

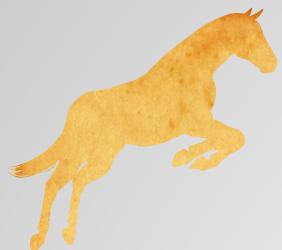


PALOMINO

OPERATIONAL EXCELLENCE
FOR DATABASES

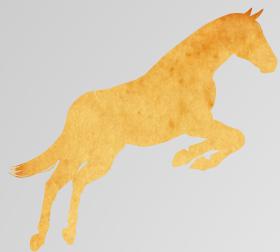
PostgreSQL FTS Solutions

FOSDEM PGDAY 2013
by Emanuel Calvo



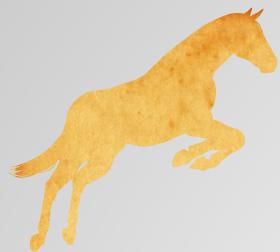
About me:

- Operational DBA at PalominoDB.
 - MySQL, Maria and PostgreSQL databases.
- Spanish Press Contact.
- Check out my LinkedIn Profile at: <http://es.linkedin.com/in/ecbccbc/>



Credits

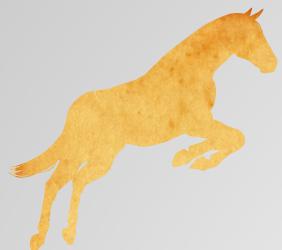
- Thanks to:
 - Andrew Atanasoff
 - Vlad Fedorkov
 - All the PalominoDB people that help out !



Palomino - Service Offerings

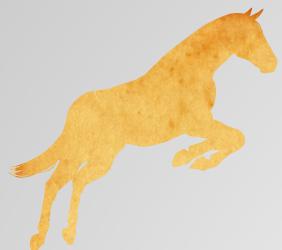
- Monthly Support:
 - Being renamed to Palomino DBA as a service.
 - Eliminating 10 hour monthly clients.
 - Discounts are based on spend per month (0-80, 81-160, 161+)
 - We will be penalizing excessive paging financially.
 - Quarterly onsite day from Palomino executive, DBA and PM for clients using 80 hours or more per month.
 - Clients using 80-160 hours get 2 new relic licenses. 160 hours plus get 4.
- Adding annual support contracts:
 - Consultation as needed.
 - Emergency pages allowed.
 - Small bucket of DBA hours (8, 16 or 24)

For more information, please go to: [Spreadsheet](#)



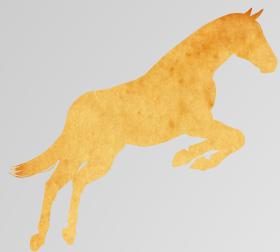
Agenda

- What we are looking for?
- Concepts
- Native Postgres Support
 - <http://www.postgresql.org/docs/9.2/static/textsearch.html>
- External solutions
 - Sphinx
 - <http://sphinxsearch.com/>
 - Solr
 - <http://lucene.apache.org/solr/>



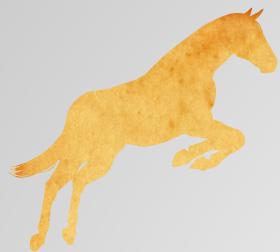
Goals of FTS

- Add complex searches using synonyms, specific operators or spellings.
 - Improving performance sacrificing accuracy.
- Reduce IO and CPU utilization.
 - Text consumes a lot of IO for read and CPU for operations.
- FTS can be handled:
 - Externally
 - using tools like Sphinx or Solr
 - Internally
 - native FTS support.
- Order words by relevance
- Language sensitive
- Faster than regular expressions or LIKE operands



Concepts

- Parsers
 - 23 token types (url, email, file, etc)
- Token
- Stop word
- Lexeme
 - array of lexemes + position + weight = tsvector
- Dictionaries
 - [Simple Dictionary](#)
 - The simple dictionary template operates by converting the input token to lower case and checking it against a file of stop words.
 - [Synonym Dictionary](#)
 - [Thesaurus Dictionary](#)
 - [Ispell Dictionary](#)
 - [Snowball Dictionary](#)

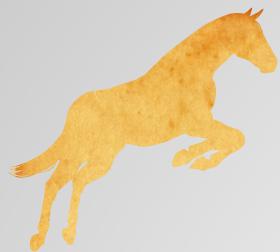


Limitations

- The length of each lexeme must be less than 2K bytes
- The length of a tsvector (lexemes + positions) must be less than 1 megabyte
- The number of lexemes must be less than 264
- Position values in tsvector must be greater than 0 and no more than **16,383**
- No more than 256 positions per lexeme
- The number of nodes (lexemes + operators) in a tsquery must be less than 32,768
- Those limits are hard to be reached!

*For comparison, the PostgreSQL 8.1 documentation contained **10,441** unique words, a total of 335,420 words, and the most frequent word “postgresql” was mentioned 6,127 times in 655 documents.*

Another example — the PostgreSQL mailing list archives contained 910,989 unique words with 57,491,343 lexemes in 461,020 messages.



psql commands

- `\dF[+] [PATTERN]` list text search configurations
- `\dFd[+] [PATTERN]` list text search dictionaries
- `\dFp[+] [PATTERN]` list text search parsers
- `\dFt[+] [PATTERN]` list text search templates



Elements

```
full_text_search=# \dFd+ *
```

List of text search dictionaries

| Schema | Name | Template | Init options | Description |
|------------|-----------------|---|---|--|
| pg_catalog | danish_stem | pg_catalog.snowball language = 'danish', stopwords = 'danish' | | snowball stemmer for danish language |
| pg_catalog | dutch_stem | pg_catalog.snowball language = 'dutch', stopwords = 'dutch' | | snowball stemmer for dutch language |
| pg_catalog | english_stem | pg_catalog.snowball language = 'english', stopwords = 'english' | | snowball stemmer for english language |
| pg_catalog | finnish_stem | pg_catalog.snowball language = 'finnish', stopwords = 'finnish' | | snowball stemmer for finnish language |
| pg_catalog | french_stem | pg_catalog.snowball language = 'french', stopwords = 'french' | | snowball stemmer for french language |
| pg_catalog | german_stem | pg_catalog.snowball language = 'german', stopwords = 'german' | | snowball stemmer for german language |
| pg_catalog | hungarian_stem | pg_catalog.snowball language = 'hungarian', stopwords = 'hungarian' | snowball stemmer for hungarian language | |
| pg_catalog | italian_stem | pg_catalog.snowball language = 'italian', stopwords = 'italian' | snowball stemmer for italian language | |
| pg_catalog | norwegian_stem | pg_catalog.snowball language = 'norwegian', stopwords = 'norwegian' | snowball stemmer for norwegian language | |
| pg_catalog | portuguese_stem | pg_catalog.snowball language = 'portuguese', stopwords = 'portuguese' | snowball stemmer for portuguese language | |
| pg_catalog | romanian_stem | pg_catalog.snowball language = 'romanian' | | snowball stemmer for romanian language |
| pg_catalog | russian_stem | pg_catalog.snowball language = 'russian', stopwords = 'russian' | | snowball stemmer for russian language |
| pg_catalog | simple | pg_catalog.simple | simple dictionary: just lower case and check for stopword | |
| pg_catalog | spanish_stem | pg_catalog.snowball language = 'spanish', stopwords = 'spanish' | | snowball stemmer for spanish language |
| pg_catalog | swedish_stem | pg_catalog.snowball language = 'swedish', stopwords = 'swedish' | | snowball stemmer for swedish language |
| pg_catalog | turkish_stem | pg_catalog.snowball language = 'turkish', stopwords = 'turkish' | | snowball stemmer for turkish language |

(16 rows)



Elements

```
postgres=# \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)
```



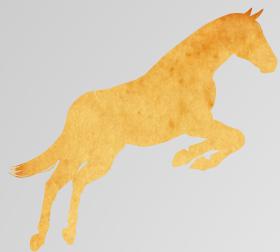
Elements

List of data types

| Schema | Name | Description |
|------------|-----------|---|
| pg_catalog | gtsvector | GiST index internal text representation for text search |
| pg_catalog | tsquery | query representation for text search |
| pg_catalog | tsvector | text representation for text search |
| (3 rows) | | |

Some operators:

- @@ (tsvector against tsquery)
- || concatenate tsvectors (it reorganises lexemes and ranking)



Small Example

```
full_text_search=# create table basic_example (i serial PRIMARY KEY, whole text, fulled tsvector, dictionary regconfig);
postgres=# CREATE TRIGGER tsvectorupdate BEFORE INSERT OR UPDATE
ON basic_example FOR EACH ROW EXECUTE PROCEDURE tsvector_update_trigger(fulled, "pg_catalog.
english", whole);
CREATE TRIGGER
postgres=# insert into basic_example(whole,dictionary) values ('This is an example','english'::regconfig);
INSERT 0 1
full_text_search=# create index on basic_example(to_tsvector(dictionary,whole));
CREATE INDEX
full_text_search=# create index on basic_example using GIST(to_tsvector(dictionary,whole));
CREATE INDEX
postgres=# select * from basic_example;
 i |    whole      | fulled   | dictionary
---+-----+-----+
 5 | This is an example | 'exampl':4 | english
(1 row)
```



Pre processing

- Documents into tokens
 - Find and clean
- Tokens into lexemes
 - Token normalised to a language or dictionary
 - Eliminate stop words (high frequently words)
- Storing
 - Array of lexemes (tsvector)
 - the position of the word respect the presence of stop words, although they are not stored
 - Stores positional information for proximity info



Highlighting

- `ts_headline`
 - it doesn't use tsvector and needs to use the entire document, so could be expensive.
- Only for certain type of queries or titles

```
postgres=# SELECT ts_headline('english','Just a simple example of a highlighted query and similarity.',  
to_tsquery('query & similarity'),'StartSel = <, StopSel = >');  
          ts_headline  
-----
```

Just a simple example of a highlighted <query> and <similarity>. (1 row)

Default:

`StartSel=, StopSel=, MaxWords=35, MinWords=15, ShortWord=3, HighlightAll=FALSE, MaxFragments=0, FragmentDelimiter=" ... "`



Ranking

- Weights: (A B C D)
- Ranking functions:
 - ts_rank
 - ts_rank_cd
- Ranking is expensive cause re process and check each tsvector.

```
SELECT to_tsquery('english', 'Fat | Rats:AB');
```

```
to_tsquery
```

```
'fat' | 'rat':AB
```

Also, * can be attached to a lexeme to specify prefix matching:

```
SELECT to_tsquery('supern:*A & star:A*B');
```

```
to_tsquery
```

```
'supern':*A & 'star':*AB
```



Manipulating tsvectors and tsquery

- Manipulating tsvectors
 - `setweight(vector tsvector, weight "char")` returns tsvector
 - `length (tsvector)` : number of lexemes
 - `strip (tsvector)`: returns tsvector without additional position as weight or position
- Manipulating Queries
- If you need a dynamic input for a query, parse it with *numnode(tsquery)*, it will avoid unnecessary searches if contains a lot of stop words
 - `numnode(plainto_tsquery('a the is'))`
 - clean the queries using `querytree` also, is useful



Example

```
postgres=# select * from ts_debug('english','The doctor says I"m sick.');?>
   alias  |  description  | token  |  dictionaries  | dictionary  | lexemes
-----+-----+-----+-----+-----+
asciiword | Word, all ASCII | The | {english_stem} | english_stem | {}
blank    | Space symbols  | {}    |             |             |
asciiword | Word, all ASCII | doctor | {english_stem} | english_stem | {doctor}
blank    | Space symbols  | {}    |             |             |
asciiword | Word, all ASCII | saysd | {english_stem} | english_stem | {said}
blank    | Space symbols  | {}    |             |             |
asciiword | Word, all ASCII |  I  | {english_stem} | english_stem | {}
blank    | Space symbols  | '   | {}    |             |
asciiword | Word, all ASCII | m   | {english_stem} | english_stem | {m}
blank    | Space symbols  | .   | {}    |             |
asciiword | Word, all ASCII | sick | {english_stem} | english_stem | {sick}
blank    | Space symbols  |     | {}    |             |
(12 rows)
```

```
postgres=# select numnode(plainto_tsquery('The doctor says I"m sick.')), plainto_tsquery('The doctor says I"m sick.'),  
to_tsvector('english','The doctor says I"m sick.'), ts_lexize('english_stem','The doctor says I"m sick.');
numnode |      plainto_tsquery      |      to_tsvector      |      ts_lexize
-----+-----+-----+
 7 | 'doctor' & 'said' & 'm' & 'sick' | 'doctor':2 'm':5 'said':3 'sick':6 | {"the doctor says i'm sick."}
(1 row)
```



Manipulating tsquery

```
postgres=# SELECT querytree(to_tsquery('!defined'));  
querytree
```

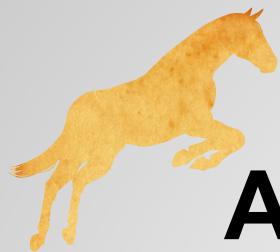
```
-----  
T  
(1 row)
```

```
postgres=# SELECT querytree(to_tsquery('cat & food | (dog & run & food)'));  
querytree
```

```
-----  
'cat' & 'food' | 'dog' & 'run' & 'food'  
(1 row)
```

```
postgres=# SELECT querytree(to_tsquery('the '));  
NOTICE: text-search query contains only stop words or doesn't contain lexemes, ignored  
querytree
```

```
-----  
(1 row)
```



Automating updates on tsvector

- Postgresql provide standard functions for this:
 - `tsvector_update_trigger(tsvector_column_name, config_name, text_column_name [, ...])`
 - `tsvector_update_trigger_column(tsvector_column_name, config_column_name, text_column_name [, ...])`

```
CREATE TABLE messages (
```

```
title text,
```

```
body text,
```

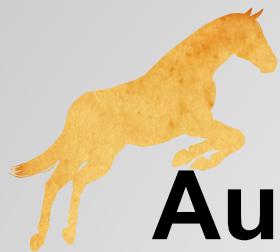
```
tsv tsvector
```

```
);
```

```
CREATE TRIGGER tsvectorupdate BEFORE INSERT OR UPDATE
```

```
ON messages FOR EACH ROW EXECUTE PROCEDURE
```

```
tsvector_update_trigger(tsv, 'pg_catalog.english', title, body);
```

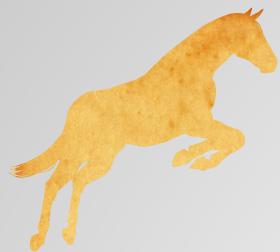


Automating updates on tsvector (2)

If you want to keep a custom weight:

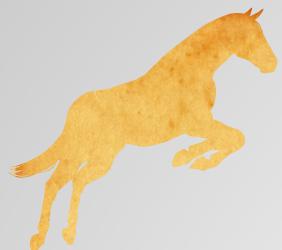
```
CREATE FUNCTION messages_trigger() RETURNS trigger AS $$  
begin  
new.tsv :=  
setweight(to_tsvector('pg_catalog.english', coalesce(new.title,'')), 'A') ||  
setweight(to_tsvector('pg_catalog.english', coalesce(new.body,'')), 'D');  
return new;  
end  
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER tsvectorupdate BEFORE INSERT OR UPDATE  
ON messages FOR EACH ROW EXECUTE PROCEDURE messages_trigger();
```



Tips and considerations

- Store the text externally, index on the database
 - requires superuser
- Store the whole document on the database, index on Sphinx/Solr
- Don't index everything
 - Solr /Sphinx are not databases, just index only what you want to search.
Smaller indexes are faster and easy to maintain.
- ts_stats
 - can help you out to check your FTS configuration
- You can parse URLs, mails and whatever using ts_debug function for non intensive operations



Tips and considerations

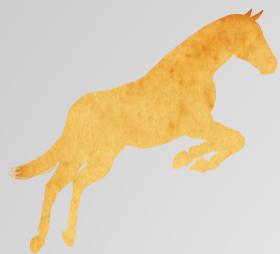
- You can index by language

```
CREATE INDEX pgweb_idx_en ON pgweb USING gin(to_tsvector('english', body))  
WHERE config_language = 'english';
```

```
CREATE INDEX pgweb_idx_fr ON pgweb USING gin(to_tsvector('french', body))  
WHERE config_language = 'french';
```

```
CREATE INDEX pgweb_idx ON pgweb USING gin(to_tsvector(config_language,  
body));
```

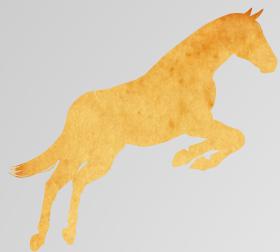
```
CREATE INDEX pgweb_idx ON pgweb USING gin(to_tsvector('english', title || ' ' ||  
body));
```



Features on 9.2

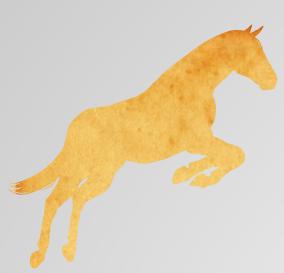
- Move tsvector most-common-element statistics to new [pg_stats](#) columns (Alexander Korotkov)
- Consult `most_common_elems` and `most_common_elem_freqs` for the data formerly available in `most_common_vals` and `most_common_freqs` for a tsvector column.

```
most_common_elems      | {exampl}
most_common_elem_freqs | {1,1,1}
```

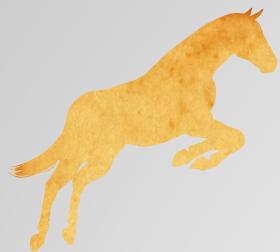


Links

- <http://www.postgresql.org/docs/9.2/static/textsearch.htm>
- <http://www.postgresql.org/docs/9.2/static/textsearch-migration.html> > migration from version pre-8.3



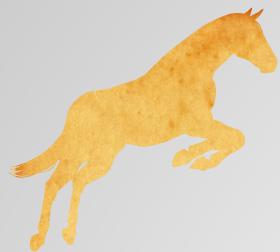
Sphinx



Sphinx

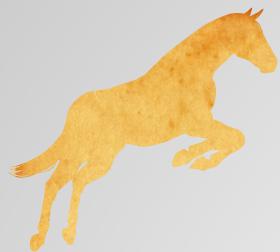
- Standalone daemon written on C++
- Highly scalable
 - Known installation consists 50+ Boxes, 20+ Billions of documents
- Extended search for text and non-full-text data
 - Optimized for faceted search
 - Snippets generation based on language settings
- Very fast
 - Keeps attributes in memory
 - See Percona benchmarks for details
- Receiving data from PostgreSQL
 - Dedicated PostgreSQL datasource type.

<http://sphinxsearch.com>



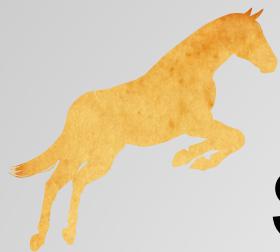
Key features- Sphinx

- Scalability & failover
- Extended FT language
 - Faceted search support
 - GEO-search support
- Integration and pluggable architecture
 - Dedicated PostgreSQL source, UDF support
- Morphology & stemming
- Both batch & real-time indexing is available
- Parallel snippets generation



What's new on Sphinx

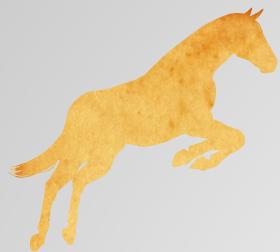
- 1. added AOT (new morphology library, lemmatizer) support
 - Russian only for now; English coming soon; small 10-20% indexing impact; it's all about search quality (much much better "stemming")
- 2. added JSON support
 - limited support (limited subset of JSON) for now; JSON sits in a column; you're able to do thing like WHERE jsoncol.key=123 or ORDER BY or GROUP BY
- 3. added subselect syntax that reorders result sets, SELECT * FROM (SELECT ... ORDER BY cond1 LIMIT X) ORDER BY cond2 LIMIT Y
- 4. added bigram indexing, and quicker phrase searching with bigrams (bigram_index, bigram_freq_words directives)
 - improves the worst cases for social mining
- 5. added HA support, ha_strategy, agent_mirror directives
- 6. added a few new geofunctions (POLY2D, GEOPOLY2D, CONTAINS)
- 7. added GROUP_CONCAT()
- 8. added OPTIMIZE INDEX rtindex, rt_merge_iops, rt_merge_maxiosize directives
- 9. added TRUNCATE RTINDEX statement



Sphinx - Postgres compilation

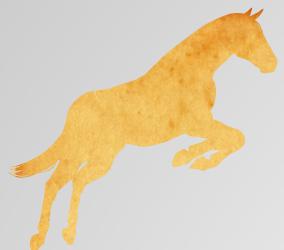
```
[root@ip-10-55-83-238 ~]# yum install gcc-c++.noarch  
[root@ip-10-55-83-238 sphinx-2.0.6-release]# ./configure --prefix=/opt/sphinx --  
without-mysql --with-pgsql-includes=$PGSQL_INCLUDE --with-pgsql-  
libs=$PGSQL_LIBS --with-pgsql  
[root@ip-10-55-83-238 sphinx]# /opt/pg/bin/psql -Upostgres -hmaster test <  
etc/example-pg.sql
```

* Package is compiled with mysql libraries dependencies

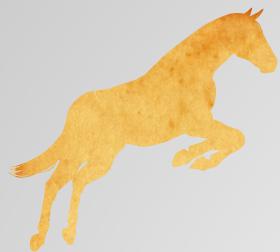


Sphinx - Daemon

- For speed
 - to offload main database
 - to make particular queries faster
 - Actually most of search-related
- For failover
 - It happens to best of us!
- For extended functionality
 - Morphology & stemming
 - Autocomplete, “do you mean” and “Similar items”

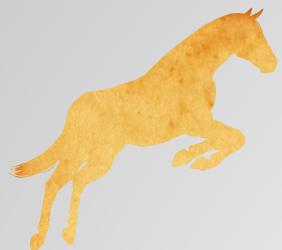


SOLR/Lucene



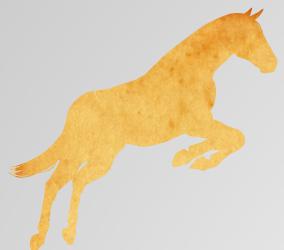
Solr Features

- Advanced Full-Text Search Capabilities
- Optimized for High Volume Web Traffic
- Standards Based Open Interfaces - XML, JSON and HTTP
- Comprehensive HTML Administration Interfaces
- Server statistics exposed over JMX for monitoring
- Linearly scalable, auto index replication, auto failover and recovery
- Near Real-time indexing
- Flexible and Adaptable with XML configuration
- Extensible Plugin Architecture



Solr

- <http://lucene.apache.org/solr/features.html>
- Solr uses Lucene Library



Thanks!

Contact us!
We are hiring!
emanuel@palominodb.com