

# Sphinx

## What is Sphinx?

“Sphinx is a full-text search engine, distributed under GPL version 2. Commercial license is also available for embedded use.”



# What It Does

- Creates an external inverted index for any data source
- Searches return an id to be looked up

doctor → ids [1,2,3 ]  
venture → ids [ 1,2,3 ]  
brock → ids [ 4,5,6 ]  
samson → ids [ 4,5,6 ]



# Features

- Blazing fast indexing and searching speed
- Data index and attributes are stored in RAM, so access is lightening fast
- For extremely large datasets, all data can be kept on disk



# Features

- Allows flexibility with how the data is indexed / retrieved
- Can be built into MySQL as a storage engine
- Distributed searches are built in
- API's in almost any language



# Features

- Top speed  $> 20,000$  docs per second indexed, searches for rare terms in 1/2 gig of data finish  $< .01$  seconds.
- Speed depends on size of the documents, filters in use, indexes being searched



# When to Use Sphinx

- Free-form queries that might search any/all columns in a DB
- Would lead to a WHERE clause with many "col1 LIKE 'search' OR col2 LIKE 'search' ..." which is slow



# When to Use Sphinx

- Queries using a LIKE operator take a significant toll on the DB server



# Using Sphinx

1. Build SQL queries that will select the data you want to index
2. Decide on any attributes that you need (for filtering, grouping, sorting)
3. Build Sphinx config using Sphinx::Config
4. Run indexer (make sure searchd is running first)  
'indexer --rotate --all'
5. Query the index with Sphinx::Search and fetch result docs from DB



# Using Sphinx

- Steps 1-4 only have to be done when data is added or modified.
- This means rapidly changing data needs to be done in batches



# Data Sources

- Most common source is MySQL or ODBC database query
- Fine-grained control is possible: throttling, specific row ranges
- Generic data can be fed into Sphinx via xmlpipe



# Data Sources

Example: Build your own Google with  
WWW::Mechanize → XML → Sphinx



# Data Sources

```
<sphinx:docset>  
  <sphinx:schema>  
    <sphinx:field name="url"/>  
    <sphinx:field name="content"/>  
    <sphinx:attr name="published" type="timestamp"/>  
  </sphinx:schema>  
  <sphinx:document id="1234">  
    <url>www.google.com</url>  
    <content>Google homepage content</content>  
  </sphinx:document>  
</sphinx:docset>
```



# Fields & Attributes

## Fields

- Fields are the content to be indexed
- You can limit a search to just a given field
- e.g. “@field search\_value”



# Fields & Attributes

## Attributes

- Attributes are document properties
- Not searched on, but returned (though you can if you have to)
- Provide the means for grouping and ordering results
- Attribute types include: integer, timestamp, Multi-Value Attribute (MVA), and (new!) String



# Fields & Attributes

Process is:

1. Search index fields for hits (all fields by default)
2. Filter results on attr (optional)
3. Group results on attr (optional)



# Perl API

Modules by Jon Schutz:

- Sphinx::Config
- Sphinx::Search
- Sphinx::Manager



# Perl API

Synopsis:

```
use Sphinx::Config;
my $c = Sphinx::Config->new();
$c->parse($filename);
$path = $c->get('index', 'test1', 'path');
$c->set('index', 'test1', 'path', $path);
$c->save($filename);
```



# Perl API

Synopsis:

```
use Sphinx::Search;  
$sphinx = Sphinx::Search->new();  
$results = $sphinx  
  ->SetMatchMode(SPH_MATCH_ALL)  
  ->SetSortMode(SPH_SORT_RELEVANCE)  
  ->Query("search terms");
```



# Perl API

Not quite done:

- Still need to retrieve the actual content from the original data source

```
$sth = $dbh->prepare("SELECT * FROM table WHERE id=?");  
foreach my $match (@{ $results->{matches} }){  
    $sth->execute($match->{doc});  
    print Dumper($sth->fetchrow_hashref);  
}
```



# Demo

Problem: Web backend spending all its time on expensive queries against varchar/text columns

Solution: Index those columns with Sphinx, point search form to Sphinx

Example: GeoIP database with 3.3 Million rows



# Tricks

- Add chars to the charset to change 1.2.3.4 from "1<ignored> 2<ignored> 3<ignored> 4" to the full string "1.2.3.4"
- Charsets can be changed on a per-index basis



# Tricks

- Generate unique id by using numberspaces:  
primary\_key\_part\_1: 25  
primary\_key\_part\_2: 33
- Docid = primary\_key\_part\_1 \* 1,000,000 +  
primary\_key\_part\_2 = 25,000,033



# Limitations

- Indexes cannot be updated incrementally or in real-time, only in batches
- Wildcard (infix and prefix) searches are very slow to index, take up a lot of space, and make searches slower.
- “TEST” infix indexed as “T”, “TE”, “TES”, “TEST”, “EST”, “ST”
- Prefixes are a faster alternative, but also carry a penalty
- Workaround: Only make certain columns wildcard searchable, or only have the wildcard at the end of the word (stuff\* versus \*stuff\*)



# Limitations

- Docid **\*MUST\*** be an unsigned integer. This can be a bit of a challenge to create if it's not already inherent to the data.



# Alternatives

- Lucene: The granddaddy of open-source full-text search
- MySQL fulltext search (only available for MyISAM FULLTEXT columns)
- Native DB WHERE clauses with Search::QueryParser

