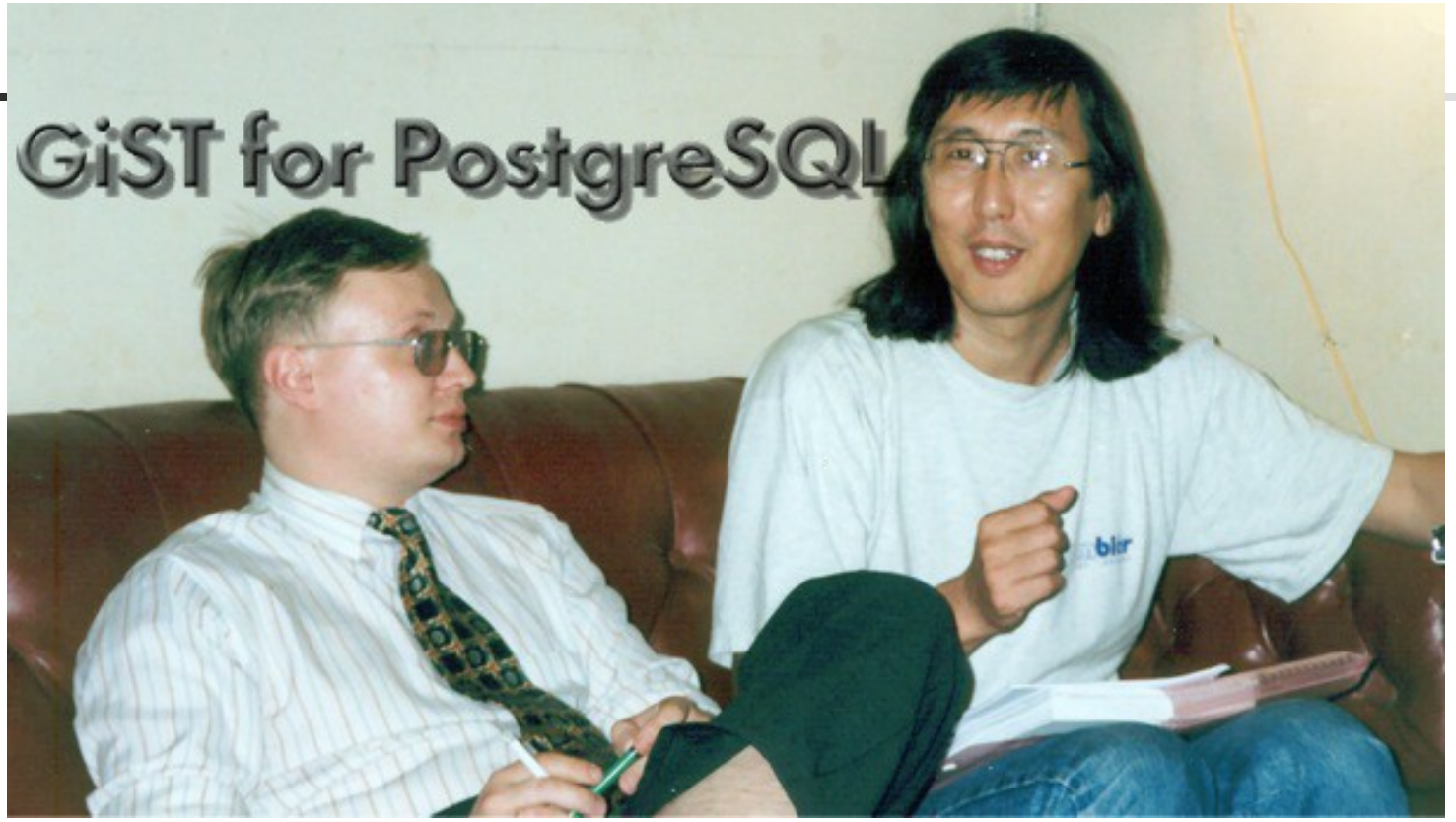
«Анатомия» полнотекстового поиска PostgreSQL

Олег Бартунов
ГАИШ-МГУ

Об авторах FTS



- Major developers of PostgreSQL
- Новые типы данных и индексы: GiST, GIN, SP-GiST
- Разработчики FTS, ltree, pg_trgm, hstore, intarray,..
- Контакты для предложений : obartunov@gmail.com

Agenda

- Что такое PostgreSQL
- Полнотекстовый поиск в PostgreSQL
  - Типы данных и операторы FTS
  - Парсер, словари, конфигурация
  - Полнотекстовые индексы (GiST, GIN)
  - Вспомогательные функции, примеры, tips
- Дополнительные возможности
  - Поиск фраз
  - Миллисекундный поиск !

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-грес**, **пэ-жэ-эс-ку-эль**

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

Лицензия: **BSD**

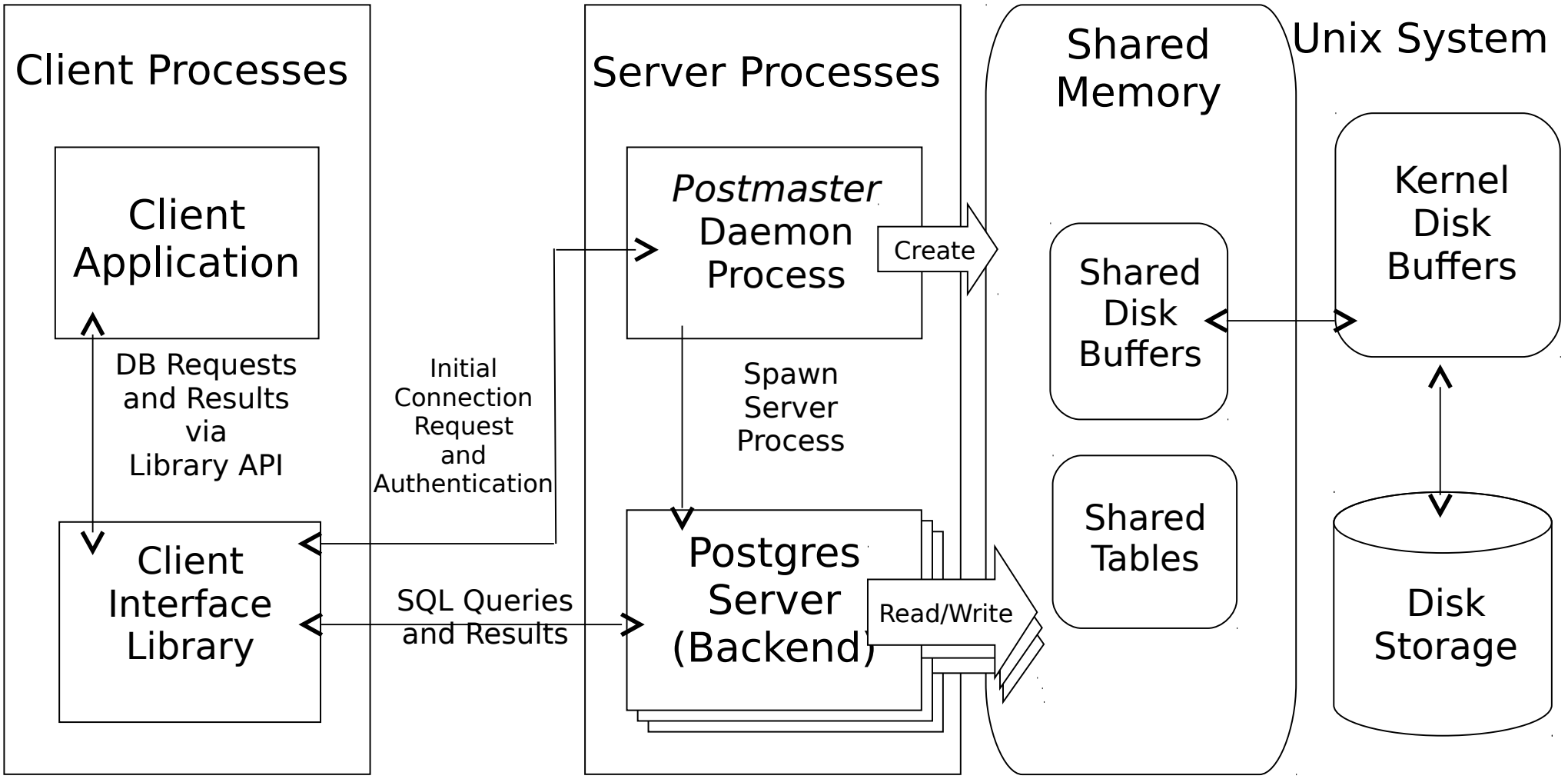
CODD 1969,1970
Relational model
1974-1975
IBM System R
1973 QUEL
UC Berkeley INGRES
INteractive Graphics REtrieval System
1978 1979 SQL
1981 SQL/DS
1983 DB2 $$$
SDL, RSI 1979
1983 ORACLE
1984 $$$
RDb/VMS
1994
2001 INFORMIX $$$
1984
1984-1987 NonStop SQL
1995 ILLUSTRA $$$
1992
1979-1982 Ingres Co
1994 CA Ingres r3
2004 $$$
1985-1988 Postgres
1989 V1
1993 V4.2
1994 V5
Postgres95 SQL
1996 V6
PostgreSQL
2005 V8
1987 Sybase ASE
1995 $$$
1992 MS SQL $$$
1993
PGDG
1997-04-03
Agatha Christie
PostgreSQL
2008 8.3
TelegraphCQ StreamBase $$$
C-Store Vertica $$$

Some forks:

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

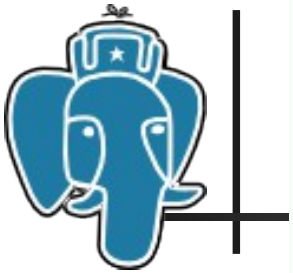
PostgreSQL Fall 2012 9.2

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 – 32Tб
- 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
- 1C: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: 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, эффективный knn
  - Обновления дерева

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, эффективный knn
  - Обновления дерева

Расширяемость PostgreSQL: GiST

- Конкурентность и восстановление после сбоев
- Поддерживает расширяемый набор запросов (в отличие от фиксированных операций сравнения B-tree)
- 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 (7 функций):
  - 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

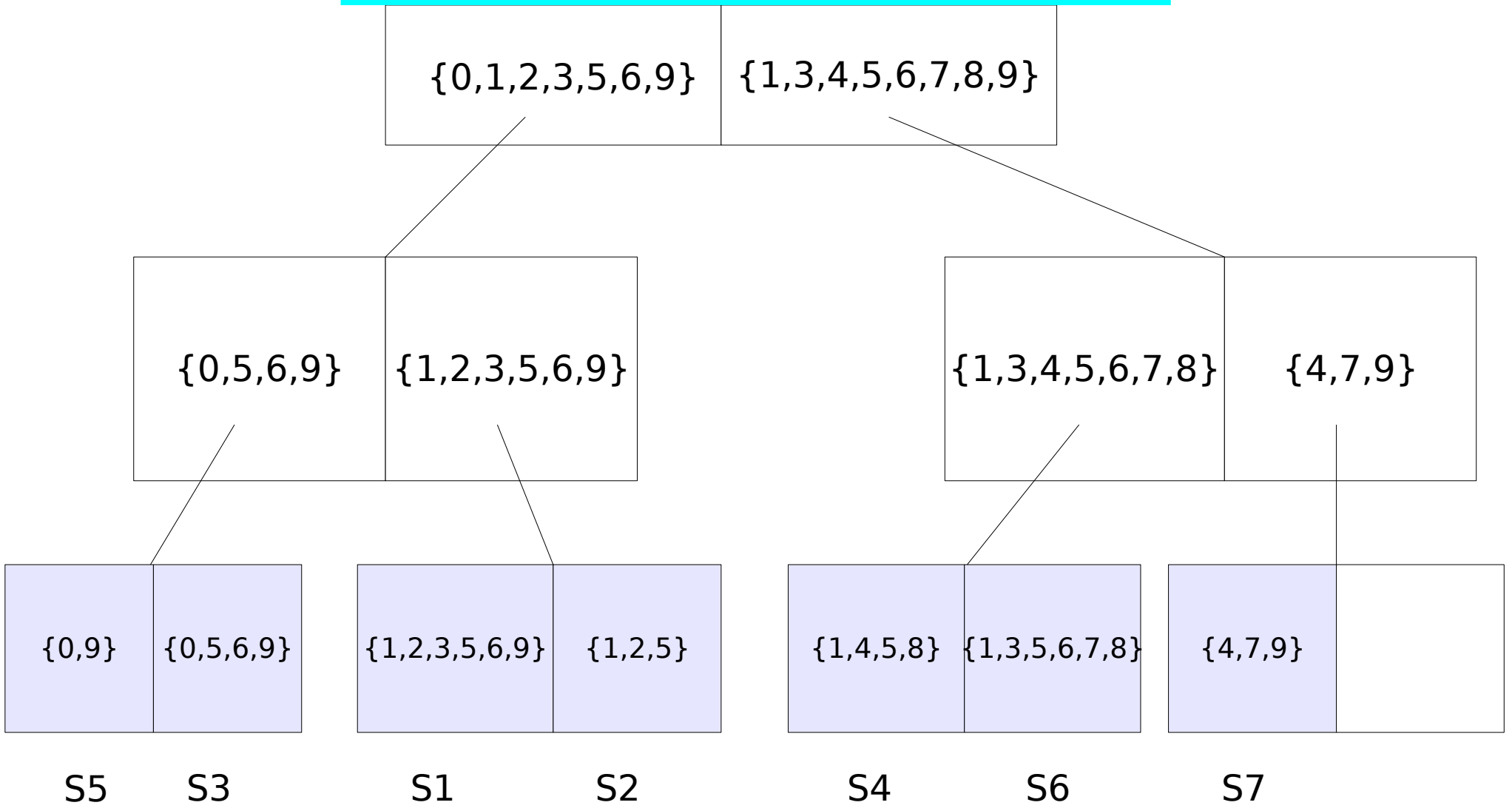
S1 = {1,**2**,3,5,6,**9**}
S2 = {1,**2**,5}
S3 = {0,5,6,**9**}
S4 = {1,4,5,8}
S5 = {0,9}
S6 = {3,5,6,7,8}
S7 = {4,7,**9**}

Q = {**2,9**}

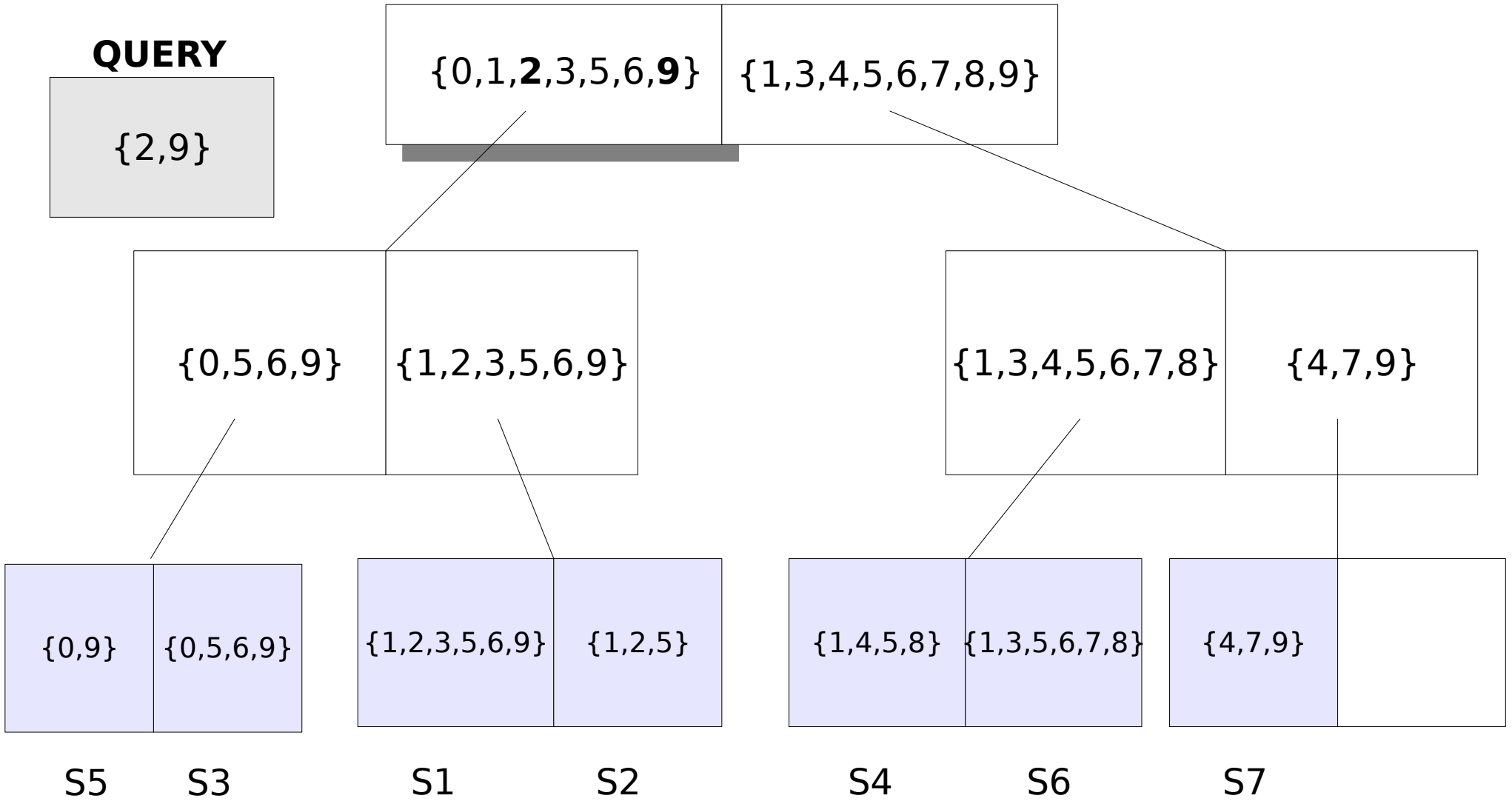
"THE RD-TREE: AN INDEX STRUCTURE FOR SETS", Joseph M. Hellerstein

RD-Tree

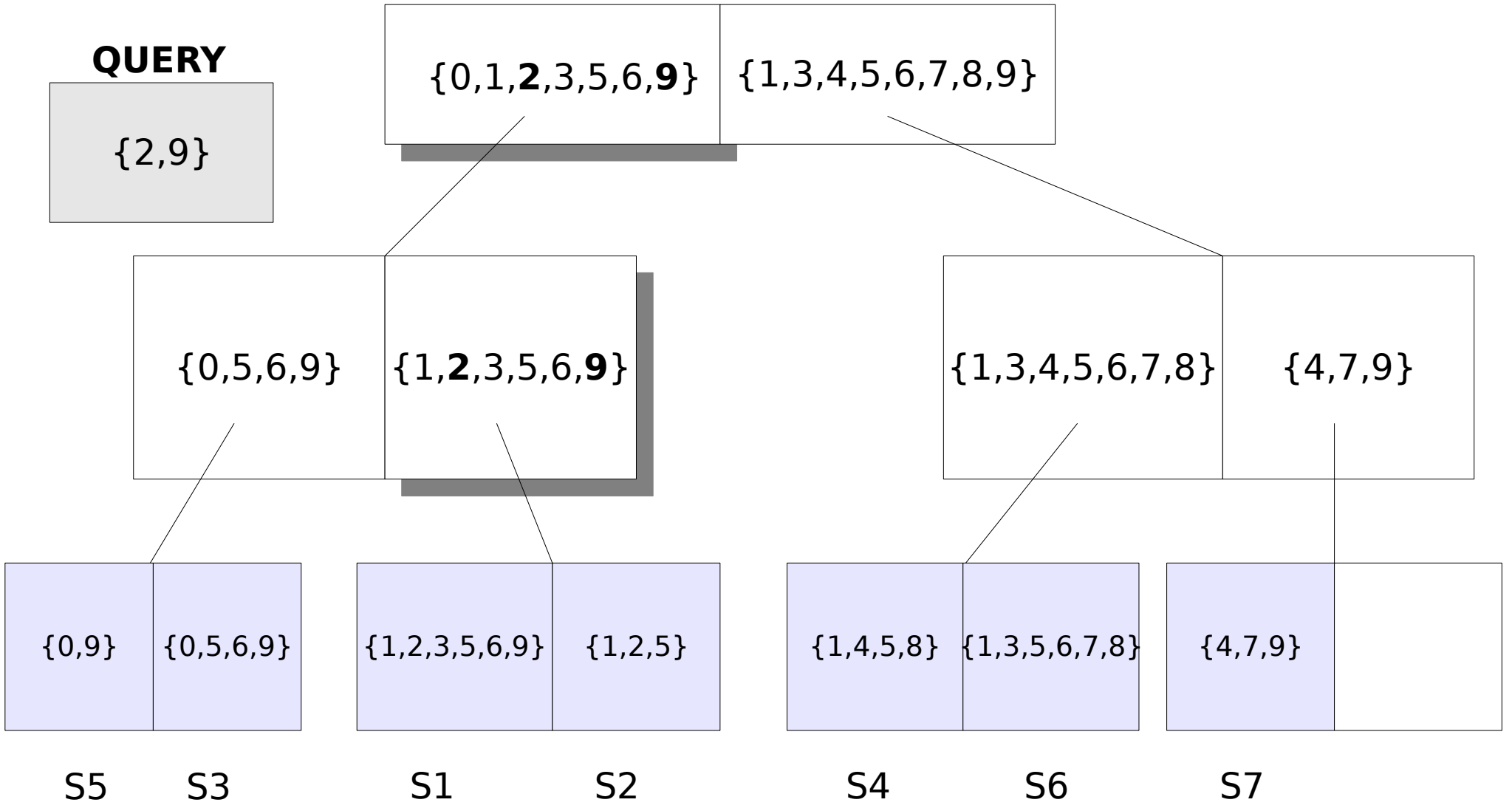
Каждый узел включает всех потомков



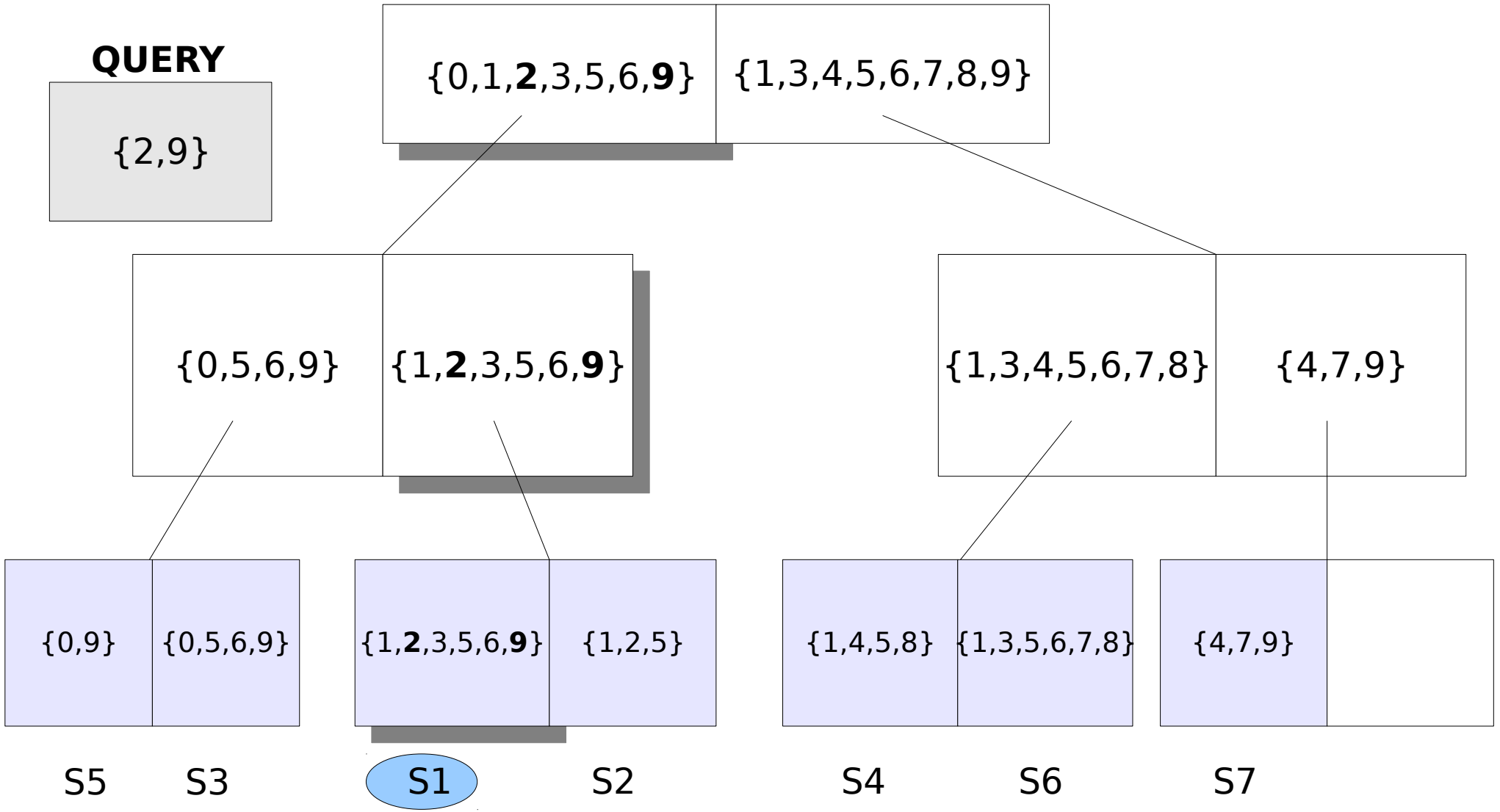
RD-Tree


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)

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

GIN
Обобщенный обратный индекс

Обратный индекс

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
aerospace instrumentation, 61
aerospace propulsion, 52
aerospace robotics, 68
aluminium, 17
amorphous state, 67
angular velocity measurement, 58
antenna phased arrays, 41, 46, 66
argon, 21
assembling, 22
atomic force microscopy, 13, 27, 35
atomic layer deposition, 15
attitude control, 60, 61
attitude measurement, 59, 61
automatic test equipment, 71
automatic testing, 24

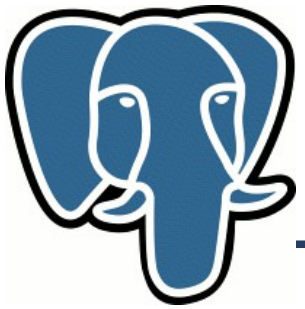
B

backward wave oscillators, 45
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

D

design for manufacture, 25
design for testability, 25
diamond, 3, 27, 43, 54, 67
dielectric losses, 31, 42
dielectric polarisation, 31
dielectric relaxation, 64
dielectric thin films, 16
differential amplifiers, 28
diffraction gratings, 68
discrete wavelet transforms, 72
displacement measurement, 11
display devices, 56
distributed feedback lasers, 38

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**

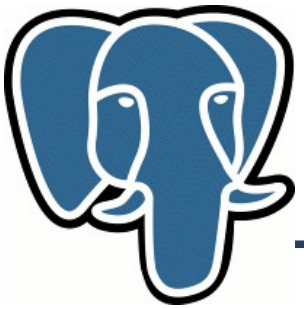
attitude measurement, 59, 61
automatic test equipment, 71
automatic testing, 24

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

ENTRY TREE

Report Index

A

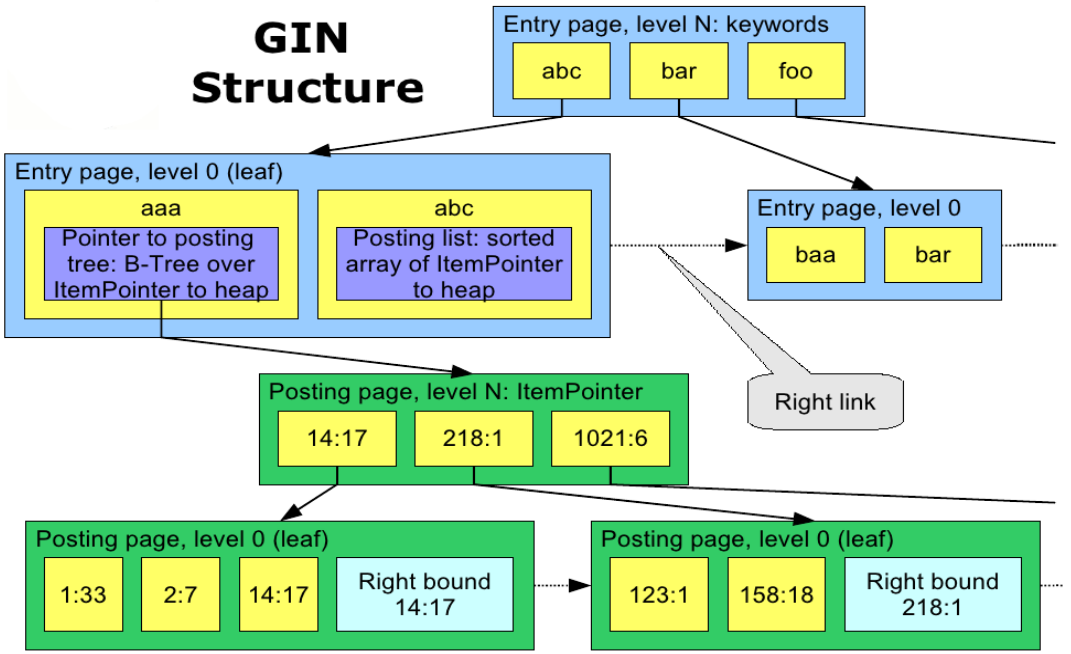
abrasives, 27
acceleration measurement, 58
accelerometers, 5, 10, 25, 28, 30, 36, 58, 59, 61, 73, 74
actuators, 4, 37, 46, 49
adaptive Kalman filters, 60, 61
adhesion, 63, 64
adhesive bonding, 15
adsorption, 44
aerodynamics, 29
aerospace instrumentation, 6
aerospace propulsion, 52
aerospace robotics, 68
aluminium, 17
amorphous state, 67
angular velocity measurement
antenna phased arrays, 41, 4
argon, 21
assembling, 22
atomic force microscopy, 13,
atomic layer deposition, 15
attitude control, 60, 61
attitude measurement, 59, 61
automatic test equipment, 71
automatic testing, 24

B

backward wave oscillators, 45

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

Posting list
Posting tree



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

Generalized Inverted Index:API

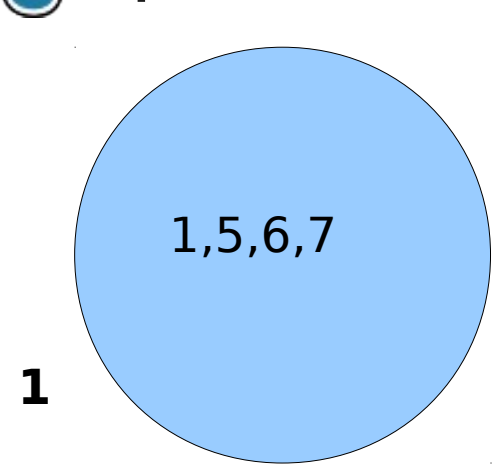
Разработчик предоставляет 4 (5) функции:

- 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)

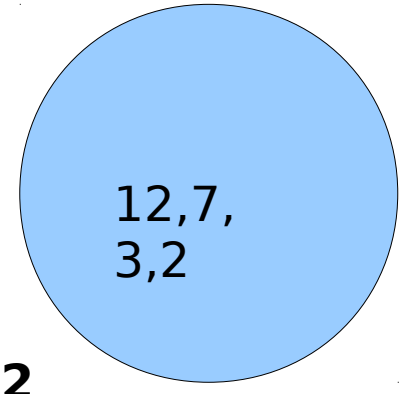
GIN

- 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

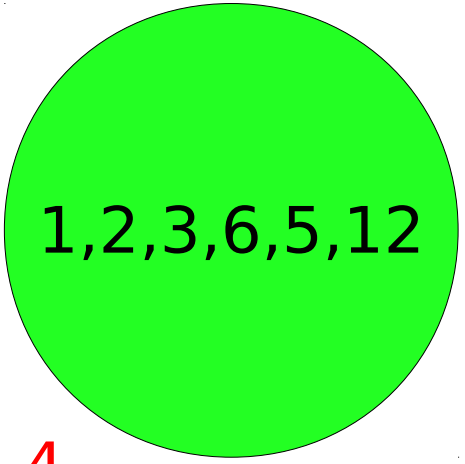
GIN: Update problem



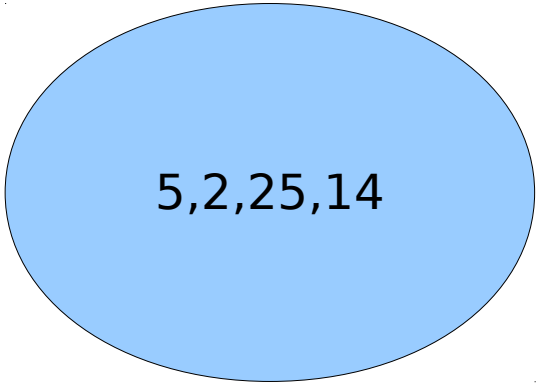
1



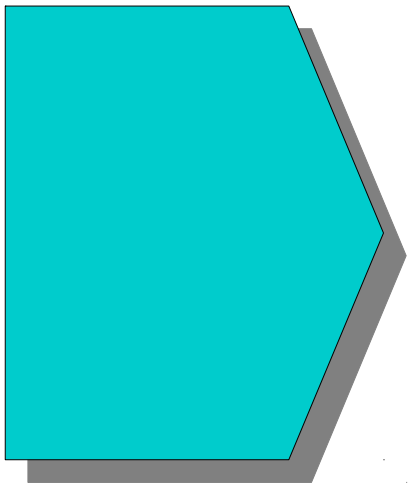
2



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: 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

Приложения GiST, GIN

- 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)
- ……..

FTS in Databases

- Полнотекстовый поиск
  - найти документы удовлетворяющие запросу
  - отсортировать их в некотором порядке

Найти документы содержащие **все** слова из запроса и вернуть их отсортированными по похожести

- **Требования к FTS**
  - **полная интеграция с СУБД**
    - транзакционность
    - конкурентный доступ
    - восстановление после сбоев
    - online индекс
  - Конфигурируемость (парсеры, словари,...)
  - Масштабируемость

Наиболее привычный вид поиска

FTS in Databases

- Обычные поисковые машины не могут индексировать базы данных
- Web-site – интерфейс к БД
- Базы данных как часть **Hidden,Invisible, Dark, Deep** Web
  - динамические страницы
  - страницы, на которые никто не ссылается
  - ограничение доступа
  - javascipt, flash генерируемые линки
  - бинарный контент

Что такое Документ ?

- Произвольный текстовый атрибут
- Комбинация текстовых атрибутов из одной или разных (join) таблиц

Title || Abstract || Keywords || Body || Author

Text Search Operators

- Традиционные операции текстового поиска (TEXT op TEXT, op - ~, ~*, LIKE, ILIKE)

```
=# select title  from apod where title ~* 'x-ray' limit 5;
                title
-------------------------------------
 The X-Ray Moon
 Vela Supernova Remnant in X-ray
 Tycho's Supernova Remnant in X-ray
 ASCA X-Ray Observatory
 Unexpected X-rays from Comet Hyakutake
(5 rows)

=# select title  from apod where title ilike '%x-ray%' limit 5;
```

What's wrong ?

- Нет поддержки лингвистики
  - что есть слово ?
  - что индексировать ?
  - «нормализация» слов
  - стоп-слова (noise-words)
- Нет релевантности
  - все документы одинаково «похожи»
- Медленно, документы каждый раз сканируются

  В 9.3+ появилась индексная поддержка (pg_trgm)

```
select * from man_lines where man_line ~* '(?:
(?:p(?:ostgres(?:ql)?|g?sql)|sql)) (?:(?:(?:mak|us)e|do|is))';
```

FTS in PostgreSQL

- OpenFTS — 2000, Pg as a storage
- GiST index — 2000, thanks Rambler
- Tsearch — 2001, contrib:no ranking
- Tsearch2 — 2003, contrib:config
- GIN —2006, thanks, JFG Networks
- FTS — 2006, in-core, thanks,EnterpriseDB
- E-FTS — Enterprise FTS, thanks ???

FTS in PostgreSQL

- **tsvector** – хранилище для документов, оптимизированное для поиска
  - отсортированный массив лексем
  - позиционная информация
  - структурная информация (важность)
- **tsquery** – текстовый тип для запроса с логическими операторами & | ! ()
- **Полнотекстовый оператор:** tsvector @@ tsquery
- Операторы @>, <@ для tsquery
- **Функции**: to_tsvector, to_tsquery, plainto_tsquery, ts_lexize, ts_debug, ts_stat, ts_rewrite,ts_headline, ts_rank, ts_rank_cd, setweight
- **Индексы**: GiST, GIN

FTS in PostgreSQL

Где выигрыш ?

документ обрабатывается при индексировании – не тратится время на обработку при поиске.

- документ разбивается на токены с помощью подключаемого парсера
- токены превращаются в лексемы с помощью подключаемых словарей
- запоминаются позиционная информация и важность лексемы, используется для ранжирования результатов
- стоп-слова игнорируются

FTS in PostgreSQL

- Query обрабатывается при **поиске**
  - разбивается на токены
  - токены превращаются в лексемы
  - Токены могут иметь веса
  - убираются стоп-слова
  - можно ограничивать область поиска
    `'fat:ab & rats & ! (cats | mice)'`
  - может изменяться с помощью **query rewriting** «на ходу»

FTS in PostgreSQL

- Парсер разбивает текст на токены

Парсер

```
=# 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

```
=# \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
```

ts_lexize('english_stem','stars')

---

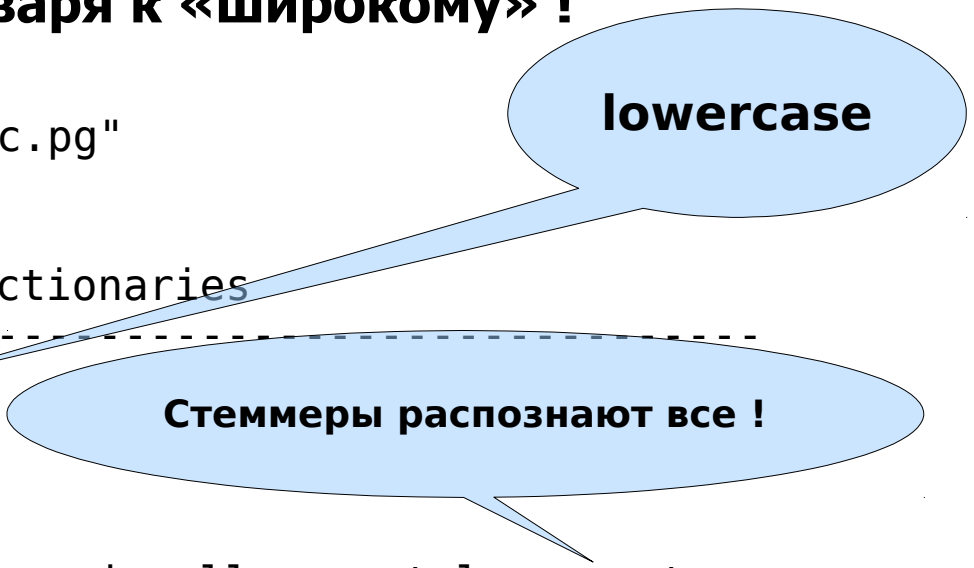
star

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)

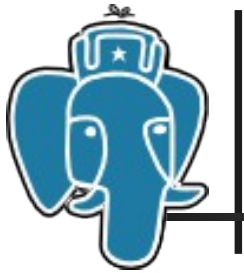
```
1. Unaccent dictionary does nothing and returns NULL.
(lexeme 'Hotels' will be passed to the next dictionary if any)

=# 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

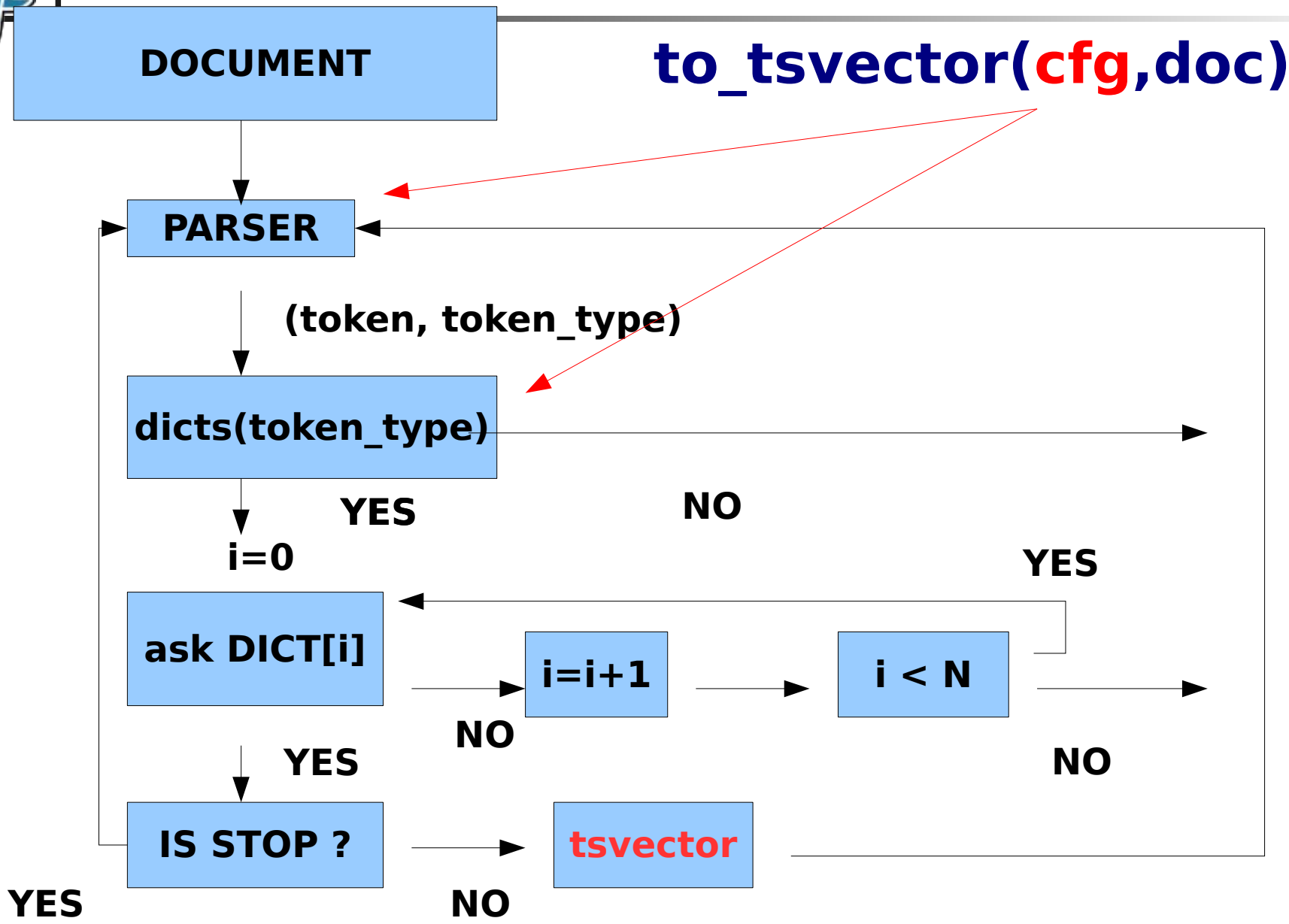
\dF{,d,p}[+] [PATTERN]

FTS configuration

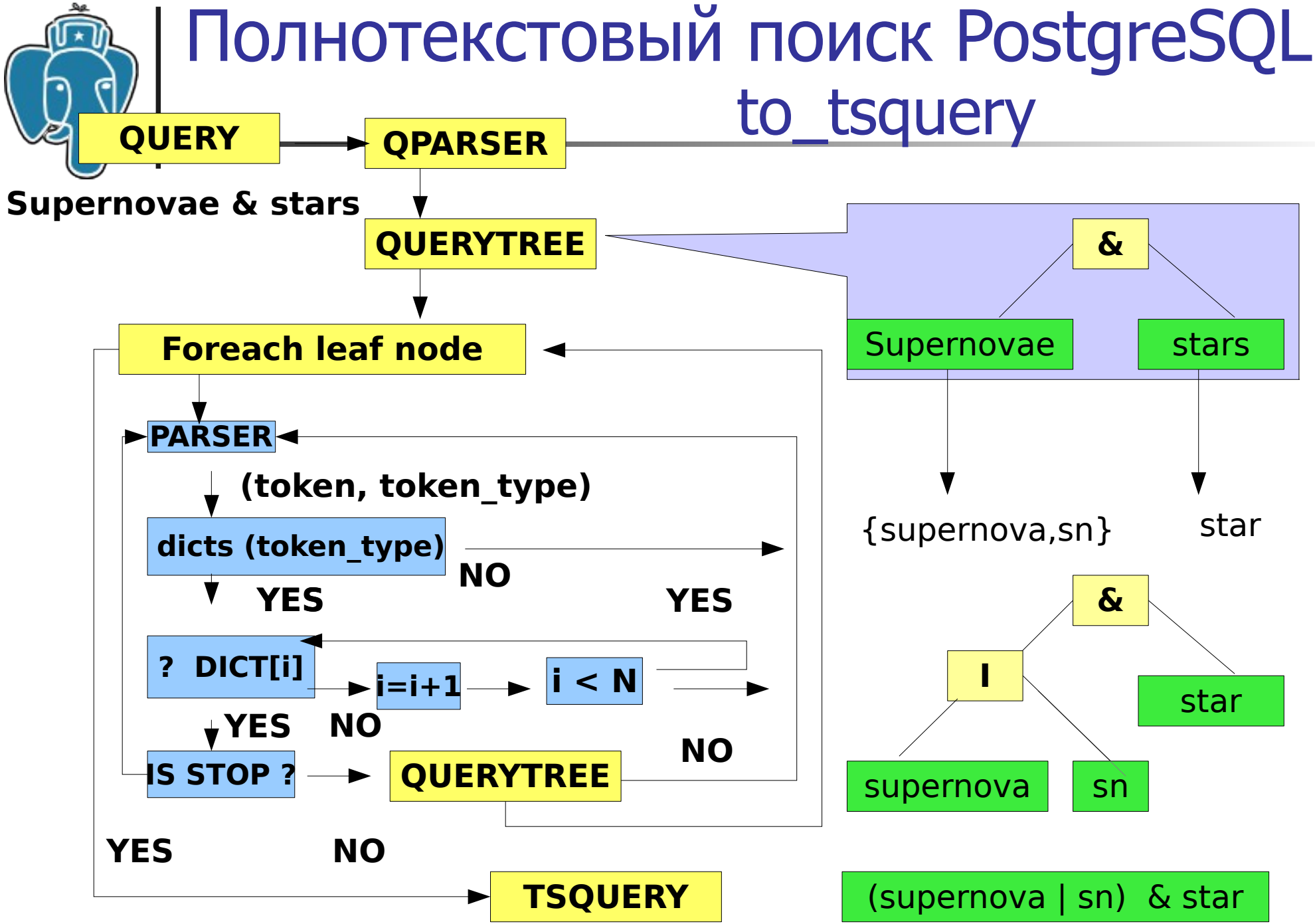
16 конфигураций для 15 языков

```
=# \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)
```

Полнотекстовый поиск PostgreSQ



Полнотекстовый поиск PostgreSQL
to_tsquery



Индексы !

- 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)

FTS Index (GiST): RD-Tree

- 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

FTS Index (GiST): RD-Tree

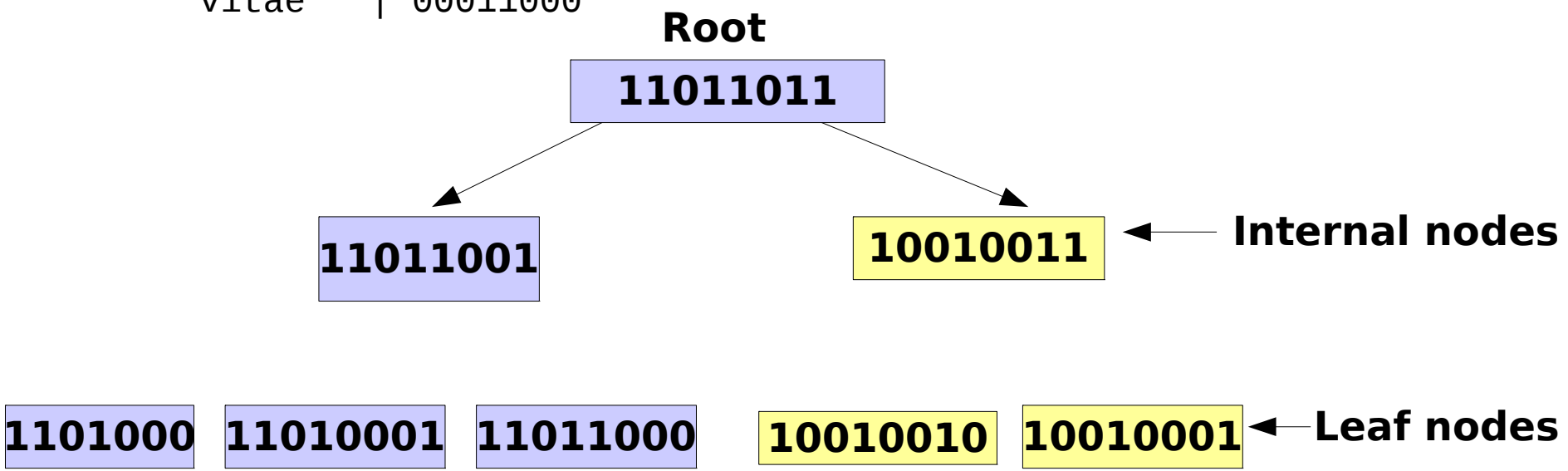
- Пример — латинские поговорки

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

FTS Index (GiST): RD-Tree

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

QUERY



GiST index - RD-Tree

Contrib module Gevel используется для изучения поискового дерева.

Листья дерева

```
arxiv=# select * from gist_print('gist_idx_90') as
        t(level int,valid bool, fts gtsvector) where level =4;
 level | valid |              fts
------+------ +---------------------------------
     4 | t       | 130 true bits, 1886 false bits
     4 | t       | 95 unique words
     4 | t       | 33 unique words
.....................................
     4 | t       | 61 unique words
(417366 rows)
```

Внутренние узлы

```
arxiv=# select * from gist_print('gist_idx_90') as
        t(level int, valid bool, fts gtsvector) where level =3;

 level | valid |              fts
------+-------+---------------------------------
     3 | t        | 852 true bits, 1164 false bits
     3 | t        | 861 true bits, 1155 false bits
     3 | t        | 858 true bits, 1158 false bits
.........................................................
     3 | t        | 773 true bits, 1243 false bits
(17496 rows)
```

RD-Tree (GiST)

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

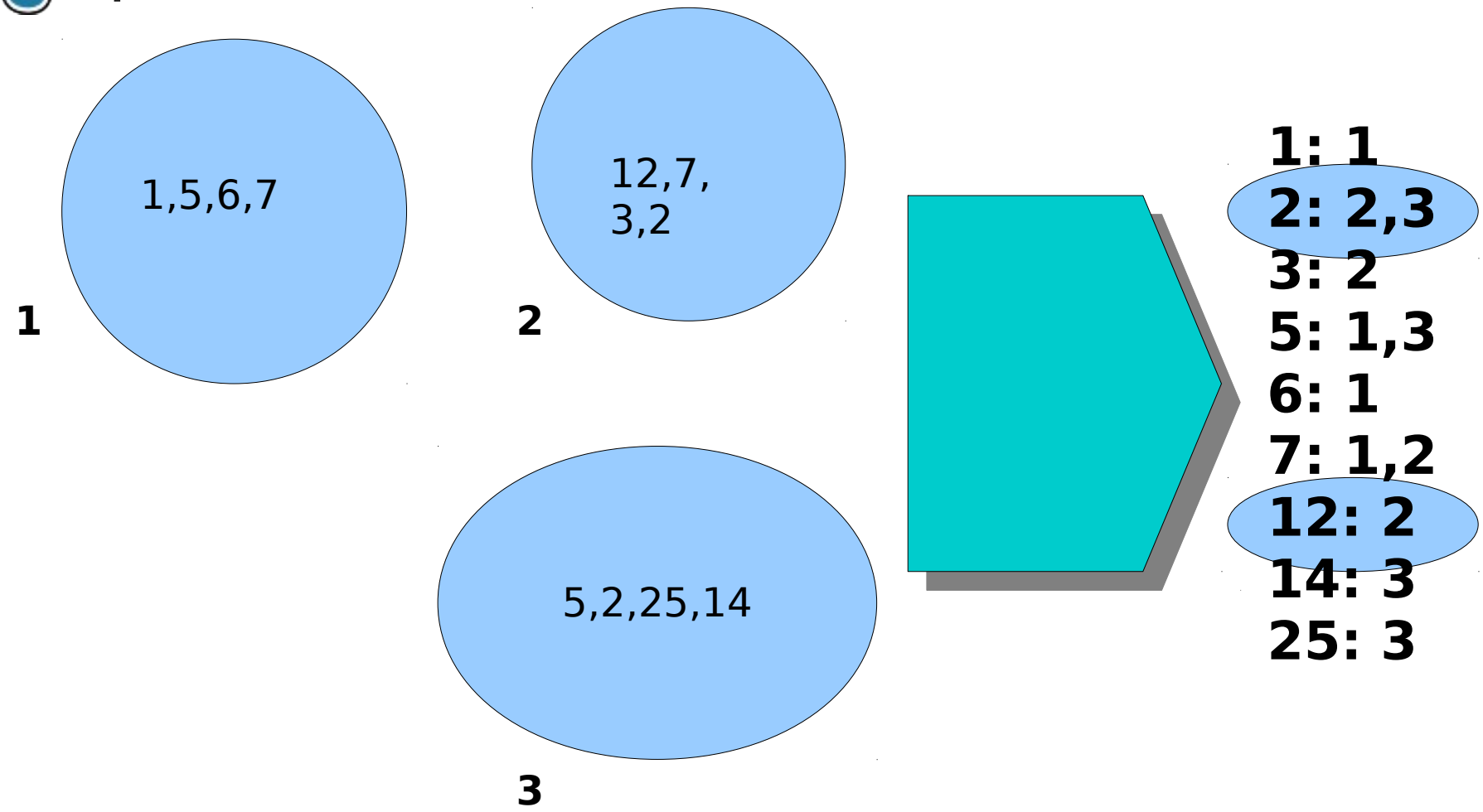
False drop

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

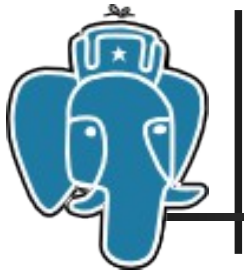
FTS Indexes

- Используйте индексы
  - GiST индекс для изменяющихся данных
    - быстро обновляется
    - не очень хорошо шкалируется
    - зависит от количества уникальных слова
  - GiN индекс для архивных таблиц
    - дольше обновляется (при вставке документа из 1000 слов требуется сделать 1000 updates). Gin Fast Update проблему сильно ослабил !
    - хорошо шкалируется
    - очень слабо зависит от числа уникальных слов
- Оба индекса конкурентны и поддерживают восстановление после сбоев

GIN



Query: 2 & 12
Result: 2

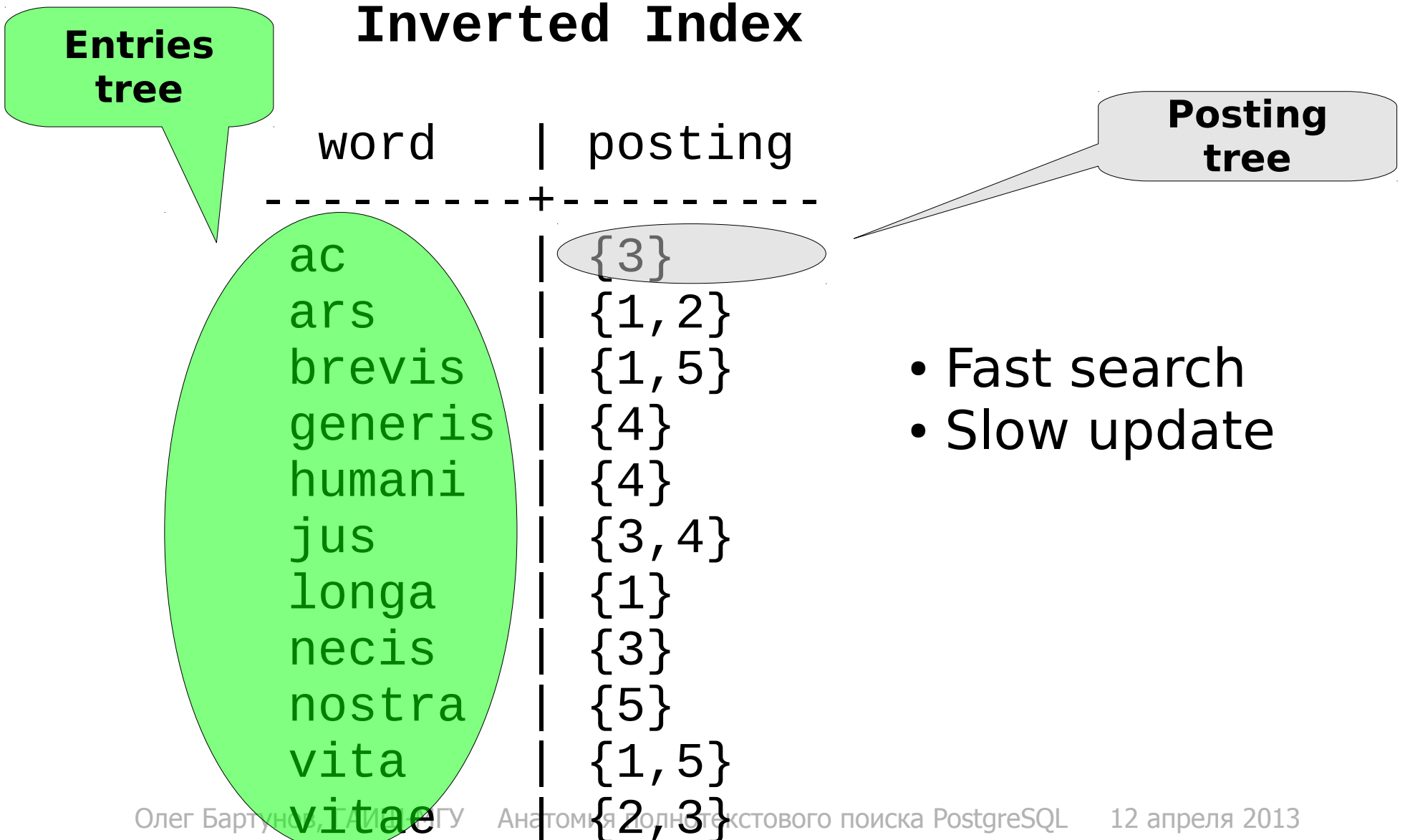


GIN Index

Demo collections – latin proverbs

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

GIN Index



- Fast search
- Slow update

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_rank_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

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

790 times

```
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
```

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 X-Rays	**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	595041
1163611	Vela Supernova Remnant in Optical	0. 1655
1161686	Vela Supernova Remnant in Optical	0.5 55

new document

```
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

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
```

ACID overhead is really big :(

- Foreign solutions: Sphinx, Solr, Lucene....
  - Crawl database and index (time lag)
  - No access to attributes
  - Additional complexity
  - BUT: **Very fast !**

Can we improve native FTS ?

Can we improve native FTS ?

156676 Wikipedia articles:

```
postgres=# explain analyze
SELECT docid, ts_rank(text_vector, to_tsquery('english', 'title')) AS rank
FROM ti2
WHERE text_vector @@ to_tsquery('english', 'title')
ORDER BY rank DESC
LIMIT 3;

 Limit  (cost=8087.40..8087.41 rows=3 width=282) (actual time=433.749..433.752 rows=3 loops=
   ->  Sort  (cost=8087.40..8206.63 rows=47692 width=282)
(actual time=433.749..433.749 rows=3 loops=1)
         Sort Key: (ts_rank(text_vector, '''titl'''::tsquery))
         Sort Method: top-N heapsort  Memory: 25kB
         ->  Bitmap Heap Scan on ti2  (cost=529.61..7470.99 rows=47692 width=282)
(actual time=15.094..423.452 rows=47855 loops=1)
               Recheck Cond: (text_vector @@ '''titl'''::tsquery)
               ->  Bitmap Index Scan on ti2_index  (cost=0.00..517.69 rows=47692 width=0)
(actual time=13.736..13.736 rows=47855 loops=1)
                     Index Cond: (text_vector @@ '''titl'''::tsquery)
 Total runtime: 433.787 ms
```

HEAP IS SLOW
400 ms !

Can we improve native FTS ?

156676 Wikipedia articles:

```
postgres=# explain analyze
SELECT docid, ts_rank(text_vector, to_tsquery('english', 'title')) AS rank
FROM ti2
WHERE text_vector @@ to_tsquery('english', 'title')
ORDER BY text_vector>< plainto_tsquery('english','title')
LIMIT 3;
```

What if we have this plan ?

```
Limit  (cost=20.00..21.65 rows=3 width=282) (actual time=18.376..18.427 rows=3 loops=1)
   ->  Index Scan using ti2_index on ti2  (cost=20.00..26256.30 rows=47692 width=282)
(actual time=18.375..18.425 rows=3 loops=1)
         Index Cond: (text_vector @@ '''titl'''::tsquery)
         Order By: (text_vector >< '''titl'''::tsquery)
 Total runtime: 18.511 ms vs 433.787 ms
```

We'll be FINE !

FTS tips — Очепятки (pg_trgm)

```
=# select show_trgm('supyrnova');
                        show_trgm
-----------------------------------------------
 {"  s"," su",nov,ova,pyr,rno,sup,upy,"va ",yrn}
```

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

=# \d apod_words
  Table "public.apod_words"
 Column |  Type   | Modifiers
--------+---------+-----------
 word   | text    |
 ndoc   | integer |
 nentry | integer |
```

собираем статистику по словам

```
=# create index trgm_idx on apod_words using gist(word gist_trgm_ops);

=# select word, similarity(word, 'supyrnova') AS sml
from apod_words where word % 'supyrnova' order by sml desc, word;

   word    |   sml
-----------+----------
 supernova | 0.538462
```

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
  было:
  T(freq & rare)=T(rare & freq) ~ T(freq)
  стало:
  T(freq & rare)>>T(rare & freq) ~ T(rare)

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.**

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.**

Phrase Search

- 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 $0 \le P_B - P_A \le 1$ (distance condition)

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

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

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 !

Спасибо за внимание !