Full Text Search Throwdown

Bill Karwin, Percona Inc.

In a full text search, the search engine examines all of the words in every stored document as it tries to match search words supplied by the user.

http://www.flickr.com/photos/tryingyouth/

StackOverflow Test Data

- Data dump, exported December 2011
- 7.4 million Posts = 7.5 GB

stackoverflow | Questions | Tags | Users | Badges | Unanswered | Ask Question

Bill Karwin less info

edit prefs flair my logins | network profile

104,700
reputation

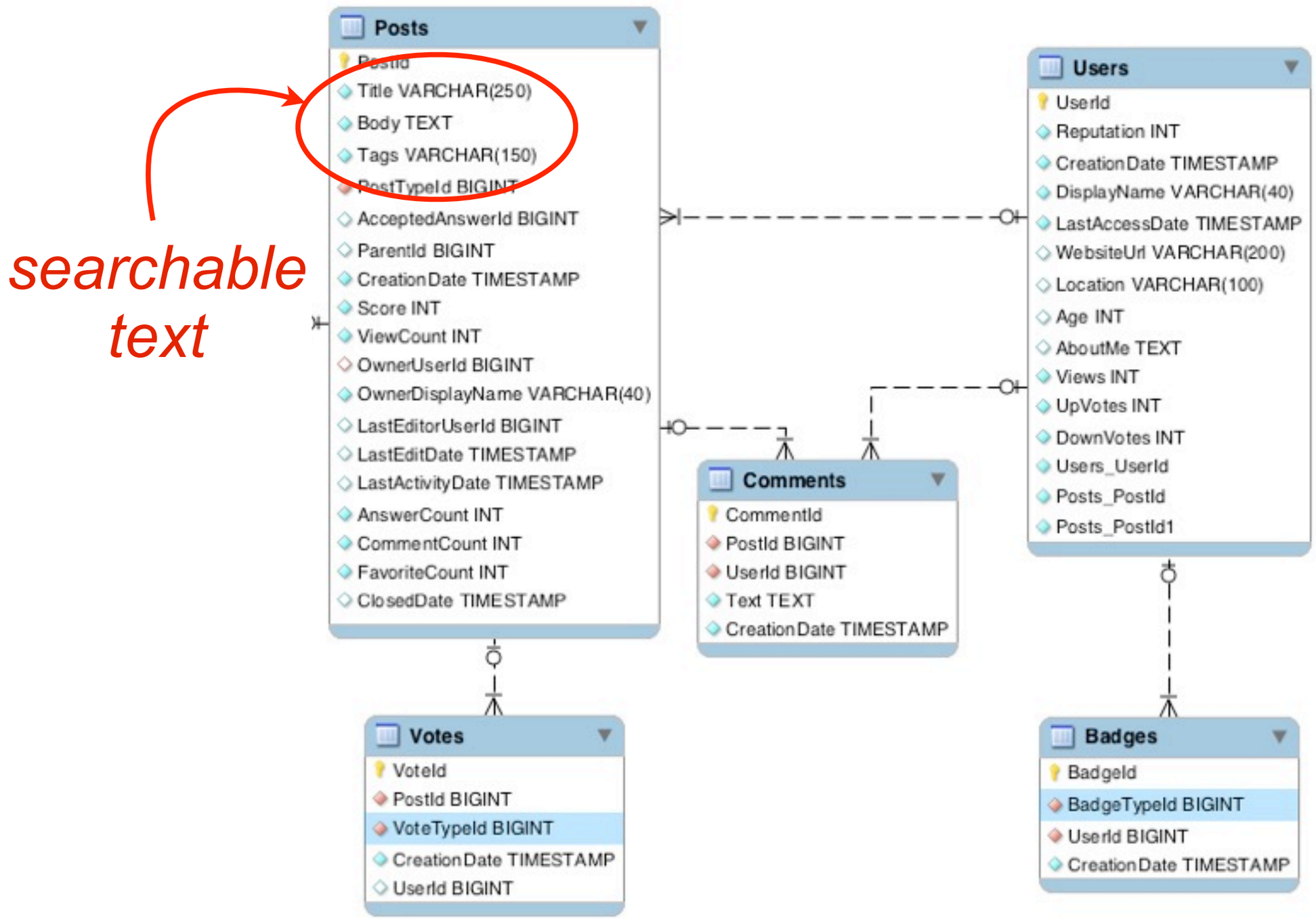
•15 •114 •254

bio	website	karwin.com
	location	California
	email	bill@karwin.com
	real name	Bill Karwin
	age	44
visits	member for	3 years, 6 months
	visited	1004 days, 36 consecutive
	seen	just now
stats	profile views	12,150

I'm a MySQL Performance Consultant working at Percona, Inc. I've been a software engineer since 1987, and my specialty is as an SQL maven. I also have experience programming in Java, PHP, Perl, C, JavaScript, and I have many other coding skills.

I've written a book, SQL Antipatterns: Avoiding the Pitfalls of Database Programming from Pragmatic Bookshelf, based on the most common SQL problems I've answered on Stack Overflow and other forums, mailing lists, and newsgroups over the past 15 years.

StackOverflow ER diagram



The Baseline:
Naive Search Predicates

Some people, when confronted with a problem, think

"I know, I'll use regular expressions."

Now they have two problems.

— Jamie Zawinsky

Accuracy issue

- Irrelevant or false matching words ‘one’, ‘money’, ‘prone’, etc.:

  WHERE Body LIKE ‘%one%’

- Regular expressions in MySQL support escapes for word boundaries:

  WHERE Body RLIKE ‘[[:<:]]one[[:>:]]’

Performance issue

- LIKE with wildcards:

```
SELECT * FROM Posts
  WHERE title LIKE ‘%performance%‘
      OR body LIKE ‘%performance%’
      OR tags LIKE ‘%performance%’;
```

1 min 41 sec

- POSIX regular expressions:

```
SELECT * FROM Posts
  WHERE title RLIKE ‘performance’
      OR body RLIKE ‘performance’
      OR tags RLIKE ‘performance’;
```

7 min 39 sec

Why so slow?

```
CREATE TABLE TelephoneBook (
    FullName VARCHAR(50)
);

CREATE INDEX name_idx ON TelephoneBook
    (FullName);

INSERT INTO TelephoneBook VALUES
    ('Riddle, Thomas'),
    ('Thomas, Dean');
```

Why so slow?

- Search for all with last name “Thomas”

```
SELECT * FROM telephone_book
  WHERE full_name LIKE ‘Thomas%’
```

uses index

- Search for all with first name “Thomas”

```
SELECT * FROM telephone_book
  WHERE full_name LIKE ‘%Thomas’
```

can’t use index

B-Tree indexes can't search for substrings

- FULLTEXT in MyISAM
- FULLTEXT in InnoDB
- Sphinx Search
- Apache Solr
- Trigraphs

FULLTEXT in MyISAM

FULLTEXT Index with MyISAM

- Special index type for MyISAM
- Integrated with SQL queries
- Indexes always in sync with data
- Balances features vs. speed vs. space

Build Index on Data

```
CREATE FULLTEXT INDEX PostText
    ON Posts(Title, Body, Tags);
```

time: 52 min 28 sec

Insert Data into Index

```
INSERT INTO Posts
  SELECT * FROM PostsSource;
```

time: 53 min 45 sec

Querying

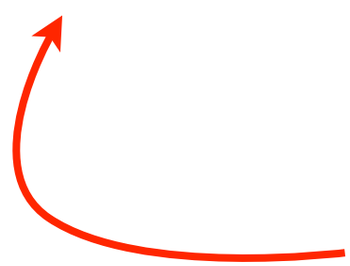
SELECT * FROM Posts
WHERE MATCH(*column(s)*)
AGAINST('*query pattern*');

must include all columns of your index, in the order you defined

Natural Language Mode (MyISAM)

- Searches concepts with free text queries:

```
SELECT * FROM Posts
  WHERE MATCH( Title, Body, Tags )
  AGAINST('mysql performance'
  IN NATURAL LANGUAGE MODE)
  LIMIT 100;
```



time with index:
150 milliseconds

Query Profile: Natural Language Mode (MyISAM)

Status	Duration
starting	0.000064
checking permissions	0.000006
Opening tables	0.000020
System lock	0.000009
init	0.000028
optimizing	0.000009
statistics	0.000015
preparing	0.000008
FULLTEXT initialization	0.149113
executing	0.000012
Sending data	0.000910
end	0.000005
query end	0.000003
closing tables	0.000012
freeing items	0.000413
cleaning up	0.000003

Boolean Mode (MyISAM)

- Searches words using mini-language:

```
SELECT * FROM Posts
  WHERE MATCH( Title, Body, Tags )
AGAINST('+mysql +performance'
IN BOOLEAN MODE)
LIMIT 100;
```

time with index:
3 milliseconds

Query Profile: Boolean Mode (MyISAM)

Status	Duration
starting	0.000357
checking permissions	0.000007
Opening tables	0.000017
System lock	0.000008
init	0.000018
optimizing	0.000008
statistics	0.000016
preparing	0.000007
FULLTEXT initialization	0.000014
executing	0.000003
Sending data	0.002603
end	0.000004
query end	0.000002
closing tables	0.000007
freeing items	0.000026
cleaning up	0.000001

FULLTEXT in InnoDB

FULLTEXT Index with InnoDB

- Under development in MySQL 5.6
- Usage very similar to FULLTEXT in MyISAM

Build Index on Data (InnoDB)

```
CREATE FULLTEXT INDEX PostText
    ON Posts(Title, Body, Tags);
```

time: 39 min 25 sec

Insert Data into Index (InnoDB)

```
INSERT INTO Posts
  SELECT * FROM PostsSource;
```

time: 63 min 4 sec

Natural Language Mode (InnoDB)

- Searches concepts with free text queries:

```
SELECT * FROM Posts
  WHERE MATCH( Title, Body, Tags )
  AGAINST('mysql performance'
  IN NATURAL LANGUAGE MODE)
  LIMIT 100;
```

time with index:
691 milliseconds

Query Profile: Natural Language Mode (InnoDB)

Status	Duration
starting	0.000064
checking permissions	0.000006
Opening tables	0.000018
System lock	0.000008
init	0.000028
optimizing	0.000008
statistics	0.000016
preparing	0.000007
FULLTEXT initialization	0.538327
executing	0.000011
Sending data	0.095616
end	0.000011
query end	0.000004
closing tables	0.000011
freeing items	0.057301
cleaning up	0.000014

Boolean Mode (InnoDB)

- Searches words using mini-language:

```
SELECT * FROM Posts
  WHERE MATCH( Title, Body, Tags )
AGAINST('+mysql +performance'
IN BOOLEAN MODE)
LIMIT 100;
```

time with index:
306 milliseconds

Query Profile: Boolean Mode (InnoDB)

Status	Duration
starting	0.000064
checking permissions	0.000006
Opening tables	0.000024
System lock	0.000008
init	0.000028
optimizing	0.000009
statistics	0.000016
preparing	0.000008
FULLTEXT initialization	0.300187
executing	0.000011
Sending data	0.004977
end	0.000008
query end	0.000023
closing tables	0.000011
freeing items	0.001045
cleaning up	0.000006

Apache Solr

Apache Solr

- Formerly known as Lucene
- Lucene Project started 2001
- Apache License
- Java implementation
- Web service architecture
- Many sophisticated search features

DataImportHandler

conf/solrconfig.xml:

```
. . .
<requestHandler name="/dataimport"
    class="org.apache.solr.handler.dataimport.DataImportHandler">
  <lst name="defaults">
    <str name="config">data-config.xml</str>
  </lst>
</requestHandler>
. . .
```

DataImportHandler

conf/data-config.xml:

```
<dataConfig>
  <dataSource type="JdbcDataSource"
          driver="com.mysql.jdbc.Driver"
          url="jdbc:mysql://localhost/testpattern?useUnicode=true"
          batchSize="-1"
          user="xxxx"
          password="xxxx"/>
  <document>
    <entity name="id"
          query="SELECT PostId, ParentId, Title, Body, Tags FROM Posts">
    </entity>
  </document>
</dataConfig>
```

extremely important to avoid buffering the whole query result!

DataImportHandler

conf/schema.xml:

```
. . .
<fields>
  <field name="PostId" type="string" indexed="true" stored="true" required="true" />
   <field name="ParentId" type="string" indexed="true" stored="true" required="false" />
   <field name="Title" type="text_general" indexed="false" stored="false" required="false" />
   <field name="Body" type="text_general" indexed="false" stored="false" required="false" />
   <field name="Tags" type="text_general" indexed="false" stored="false" required="false" />

   <field name="text" type="text_general" indexed="true" stored="false" multiValued="true"/>
<fields>

<uniqueKey>PostId</uniqueKey>
<defaultSearchField>text</defaultSearchField>

<copyField source="Title" dest="text"/>
<copyField source="Body" dest="text"/>
<copyField source="Tags" dest="text"/>
. . .
```

Insert Data into Index (Solr)

- http://localhost:8983/solr/dataimport?command=full-import

time: 14 min 28 sec

DataImportHandler

- http://localhost:8983/solr/select/?q=mysql+AND+performance

time: 1-79ms

Query results are cached (like MySQL Query Cache), so they return much faster on a second execution

Sphinx Search

Sphinx Search

- Started in 2001
- GPLv2 license
- C++ implementation
- SphinxSE storage engine for MySQL
- Supports MySQL protocol, SQL-like queries
- Many sophisticated search features

sphinx.conf

```
source src1
  {
     type = mysql
     sql_host = localhost
     sql_user = xxxx
     sql_pass = xxxx
     sql_db = testpattern
     sql_query = SELECT PostId, ParentId, Title,
        Body, Tags FROM Posts
     sql_query_info = SELECT * FROM Posts \
        WHERE PostId=$id
  }
```

sphinx.conf

```
index test1
  {
     source = src1
     path = C:\Sphinx\data
  }
```

Insert Data into Index (Sphinx)

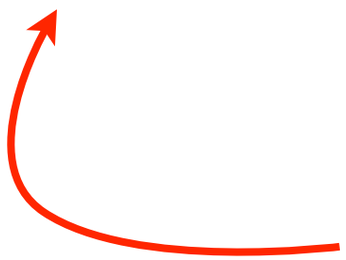
C:\Sphinx> indexer.exe -c sphinx.conf.in --verbose test1

```
Sphinx 2.0.4-release (r3135)

using config file 'sphinx.conf.in'...
indexing index 'test1'...
collected 7397507 docs, 5766.1 MB
sorted 925.5 Mhits, 100.0% done
total 7397507 docs, 5766129897 bytes
total 596.181 sec, 9671776 bytes/sec, 12408.15 docs/sec
total 48 reads, 4.935 sec, 75395.7 kb/call avg, 102.8 msec/call avg
total 7038 writes, 5.392 sec, 1021.5 kb/call avg, 0.7 msec/call avg

Execution time: 596.205 s
```

time: 9 min 56 sec

Querying index

```
search -c sphinx.conf.in
   -i test1 -b “mysql & performance”
```



time: 16ms

Trigraphs

Three-Letter Sequences

```
CREATE TABLE AtoZ (
    c       CHAR(1),
    PRIMARY KEY (c));

INSERT INTO AtoZ (c) VALUES ('a'), ('b'), ('c'), ...

CREATE TABLE Trigraphs (
    Tri     CHAR(3),
    PRIMARY KEY (Tri));

INSERT INTO Trigraphs (Tri)
SELECT CONCAT(t1.c, t2.c, t3.c)
FROM AtoZ t1 JOIN AtoZ t2 JOIN AtoZ t3;
```

Joined to Documents

```
CREATE TABLE PostsTrigraph (
    Tri          CHAR(3),
    PostId       INT UNSIGNED,
    PRIMARY KEY (Tri, PostId)
);

INSERT INTO PostsTrigraph (Tri, PostId)
SELECT t.Tri, p.PostId
FROM Trigraphs t JOIN Posts p ON
    CONCAT(p.Title, p.Body, p.Tags)
    LIKE CONCAT('%', t.Tri, '%');
```

this takes a very long time!

Insert Data Into Index

```
my $sth = $dbh1->prepare("SELECT * FROM Posts") or die $dbh1->errstr;
$sth->execute() or die $dbh1->errstr;
$dbh2->begin_work;
my $i = 0;
while (my $row = $sth->fetchrow_hashref ) {
  my $text = lc(join('|', ($row->{title}, $row->{body}, $row->{tags})));
  my %tri;
  map($tri{$_}=1, ( $text =~ m/[[:alpha:]]{3}/g ));
  next unless %tri;
  my $tuple_list = join(",", map("('$_',$row->{postid})", keys %tri));
  my $sql = "INSERT IGNORE INTO PostsTrigraph (tri, PostId) VALUES $tuple_list";
  $dbh2->do($sql) or die "SQL = $sql, ".$dbh2->errstr;
  if (++$i % 1000 == 0) {
    print ".";
    $dbh2->commit;
    $dbh2->begin_work;
  }
}
print ".\n";
$dbh2->commit;
```

time: 116 min 50 sec

space: 16.2GiB

rows: 519 million

Indexed Lookups

```
SELECT p.*
FROM Posts p
JOIN PostsTrigraph t1 ON
    t1.PostId = p.PostId AND t1.Tri = 'mys'
```

time: 46 sec

Search Among Fewer Matches

time: 19 sec

```
SELECT p.*
FROM Posts p
JOIN PostsTrigraph t1 ON
    t1.PostId = p.PostId AND t1.Tri = 'mys'
JOIN PostsTrigraph t2 ON
    t2.PostId = p.PostId AND t2.Tri = 'per'
```

Search Among Fewer Matches

time: 22 sec

```
SELECT p.*
FROM Posts p
JOIN PostsTrigraph t1 ON
    t1.PostId = p.PostId AND t1.Tri = 'mys'
JOIN PostsTrigraph t2 ON
    t2.PostId = p.PostId AND t2.Tri = 'per'
JOIN PostsTrigraph t3 ON
    t3.PostId = p.PostId AND t3.Tri = 'for'
```

Search Among Fewer Matches

time: 13.6 sec

```
SELECT p.*
FROM Posts p
JOIN PostsTrigraph t1 ON
    t1.PostId = p.PostId AND t1.Tri = 'mys'
JOIN PostsTrigraph t2 ON
    t2.PostId = p.PostId AND t2.Tri = 'per'
JOIN PostsTrigraph t3 ON
    t3.PostId = p.PostId AND t3.Tri = 'for'
JOIN PostsTrigraph t4 ON
    t4.PostId = p.PostId AND t4.Tri = 'man'
```

Narrow Down Further

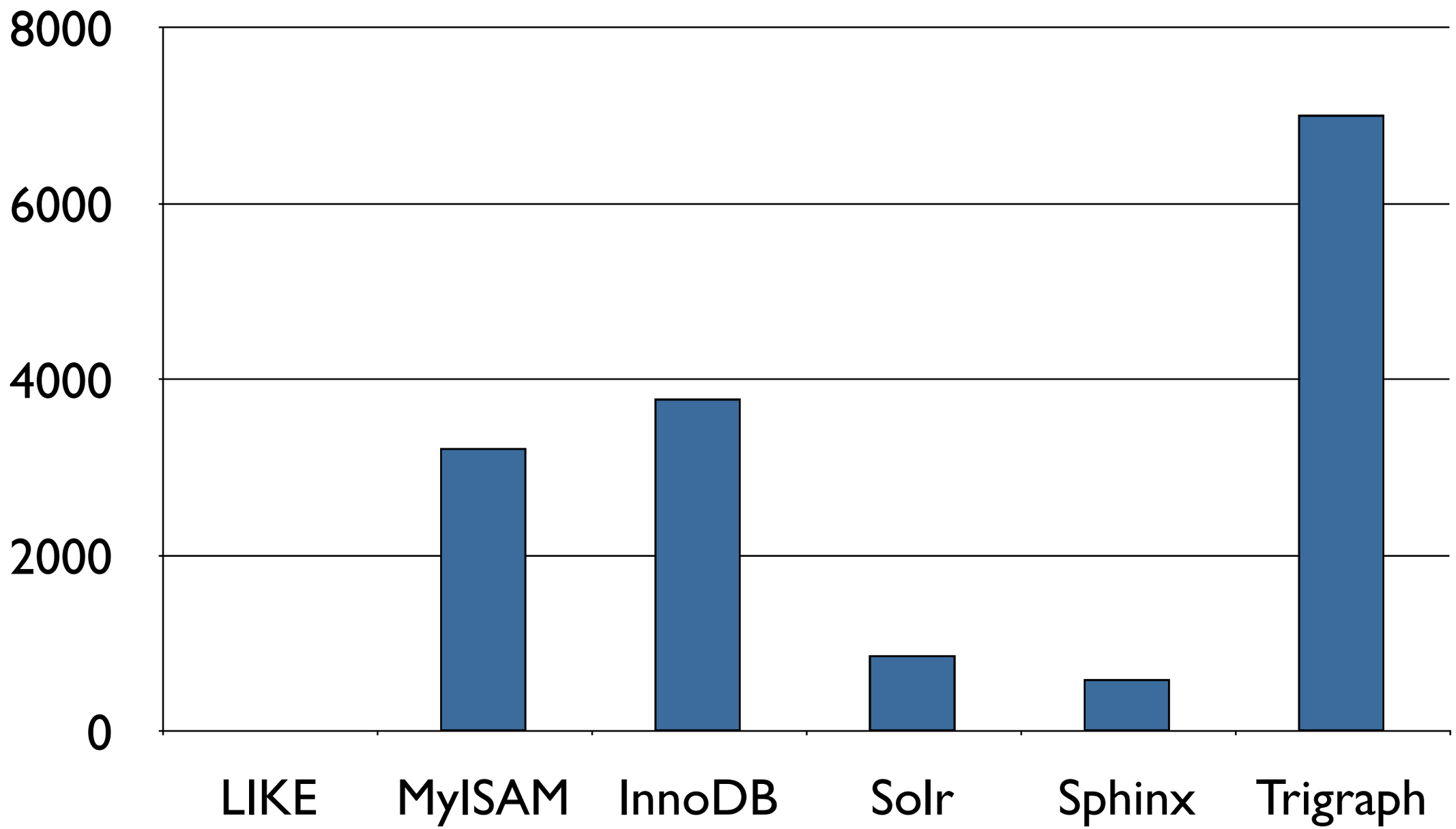
time: 13.8 sec

```
SELECT p.*
FROM Posts p
JOIN PostsTrigraph t1 ON
    t1.PostId = p.PostId AND t1.Tri = 'mys'
JOIN PostsTrigraph t2 ON
    t2.PostId = p.PostId AND t2.Tri = 'per'
JOIN PostsTrigraph t3 ON
    t3.PostId = p.PostId AND t3.Tri = 'for'
JOIN PostsTrigraph t4 ON
    t4.PostId = p.PostId AND t4.Tri = 'man'
WHERE CONCAT(p.Title, p.Body, p.Tags) LIKE '%mysql%'
    AND CONCAT(p.Title, p.Body, p.Tags) LIKE '%performance%';
```

Time to Insert Data into Index

LIKE expression	None
FULLTEXT MyISAM	53 min, 43 sec
FULLTEXT InnoDB	63 min, 4 sec
Apache Solr	14 min, 28 sec
Sphinx Search	9 min, 56 sec
Trigraphs	116 min, 50 sec

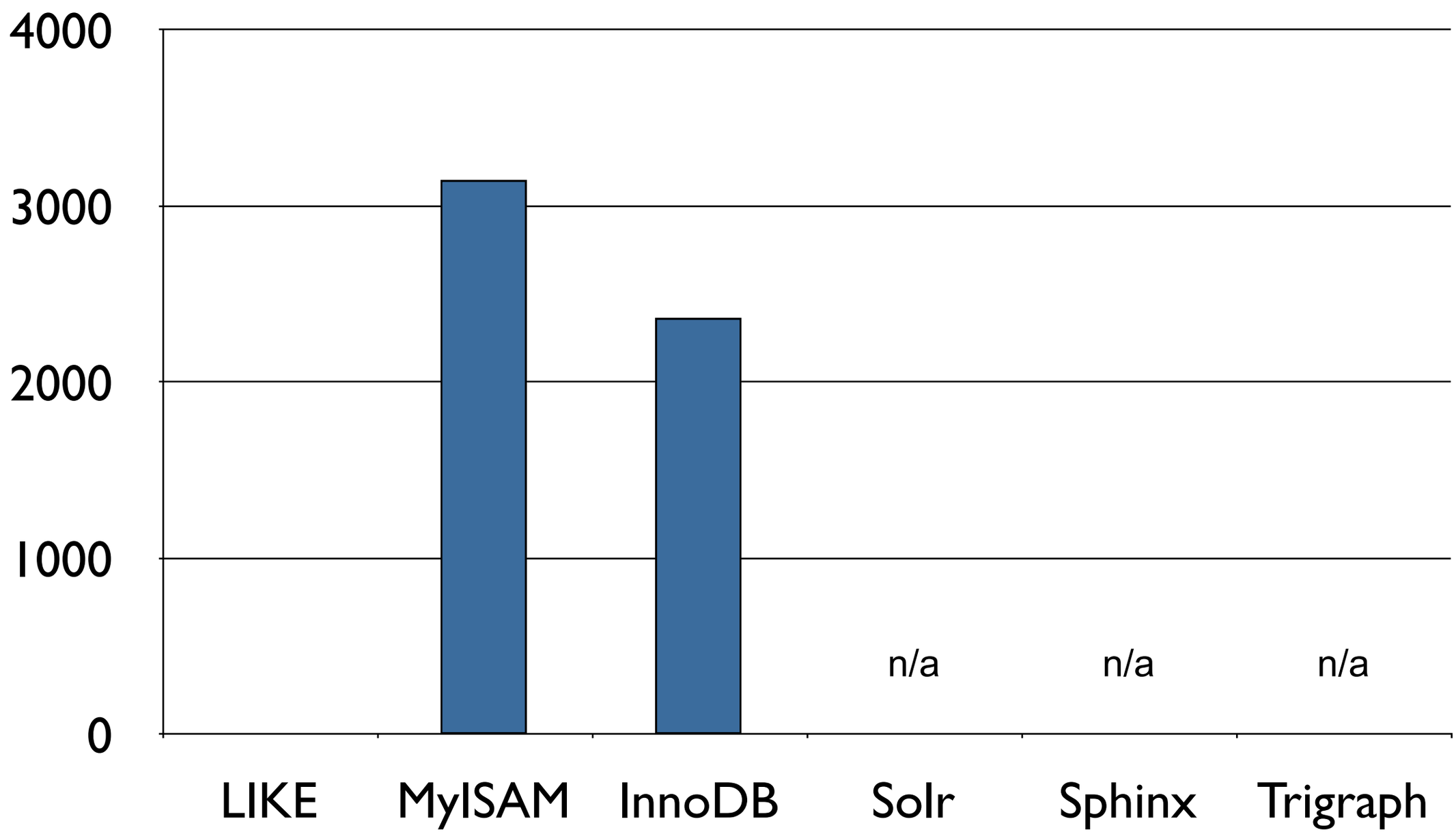
Time to Insert Data into Index



Time to Build Index on Data

LIKE expression	None
FULLTEXT MyISAM	52 min, 28 sec
FULLTEXT InnoDB	39 min, 25 sec
Apache Solr	n/a
Sphinx Search	n/a
Trigraphs	n/a

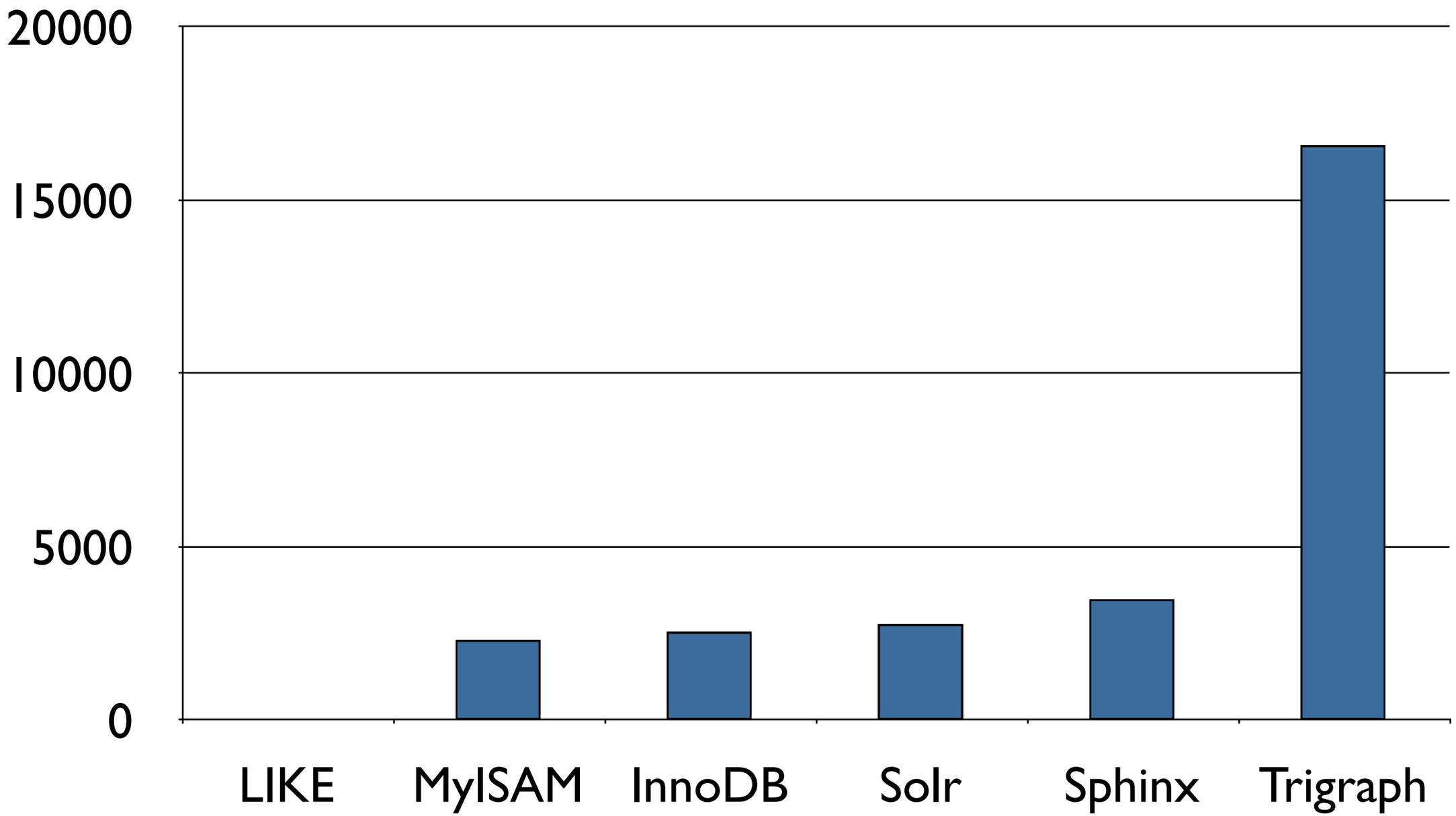
Time to Build Index on Data



Index Storage

LIKE expression	None
FULLTEXT MyISAM	2310MiB
FULLTEXT InnoDB	2544MiB
Apache Solr	2766MiB
Sphinx Search	3487MiB
Trigraphs	16580MiB

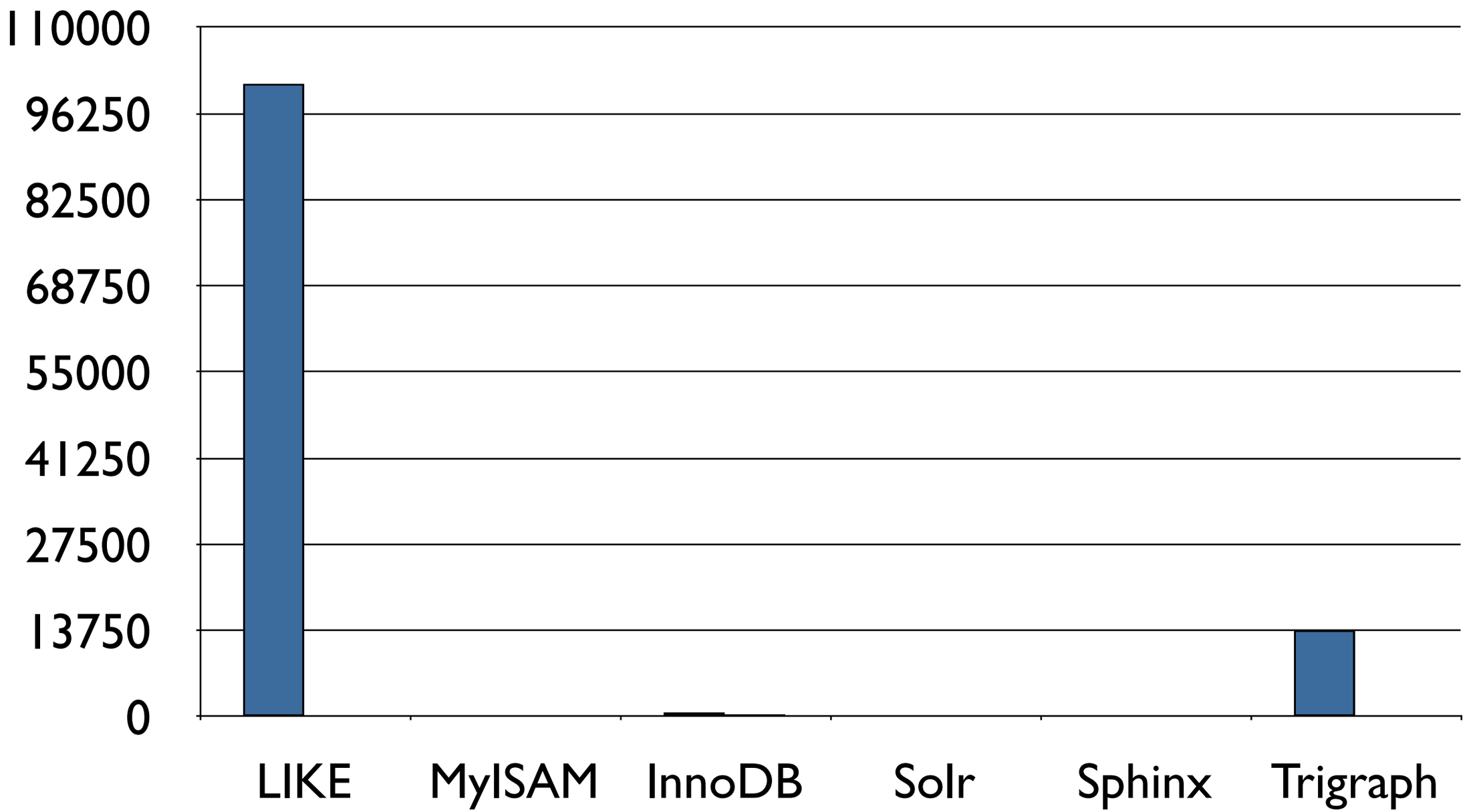
Index Storage



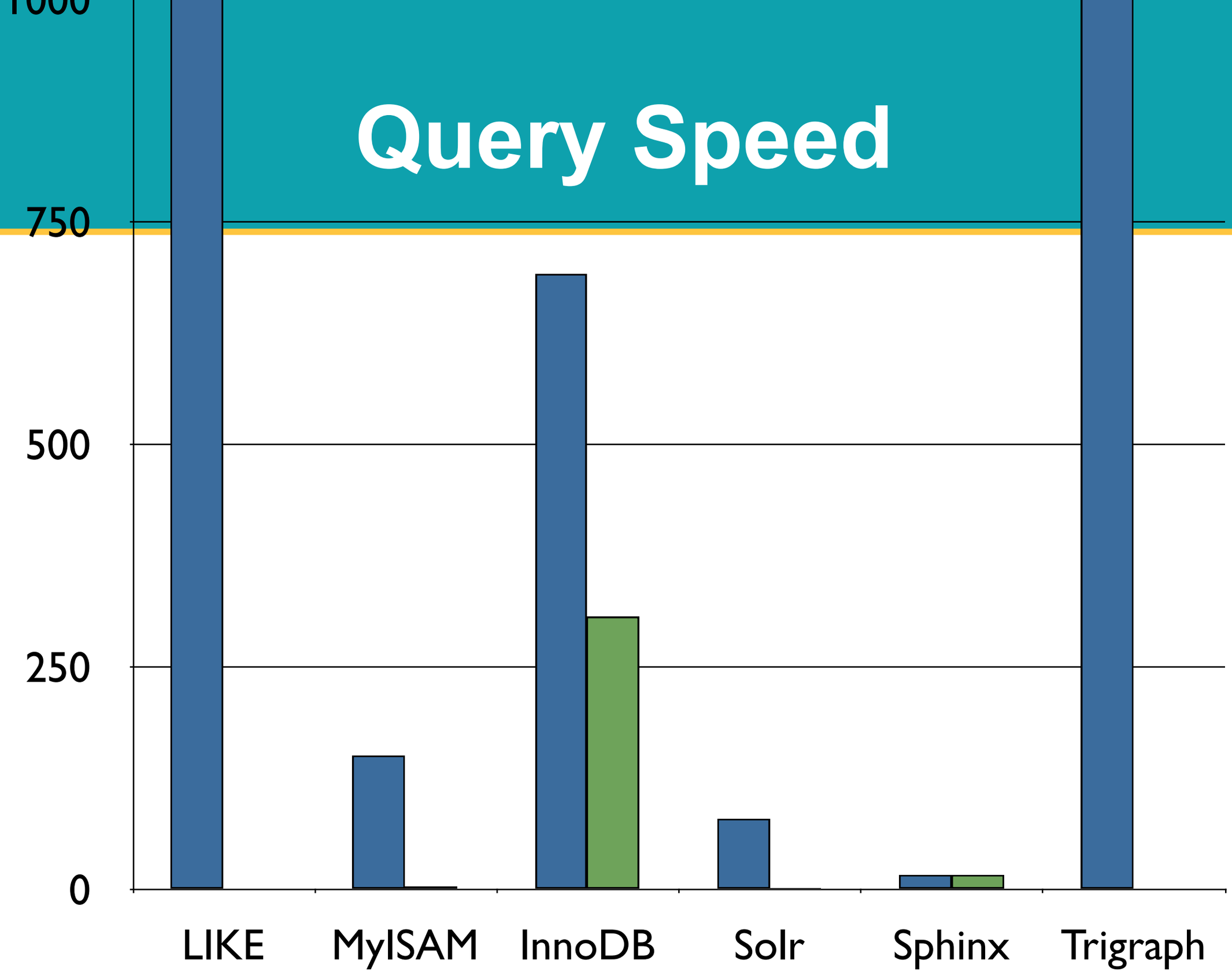
Query Speed

LIKE expression	1 min, 41 sec (101000ms)
FULLTEXT MyISAM	3 - 150ms
FULLTEXT InnoDB	306 - 691ms
Apache Solr	1 - 79ms
Sphinx Search	16ms
Trigraphs	13.8 sec

Query Speed


110000
96250
82500
68750
55000
41250
27500
13750
0
LIKE
MyISAM
InnoDB
Solr
Sphinx
Trigraph

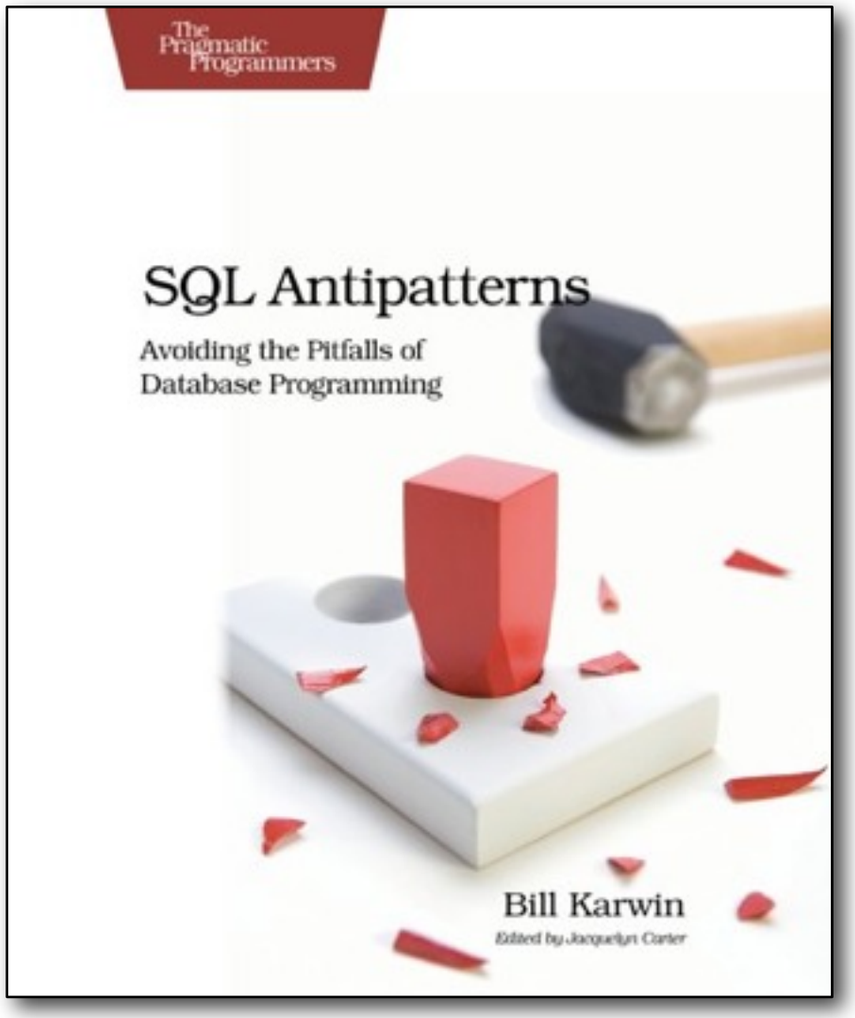
Query Speed



Bottom Line

	build	*insert*	*storage*	*query*	*solution*
LIKE expression	0	0	0	101000ms	SQL
FULLTEXT MyISAM	53:28	52:28	2310MiB	3-150ms	MySQL
FULLTEXT InnoDB	39:25	63:04	2545MiB	306-691ms	MySQL 5.6
Apache Solr	n/a	14:28	2766MiB	1-79ms	Java
Sphinx Search	n/a	9:56	3487MiB	16ms	C++
Trigraphs	n/a	116:50	16580MiB	13800ms	SQL

SQL Antipatterns



http://www.pragprog.com/titles/bksqla/