MySQL Fulltext Search

Sergei Golubchik

International PHP Conference
Frankfurt/Main, 2003

MySQL Fulltext Search: Session Outline

- Fulltext search in MySQL
  - ✓ overview
  - ✓ modes of operation
  - ✓ syntax
- Boolean Fulltext Search
  - ✓ purpose
  - ✓ operators
  - ✓ relevance ranking
- Natural Language Search
  - ✓ purpose
  - ✓ relevance ranking
  - ✓ query expansion *new!*
- Internal Structure of Fulltext Index
- TODO: directions of development

Who am I ?

- My name is Sergei Golubchik
- Originally from Ukraine, now living in Osnabrück
- Primary developer of MySQL Fulltext Search
- Full-time MySQL AB employee since March 2000
- Doing other things, besides fulltext search:
  - ✓ indexes in MERGE tables
  - ✓ bulk inserts
  - ✓ parallel repair
  - ✓ `HANDLER`
  - ✓ `ALTER TABLE ... ENABLE/DISABLE KEYS`
  - ✓ `INSERT ... ON DUPLICATE KEY UPDATE`
  - ✓ security@mysql.com
  - ✓ bug policeman

Fulltext search QuickPoll

How many of you...

- have used MySQL Fulltext Search in production ?
  - ✓ have at least tried using MySQL Fulltext search ?
  - ✓ have used fulltext search in other products ?
- are interested in boolean fulltext search ?
  - ✓ natural language search ?
  - ✓ in our future plans for fulltext search ?
- are interested in how to tune and optimize your fulltext search application ?
- are interested in how MySQL Fulltext Search works internally ?
  - ✓ have contributed code to Open Source products ?

History of MySQL Fulltext search

- 1995–1998
  - ✓ I was using various fulltext engines. No one could do complex queries on structured data
  - ✓ Relational DBMSes could do it perfectly, and SQL was very capable – but they had no fulltext search capabilities
  - ✓ MySQL RDBMS was Open Source and it was known to be very fast. It was MySQL 3.22.7
- Oct. 1998: First version of fulltext search engine for ISAM
- May 1999: It was rewritten for MyISAM
- Oct. 1999: First public release
- Jun. 2000: `MATCH ... AGAINST` syntax
- Dec. 2001: Boolean fulltext search went public
- Jan. 2003: Major rewrite of the index structure code
- Sep. 2003: Unicode support, query expansion

Problem to solve

- Having a collection of documents, quickly find documents, that
  - ✓ contain certain words, or
  - ✓ are *about* some topic
- Applications are:
  - ✓ digital libraries
  - ✓ text (e.g. mail, news) archives
  - ✓ you name it
- Requirements:
  - ✓ fully dynamic index (no need for periodical reindexing)
  - ✓ native SQL-like interface
  - ✓ fast inserts/updates and searches
  - ✓ reasonable effectiveness for natural language queries
  - ✓ configurable, but easy to use
  - ✓ moderate index size

Modes of Operation

MySQL Fulltext Search engine supports two different search modes:

- Natural Language Search:
  - ✔ `MATCH (col1, col2, ...) AGAINST ("query phrase" [ WITH QUERY EXPANSION ] )`
  - ✔ is supposed to find documents (rows) that are *about* the topic described in the *query phrase*
  - ✔ the query is a phrase in *natural human language* (e.g. English)
- Boolean Search
  - ✔ `MATCH (col1, col2, ...) AGAINST ("query expression" IN BOOLEAN MODE )`
  - ✔ finds only rows that satisfy *query expression* exactly
  - ✔ the query is written on special *query language*

Example 1

```
mysql> CREATE TABLE articles (
    ->   id INT UNSIGNED NOT NULL PRIMARY KEY,
    ->   title VARCHAR(200),
    ->   body TEXT,
    ->   FULLTEXT (title,body)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> LOAD DATA INFILE 'test.txt' INTO TABLE articles;
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

Example 2

```
mysql> SELECT * FROM articles
    ->       WHERE MATCH (title,body) AGAINST ('database');
+----+-------------------+----------------------------------+
| id | title             | body                             |
+----+-------------------+----------------------------------+
|  5 | MySQL vs. YourSQL | In the following database compa...|
|  1 | MySQL Tutorial    | DBMS stands for DataBase. In th...|
+----+-------------------+----------------------------------+
2 rows in set (0.00 sec)
```

Example 3

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
    -> ('Security implications of running MySQL as root')
    -> AS score
    -> FROM articles WHERE MATCH (title,body) AGAINST
    -> ('Security implications of running MySQL as root');
+----+-------------------------------------+-----------------+
| id | body                                | score           |
+----+-------------------------------------+-----------------+
|  4 | 1. Never run mysqld as root. 2. ... | 1.5055546709332 |
|  6 | When configured properly, MySQL ... |   1.31140957288 |
+----+-------------------------------------+-----------------+
2 rows in set (0.00 sec)
```

Example 4

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
    -> AGAINST ('MySQL');
0 rows in set (0.00 sec)

mysql> SELECT * FROM articles WHERE MATCH (title,body)
    ->     AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+----+-------------------------+------------------------------+
| id | title                   | body                         |
+----+-------------------------+------------------------------+
|  1 | MySQL Tutorial          | DBMS stands for DataBase M...|
|  2 | How To Use MySQL Effi...| After you went through a l...|
|  3 | Optimising MySQL        | In this tutorial we will s...|
|  4 | 1001 MySQL Tricks       | 1. Never run mysqld as root. |
|  6 | MySQL Security          | When configured properly M...|
+----+-------------------------+------------------------------+
5 rows in set (0.00 sec)
```

Boolean Fulltext Search

- Search is performed for *words*
- Query specifies what words *must* be present in every found row and what words *must not*
- Complex expressions can be created with boolean operators and parentheses
- No matter how complex search expression is, each row either satisfies it – and does it absolutely, result of comparison is TRUE – or does not satisfy it – and again, completely, value is FALSE
- For each row it is easy to say whether it matches given search expression or not
- Formulating a proper query can be very difficult

Operators in Boolean Fulltext Search

- apple banana
- +apple +juice
- +apple -macintosh
- +apple +(pie strudel)
- apple*
- "Big Apple"
- +apple baked
- +apple ~baked
- +apple +(>pie <strudel)

- ✓ apple banana ↔ apple OR banana
- ✓ +apple +juice ↔ apple AND juice
- ✓ +apple baked ↔ ???

Relevance Ranking

- There is no concept of relevance in boolean search
- “Extended boolean” search models provide relevance but break strict matching, they return partially matched rows
- In MySQL, boolean search is strictly boolean, yet some “relevance” value is available
- The logic is (in and/or syntax for simplicity):

✓ A or B	weight(A)=weight(B)=1
✓ A and B and C	w(A)=w(B)=w(C)=⅓
✓ A or (B and (C or D))	w(A)=1, w(B)=w(C)=w(D)=½

- ✓ “relevance” of the row is the sum of weights of matched words, it cannot be less than 1

Natural Language Search

- Looks for documents that are *relevant* to the free-text natural human language query
  - ✓ e.g. “join algorithms in relational DBMS-es”
- There is no “absolute matching” here – document can be only partially relevant (and partially irrelevant)
- There is no way to define this “relevance” – different people will have different opinions on how relevant is one text to another
  - ✓ search engines are trained on sample document/query sets where relevance judgments were provided by human experts
- One should **never** assume that `MATCH ... AGAINST()` will return rows that contain query words
  - ✓ e.g. “Optimizing nested-loop joins in MySQL” is very relevant to the above query, but has no common words with it

Relevance Ranking

- Vector Space Model:
  - ✔ n-dimensional space, where n – is number of unique words in the collection
  - ✔ every document and every query is a vector in this space, with i^{th}-coordinate to be weight of the word i in this document
  - ✔ similarity between document and query is computed as a scalar product of their vectors, usually with a normalization factor
- There are many different theories proposing estimation of the word weight in a document
  - ✔ generally, it can be decomposed as: $w_i = w_{i,local} \cdot w_{i,global}$
  - ✔ In MySQL

$$w_{i,local} = w_{log} = \frac{\ln tf_i + 1}{\langle \ln tf_i + 1 \rangle} \qquad w_{i,global} = w_{prob} = \ln\left(\frac{N}{df_i} - 1\right) \qquad \alpha_{norm} = \frac{1}{1 + 0.0115\, n_{uniq}}$$

Query Expansion

- Also known as “automatic relevance feedback”
- Simple and proven technique for improving recall for short natural-language queries
- Natural language searches are known to work the best on relatively long queries
- Short queries carry too few semantic data to describe an implied (by the author) topic of search
- Query expansion works according to the following
  - ✓ perform the search
  - ✓ get the few most relevant document and add them to the original query
  - ✓ repeat the search using the new expanded query
- Available in MySQL 4.1.1

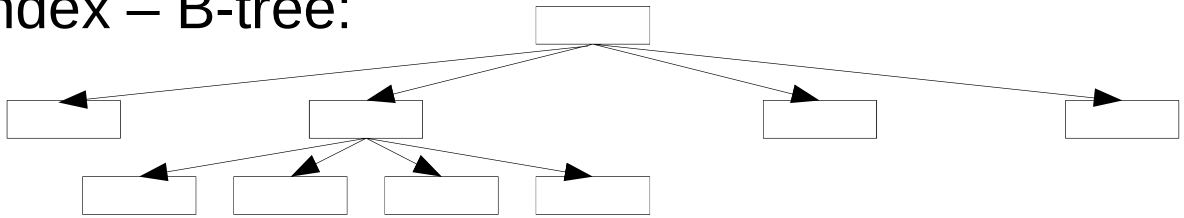
Search modes side-by-side

- Boolean mode
  - searches for **words**
  - uses query language
  - suitable for experienced users
  - provides only simplistic (and rather arbitrary) relevance ranking
  - can work without FULLTEXT index
  - search is performed incrementally
  - is faster

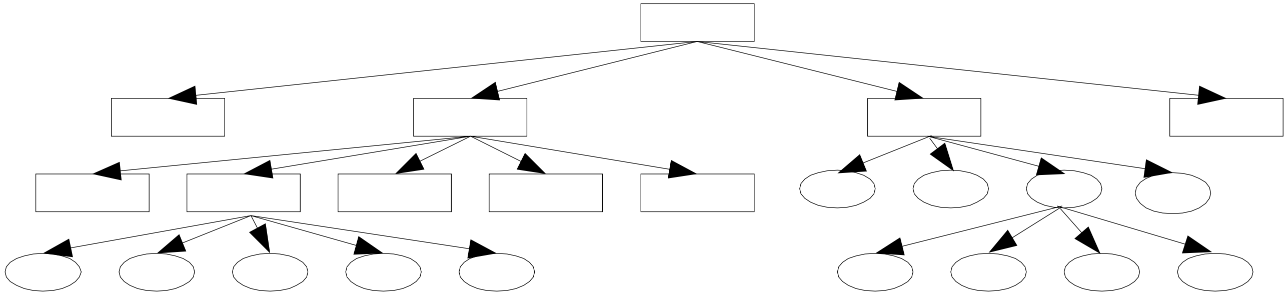
- Natural language mode
  - searches for **concepts**
  - free text queries
  - does not require computer literacy
  - computes complicated theoretically grounded relevance ranking
  - requires FULLTEXT index for collection-wide statistics
  - all the documents are found at once
  - is $O(\log N)$ slower

Fulltext Index Internals

- Key entry structure:
  - ✔ `{ word(varchar), weight(float), rowid }`
- MyISAM index – B-tree:



- Two level B-tree (MySQL 4.1):
  - ✔ `{ word(varchar), count(long) }`
  - ✔ `{ weight(float), rowid }`



Fulltext TODO

- Boolean Search:
  - ✓ benefit from the new index structure
  - ✓ proximity operators
- Natural Language Search:
  - ✓ benefit from the new index structure
  - ✓ new, more adequate weighting scheme
  - ✓ language-dependent stem and stopwords
- General features:
  - ✓ per-index settings
  - ✓ support for binary charsets (UCS-2)
  - ✓ support for MERGE tables
  - ✓ “always index” word list (for “C++”, “TCP/IP”, etc)
  - ✓ UDF (pre-)parser (indexing HTML, PDF, MS Word, etc.)

That's all !

- If you have any questions, you can ask them now
- Or you can ask me after the presentation
- Thanks for listening

☺

Tuning & Optimization

- Key_buffer_size

- ft_stopword_file
- ft_min_word_len
- ft_max_word_len

- ft_query_expansion_limit
- myisam_sort_buffer_size
- myisam_max_sort_file_size

- searching for ”search*ing”:
  - ✓
    ```
    MATCH col AGAINST ('search*' IN BOOLEAN MODE)
    AND col REGEXP
    '[[:<:]]search[[:alnum:]]+ing[[:>:]]'
    ```

Benefits of new structure

- Natural Language Search:
  - ✓ unsafe optimization: skip words with $w_{i,global} < \varepsilon$
  - ✓ reduced noise
  - ✓ lesser number of rows returned
    - reduced I/O
    - reduced $\mathrm{O}(\log N)$ speed penalty
- Boolean Search:
  - ✓ in “+apple +avocado” queries, instead of intersecting long list of rows, containing word “apple” and the short list of rows with “avocado”, MySQL can look up rows from the short “avocado” list in the “apple” sub-B-tree, thus avoiding index scan