Full-text search in MySQL

As data grows more and more, the problem of finding accurate information becomes even more important. With complex and large volumes of data, the problem is how to find the right and the right information. People d & u

As data grows more and more, the problem of finding accurate information becomes even more important. With complex and large volumes of data, the problem is how to find the right and the right information. Users do not want to search for a word but have a million answers, they need accuracy and eliminate the words that cause interference. At that time, users will need full-text search feature.

Full text search (TKTV) has been supported in MySQL version 3.23.23. The indexed VARCHAR and TEXT columns with FULLTEXT can be used with special SQL statements to perform full-text search in MySQL. To version 4.1, this feature becomes complete with full boolean search support.

Full-text search in MySQL

TKTV is a function available in MySQL that allows users to search for pieces of information that match a string on one or more specific tables, rather than finding matching form "SELECT LIKE" on each row of a certain field. .

A full-text index in MySQL is an index of type FULLTEXT. FULLTEXT indexes are only used with MyISAM tables and can be created from CHAR, VARCHAR, or TEXT columns at table creation with CREATE TABLE or later with ALTER TABLE or CREATE INDEX.

TKTV index is very similar to other indexes: it is a list of keys that are ordered. These keys refer to the records in the data file. Each key consists of (format of version 4.1):

{

Word - VARCHAR. One word inside the text section.

Count - LONG. How many times that word appears in the text.

}

{

Weight - FLOAT. Evaluate the importance of words.

Rowid - a pointer to a specific row in the data file.

}

Some key features of TKTV feature in MySQL:

• Remove words with less than 4 letters.

• Words with a dash between them are considered 2 words.

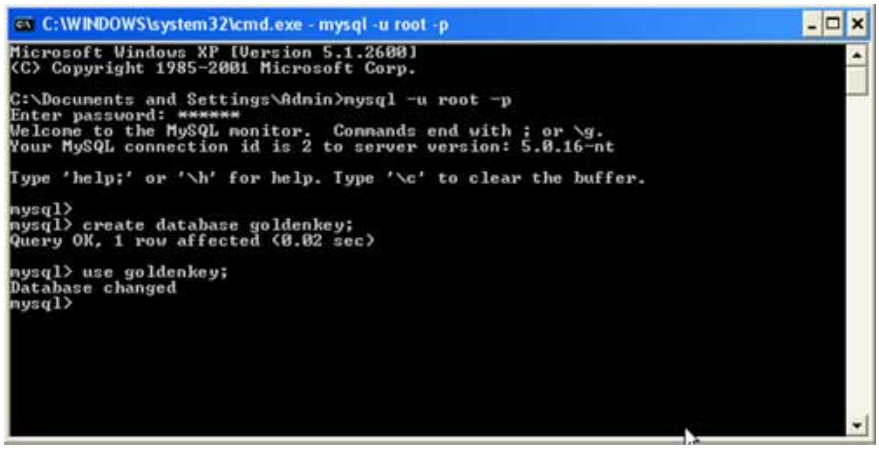
• Items are returned in the proper order, from high to low

• The words in the list of common words in English are also removed from the list of search results. This word list is in the file myisam / ft_static.c. When you need to filter common words for another language, for example Vietnamese, you need to edit this file, recompile MySQL, and rebuild the index!

Prepare data

We try to implement a simple example to better visualize the mechanism of operation of TKTV.

First, we create a goldenkey database from the command line (console) of MySQL:



Next, we create a table using MySQL's FULLTEXT clause to specify the fields we want to index the search for:

create Staff table

(

pk_id int auto_increment not null,

firstName varchar (20),

lastName varchar (20),

age int,

details text,

primary key (pk_id),

unique id (pk_id),

fulltext (firstName, lastName, details)

) ENGINE = MyISAM;

After executing this statement, you get a table with the following structure:

The first pk_id field is used as the primary key. We used the FULLTEXT clause to index the 3 fields group first first, lastName and details.

If you have created the table above and want to change the indexed field, use the following command:

ALTER TABLE Staff ADD FULLTEXT (field1, field2);

Here, you notice the *fulltext* line *(firstName, lastName, details)* . This line tells MySQL to set an index on the firstName, lastName, and details fields of the Staff table. Indexes can only be created on fields that are VARCHAR and TEXT. Once these fields have an index, the database is ready to exploit the TKTV feature to find the appropriate search records based on the values ??contained in these three fields.

To test, we add data to the new table created with the following statements:

insert into Staff values ??(0, Jeff, Holmes, 52, Mr. Jeff Holmes is a senior teacher in Golden Key. He likes Business, Technology and Finance. He is responsible for English for Information Technology course in Golden Key.);

insert into Staff values ??(0, Beth, Adams, 29, Mrs. Beth Adams is the Director of Studies of Golden Key Language Center She was born in England. She is very nice and professional.);

insert into Staff values ??(0, Jason, Bell, 33, Mr. Jason Bell is a business assistant in Golden Key. He graduated from London Business Management School. His major is Law in Business.);

Note that TKTV is designed for large data tables, when the larger the data, the more reliable the result returned.

Perform a search

Using TKTV command:

select firstName from Staff where match (firstName, lastName, details) against (business);

The results returned are as follows:

We will analyze to see the difference. First, consider the SELECT and FROM sections in the query:

select firstName from Staff

There is no difference here from other normal SELECT statements. But the difference is in the WHERE clause followed by:

where match (firstName, lastName, details) against (business);

This is the place to promote the power of TKTV. In the first part of this query, use the MATCH statement. This command will match the search request with the values ??of the fields firstName, lastName and details.

When the MATCH statement is used in the SELECT clause, it returns an order sorted by relevance, determined by a positive decimal. The closer this number is to 0, the less appropriate the record. This appropriate value is determined based on the search expression, the number of words included in the indexed fields as well as the total number of records searched.

The AGAINST statement accepts only one parameter. That's the string we need to find.

However, until now we have not seen the difference with traditional search statements based on LIKE statements:

select firstName from Staff where details like business;

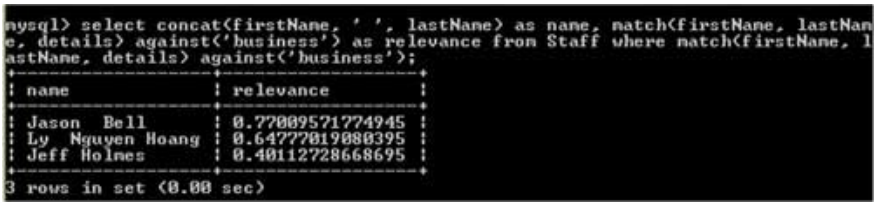
Please execute the above command to see if they return the same result set? The answer may be: yes and no. It is the intervention of the order arranged according to the relevance that makes this result set different from the result set from TKTV.

Sort by relevance

To test the appropriate rating, they execute the following query statement:

select concat (firstName,, lastName) as name, match (firstName, lastName, details) against (business) as relevance from Staff Where match (firstName, lastName, details) against (business);

The results returned are as follows:



In this query, we use the MATCH command in the SELECT clause to return the appropriate rating index for each record. The above statement performs TKTV on the firstName, lastName, and details fields to match the "business" string:

where match (firstName, lastName, details) against (business);

This query statement returns two records containing the "business" string located in one of the firstName, lastName or details fields. Now we will review the records that contain the "business" string:

"Ms. Nguyen Hoang Ly is the Marketing and Business Development Manager in Golden Key Language Center. If you want to talk about business cooperation, please call her."

"Mr. Jeff Holmes is a senior teacher in Golden Key. He likes Business, Technology and Finance. He is responsible for English for Information Technology course in Golden Key."

"Mr. Jason Bell is a business assistant in Golden Key. He graduated from London Business Management School. His major is Law in Business."

The relevance field returns from our query generated from the following command string:

match (firstName, lastName, details) against (business) as relevance

You can see that this command string has been used twice: once in the SELECT clause and once in the WHERE clause. MySQL chooses this solution and only performs TKTV once on the table, not twice. That means it will not "waste" the processing of queries like this. Obviously this is a great advantage to save resources and time compared to simple queries performed on a large database.

When the MATCH statement is used in the WHERE clause, MySQL automatically sorts rows from the highest to the lowest level. This is a query that returns the appropriate ranking index for all records:

select concat (firstName,, lastName) as name, match (firstName, lastName, details) against (business) as relevance from Staff;

We have removed the WHERE clause, so the returned records are not in order.

```
mysql> select concat(firstName, ' ', lastName) as name, match(firstName, lastNam
e, details) against('business') as relevance from Staff;
+-----------------+--------------------+
| name            | relevance          |
+-----------------+--------------------+
| Hai Do Song     |                  0 |
| Beth Adams      |                  0 |
| Ly  Nguyen Hoang | 0.64777019080395 |
| Hoai Cao Thu    |                  0 |
| Jeff Holmes     | 0.40112728668695 |
| Dubon Picora    |                  0 |
| Nga Doan Thanh  |                  0 |
| Jason  Bell     | 0.77009571774945 |
+-----------------+--------------------+
8 rows in set (0.00 sec)
```

At the beginning, when I listed the strengths of TKTV, I pointed out that MySQL removes jamming words and words with less than 4 characters. We will test this strength with the following two TKTV queries:

select firstName, match (firstName, lastName, details) against (he likes business) as relevance from Staff;

The result of this query is as follows:

```
mysql> select firstName, match(firstName, lastName, details) against('he likes b
usiness') as relevance from Staff;
+-----------+--------------------+
| firstName | relevance          |
+-----------+--------------------+
| Hai       |                  0 |
| Beth      |                  0 |
| Ly        | 0.64777019080395   |
| Hoai      |                  0 |
| Jeff      |  1.2638157384242   |
| Dubon     | 0.85746740940217   |
| Nga       |                  0 |
| Jason     | 0.77009571774945   |
+-----------+--------------------+
8 rows in set (0.00 sec)
```

Notice that our last query has two words that are longer than 3 characters, "business" and "likes". If we remove the word with less than 3 characters then we will get the result of the appropriate index similar to above:

select firstName, match (firstName, lastName, details) against (likes business) as relevance from Staff;

The results returned are as follows:

```
mysql> select firstName, match(firstName, lastName, details) against('likes busi
ness') as relevance from Staff;
+-----------+--------------------+
| firstName | relevance          |
+-----------+--------------------+
| Hai       |                  0 |
| Beth      |                  0 |
| Ly        | 0.64777019080395   |
| Hoai      |                  0 |
| Jeff      |  1.2638157384242   |
| Dubon     | 0.85746740940217   |
| Nga       |                  0 |
| Jason     | 0.77009571774945   |
+-----------+--------------------+
8 rows in set (0.00 sec)
```

As such, we still have the same return results. This proves that MySQL has removed the jamming words and the words with 3 characters or less from the query.

The TKTV function of MySQL ranks words based on their semantic value - common words rank lower than

uncommon words. This makes sense, because a word that exists in multiple records will have less relevance than a word that only appears in one or two records. The appropriate ranking according to meaning is used in most popular TKTV algorithms.

50% limit

MySQL removes jamming and short words but if a word is not short but is present in more than 50% of the records searched, those records will not be returned. MySQL calls it "50% threshold". To some extent this makes sense because it filters out all low-indexed records.

Full-text search with boolean operators

By combining multiple operators inside the search string, you can include or exclude other words, change word combinations to change the appropriate values ??. Here are some common boolean operators used in MySQL:

• **+** The plus sign at the beginning indicates that the word must appear in all returned rows.

• **-** The minus sign at the beginning indicates that the word cannot be present in all returned items.

• The default (when there is no minus sign or plus sign) the search word is optional, but the row containing the word will be appreciated more.

• **>** These two operators are used to change the word contribution to the appropriate value of a row. The operator increases the contribution.

• **()** Parentheses are used to group words into a child expression.

• **~** The tilde is at the beginning with a negative operator function, making the word contribution into the appropriate value of the negative row. This symbol is useful when evaluating interfering words. A row containing such a word will be underestimated, but does not mean that it is excluded, as is the case with the - operator.

• ***** An asterisk is a removal operator. Unlike other operators, it is connected to the word rather than being placed before the word.

• **"** *The clause is enclosed in quotation marks* **"** , will only match rows that contain that clause.

A boolean search is done quite similar to normal TKTV. However, it contains the keyword IN BOOLEAN MODE, as the example below:

select * from Staff where match (firstName, lastName, details) against (+ business -English in boolean mode);

In the above query, we ordered that all returned rows must contain the word "business" and should not contain the word "English". Similarly, we try to execute the query below:

select firstName, match (firstName, lastName, details) against (+ English -business> manager in boolean mode) as relevance from Staff where match (firstName, lastName, details) against (+ English - business> manager in boolean mode);

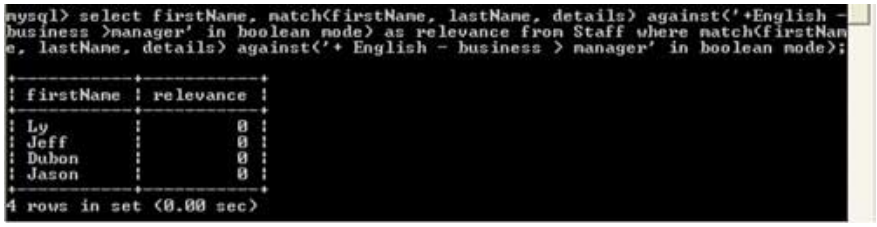
Looking at the above query, we see that the returned item will be placed in a higher position if:

• It contains the word "English"

• It does not contain the word "business"

• The word "manager" will appear in that row one or more times

The following results:

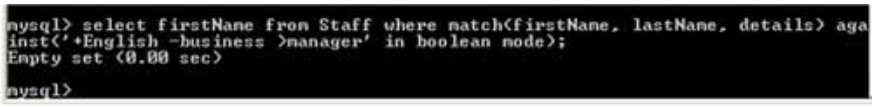


All returned records have an appropriate rating value of 0 and therefore will not be present in the returned results.

You can verify this result by executing:

select firstName from Staff where match (firstName, lastName, details) against (+ English -business> manager in boolean mode);

You will see:



Conclude

TKTV allows you to search smart, fast and minimize the need to write complex search queries. With TKTV, you will save resources, processing time and help increase the relevance of search results with databases with a large amount of content, bringing satisfaction to users.

Pham Cong Dinh
Language Center Golden Key

References:
1. Full text search in MySQL , http://dev.mysql.com/doc/mysql/en/fulltext-search.html
2. Golden Key Research Materials , http://www.goldenkey.edu.vn

guides. Thank you for reading and for following us regularly.