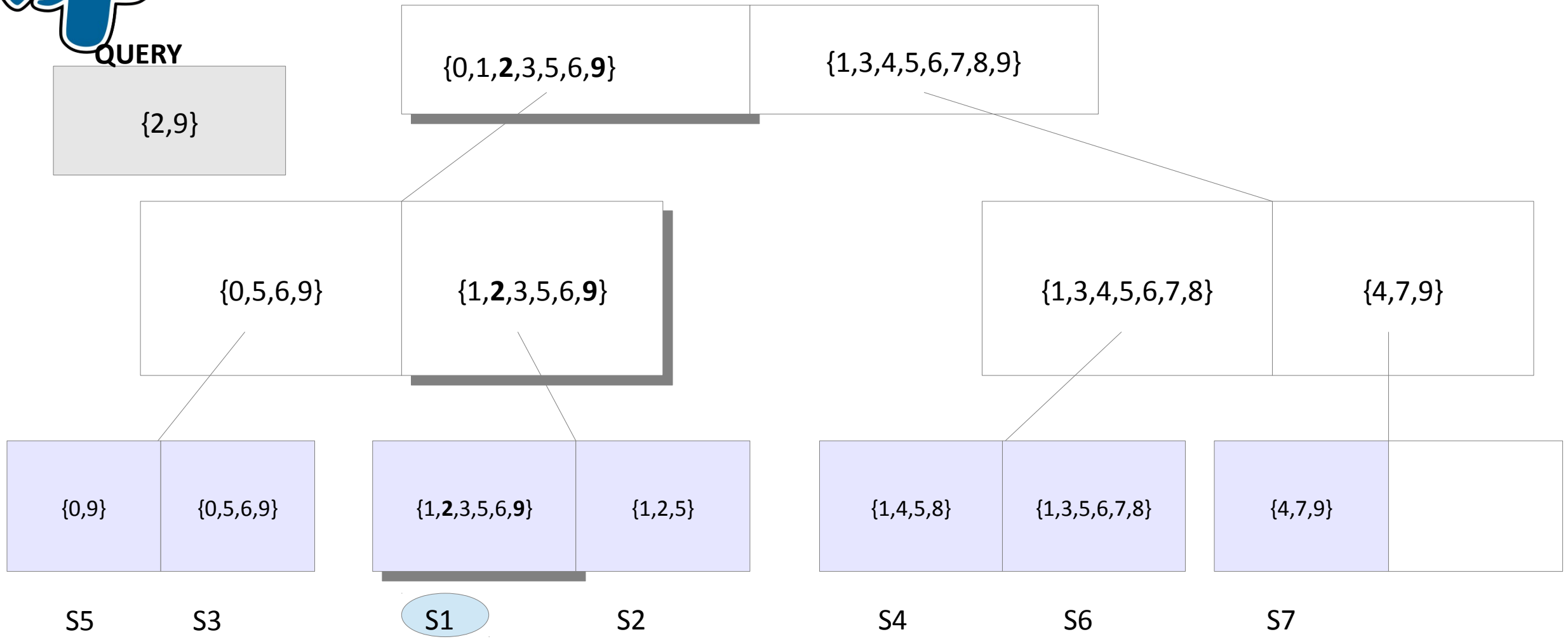
# RD-Tree

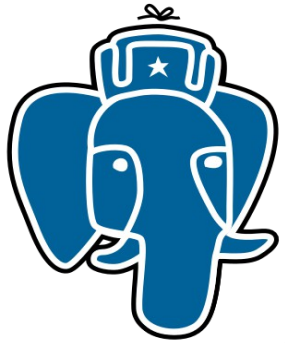

QUERY
{2,9}
{0,1,2,3,5,6,9}
{1,3,4,5,6,7,8,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,3,4,5,6,7,8}
{4,7,9}
{0,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,2,5}
{1,4,5,8}
{1,3,5,6,7,8}
{4,7,9}
S5
S3
S1
S2
S4
S6
S7

# RD-Tree


QUERY
{2,9}
{0,1,2,3,5,6,9}
{1,3,4,5,6,7,8,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,3,4,5,6,7,8}
{4,7,9}
{0,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,2,5}
{1,4,5,8}
{1,3,5,6,7,8}
{4,7,9}
S5
S3
S1
S2
S4
S6
S7

# RD-Tree


QUERY
{2,9}
{0,1,2,3,5,6,9}
{1,3,4,5,6,7,8,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,3,4,5,6,7,8}
{4,7,9}
{0,9}
{0,5,6,9}
{1,2,3,5,6,9}
{1,2,5}
{1,4,5,8}
{1,3,5,6,7,8}
{4,7,9}
S5
S3
S1
S2
S4
S6
S7



# RD-Tree (GiST)

- Сигнатура слова — слово хэшируется в позицию '1'

  w1 -> S1: 01000000  Document: w1 w2 w3

  w2 -> S2: 00010000

  w3 -> S3: 10000000

- Сигнатура документа (запроса) — суперпозиция (bit-wise OR) индивидуальных сигнатур

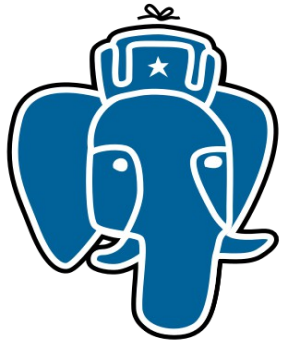
  S: 11010000

- Фильтр Блюма (Bloom filter)

  Q1: 00000001 – exact not

  Q2: 01010000 - may be contained in the document, **false drop**

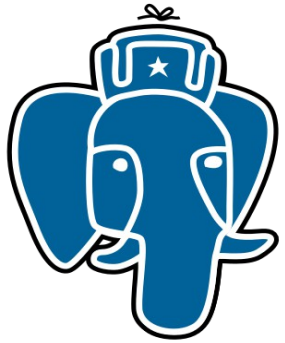
- Сигнатура — неточное (lossy) представление док-та
  - + fixed length, compact, + fast bit operations
  - - lossy (false drops), - saturation with #words grows



# RD-Tree (GiST)

- Пример — латинские поговорки

```
 id |          proverb
----+--------------------------
  1 | Ars longa, vita brevis
  2 | Ars vitae
  3 | Jus vitae ac necis
  4 | Jus generis humani
  5 | Vita nostra brevis
```

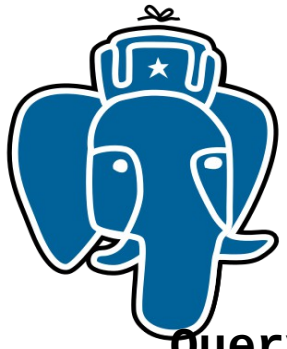


# RD-Tree (GiST)

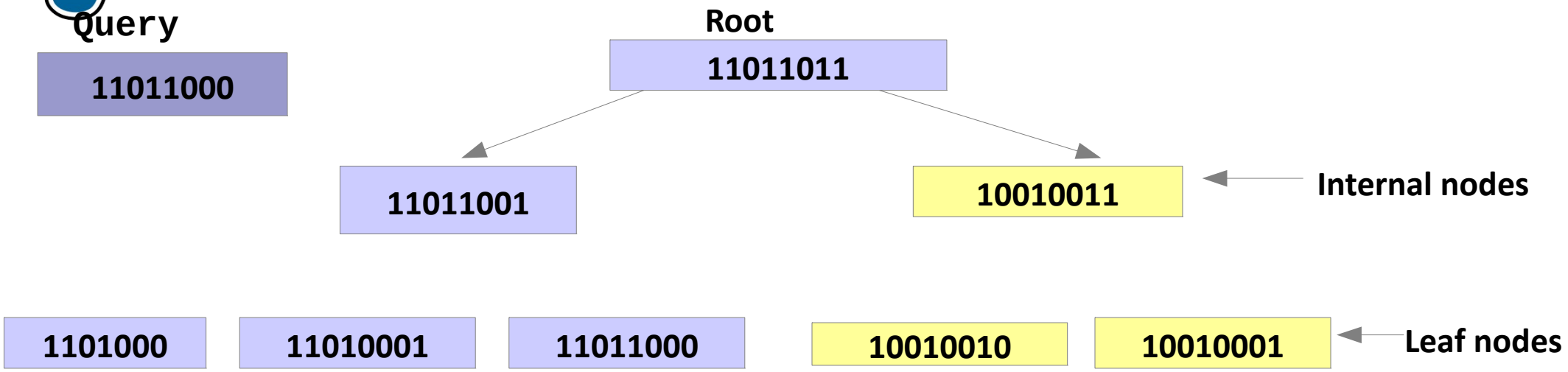
| word | signature |
|---|---|
| ac | 00000011 |
| **ars** | **11000000** |
| brevis | 00001010 |
| generis | 01000100 |
| humani | 00110000 |
| jus | 00010001 |
| longa | 00100100 |
| necis | 01001000 |
| nostra | 10000001 |
| vita | 01000001 |
| vitae | 00011000 |

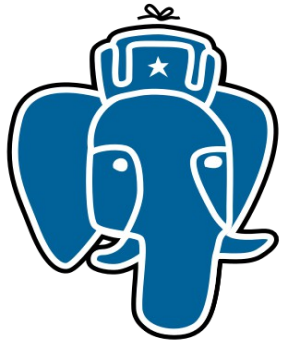
| id | proverb | signature |
|---|---|---|
| 1 | Ars longa, vita brevis | 11101111 |
| 2 | Ars vitae | 11011000 |
| 3 | Jus vitae ac necis | 01011011 |
| 4 | Jus generis humani | 01110101 |
| 5 | Vita nostra brevis | 11001011 |

False drop



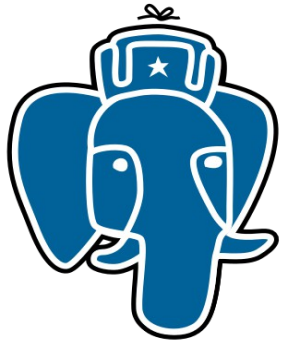
# RD-Tree (GiST)


Query
11011000
Root
11011011
11011001
10010011
Internal nodes
1101000
11010001
11011000
10010010
10010001
Leaf nodes



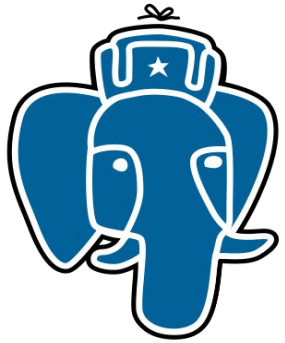
# RD-Tree (GiST)

- Проблемы
  - Плохо шкалируется с ростом количества уникальных элементов (cardinality) и количеством записей
  - Индекс неточный (lossy), требует проверки false drops



# KNN-GiST

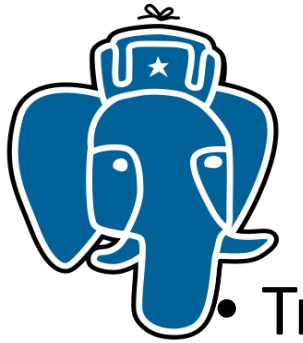
## Эффективный поиск ближайших соседей



# Knn-search: The Problem

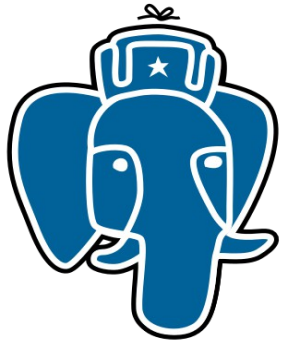
```
knn=# select id, date, event from events
       order by date <-> '1957-10-04'::date asc limit 10;
   id   |    date    |                                 event
--------+------------+-------------------------------------------------------------
  58137 | 1957-10-04 | U.S.S.R. launches Sputnik I, 1st artificial Earth satellite
  58136 | 1957-10-04 | "Leave It to Beaver," debuts on CBS
 117062 | 1957-10-04 | Gregory T Linteris, Demarest, New Jersey, astronaut, sk: STS 83
 117061 | 1957-10-04 | Christina Smith, born in Miami, Florida, playmate, Mar, 1978
 102670 | 1957-10-05 | Larry Saumell, jockey
  31456 | 1957-10-03 | Willy Brandt elected mayor of West Berlin
  58291 | 1957-10-05 | 12th Ryder Cup: Britain-Ireland, 7 -4 at Lindrick GC, England
  58290 | 1957-10-05 | 11th NHL All-Star Game: All-Stars beat Montreal 5-3 at Montreal
  58292 | 1957-10-05 | Yugoslav dissident Milovan Djilos sentenced to 7 years
 102669 | 1957-10-05 | Jeanne Evert, tennis player, Chris' sister
(10 rows)

Time: 115.548 ms
```

- Very inefficient:
  - Full table scan, btree index on date won't help.
  - Sort full table



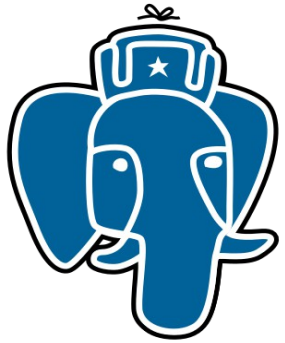
# Knn-search: Existing solutions

- Traditional way to speedup query
  - Use indexes - very inefficient (no search query !)
    - Scan full index
    - Full table scan, but in random order !
    - Sort full table
    - Better not to use index at all !
  - Constrain data space (range search)
    - Incremental search → to many queries
    - Need to know in advance the size of neighbourhood, how ? 1 Km is ok for Paris, but too small for Siberia
    - Maintain 'density map' ?



# Knn-search: What do we want !

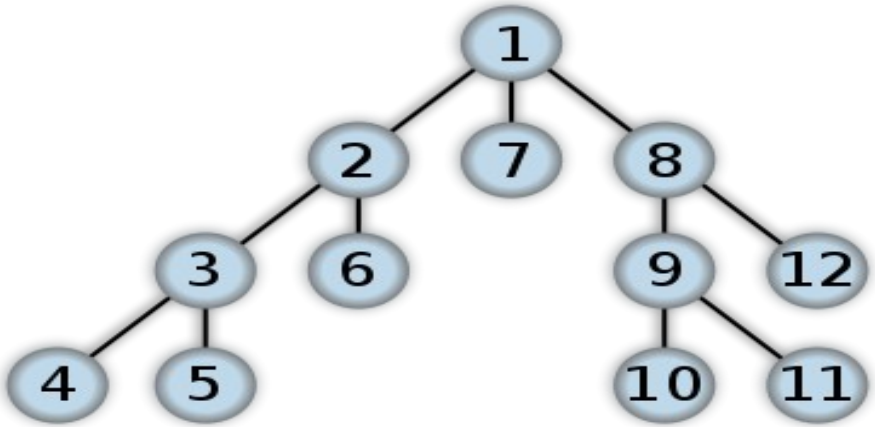
- We want to avoid full table scan – read only right tuples
  - So, we need index
- We want to avoid sorting – read  right tuples in right order
  - So, we need special strategy to traverse index
- We want to support tuples visibility
  - So, we should be able to resume  index traverse



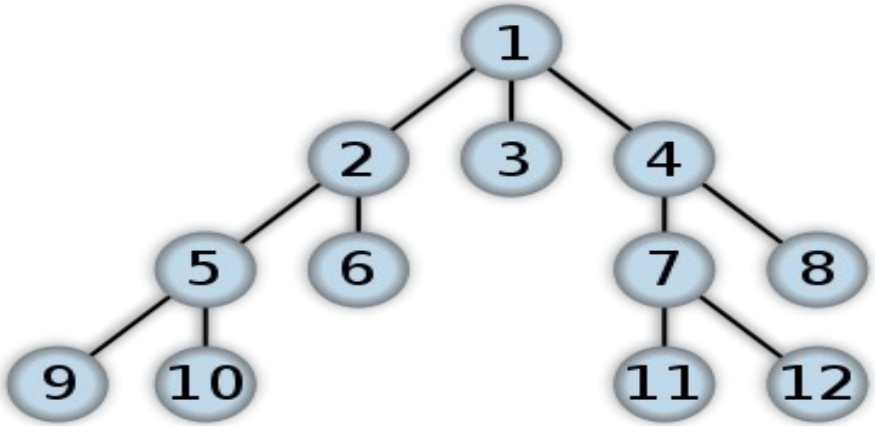
# Knn-search: Index traverse

- Depth First Search (stack, LIFO)

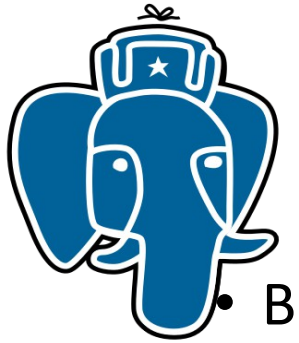
  R-tree search




- Breadth First Search (queue, FIFO)

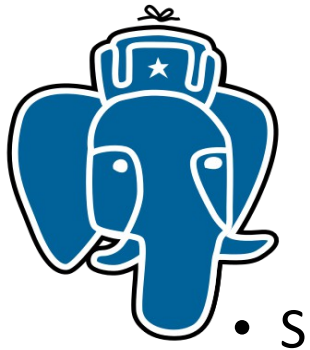



- Both strategies are not good – full index scan



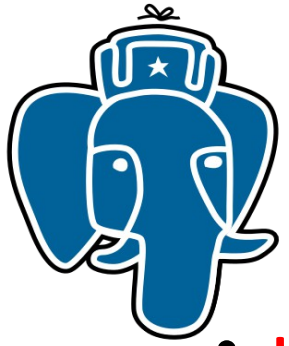
# Knn-search: Index traverse

- Best First Search (PQ, priority queue). Maintain order of items in PQ according their distance from given point
  - Distance to MBR (rectangle for Rtree) for internal pages – minimum distance of all items in that MBR
  - Distance = 0 for MBR with given point
  - Distance to point for leaf pages
- Each time we extract point from PQ we output it – it is next closest point ! If we extract rectangle, we expand it by pushing their children (rectangles and points) into the queue.
- We traverse index by visiting only interesting nodes !



# Knn-search: Performance

- SEQ (no index) – base performance
  - Sequentually read full table + Sort full table (can be very bad, sort_mem !)
- DFS – very bad !
  - Full index scan + Random read full table + Sort full table
- BFS – the best for small k !
  - Partial index scan + Random read k-records
    T(index scan) ~ Height of Search tree ~ log(n)
  - Performance win BFS/SEQ ~ Nrelpages/k, for small k. The more rows, the more benefit !
  - Can still win even for k=n (for large tables) - no sort !

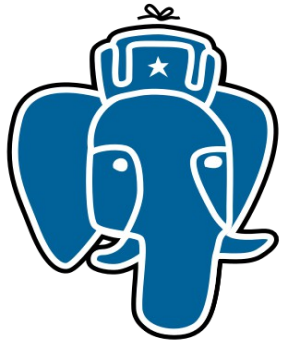


# Knn-search: What do we want !

- + We want to avoid full table scan – read only right tuples
  - So, we need index
- + We want to avoid sorting – read right tuples in right order
  - So, we need special strategy to traverse index
- + We want to support tuples visibility
  - So, we should be able to resume index traverse
- We want to support many data types
  - So, we need to modify GiST

# Knn-search: modify GiST

- GiST – Generalized Search Tree, provides
  - API to build custom disk-based search trees (any tree, where key of internal page is a Union of keys on children pages)
  - Recovery and Concurrency
  - Data type and query extendability
- GiST is widely used in GIS (PostGIS), text search, astro,bio,...
- Current strategy of search tree traverse is DFS
  - Change to BFS (Best First Search) strategy
  - Retain API compatibility



# GiST: Technical details

**Depth First Search**

```
push Stack, Root;
While Stack {
   If p is heap {
            output p;
   else {
            children = get_children(p);
            push Stack, children;
        }
}
```

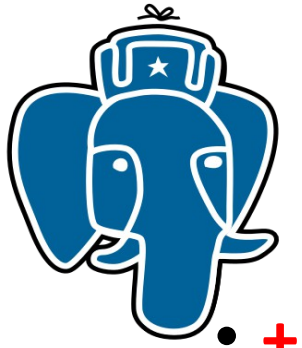
**Best First Search**

```
push PQ, Root;
While PQ {
   If p is heap {
            output p;
   else {
            Children = get_children(p);
            push PQ, children;
        }
}
```

- For non-knn search all distances are zero, so

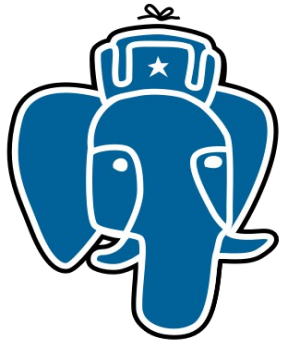
  PQ => Stack and BFS => DFS

- We can use only one strategy (BFS) for both – normal search and knn-search !



# Knn-search: What do we want !

- + We want to avoid full table scan – read only <right> tuples
  - So, we need index
- + We want to avoid sorting – read  <right> tuples in <right> order
  - So, we need special strategy to traverse index
- + We want to support tuples visibility
  - So, we should be able to resume  index traverse
- + We want to support many data types
  - So, we need to modify GiST

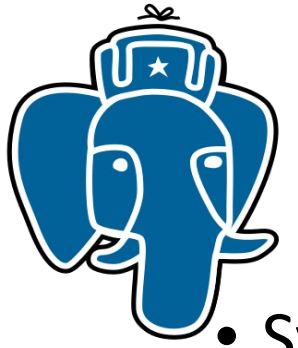


# Knn-search: syntax

- Knn-query uses ORDER BY clause

```
SELECT … FROM … WHERE …
ORDER BY p <-> '(0.,0.)'::point
 LIMIT k;
```

<-> - distance operator, should be provided for data type



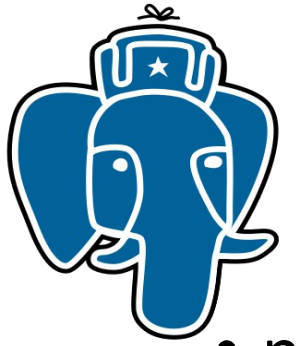
# Knn-search: Examples

- Synthetic data – 1,000,000 randomly distributed points

```
create table qq ( id serial, p point, s int4);

insert into qq (p,s)  select point( p.lat, p.long), (random()*1000)::int
from ( select  (0.5-random())*180 as lat, random()*360 as long
       from ( select generate_series(1,1000000) ) as t
) as p;
create index qq_p_s_idx on qq using gist(p);
analyze qq;
```

- Query – find k-closest points to (0,0)

```
set enable_indexscan=on|off;
explain (analyze on, buffers on)
  select * from qq order by (p <-> '(0,0)') asc limit 10;
```



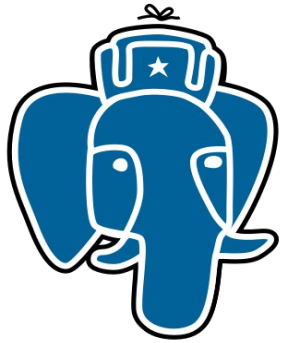
# Knn-search: Examples

- postgresql.conf:

  shared_buffers = 512MB #32MB
  work_mem = 32MB #1MB
  maintenance_work_mem = 256MB #16MB
  checkpoint_segments = 16
  effective_cache_size = 1GB #128MB

- Index statistics (n=1000,000)

```
Number of levels:          3
Number of pages:           8787
Number of leaf pages:      8704
Number of tuples:          1008786
Number of invalid tuples:  0
Number of leaf tuples:     1000000
Total size of tuples:      44492028 bytes
Total size of leaf tuples: 44104448 bytes
Total size of index:       71983104 bytes
```

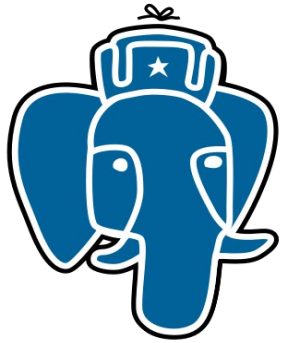


# Knn-search: Examples

**k=1, n=1,000,000**

```
Limit  (cost=24853.00..24853.00 rows=1 width=24) (actual time=469.129..469.130
rows=1 loops=1)
   Buffers: shared hit=7353
   ->  Sort  (cost=24853.00..27353.00 rows=1000000 width=24) (actual
time=469.128..469.128 rows=1 loops=1)
         Sort Key: ((p <-> '(0,0)'::point))
         Sort Method:  top-N heapsort  Memory: 25kB
         Buffers: shared hit=7353
         ->  Seq Scan on qq  (cost=0.00..19853.00 rows=1000000 width=24)
(actual time=0.007..241.539 rows=1000000 loops=1)
               Buffers: shared hit=7353
 Total runtime: 469.150 ms
-------------------------
 Limit  (cost=0.00..0.08 rows=1 width=24) (actual time=0.104..0.104
rows=1 loops=1)
   Buffers: shared hit=4
   ->  Index Scan using qq_p_idx on qq  (cost=0.00..82060.60 rows=1000000
width=24) (actual time=0.104..0.104 rows=1 loops=1)
         Sort Cond: (p <-> '(0,0)'::point)
         Buffers: shared hit=4

 Total runtime: 0.117 ms
```

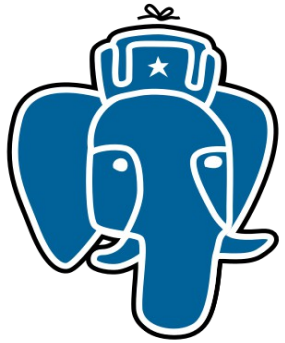
**4000 times faster !**



# Knn-search: Examples

**N=1,000,000**

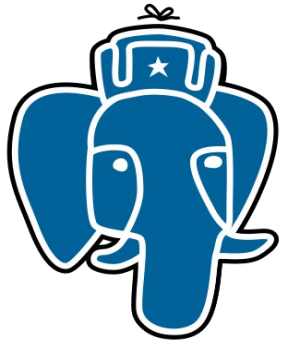
| k | :hit | :knn | : seq | :sortmem ( seq ) |
|---|---|---|---|---|
| 1 | :4 | :0.117 | :469.150 | : 25 |
| 10 | :17 | :0.289 | :471.735 | : 25 |
| 100 | :118 | :0.872 | :468.244 | : 32 |
| 1000 | :1099 | :7.107 | :473.840 | : 127 |
| 10000 | :10234 | :31.629 | :525.557 | : 1550 |
| 100000 | :101159 | :321.182 | :994.925 | : 13957 |



# Knn-search: The Problem

```
knn=# select id, date, event from events
       order by date <-> '1957-10-04'::date asc limit 10;
   id   |    date    |                                 event
--------+------------+-------------------------------------------------------------
  58137 | 1957-10-04 | U.S.S.R. launches Sputnik I, 1st artificial Earth satellite
  58136 | 1957-10-04 | "Leave It to Beaver," debuts on CBS
 117062 | 1957-10-04 | Gregory T Linteris, Demarest, New Jersey, astronaut, sk: STS 83
 117061 | 1957-10-04 | Christina Smith, born in Miami, Florida, playmate, Mar, 1978
 102670 | 1957-10-05 | Larry Saumell, jockey
  31456 | 1957-10-03 | Willy Brandt elected mayor of West Berlin
  58291 | 1957-10-05 | 12th Ryder Cup: Britain-Ireland, 7 -4 at Lindrick GC, England
  58290 | 1957-10-05 | 11th NHL All-Star Game: All-Stars beat Montreal 5-3 at Montreal
  58292 | 1957-10-05 | Yugoslav dissident Milovan Djilos sentenced to 7 years
 102669 | 1957-10-05 | Jeanne Evert, tennis player, Chris' sister
(10 rows)

Time: 115.548 ms
```

- Very inefficient:
  - Full table scan, btree index on date won't help.
  - Sort full table



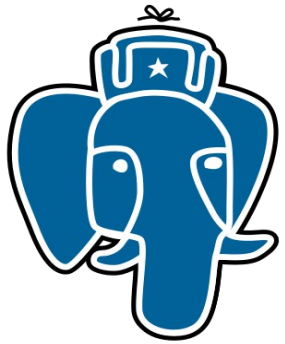
# Knn-search: The Problem

contrib/btree_gist

```
knn=# select id, date, event from events
      order by date <-> '1957-10-04'::date asc limit 10;
   id   |    date    |                                 event
--------+------------+-------------------------------------------------------------
  58137 | 1957-10-04 | U.S.S.R. launches Sputnik I, 1st artificial Earth satellite
  58136 | 1957-10-04 | "Leave It to Beaver," debuts on CBS
 117062 | 1957-10-04 | Gregory T Linteris, Demarest, New Jersey, astronaut, sk: STS 83
 117061 | 1957-10-04 | Christina Smith, born in Miami, Florida, playmate, Mar, 1978
 102670 | 1957-10-05 | Larry Saumell, jockey
  31456 | 1957-10-03 | Willy Brandt elected mayor of West Berlin
  58291 | 1957-10-05 | 12th Ryder Cup: Britain-Ireland, 7 -4 at Lindrick GC, England
  58290 | 1957-10-05 | 11th NHL All-Star Game: All-Stars beat Montreal 5-3 at Montreal
  58292 | 1957-10-05 | Yugoslav dissident Milovan Djilos sentenced to 7 years
 102669 | 1957-10-05 | Jeanne Evert, tennis player, Chris' sister
(10 rows)

Time: 0.590 ms
```

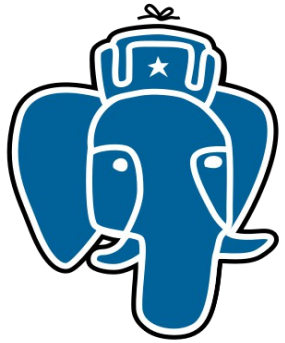
200 times faster !

- Very inefficient:
  - 8 index pages read + 10 tuples read
  - No sorting



# GIN

## Обобщенный обратный индекс



# Обратный индекс

## Report Index

### A



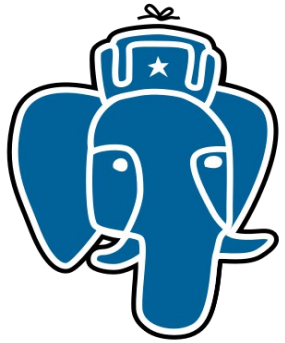
### B



### D



### E



# Inverted Index

## Report Index

### A

abrasives, 27
acceleration measurement, 58
accelerometers, 5, 10, 25, 28, 30, 36, 58, 59, 61, 73, 74
actuators, 4, 37, 46, 49
adaptive Kalman filters, 60, 61
adhesion, 63, 64
adhesive bonding, 15
adsorption, 44
aerodynamics, 29

compensation, 30, 68
compressive strength, 54
compressors, 29
computational fluid dynamics, 23, 29
computer games, 56
concurrent engineering, 14
contact resistance, 47, 66
convertors, 22
coplanar waveguide components, 40
Couette flow, 21
creep, 17
crystallisation, 64
current density, 13, 16

QUERY: compensation accelerometers

INDEX: accelerometers 5,10,25,28,**30**,36,58,59,61,73,74

compensation **30**,68

RESULT: **30**

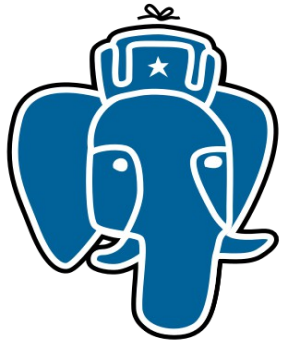
attitude measurement, 59, 61
automatic test equipment, 71
automatic testing, 24

### B

backward wave oscillators, 45

discrete wavelet transforms, 72
displacement measurement, 11
display devices, 56
distributed feedback lasers, 38

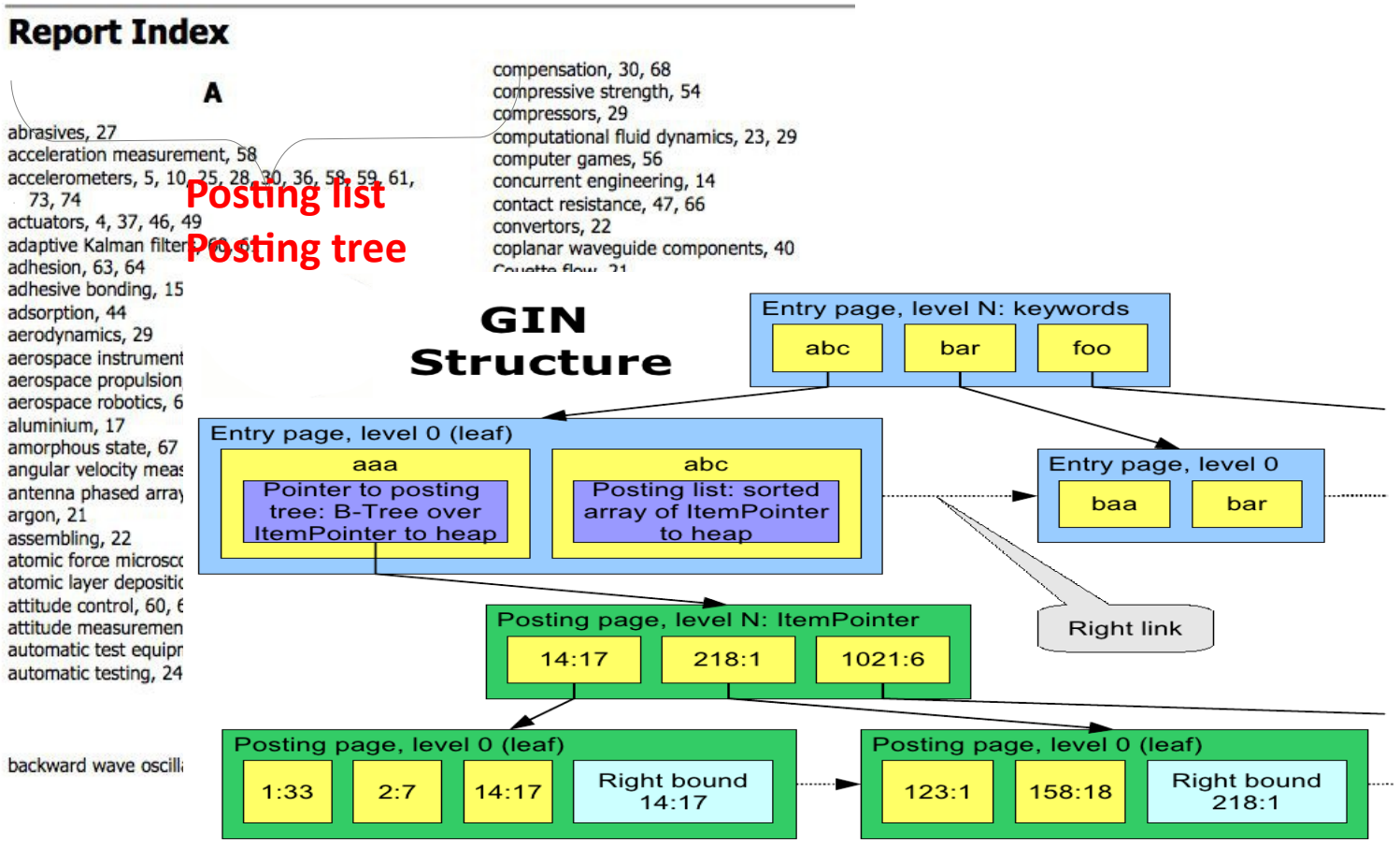
### E

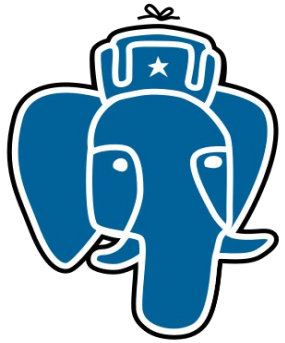


No positions in index !

# Inverted Index in PostgreSQL

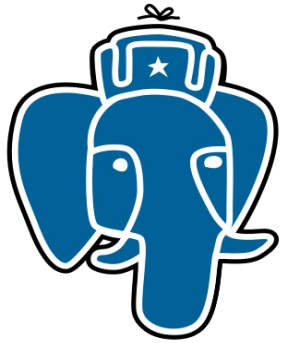
E N T R Y  T R E E





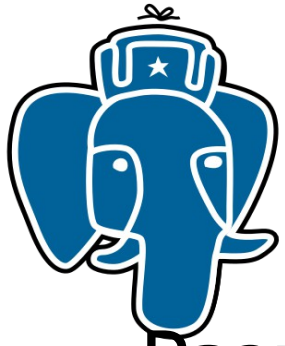
# Inverted Index

- Структура данных, которая для каждого ключа хранит список документов, содержащих этот ключ
- Тратим время на препроцессинг и экономим при поиске
- Синонимы: posting list, posting file, inverted file, инвертированный список
- GIN (Generalized Inverted Index) - *Абстрагируемся от операции* — тип данных сам определяет какую операцию ускорять



# GIN

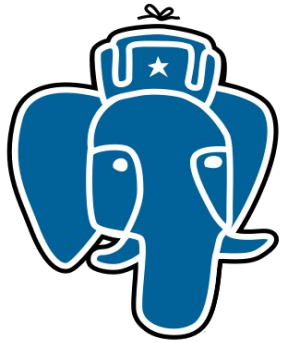
- Поддерживает разные типы данных
- Очень быстрый поиск по ключам — Btree
- Поддержка partial match
- Многоатрибутный индекс
- Хорошая масштабируемость (кол-во ключей, кол-во документов)
- Быстрое создание индекса
- «Медленное» обновление индекса
- Надежность и хороший параллелизм



# Generalized Inverted Index:API

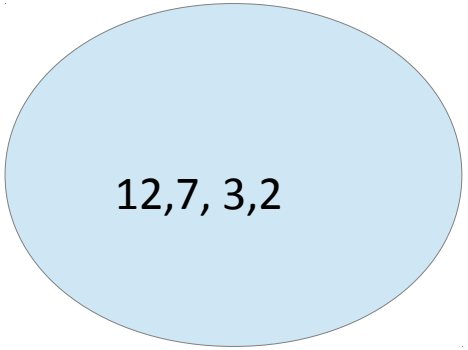
Разработчик предоставляет 4 (5) функции:

- Datum* extractValue(Datum inputValue, uint32* nentries)
- int compareEntry(Datum a, Datum b)
- Datum* extractQuery(Datum query, uint32* nentries, StrategyNumber n, bool* pmatch[])
- bool consistent(bool check[], StrategyNumber n, Datum query, bool *needRecheck)
- int comparePartial(Datum query_key, Datum indexed_key, StrategyNumber n )

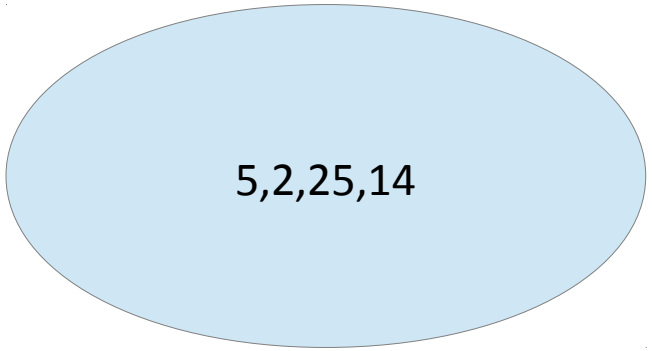


# GIN

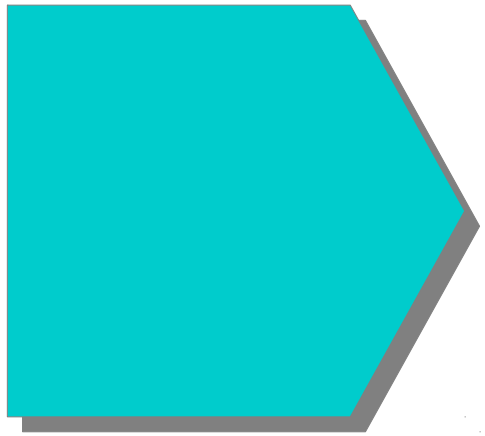

**1**



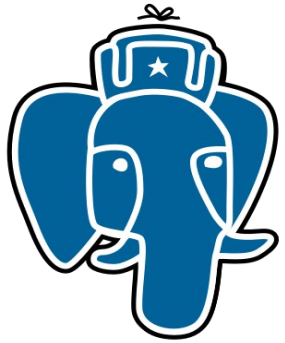

**2**




**3**

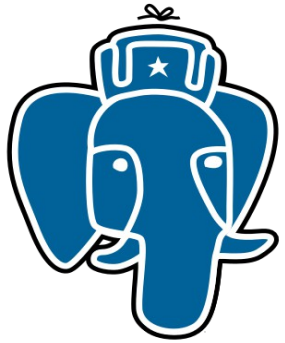


**1: 1**
**2: 2,3**
**3: 2**
**5: 1,3**
**6: 1**
**7: 1,2**
**12: 2**
**14: 3**
**25: 3**



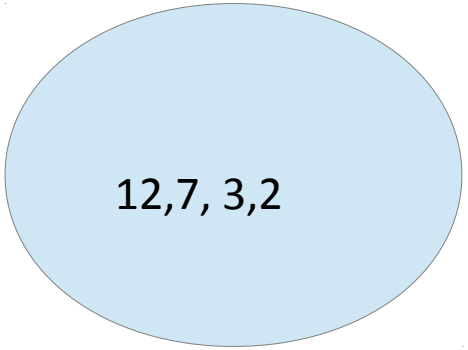
# GIN

- Поддерживает разные типы данных
- Очень быстрый поиск по ключам — Btree
- Поддержка partial match
- Многоатрибутный индекс
- Хорошая масштабируемость (кол-во ключей, кол-во документов)
- Быстрое создание индекса
- Медленное обновление индекса :(
- Надежность и хороший параллелизм

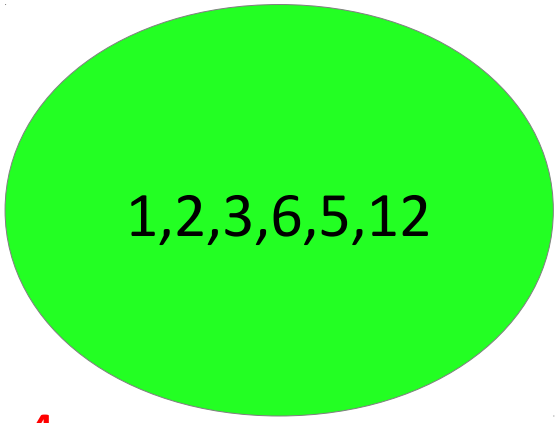


# GIN: Update problem

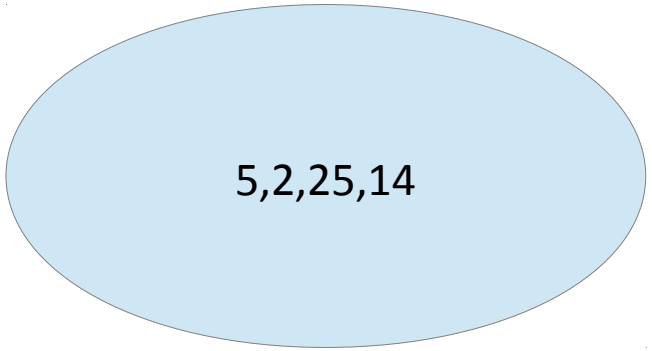

1



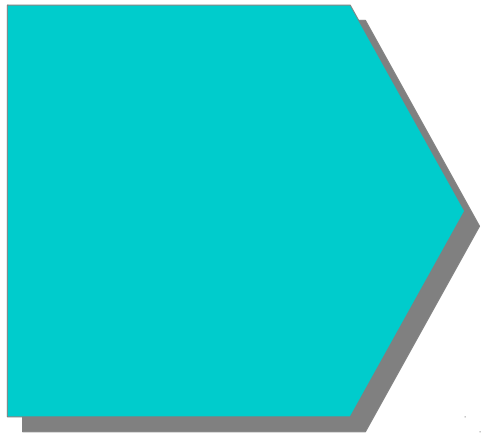

2




4

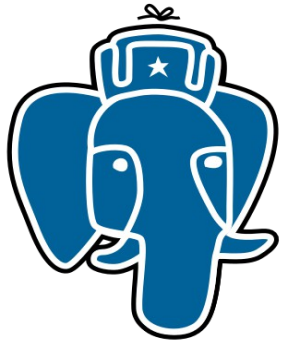



3



**1: 1,4**
**2: 2,3,4**
**3: 2,4**
**5: 1,3,4**
**6: 1,4**
**7: 1,2**
**12: 2,4**
**14: 3**
**25: 3**

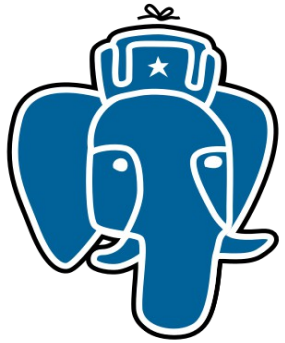
1 new object → 6 inserts into index



# GIN: The Problem

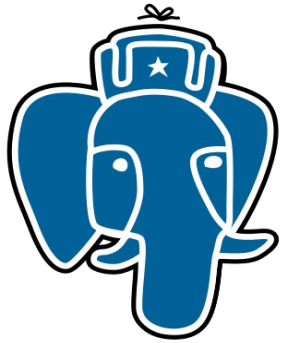
```
CREATE TABLE                    CREATE TABLE
INSERT 10,000 int[]             CREATE INDEX
CREATE INDEX                    INSERT 10,000 int[]

 3.1 s + 11 s                   ~0 s + 100 s
    13.1 s                           100 s
```

**BULK index insert ~ 10  times faster !**



# GIN: Быстрое обновление

- Постинг лист для большого количества  значений заменяется на Btree — ускоряет поиск
- Обновления в индекс *откладываются* — используется техника bulk insert, как и при создании индекса

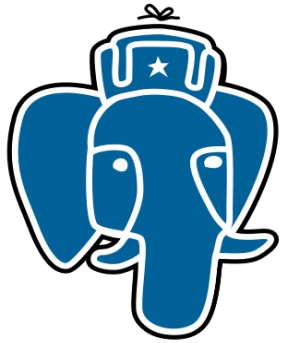


# GIN: Быстрое обновление

```
CREATE TABLE
CREATE INDEX
INSERT 10,000 int[]

~0 s + 100 s
     100 s
```

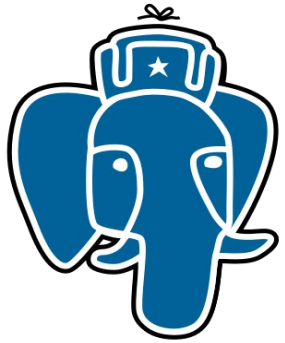
```
CREATE TABLE
CREATE INDEX
INSERT 10,000 int[]
VACUUM TABLE

~0 s + 18 s + 12 s
      30 s
```

| BULK INSERT | OLD_GIN | NEW_GIN |
|---|---|---|
| 10 s | 100 s | 30 s |



# Приложения

- Целочисленные массивы (GiST, GIN)
- Полнотекстовый поиск  (GiST, GIN)
- Данные с древовидной структурой (GiST)
- Поиск похожих слов  (GiST, GIN)
- Rtree (GiST)
- PostGIS (postgis.org) (GiST) — spatial index
- BLASTgres (GiST) — биоинформатика
- Многомерный куб (GiST)
- ……..

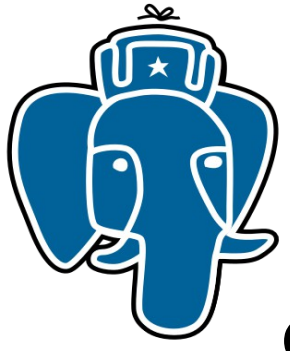


# Полнотекстовый поиск: GTSE

GTSE — Generalized Text Search Engine

- Полнотекстовые типы данных
  - Tsvector - полнотекстовое представление документа (прямой индекс)
    - лексемы с координатной информацией и важностью
  - Tsquery — полнотекстовый запрос
    - Лексемы с логическими операторами
- Полнотекстовый оператор tsvector @@ tsquery

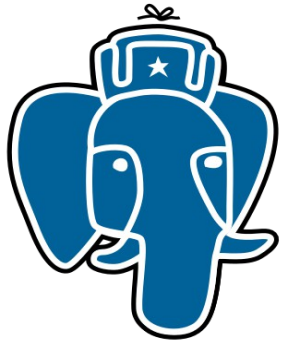
```
'a fat cat sat on a mat and ate a fat rat'::tsvector @@ 'cat & rat':: tsquery;
```



# Полнотекстовый поиск: GTSE

GTSE — Generalized Text Search Engine

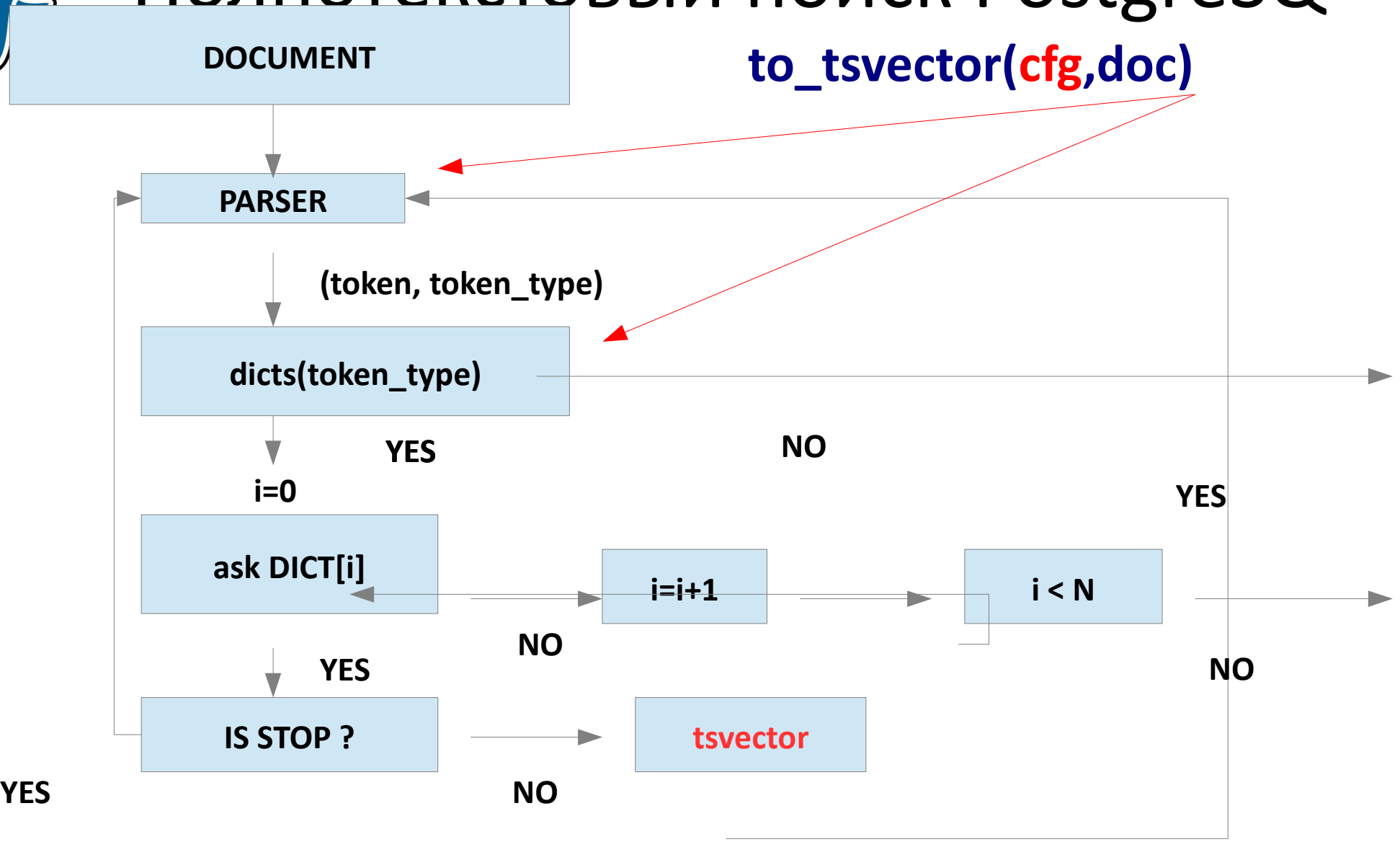
- Полная поддержка ACID
- Online index — GiST, GIN
- Можно использовать для реализации поисковых движков:
  - OpenFTS (openfts.sourceforge.net) — внешний поиск, PostgreSQL используется как хранилище и исполнитель запросов
  - Встроенный поиск в PostgreSQL



# Полнотекстовый поиск PostgreSQL

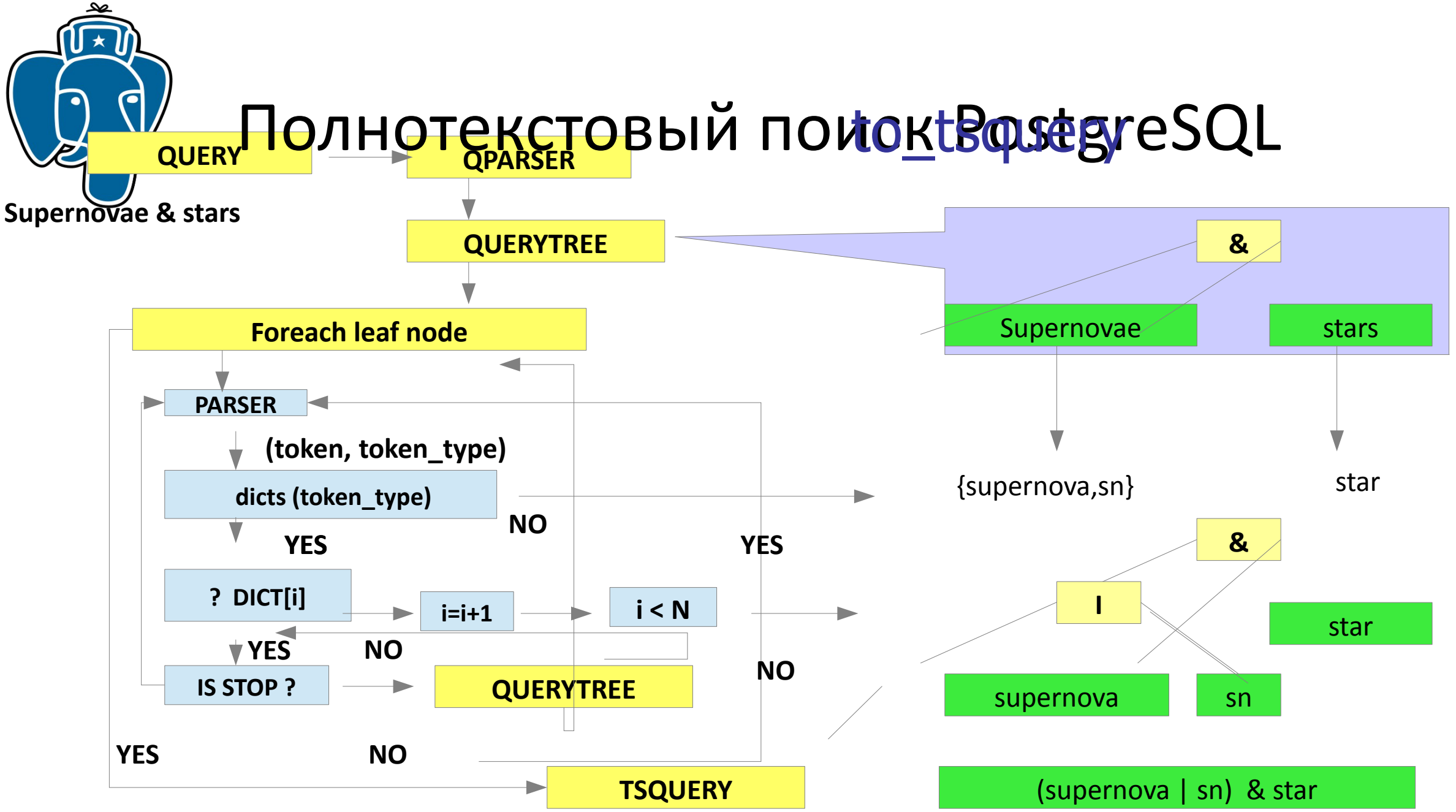
- Программная «обвязка» предоставляет
  - API для словарей, API для парсеров
- SQL интерфейс для настройки поиска
- Встроенная поддержка для всех европейских языков
- Встроенные словари — simple, ispell, stemming, thesaurus, synonym
- Встроенный парсер — 23  типа лексем
- Ранжирование результатов поиска

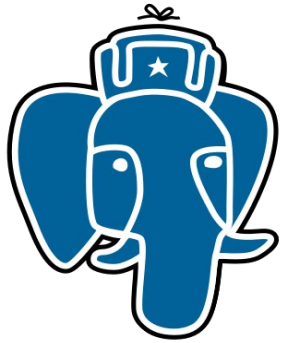
# Полнотекстовый поиск PostgreSQ



# Полнотекстовый поиск в PostgreSQL

to_tsquery

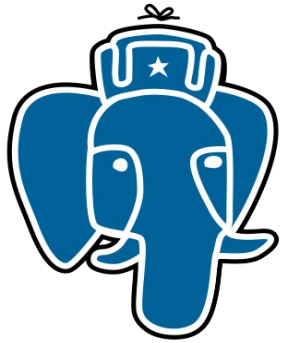




# FTS in PostgreSQL

- Парсер разбивает текст на токены

```
=# select * from ts_token_type('default');
 tokid |      alias      |                description
-------+-----------------+-------------------------------------------
     1 | asciiword       | Word, all ASCII
     2 | word            | Word, all letters
     3 | numword         | Word, letters and digits
     4 | email           | Email address
     5 | url             | URL
     6 | host            | Host
     7 | sfloat          | Scientific notation
     8 | version         | Version number
     9 | hword_numpart   | Hyphenated word part, letters and digits
    10 | hword_part      | Hyphenated word part, all letters
    11 | hword_asciipart | Hyphenated word part, all ASCII
    12 | blank           | Space symbols
    13 | tag             | XML tag
    14 | protocol        | Protocol head
    15 | numhword        | Hyphenated word, letters and digits
    16 | asciihword      | Hyphenated word, all ASCII
    17 | hword           | Hyphenated word, all letters
    18 | url_path        | URL path
    19 | file            | File or path name
    20 | float           | Decimal notation
    21 | int             | Signed integer
    22 | uint            | Unsigned integer
    23 | entity          | XML entity
(23 rows)
```

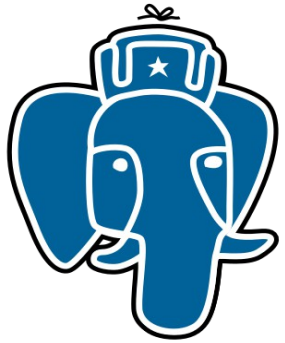


# FTS in PostgreSQL

- Каждый токен  обрабатывается словарями

```
select ts_lexize('english_stem','stars')
---------------------------------------------
          star
```

```
=# \dF+ russian
Text search configuration "pg_catalog.russian"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciihword      | english_stem
 asciiword       | english_stem
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | russian_stem
 hword_asciipart | english_stem
 hword_numpart   | simple
 hword_part      | russian_stem
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | russian_stem
```

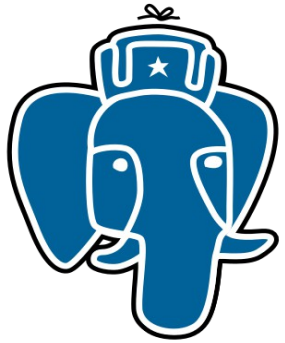


# FTS in PostgreSQL

- Слово передается от словаря к словарю пока оно не распознается.
- Если слово не распознано **всеми** словарями, то оно не индексируется.

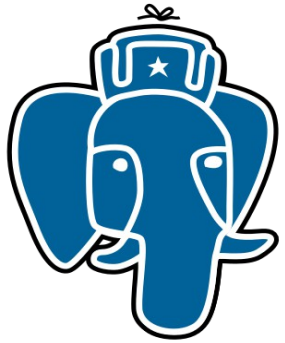
**Правило: от «узкого» словаря к «широкому» !**

```
=# \dF+ pg
                  Configuration "public.pg"
Parser name: "pg_catalog.default"
Locale: 'ru_RU.UTF-8' (default)
    Token     |                    Dictionaries
--------------+-----------------------------------------------------
 file         | pg_catalog.simple
 host         | pg_catalog.simple
 hword        | pg_catalog.simple
 int          | pg_catalog.simple
 lhword       | public.pg_dict,public.en_ispell,pg_catalog.en_stem
 lpart_hword  | public.pg_dict,public.en_ispell,pg_catalog.en_stem
 Lword        | public.pg_dict,public.en_ispell,pg_catalog.en_stem
 nlhword      | pg_catalog.simple
 nlpart_hword | pg_catalog.simple
```



# FTS in PostgreSQL

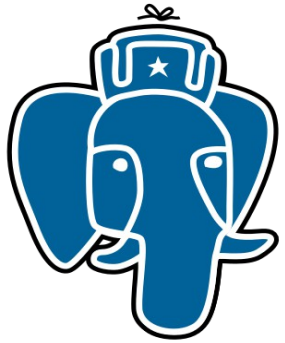
- **Словарь** – это **программа**, которая принимает на вход токен и выдает массив лексем или NULL, если распознанно стоп-слово
- API позволяет писать словари под разные задачи
  - Укорачивать длинные цифры
  - Приводить все обозначения цветов в один вид
  - Приводить URL-и к каноническому виду
- Встроенные словари-заготовки (templates)  для
  - словарей ispell, myspell, hunspell
  - snowball stemmer
  - thesaurus
  - synonym
  - simple



# Словари

- Словарь — это программа !

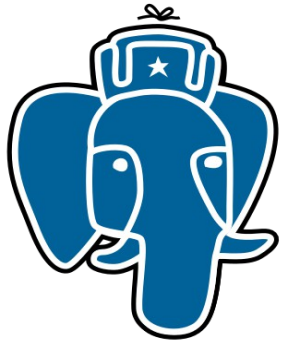
```
=# select ts_lexize('intdict', 11234567890);
  ts_lexize
----------
 {112345}
=# select ts_lexize('roman', 'XIX');
 ts_lexize
-------------
 {19}
=# select ts_lexize('colours','#FFFFFF');
ts_lexize
------------
{white}
```



# Астрономический словарь (arxiv)

## Dictionary with regexp support (pcre library)

```
# Messier objects
(M|Messier)(\s|-)?((\d){1,3}) M$3
# catalogs
(NGC|Abell|MKN|IC|H[DHR]|UGC|SAO|MWC)(\s|-)?((\d){1,6}[ABC]?) $1$3
(PSR|PKS)(\s|-)?([JB]?)(\d\d\d\d)\s?([+-]\d\d)\d? $1$4$5
# Surveys
OGLE(\s|-)?((I){1,3}) ogle
2MASS twomass
# Spectral lines
H(\s|-)?(alpha|beta|gamma) h$2
(Fe|Mg|Si|He|Ni)(\s|-)?((\d)|([IXV])+) $1$3
# GRBs
gamma\s?ray\s?burst(s?) GRB
GRB\s?(\d\d\d\d\d\d)([abcd]?) GRB$1$2
```

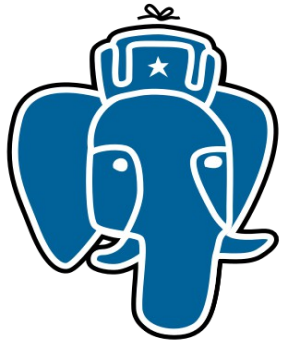


# Dictionaries - interface

```
void* dictInit(List *dictoptions)
```

- list of dictoptions actually contains list of DefElem structures (see headers)
- returns pointer to the palloc'ed dictionary structure
- Can be expensive (ispell)

```
TSLexeme* dictLexize(
    void* dictData,  // returned by dictInit()
    char* lexeme,    // not zero-terminated
    int   lenlexeme,
    DictSubState *substate   // optional
);
```



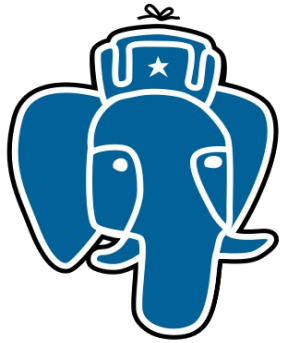
# Dictionaries – output

```
typedef struct {
      uint16      nvariant; // optional
      uint16      flags;    // optional
      char       *lexeme;
  } TSLexeme;
```

dictLexize returns NULL – dictionary doesn't recognize the lexeme

dictLexize returns array of TSLexeme (last element  TSLexeme->lexeme is NULL)

dictLexize returns empty array – dictionary recognizes the lexeme, but it's a stop-word



# Agglutinative Languages

German, norwegian, ...
http://en.wikipedia.org/wiki/Agglutinative_language

Concatenation of words without space

Query - Fotballklubber
Document - Klubb on fotballfield

How to find document ?

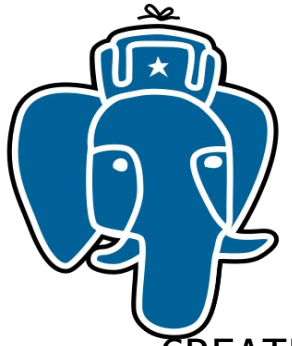
Split words and build search query

'fotbalklubber' =>
' ( fotball & klubb ) | ( fot & ball & klubb ) '

# Filter dictionary – unaccent

contrib/unaccent - unaccent text search dictionary and function to remove accents (suffix tree, ~ 25x faster *translate*() solution)

```
1. Unaccent dictionary does nothing and returns NULL.
(lexeme 'Hotels' will be passed to the next dictionary if any)

=# select ts_lexize('unaccent','Hotels') is NULL;
 ?column?
----------
 t

2. Unaccent dictionary removes accent and returns 'Hotel'.
(lexeme 'Hotel' will be passed to the next dictionary if any)

=# select ts_lexize('unaccent','Hôtel');
 ts_lexize
-----------
 {Hotel}
```

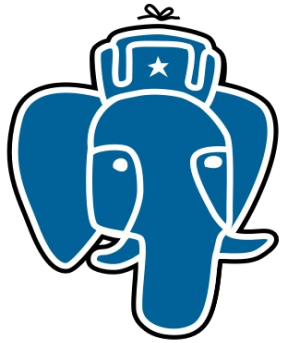


# Filter dictionary - unaccent

```
CREATE TEXT SEARCH CONFIGURATION fr ( COPY = french );
ALTER TEXT SEARCH CONFIGURATION fr ALTER MAPPING FOR hword, hword_part, word
        WITH unaccent, french_stem;


=# select to_tsvector('fr','Hôtel de la Mer') @@ to_tsquery('fr','Hotels');
 ?column?
----------
 t
```

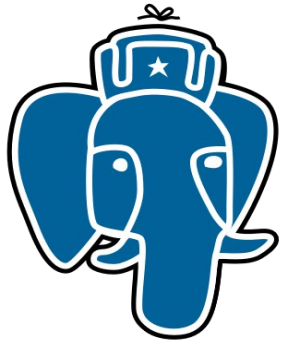
Finally, unaccent dictionary solves the known problem with headline !
( to_tsvector(remove_accent(document)) works with search, but
has problem with highlighting )

```
=# select ts_headline('fr','Hôtel de la Mer',to_tsquery('fr','Hotels'));
      ts_headline
------------------------
  <b>Hôtel</b> de la Mer
```



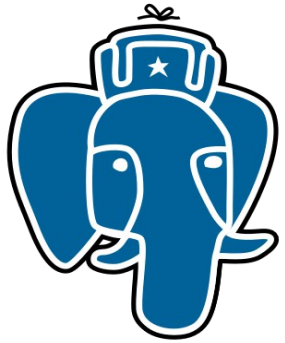
# Synonym dictionary with prefix search support

```
cat $SHAREDIR/tsearch_data/synonym_sample.syn
postgres        pgsql
postgresql      pgsql
postgre pgsql
gogle   googl
indices index*

=# create text search dictionary syn
( template=synonym,synonyms='synonym_sample');
=# select ts_lexize('syn','indices');
 ts_lexize
-----------
 {index}
```



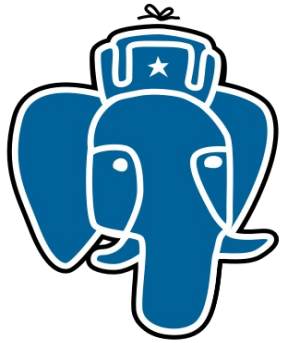
# Synonym dictionary with prefix search support

```
=# create text search configuration tst ( copy=simple);
=# alter text search configuration tst alter mapping
     for asciiword with syn;

=# select to_tsquery('tst','indices');
 to_tsquery
------------
 'index':*
=# select 'indexes are very useful'::tsvector @@  to_tsquery('tst','indices');
 ?column?
----------
 t
```



# dict_xsyn

- How to search for  'William' and any synonyms 'Will', 'Bill', 'Billy' ?  We can:
  - Index only synonyms
  - Index  synonyms and original name
  - Index only original name  - replace all synonyms.  Index size is minimal, but s*earch for specific name is  impossible*.



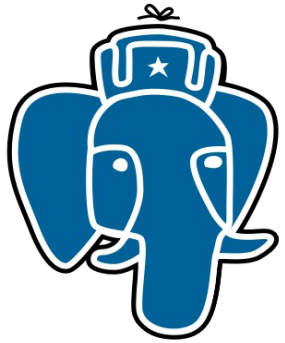
# dict_xsyn

- Old version of dict_xsyn can return only list of synonyms. It's possible to prepare synonym file to support other options:

```
William Will Bill Billy
Will William Bill Billy
Bill William Will Billy
Billy William Will Bill
```

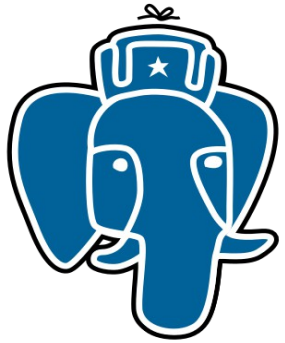
- New dict_xsyn (Sergey Karpov) allows better control:

```
CREATE TEXT SEARCH DICTIONARY xsyn (RULES='xsyn_sample',
KEEPORIG=false|true, mode='SIMPLE|SYMMETRIC|MAP');
```



# dict_xsyn

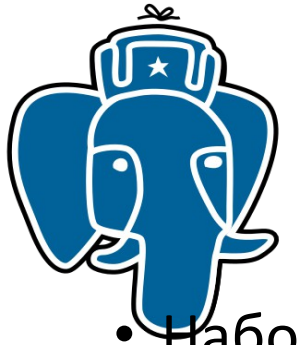
- Mode SIMPLE - accepts the original word and returns all synonyms as OR-ed list. This is default mode.
- Mode SYMMETRIC - accepts the original word **or any** of its synonyms, and return all others as OR-ed list.
- Mode MAP - accepts any synonym and returns the original word.



# dict_xsyn

**EXAMPLES:**

```
=# ALTER TEXT SEARCH DICTIONARY xsyn (RULES='xsyn_sample',
KEEPORIG=false, mode='SYMMETRIC');

=# select ts_lexize('xsyn','Will') as Will,
          ts_lexize('xsyn','Bill') as Bill,
          ts_lexize('xsyn','Billy') as Billy;

          will          |          bill          |         billy
------------------------+------------------------+-----------------------
 {william,bill,billy} | {william,will,billy} | {william,will,bill}

Mode='MAP'

   will     |   bill     |   billy
------------+------------+------------
 {william} | {william} | {william}
```



# FTS in PostgreSQL

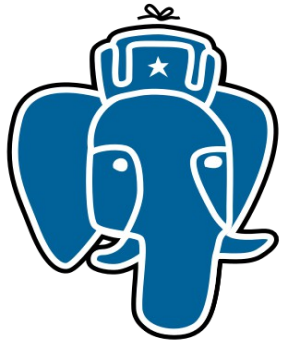
- Набор функций для получения tsvector и tsquery
  - to_tsvector(ftscfg, text)
  - to_tsquery(ftscfg, text)

```
=# select to_tsvector('english', 'as supernovae stars');
    to_tsvector
----------------------
 'star':3 'supernova':2
```

стоп-слово

position

```
=# select * from ts_debug('english', 'a supernovae stars');
   alias   |   description   |   token    |  dictionaries  |  dictionary  |   lexemes
-----------+-----------------+------------+----------------+--------------+-------------
 asciiword | Word, all ASCII | a          | {english_stem} | english_stem | {}
 blank     | Space symbols   |            | {}             |              |
 asciiword | Word, all ASCII | supernovae | {english_stem} | english_stem | {supernova}
 blank     | Space symbols   |            | {}             |              |
 asciiword | Word, all ASCII | stars      | {english_stem} | english_stem | {star}
(5 rows)
```



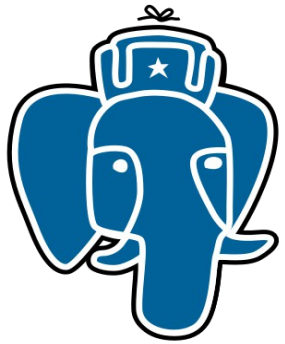
# FTS configuration

- FTS конфигурация определяет
  - какой парсер используется для разбивания текста на токены
  - какие токены, какими словарями и в каком порядке обрабатываются
- Конфигурация задается с помощью SQL команд

```
{CREATE | ALTER | DROP} TEXT SEARCH {CONFIGURATION | DICTIONARY | PARSER}
```

- FTS конфигураций может быть много, поддерживаются схемы
- Информация о конфигурации доступна в psql

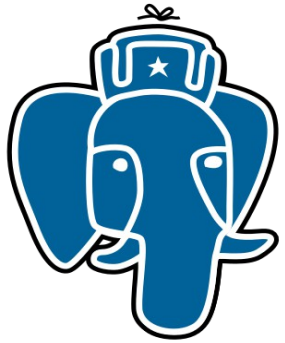
**\dF{,d,p}[+] [PATTERN]**



# FTS configuration

**16 конфигураций для 15 языков**

```
=# \dF
                  List of text search configurations
   Schema   |    Name    |                  Description
------------+------------+-----------------------------------
 pg_catalog | danish     | configuration for danish language
 pg_catalog | dutch      | configuration for dutch language
 pg_catalog | english    | configuration for english language
 pg_catalog | finnish    | configuration for finnish language
 pg_catalog | french     | configuration for french language
 pg_catalog | german     | configuration for german language
 pg_catalog | hungarian  | configuration for hungarian language
 pg_catalog | italian    | configuration for italian language
 pg_catalog | norwegian  | configuration for norwegian language
 pg_catalog | portuguese | configuration for portuguese language
 pg_catalog | romanian   | configuration for romanian language
 pg_catalog | russian    | configuration for russian language
 pg_catalog | simple     | simple configuration
 pg_catalog | spanish    | configuration for spanish language
 pg_catalog | swedish    | configuration for swedish language
 pg_catalog | turkish    | configuration for turkish language
(16 rows)
```



# Full-text search tips

- Stable to_tsquery
- Find documents with specific token type
- Getting words from tsvector
- Confuse with text search
- Antimat constraint
- APOD example (ts_headline, query rewriting)
- FTS without tsvector column
- Strip tsvector
- Fast approximated statistics

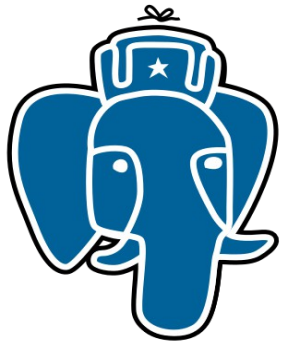
# Stable to_tsquery

Result of to_tsquery() can't be used as a cache key,
since to_tsquery() does preserve an order,
which isn't good for cacheing.

Little function helps:

```
CREATE OR REPLACE FUNCTION stable_ts_query(tsquery)
RETURNS tsquery AS
$$
   SELECT ts_rewrite( $1 , 'dummy_word', 'dummy_word');
$$
LANGUAGE SQL RETURNS NULL ON NULL INPUT IMMUTABLE;
```

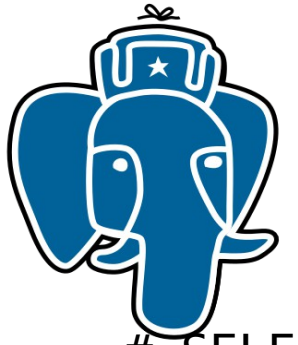
Note:  Remember about text search
configuraton to have really good cache key !



# Find documents with specific token type

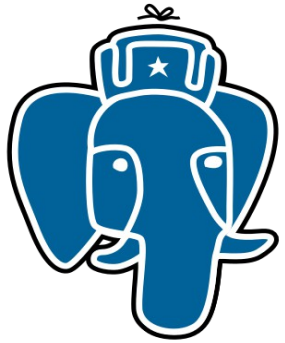
How to find documents, which contain emails ?

```
CREATE OR REPLACE FUNCTION document_token_types(text)
RETURNS _text AS
$$

SELECT ARRAY (
    SELECT
        DISTINCT alias
    FROM
        ts_token_type('default') AS tt,
        ts_parse('default', $1) AS tp
    WHERE
        tt.tokid = tp.tokid
    );
$$ LANGUAGE SQL immutable;
```



# Find documents with specific token type

```
=# SELECT document_token_types(title) FROM papers
        LIMIT 10;
                      document_token_types
-------------------------------------------------------------
 {asciihword,asciiword,blank,hword_asciipart}
 {asciiword,blank}
 {asciiword,blank}
 {asciiword,blank}
 {asciiword,blank}
 {asciiword,blank,float,host}
 {asciiword,blank}
 {asciihword,asciiword,blank,hword_asciipart,int,numword,uint}
 {asciiword,blank}
 {asciiword,blank}
(10 rows)

CREATE INDEX fts_types_idx ON papers USING
                        gin( document_token_types (title) );
```



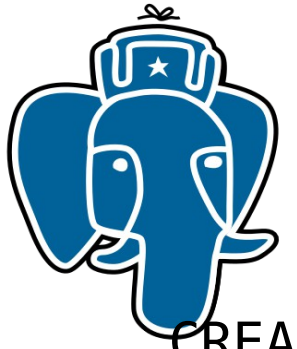
# Find documents with specific token type

How to find documents, which contain emails ?

```
SELECT comment FROM papers
WHERE document_token_types(title) && '{email}';
```

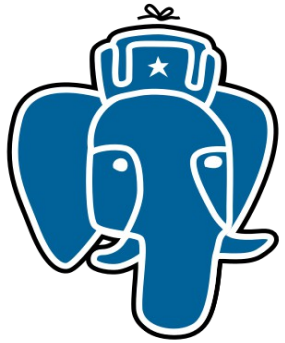
The list of available token types:

```
SELECT * FROM ts_token_type('default');
```



# Getting words from tsvector

```
CREATE OR REPLACE FUNCTION ts_stat(tsvector, OUT word text,
OUT ndoc integer, OUT nentry integer)
RETURNS SETOF record AS $$
SELECT ts_stat('SELECT ' || quote_literal( $1::text )
                                || '::tsvector');
$$ LANGUAGE SQL RETURNS NULL ON NULL INPUT IMMUTABLE;

SELECT id, (ts_stat(fts)).* FROM apod WHERE id=1;
 id |    word    | ndoc | nentry
----+------------+------+--------
  1 | 1          |    1 |      1
  1 | 2          |    1 |      2
  1 | io         |    1 |      2
  1 | may        |    1 |      1
  1 | new        |    1 |      1
  1 | red        |    1 |      1
  1 | two        |    1 |      1
```



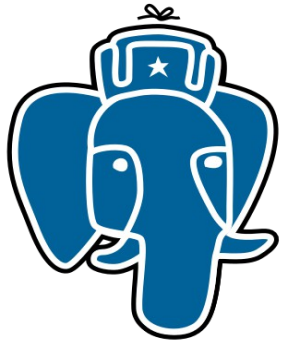
# Confuse with text search

One expected **true** here, but result is disappointing **false**

```
=# select to_tsquery('ob_1','inferences') @@
            to_tsvector('ob_1','inference');
 ?column?
----------
 f
```

Use ts_debug() to understand the problem

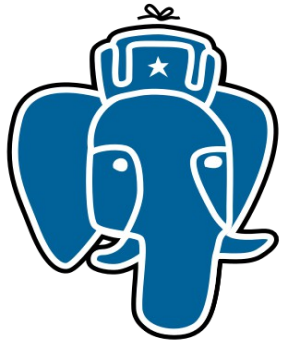
```
'inferences':
{french_ispell,french_stem} | french_stem   | {inferent}

'inference':
{french_ispell,french_stem} | french_ispell | {inference}
```



# Confuse with text search

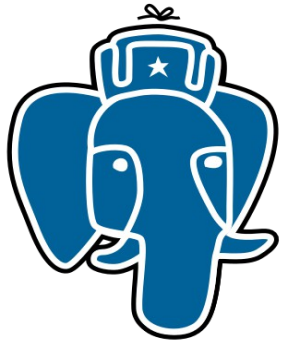
- Use synonym dictionary as a first dictionary {synonym,french_ispell,french_stem}

  with rule '`inferences inference`'
  - Don't forget to reindex !
- Use ts_rewrite()
  - Don't need to reindex



# Antimat constraint

```
CREATE TABLE nomat (i int, t text,
   CHECK (NOT (to_tsvector(t) @@ 'f.ck'::tsquery))
);

=# INSERT INTO nomat(i,t) VALUES(1,'f.ck him');

ERROR: new row for relation "nomat" violates check constraint
"nomat_t_check"
DETAIL: Failing row contains (1, f.ck him).

=# INSERT INTO nomat(i,t) VALUES(1,'f.cking him');
ERROR: new row for relation "nomat" violates check constraint
"nomat_t_check"
DETAIL: Failing row contains (1, f.cking him).
=# INSERT INTO nomat(i,t) VALUES(1,'kiss him');
INSERT 0 1
```

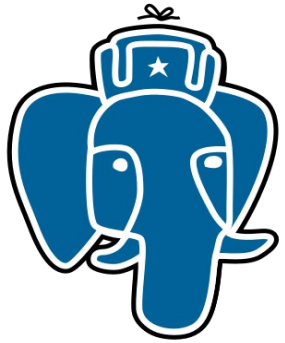


# APOD example

**http://www.astronet.ru/db/apod.html**

- curl -O http://www.sai.msu.su/~megera/postgres/fts/apod.dump.gz
- zcat apod.dump.gz | psql postgres
- psql postgres

```
postgres=# \d apod
        Table "public.apod"
  Column  |   Type    | Modifiers
----------+-----------+-----------
 id       | integer   | not null
 title    | text      |
 body     | text      |
 sdate    | date      |
 keywords | text      |

postgres=# show default_text_search_config;
    default_text_search_config
-------------------------------
 pg_catalog.russian
```



# APOD example: FTS configuration

```
=# \dF+ russian
Text search configuration "pg_catalog.russian"
Parser: "pg_catalog.default"
      Token      | Dictionaries
-----------------+--------------
 asciihword      | english_stem
 asciiword       | english_stem
 email           | simple
 file            | simple
 float           | simple
 host            | simple
 hword           | russian_stem
 hword_asciipart | english_stem
 hword_numpart   | simple
 hword_part      | russian_stem
 int             | simple
 numhword        | simple
 numword         | simple
 sfloat          | simple
 uint            | simple
 url             | simple
 url_path        | simple
 version         | simple
 word            | russian_stem
```

# APOD example: FTS index

```
postgres=# alter table apod add column fts tsvector;

postgres=# update apod  set fts=
                                setweight( coalesce( to_tsvector(title),''),'B')          ||
                                setweight( coalesce( to_tsvector(keywords),''),'A') ||
                                setweight( coalesce( to_tsvector(body),''),'D');
```

**if NULL then ''**

**NULL || nonNULL => NULL**

```
postgres=# create index apod_fts_idx on apod using gin(fts);
postgres=# vacuum analyze apod;

postgres=# select title from apod where fts  @@ plainto_tsquery('supernovae stars') limit 5;
                title
-----------------------------------------
 Runaway Star
 Exploring The Universe With IUE 1978-1996
 Tycho Brahe Measures the Sky
 Unusual Spiral Galaxy M66
 COMPTEL Explores The Radioactive Sky
```

# APOD example: Search

```
postgres=# select title,ts_rank_cd(fts, q)as rank from apod,
to_tsquery('supernovae & x-ray') q
where fts  @@ q order by rank_cd desc limit 5;
```

| title | rank |
|---|---|
| Supernova Remnant E0102-72 from Radio to X-Ray | 1.59087 |
| An X-ray Hot Supernova in M81 | 1.47733 |
| X-ray Hot Supernova Remnant in the SMC | 1.34823 |
| Tycho's Supernova Remnant in X-ray | 1.14318 |
| Supernova Remnant and Neutron Star | 1.08116 |

(5 rows)

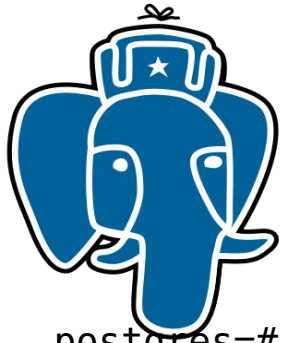
Time: 1.965 ms

**ts_rank_cd не нормирован, так как используется только локальная информация !**

**0 < rank/(rank+1) < 1**

**ts_rank_cd('{0.1, 0.2, 0.4, 1.0 }',fts, q)**

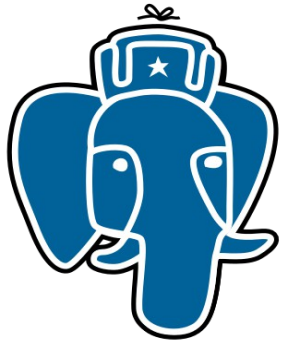
**D C B A**



# APOD example: headline

```
postgres=# select ts_headline(body,q,'StartSel=<,StopSel=>,MaxWords=10,MinWords=5'),
ts_ank_cd(fts, q) from apod, to_tsquery('supernovae & x-ray') q where fts  @@
q order by rank_cd desc limit 5;

                              headline                                | ts_rank_cd
----------------------------------------------------------------------+----------
 <supernova> remnant  E0102-72, however, is giving astronomers a clue | 1.59087
 <supernova> explosion. The picture was taken in  <X>-<rays>          | 1.47733
  <X>-<ray> glow is produced by  multi-million degree                 | 1.34823
  <X>-<rays> emitted by this shockwave made by a telescope            | 1.14318
  <X>-<ray> glow. Pictured is the <supernova>                         | 1.08116
(5 rows)

Time: 39.298 ms
```

**Медленно** ! Надо использовать **subselect**. Об этом подробнее в советах.

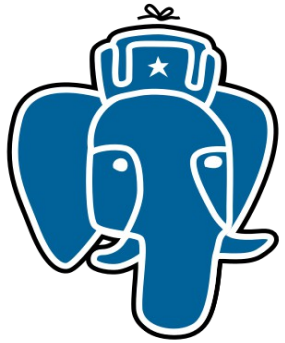


# APOD example

- Используя один индекс можно иметь разные поиски
  - поиск только в заголовках – поиск среди лексем, маркированных «важностью» 'b'.

```
=# SELECT title,ts_rank_cd(fts, q) AS rank FROM apod,
to_tsquery('supernovae:b & x-ray') q
WHERE fts  @@ q ORDER BT rank_cd DESC LIMIT 5;

                      title                       | rank
--------------------------------------------------+----------
 Supernova Remnant E0102-72 from Radio to X-Ray | 1.59087
 An X-ray Hot Supernova in M81                  | 1.47733
 X-ray Hot Supernova Remnant in the SMC         | 1.34823
 Tycho's Supernova Remnant in X-ray             | 1.14318
 Supernova Remnant and Neutron Star             | 1.08116
(5 rows)
```

**to_tsquery('supernovae:ab')** - поиск среди заголовков и ключевых слов

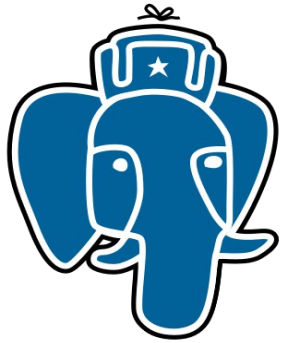


# FTS without tsvector column

- Use functional index (GiST or GiN)
  - no ranking, use other ordering

```
create index gin_text_idx on test using gin (
( coalesce(to_tsvector(title),'') || coalesce(to_tsvector(body),'') )
);
```

```
apod=# select title from test where
(coalesce(to_tsvector(title),'') || coalesce(to_tsvector(body),'') ) @@
to_tsquery('supernovae') order by sdate desc limit 10;
```



# FTS tips

```
select id,ts_headline(body,q),ts_rank(fts,q) as rank
from apod, to_tsquery('stars') q
where fts @@ q order by rank desc limit 10;

Time: 723.634 ms
```

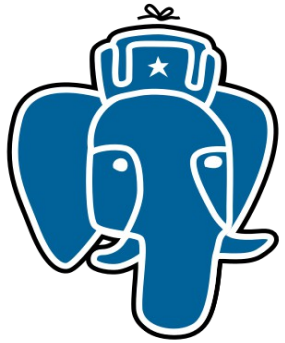
10 times !

```
select id,ts_headline(body,q),ts_rank from (
  select id,body,q, rank(fts,q) as rank from apod,
to_tsquery('stars') q
  where fts @@ q order by rank desc limit 10
) as foo;

Time: 21.846 ms
```

```
=#select count(*)from apod where fts @@ to_tsquery('stars');
 count
-------
   790
```

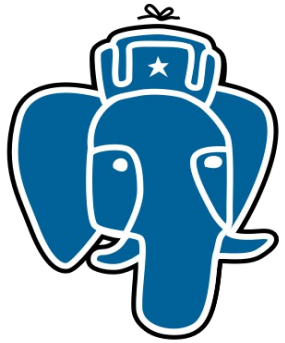
790 times

- ts_headline() функция медленная – используйте **subselect**



# FTS tips – Query rewriting

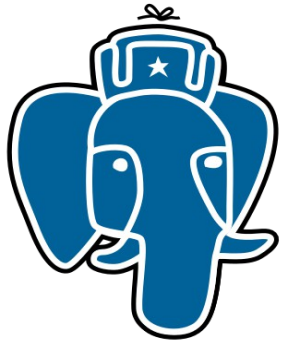
- Изменение запроса **online**
  - расширение запроса
    - синонимы ( new york => Gottham, Big Apple, ...)
  - Сужение запроса
    - Курск => подводная лодка Курск
- Похоже на словарь тезаурус (синонимов), но не требует переиндексации



# FTS tips – Query rewriting

**ts_rewrite (tsquery, tsquery, tsquery)**
**ts_rewrite (ARRAY[tsquery,tsquery,tsquery]) from aliases**
**ts_rewrite (tsquery,'select tsquery,tsquery from aliases')**

```
create table aliases( t tsquery primary key, s tsquery);

insert into aliases values(to_tsquery('supernovae'),
to_tsquery('supernovae|sn'));

apod=# select ts_rewrite(to_tsquery('supernovae'),
'select * from aliases');

      ts_rewrite
---------------------
 'supernova' | 'sn'
```

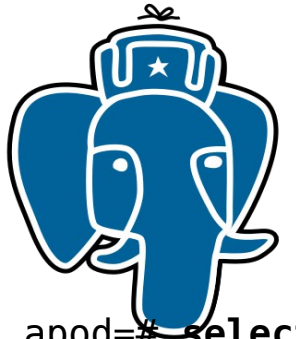


# FTS tips – Query rewriting

```
apod=# select title, coalesce(ts_rank_cd(fts,q,1),2) as rank
from apod, to_tsquery('supernovae') q
where fts @@ q  order by rank desc limit 10;
```

| title | rank |
|---|---|
| The Mysterious Rings of Supernova 1987A | 0.669633 |
| Tycho's Supernova Remnant in X-ray | 0.598556 |
| Tycho's Supernova Remnant in X-ray | 0.598556 |
| Vela Supernova Remnant in Optical | 0.591655 |
| Vela Supernova Remnant in Optical | 0.591655 |
| Galactic Supernova Remnant IC 443 | 0.590201 |
| Vela Supernova Remnant in X-ray | 0.589028 |
| Supernova Remnant: Cooking Elements In The LMC | 0.585033 |
| Cas A Supernova Remnant in X-Rays | 0.583787 |
| Supernova Remnant N132D in | **0.579241** |

**Lower limit**



# FTS tips – Query rewriting

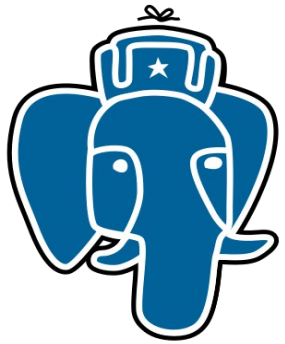
```
apod=# select id, title, coalesce(ts_rank_cd(fts,q,1),2) as rank
from apod, ts_rewrite(to_tsquery('supernovae'), 'select * from aliases') q
where fts @@ q  order by rank desc limit 10;
   id    |                   title                    |   rank
---------+--------------------------------------------+----------
 1162701 | The Mysterious Rings of Supernova 1987A    |  0.90054
 1162717 | New Shocks For Supernova 1987A             | 0.738432
 1163673 | Echos of Supernova 1987A                   | 0.658021
 1163593 | Shocked by Supernova 1987a                 | 0.621575
 1163395 | Moving Echoes Around SN 1987A              | 0.614411
 1161721 | Tycho's Supernova Remnant in X-ray         | 0.598556
 1163201 | Tycho's Supernova Remnant in X-ray         | 0.598556
 1163133 | A Supernova Star-Field                     | 0.595041
 1163611 | Vela Supernova Remnant in Optical          | 0.591655
 1161686 | Vela Supernova Remnant in Optical          | 0.591655
```

```
apod=# select  title, coalesce(rank_cd(fts,q,1),2) as rank
from apod, to_tsquery('supernovae') q
where fts @@ q and id=1162717;
```

```
            title             |   rank
------------------------------+----------
 New Shocks For Supernova 1987A | 0.533312
```

**Old rank**

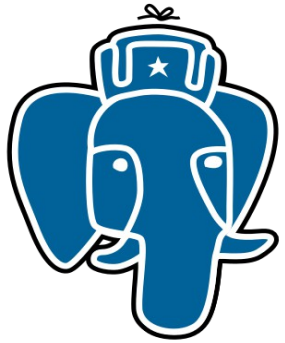
**new document**



# FTS tips — strip tsvector

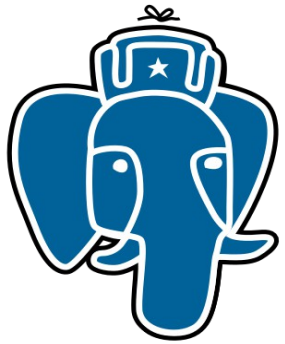
- Если не нужна релевантность, то позиционная информация не нужна — можно иметь индекс сильно меньше !

```
postgres=# select to_tsvector('w1 w3 w1 w3');
     to_tsvector
--------------------
 'w1':1,3 'w3':2,4
(1 row)

Time: 0.268 ms
postgres=# select strip(to_tsvector('w1 w3 w1 w3'));
   strip
-----------
 'w1' 'w3'
(1 row)
```



# Fast approximated statistics

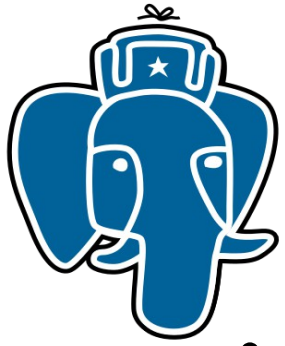
- Gevel extension — GiST/GIN indexes explorer (http://www.sai.msu.su/~megera/wiki/Gevel)
- **Fast** — uses only GIN index (no table access)
- **Approximated** — no table access, which contains visibility information, approx. for long posting lists
- For mostly **read-only** data error is small



# Fast approximated statistics

- Top-5 most frequent words (463,873 docs)

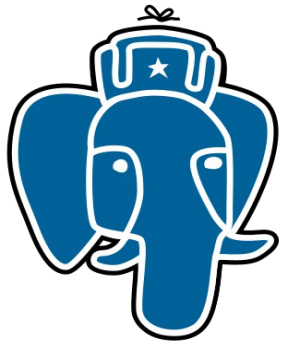
```
=# SELECT * FROM gin_stat('gin_idx') as t(word text, ndoc int)  order by
ndoc desc limit 5;

  word   |  ndoc
---------+--------
 page    | 340858
 figur   | 240366
 use     | 148022
 model   | 134442
 result  | 129010
(5 rows)
Time: 520.714 ms
```



# Fast approximated statistics

- gin_stat() vs ts_stat()

```
=# select * into stat from ts_stat('select fts from papers') order by ndoc desc, nentry
desc,word;

...wait.... 68704,182 ms

=# SELECT a.word, b.ndoc as exact, a.estimation as estimation,
round ( (a.estimation-b.ndoc)*100.0/a.estimation,2)||'%' as error
FROM (SELECT * FROM gin_stat('gin_x_idx') as t(word text, estimation int)  order by
estimation desc limit 5 ) as a, stat b
WHERE a.word = b.word;

  word   | exact  | estimation | error
--------+--------+------------+-------
 page    | 340430 |     340858 | 0.13%
 figur   | 240104 |     240366 | 0.11%
 use     | 147132 |     148022 | 0.60%
 model   | 133444 |     134442 | 0.74%
 result  | 128977 |     129010 | 0.03%
(5 rows)
Time: 550.562 ms
```



# GIN Improvements

- Store additional information (compression)(positions for FTS, array length for arrays...)
  Можно вычислять релевантность в индексе
- Output **ordered** results from index

  (no heap scan!)
- Optimize execution of (rare & frequent) query  commited 9.4
  было:
  T(freq & rare)=T(rare & freq) ~ T(freq)

  стало:

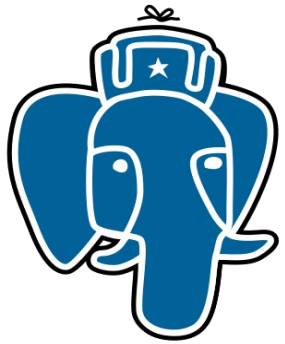
  T(freq & rare)>>T(rare & freq) ~ T(rare)

# 20 mln descriptions

| | Without patch | With patch | With patch functional index | Sphinx |
|---|---|---|---|---|
| Table size | 18.2 GB | 18.2 GB | 11.9 GB | - |
| Index size | 2.28 GB | 2.30 GB | 2.30 GB | 3.09 GB |
| Index build time | 258 sec | 684 sec | 1712 sec | 481 sec* |
| **Queries in 8 hours** | **2.67 mln.** | **38.7 mln.** | **38.7 mln.** | **26.7 mln.** |

# 6.7 mln classifieds

| | Without patch | With patch | With patch functional index | Sphinx |
|---|---|---|---|---|
| Table size | 6.0 GB | 6.0 GB | 2.87 GB | - |
| Index size | 1.29 GB | 1.27 GB | 1.27 GB | 1.12 GB |
| Index build time | 216 sec | 303 sec | 718sec | 180 sec* |
| **Queries in 8 hours** | **3,0 mln.** | **42.7 mln.** | **42.7 mln.** | **32.0 mln.** |

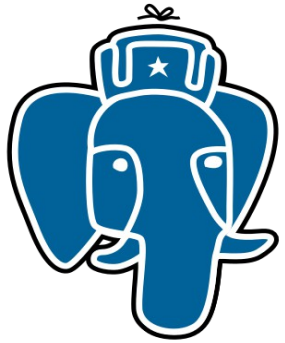


# Phrase Search (waiting for support)

- Запросы 'A & B'::tsquery и 'B & A'::tsquery дадут одинаковый результат
- Иногда хочется искать с учетом порядка — поиск фразы
- Новый оператор $ (A $ B): word 'A' followed by 'B'
  - A & B (the same priority)
  - exists at least one pair of positions $P_B$, $P_A$ , so that $0 \leq P_B - P_A \leq 1$ (distance condition)

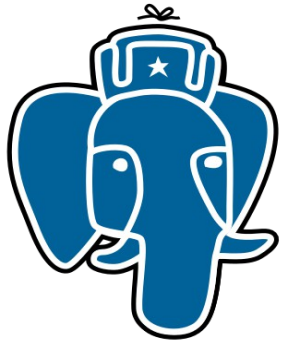
  A $[n] B: $0 \leq P_B - P_A \leq n$

  $A \ \$ \ B \neq B \ \$ \ A$



# Phrase search - transformation

```
# select '( A | B ) $ ( D | C )'::tsquery;
                        tsquery
  ----------------------------------------
   'A' $ 'D' | 'B' $ 'D' | 'A' $ 'C' | 'B' $ 'C'


# select 'A $ ( B & ( C | ! D ) )'::tsquery;
                           tsquery
  ---------------------------------------------------
   ( 'A' $ 'B' ) & ( 'A' $ 'C' | 'A' & !( 'A' $ 'D' ) )
```



# Phrase search - example

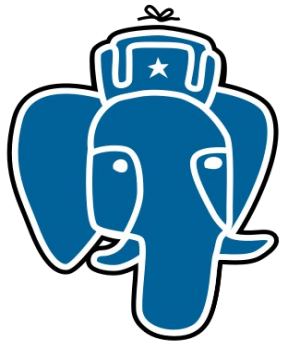
'PostgreSQL can be extended by the user in many ways' ->

```
# SELECT phraseto_tsquery('PostgreSQL can be extended
                          by the user in many ways');
                            phraseto_tsquery
  ------------------------------------------------------------------
'postgresql' $[3] ( 'extend' $[3] ( 'user' $[2] ( 'mani' $ 'way' ) ) )
```

Can be written by hand:

```
'postgresql' $[3] extend $[6] user $[8] mani $[9] way
```

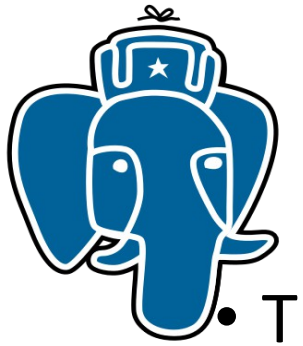
Difficult to modify, use phraseto_tsquery() function !



# SP-GiST (9.2)

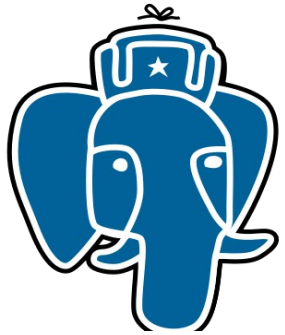
## Space-partitioning trees in PostgreSQL

Challenge: in-memory ADT → page-oriented storage



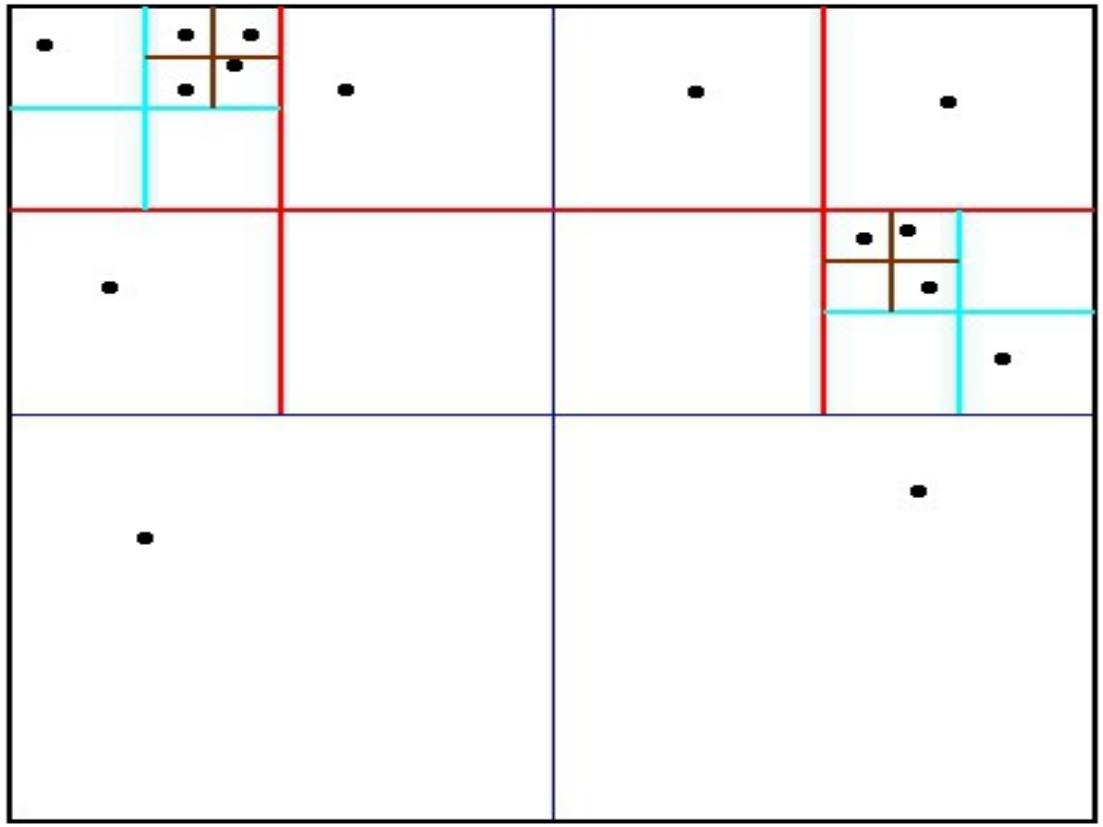
# PostgreSQL extensibility

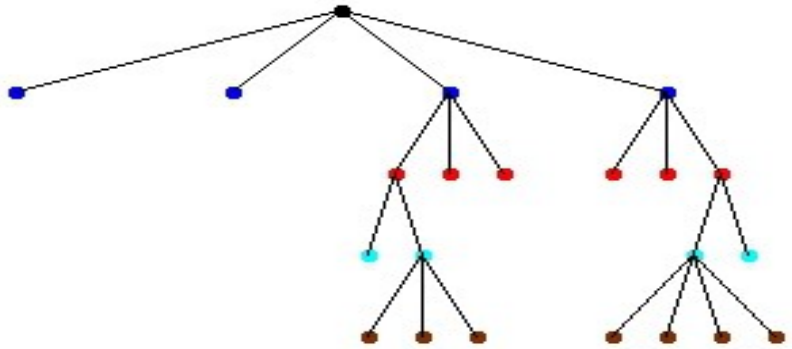
- There are many interesting data structures not available
  - K-D-tree, Quadtree and many variants
    - CAD, GIS, multimedia
  - Tries, suffix tree and many variants
    - Phone routing, ip routing, substring search
- Common features:
  - Decompose space into disjoint partitions
    - Quadtree – 4 quadrants
    - Suffix tree – 26 regions (for english alphabet)
  - Unbalanced trees

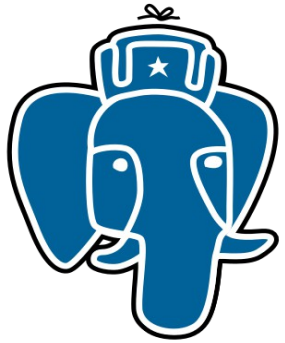


# Quadtree

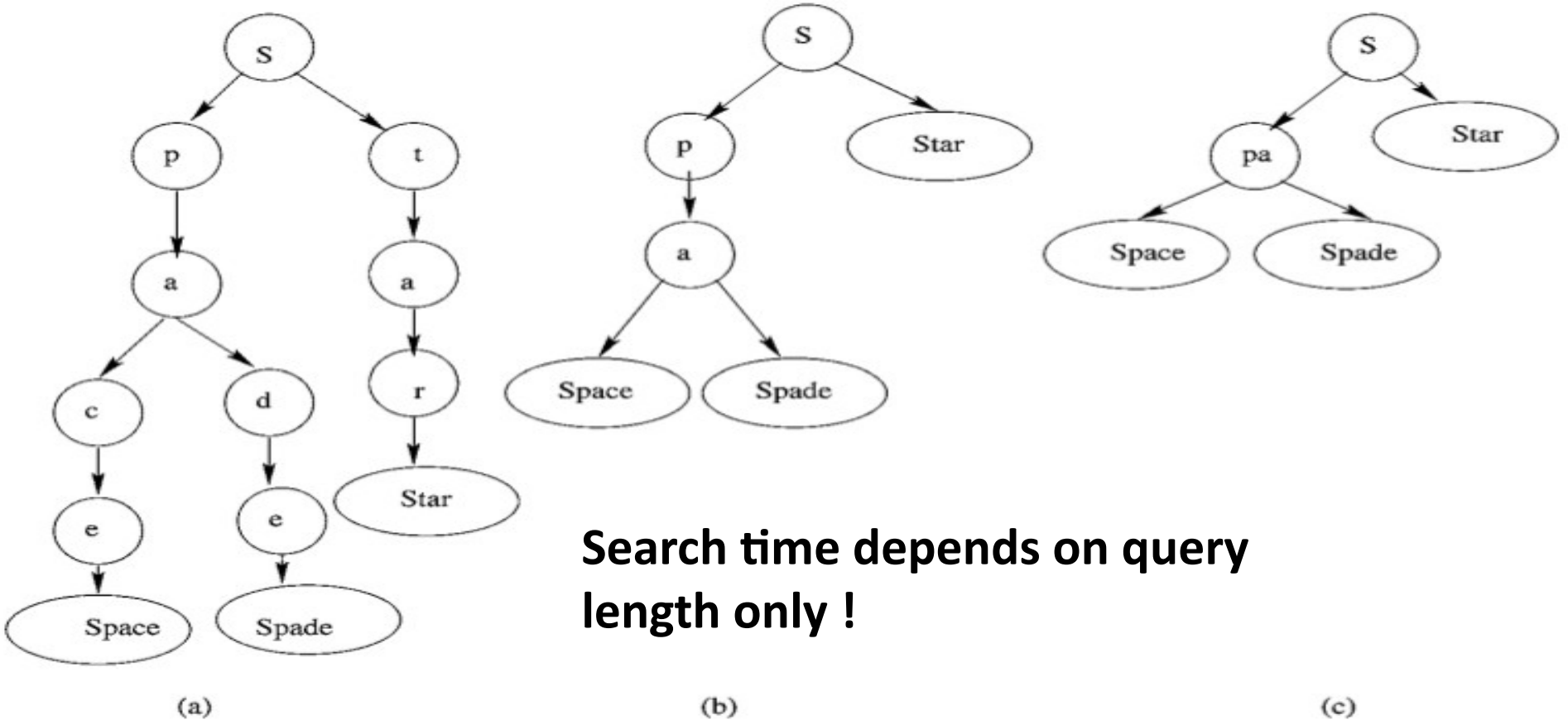
- (



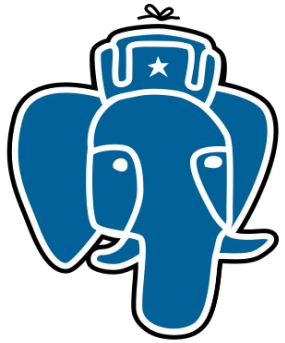




# Suffix tree

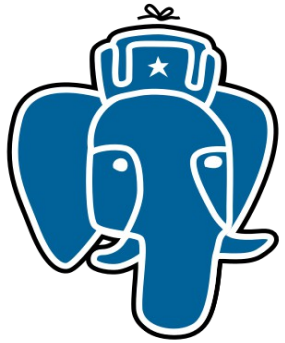



**Search time depends on query length only !**



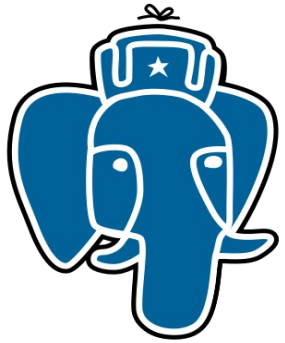
# SP-GiST

- GiST was inspired by R-tree and  doesn't supports unbalanced trees
- We need  new indexing framework for Spatial Partitioning trees:
  - Provide internal methods,  which are common for whole class of space partitioning trees
  - Provide API for implementation specific features of data type



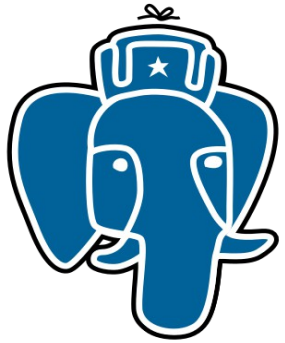
# SP-GiST

- Big Problem – Space Partitioning trees are in-memory structures and not suitable for page-oriented storage
- Several approaches:
  1. Adapt structure for disk storage – difficult and not generalized
  2. Introduce non-page oriented storage in Postgres - No way !
  3. **Add node clustering to utilize page space on disk and preserve locality (path nodes stored close)**



# Quadtree implementation

- Prefix and leaf predicate are points, node predicate is short number
- SplitFn() - just form a centroid and 4 nodes (quadrants)
- ChooseFn() - choose a quadrant (no AddNode, no split tuple)
- InnerConsistentFn() - choose quadrant(s)
- LeafConsistentFn – simple equality
- **179 lines of code**

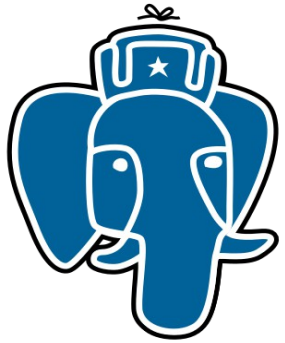


# Quadtree

- Table geo (points) : 2045446 points from US geonames
  Size: 293363712

**SeqScan**:

```
knn=# explain (analyze on, buffers on) select point from geo
where point ~= '(34.34898,-92.82934)';
                Abco (Arkansas,County of Hot Spring)

 Seq Scan on geo  (cost=0.00..36626.31 rows=10228 width=16)
(actual time=0.027..286.088 rows=1 loops=1)
   Filter: (point ~= '(34.34898,-92.82934)'::point)
   Buffers: shared hit=11057
 Total runtime: 286.118 ms
(4 rows)

Time: 286.659 ms
```



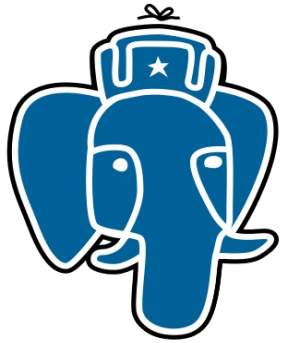
# Quadtree

- Table geo (points) : 2045446 points from US geonames
- GiST

```
knn=# create index pt_gist_idx on geo using gist(point);
CREATE INDEX
Time:  36672.283 ms
Size: 153,124,864
```

- SP-GiST

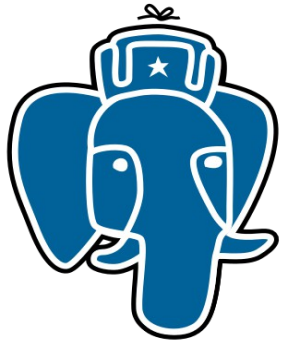
```
knn=# create index pt_spgist_idx on geo using spgist(point);
CREATE INDEX
Time: 12805.530 ms  ~ 3 times faster !
Size: 153,788,416     ~ the same size
```



# Quadtree

- GiST

```
knn=# explain (analyze on, buffers on) select point from geo where point ~=
'(34.34898,-92.82934)';

 Bitmap Heap Scan on geo  (cost=456.26..11872.18 rows=10227 width=16) (actual
time=0.188..0.188 rows=1 loops=1)
   Recheck Cond: (point ~= '(34.34898,-92.82934)'::point)
   Buffers: shared hit=12
   ->  Bitmap Index Scan on pt_gist_idx  (cost=0.00..453.70 rows=10227 width=0)
(actual time=0.179..0.179 rows=1 loops=1)
         Index Cond: (point ~= '(34.34898,-92.82934)'::point)
         Buffers: shared hit=11
 Total runtime: 0.235 ms
```



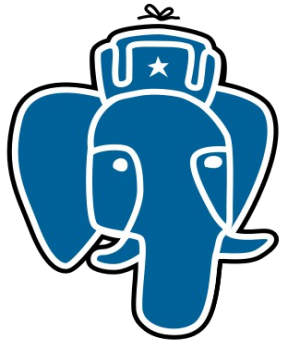
# Quadtree

- SP-GiST

```
knn=# explain (analyze on, buffers on) select point from geo where point ~=
'(34.34898,-92.82934)';

 Bitmap Heap Scan on geo  (cost=576.50..11992.42 rows=10227 width=16) (actual
time=0.041..0.041 rows=1 loops=1)
   Recheck Cond: (point ~= '(34.34898,-92.82934)'::point)
   Buffers: shared hit=6
   ->  Bitmap Index Scan on pt_spgist_idx  (cost=0.00..573.94 rows=10227 width=0)
(actual time=0.033..0.033 rows=1 loops=1)
         Index Cond: (point ~= '(34.34898,-92.82934)'::point)
         Buffers: shared hit=5
 Total runtime: 0.083 ms
```

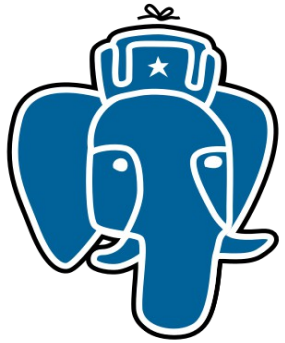
**~ 6 times faster than GiST!**

**~ 3440 times faster SeQScan !!!**



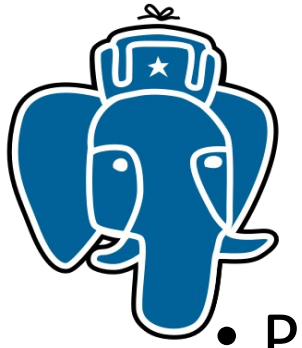
# PostgreSQL

- PostgreSQL - зрелая, развитая СУБД, свободная лицензия, большое сообщество пользователей и разработчиков (в том числе и в России)
- Расширяемость PostgreSQL позволяет добавлять новые типы данных, новые операции «на ходу»
- PostgreSQL — реальный кандидат на национальную СУБД, используется в МО.



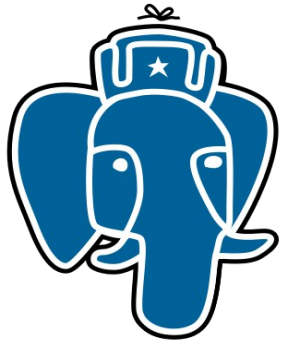
# PostgreSQL

- PostgreSQL - зрелая, развитая СУБД, свободная лицензия, большое сообщество пользователей и разработчиков (в том числе и в России)
- Расширяемость PostgreSQL позволяет добавлять новые типы данных, новые операции «на ходу»
- PostgreSQL — реальный кандидат на национальную СУБД, используется в МО.



# Однако

- Реляционные СУБД были разработаны для старой архитектуры !
  - Дорогой сервер, маломощный процессор, мало памяти, сравнительно большой диск
- Современные системы
  - Многоядерные  дешевые сервера, объединенные в сети, много памяти
- Старые задачи — подсчет денег
- Новые задачи — информационный поиск, большая конкурентность, много  разнообразных изменяющихся данных, highload

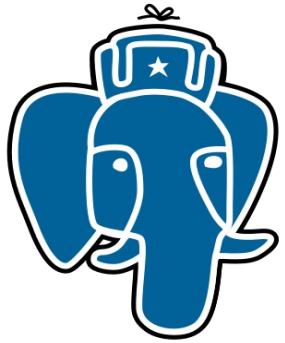


# Bandwidth VS Latency

| | Bandwidth | Latency |
|---|---|---|
| RAM, 1980 | 13 MB/s | 225ns |
| RAM, 2000 | 1600 MB/s (~125x) | 52ns (~4.5x) |
| RAM, 2012 | 12800 MB/s (~1000x) | 35ns (~6.5x) |
| HDD, 1980 | 0.6 MB/s | 48 ms |
| HDD, 2000 | 86 MB/s (~150x) | 5.7 ms (~8.5x) |
| HDD, 2012 | 300 MB/s (~300x) | 5 ms (~10x) |
| 10Mbit Ethernet | 1 MB/s | 3000 µs |
| 56Gbit Infiniband | ~5000 MB/s (~5000x) | 0.7 µs (~4300x) |

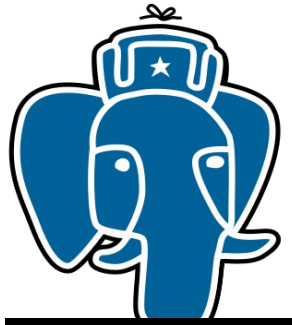
Случайный доступ — медленно
- RAM — это HDD
- HDD — это tape

CPU — аналогично

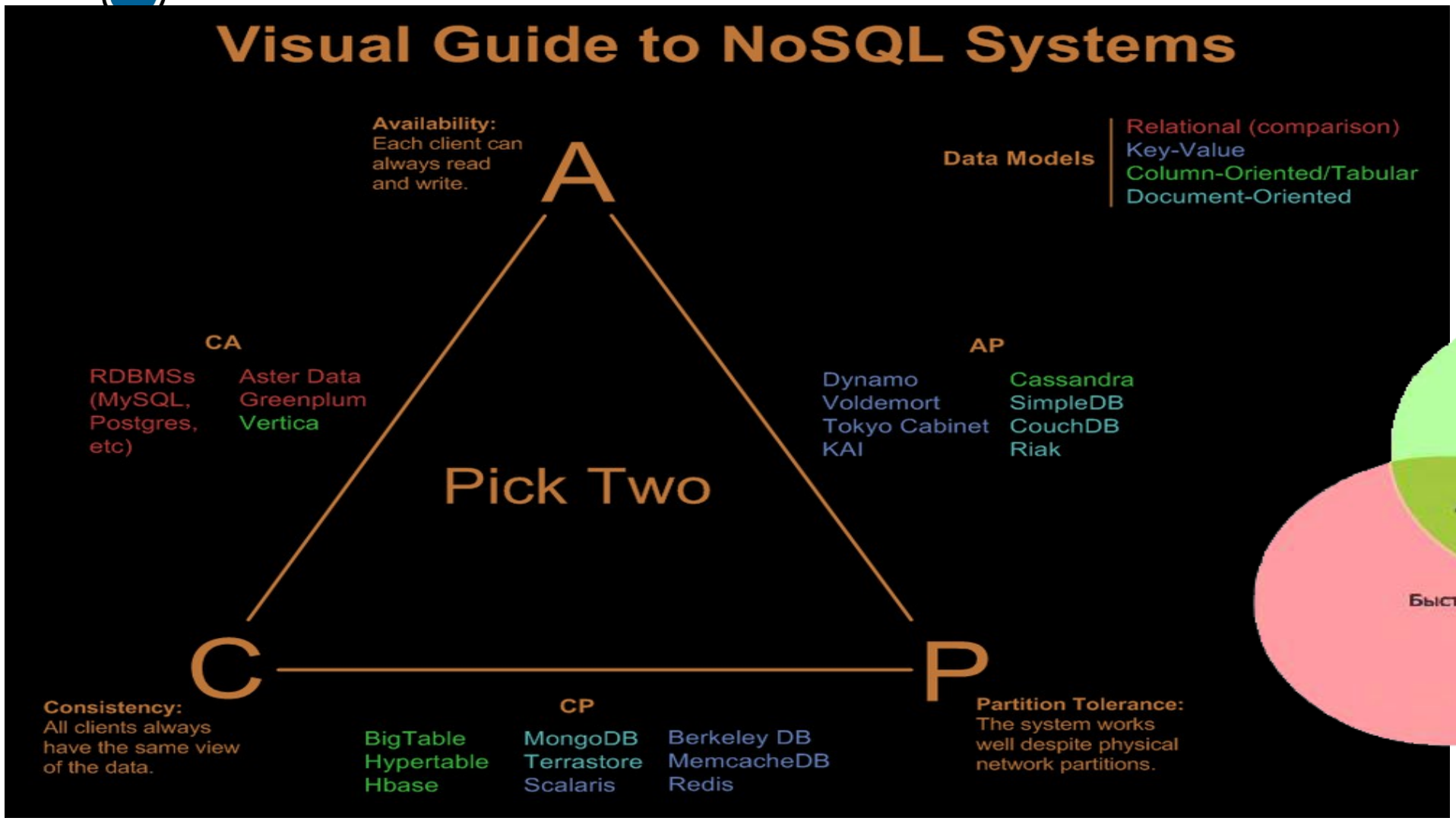


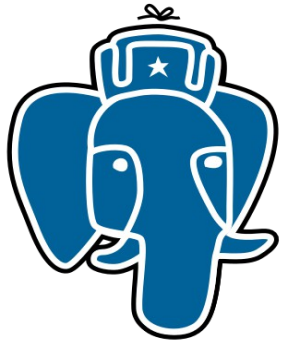
# Требования к БД

- Скорость, скорость и скорость
- Надежность
- Целостность, Транзакционность



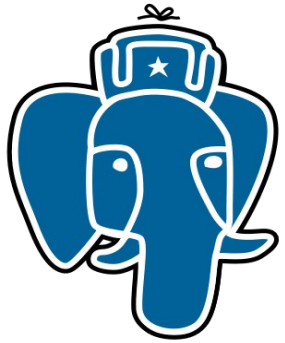
*В распределённой системе невозможно обеспечить одновременное выполнение всех трёх условий: корректности, доступности, устойчивости к сбоям* узлов.(Eric A. Brewer, CAP-theorem)



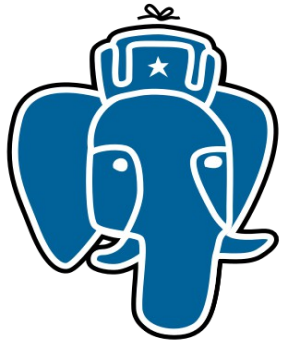
# Скорость

- In-memory
- Append only write
- Context switch (1000 cycles per switch, 10^5 csps in x86)
- Bulk net io operation
- Kqueue, epoll - libev
- Single threaded on non CPU-intensive queries



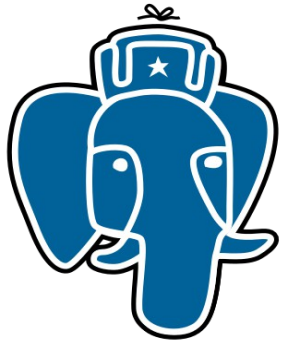
# In memory and append writes

- База целиком в памяти
- На диске — единовременные снапшоты (Copy-On-Writes)
- Изменения — в WAL



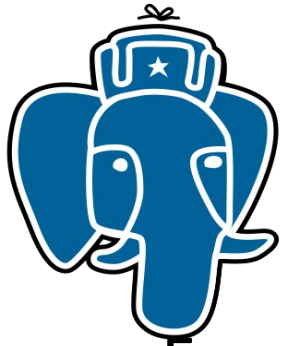
# Context switch

- Поход в ядро — дорого
  - 2 context switch
  - cache trashing
- Пакетный ввод вывод в сокет
  - 2 context switch + memcpy(small buffer)
  - копирование 8-байтовых строк 300 MB/s
- Переупорядочивание запросов (write запросы ждут диск)
- Группировка записи в WAL



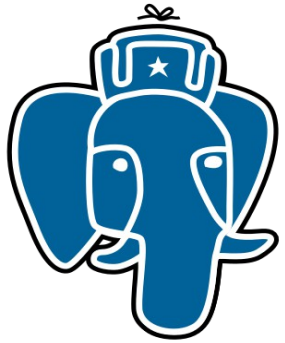
# Execution

- Денормализация
- Простые запросы — в текущем треде/контексте
  - trick — сопрограммы (Кнут)
- Сложные запросы — треды, GPU



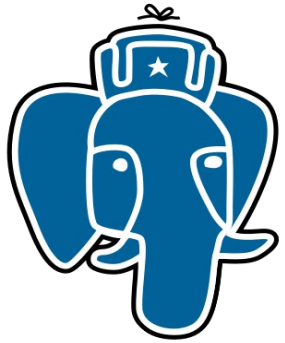
# Надежность

- Fsync, fdatasync, O_DIRECT
- Slaves
- Multimaster, paxos
- Sharding



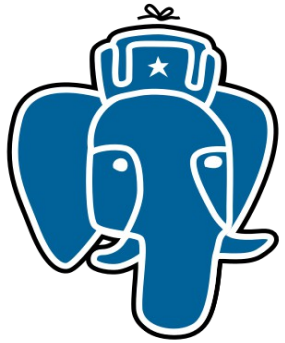
# Транзакционность

- :(
- Очереди
- Сохраняем атомарность
- Update нескольких записей за одну запись в WAL
- UNDO логи — слишком дорого



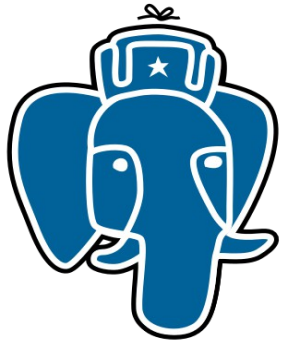
# Скорость

- Небогатый бинарный язык запросов
- Отсутствие оптимизатора/планировщика



# Реализация высокопроизводительной СУБД

- High-performance framework
- In-memory (гарантированное кол-во памяти)
- CA (корректность, доступность)
- Snapshot+WAL
- Асинхронность
- X*10^6 req/sec  - простые запросы
- X*10^5 req/sec  - «тяжелые» запросы
- X*10^4 req/sec  - модификация данных



# Реализация высокопроизводительной СУБД

- Идеальное применение
  - Мониторинг — мобильные системы, отслеживание перемещений больших групп объектов в режиме онлайн
  - Рекомендационные системы
- Возможность развития
  - Есть Framework, требуется разработать специфику задачи (пространственные данные, документы,...)
  - ~3 месяца на прототип


Спасибо за Внимание