# Лабораторная работа 16. Индексы MySQL

Цель работы: получение навыков создания многоколоночных и комбинированных индексов MySQL.

Ссылка: http://linux.yaroslavl.ru/docs/www/mysql/doc/CREATE\_INDEX.html

Синтаксис оператора CREATE INDEX

CREATE [UNIQUE|FULLTEXT] INDEX index\_name

ON tbl\_name (col\_name[(length)],... )

Команда CREATE INDEX соответствует команде ALTER TABLE в части создания индексов.

Обычно все индексы создаются в таблице во время создания самой таблицы командой CREATE TABLE. CREATE INDEX дает возможность добавить индексы к существующим таблицам.

Список столбцов в форме (col1,col2,...) создает индекс для нескольких столбцов. Величины индексов формируются путем конкатенации величин указанных столбцов.

Для столбцов типов CHAR и VARCHAR с помощью параметра col\_name(length) могут создаваться индексы, для которых используется только часть столбца (для столбцов BLOB и TEXT нужно указывать длину). Команда, приведенная ниже, создает индекс, используя первые 10 символов столбца name:

CREATE INDEX part\_of\_name ON customer (name(10));

Поскольку большинство имен обычно имеют отличия друг от друга в первых 10 символах, данный индекс не должен быть намного медленнее, чем созданный из столбца name целиком. Кроме того, используя неполные столбцы для индексов, можно сделать файл индексов намного меньше, а это позволяет сэкономить место на диске и к тому же повысить скорость операций INSERT.

Следует учитывать, что для таблиц типа MyISAM можно добавлять индекс только для столбцов, которые могут принимать величины NULL или для столбцов BLOB/TEXT.

С помощью опции FULLTEXT можно индексировать только столбцы VARCHAR и TEXT и только в таблицах MyISAM.

В таблицах InnoDB осуществляется проверка ограничений целостности внешних ключей.

Фактически для соединения двух таблиц внешние ключи не нужны.

Единственное, что MySQL в настоящее время не осуществляет (в типах таблиц, отличных от InnoDB), это проверку (CHECK) что используемые ключи действительно существуют в таблице(ах), на которые выполняется ссылка, и не удаляет автоматически записи из таблиц с определением внешних ключей. Если же ключи используются обычным образом, все будет работать.

**Задание 1.**

Выполните следующее:

CREATE TABLE person (

id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

name CHAR(60) NOT NULL,

PRIMARY KEY (id)

);

CREATE TABLE shirt (

id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

style ENUM('t-shirt', 'polo', 'dress') NOT NULL,

color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,

owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),

PRIMARY KEY (id)

);

INSERT INTO person VALUES (NULL, 'Antonio Paz');

INSERT INTO shirt VALUES

(NULL, 'polo', 'blue', LAST\_INSERT\_ID()),

(NULL, 'dress', 'white', LAST\_INSERT\_ID()),

(NULL, 't-shirt', 'blue', LAST\_INSERT\_ID());

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

INSERT INTO shirt VALUES

(NULL, 'dress', 'orange', LAST\_INSERT\_ID()),

(NULL, 'polo', 'red', LAST\_INSERT\_ID()),

(NULL, 'dress', 'blue', LAST\_INSERT\_ID()),

(NULL, 't-shirt', 'white', LAST\_INSERT\_ID());

SELECT \* FROM person;

+----+---------------------+

| id | name |

+----+---------------------+

| 1 | Antonio Paz |

| 2 | Lilliana Angelovska |

+----+---------------------+

SELECT \* FROM shirt;

+----+---------+--------+-------+

| id | style | color | owner |

+----+---------+--------+-------+

| 1 | polo | blue | 1 |

| 2 | dress | white | 1 |

| 3 | t-shirt | blue | 1 |

| 4 | dress | orange | 2 |

| 5 | polo | red | 2 |

| 6 | dress | blue | 2 |

| 7 | t-shirt | white | 2 |

+----+---------+--------+-------+

SELECT s.\* FROM person p, shirt s

WHERE p.name LIKE 'Lilliana%'

AND s.owner = p.id

AND s.color <> 'white';

+----+-------+--------+-------+

| id | style | color | owner |

+----+-------+--------+-------+

| 4 | dress | orange | 2 |

| 5 | polo | red | 2 |

| 6 | dress | blue | 2 |

+----+-------+--------+-------+

Поиск по двум ключам

MySQL пока не осуществляет оптимизации, если поиск производится по двум различным ключам, которые связаны при помощи оператора OR (поиск по одному ключу с различными частями OR оптимизируется хорошо).

**Задание 2.**

Выполните следующее:

SELECT field1\_index, field2\_index FROM test\_table WHERE field1\_index = '1'

OR field2\_index = '1'

В настоящее время данную проблему очень эффективно можно решить при помощи временной таблицы. Этот способ оптимизации также хорошо подходит при запуске очень сложных запросов, когда SQL-сервер делает оптимизацию в неправильном порядке.

**Задание 3.**

Выполните следующее:

CREATE TEMPORARY TABLE tmp

SELECT field1\_index, field2\_index FROM test\_table WHERE field1\_index = '1';

INSERT INTO tmp

SELECT field1\_index, field2\_index FROM test\_table WHERE field2\_index = '1';

SELECT \* from tmp;

DROP TABLE tmp;

Вышеупомянутый способ выполнения этого запроса - это фактически UNION (объединение) двух запросов.

Полнотекстовый поиск в MySQL

Полнотекстовые индексы в MySQL обозначаются как индексы типа FULLTEXT. Эти индексы могут быть созданы в столбцах VARCHAR и TEXT во время создания таблицы командой CREATE TABLE или добавлены позже с помощью команд ALTER TABLE или CREATE INDEX. Загрузка больших массивов данных в таблицу будет происходить намного быстрее, если таблица не содержит индекс FULLTEXT, который затем создается командой ALTER TABLE (или CREATE INDEX). Загрузка данных в таблицу, уже имеющую индекс FULLTEXT, будет более медленной.

Полнотекстовый поиск выполняется с помощью функции MATCH().

**Задание 4.**

Выполните следующее, добавив номер своего варианта к началам строковых констант:

CREATE TABLE articles (

id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,

title VARCHAR(200),

body TEXT,

FULLTEXT (title,body)

);

Query OK, 0 rows affected (0.00 sec)

INSERT INTO articles VALUES

(0,'MySQL Tutorial', 'DBMS stands for DataBase ...'),

(0,'How To Use MySQL Efficiently', 'After you went through a ...'),

(0,'Optimising MySQL','In this tutorial we will show ...'),

(0,'1001 MySQL Trick','1. Never run mysqld as root. 2. ...'),

(0,'MySQL vs. YourSQL', 'In the following database comparison ...'),

(0,'MySQL Security', 'When configured properly, MySQL ...');

Query OK, 6 rows affected (0.00 sec)

Records: 6 Duplicates: 0 Warnings: 0

SELECT \* FROM articles

WHERE MATCH (title,body) AGAINST ('database');

+----+-------------------+------------------------------------------+

| id | title | body |

+----+-------------------+------------------------------------------+

| 5 | MySQL vs. YourSQL | In the following database comparison ... |

| 1 | MySQL Tutorial | DBMS stands for DataBase ... |

+----+-------------------+------------------------------------------+

2 rows in set (0.00 sec)

Функция MATCH() выполняет поиск в естественном языке, сравнивая строку с содержимым текста (совокупность одного или более столбцов, включенных в индекс FULLTEXT). Строка поиска задается как аргумент в выражении AGAINST(). Поиск выполняется без учета регистра символов. Для каждой строки столбца в заданной таблице команда MATCH() возвращает величину релевантности, т.е. степень сходства между строкой поиска и текстом, содержащимся в данной строке указанного в списке оператора MATCH() столбца.

Когда команда MATCH() используется в выражении WHERE (см. пример выше), возвращенные строки столбцов автоматически сортируются, начиная с наиболее релевантных. Величина релевантности представляет собой неотрицательное число с плавающей точкой. Релевантность вычисляется на основе количества слов в данной строке столбца, количества уникальных слов в этой строке, общего количества слов в тексте и числа документов (строк), содержащих отдельное слово.

Поиск возможен также в логическом режиме, это объясняется ниже в данном разделе.

Предыдущий пример представляет собой общую иллюстрацию использования функции MATCH(). Строки возвращаются в порядке уменьшения релевантности.

В следующем примере показано, как извлекать величины релевантности в явном виде. В случае отсутствия выражений WHERE и ORDER BY возвращаемые строки не упорядочиваются.

**Задание 5.**

Выполните следующее:

SELECT id,MATCH (title,body) AGAINST ('Tutorial') FROM articles;

+----+-----------------------------------------+

| id | MATCH (title,body) AGAINST ('Tutorial') |

+----+-----------------------------------------+

| 1 | 0.64840710366884 |

| 2 | 0 |

| 3 | 0.66266459031789 |

| 4 | 0 |

| 5 | 0 |

| 6 | 0 |

+----+-----------------------------------------+

6 rows in set (0.00 sec)

Следующий пример - более сложный. Запрос возвращает значение релевантности и, кроме того, сортирует строки в порядке убывания релевантности. Чтобы получить такой результат, необходимо указать MATCH() дважды. Это не приведет к дополнительным издержкам, так как оптимизатор MySQL учтет, что эти два вызова MATCH() идентичны, и запустит код полнотекстового поиска только однажды. **Выполните** следующее:

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)

Для разбивки текста на слова MySQL использует очень простой синтаксический анализатор. «Словом» является любая последовательность символов, состоящая из букв, чисел, знаков `'' и `\_'. Любое «слово», присутствующее в стоп-списке (stopword) или просто слишком короткое (3 символа или меньше), игнорируется.

Каждое правильное слово в наборе проверяемых текстов и в данном запросе оценивается в соответствии с его важностью в этом запросе или наборе текстов. Таким образом, слово, присутствующее во многих документах, будет иметь меньший вес (и даже, возможно, нулевой), как имеющее более низкое смысловое значение в данном конкретном наборе текстов. С другой стороны, редко встречающееся слово получит более высокий вес. Затем полученные значения весов слов объединяются для вычисления релевантности данной строки столбца.

Описанная техника подсчета лучше всего работает для больших наборов текстов (фактически она именно для этого тщательно настраивалась). Для очень малых таблиц распределение слов не отражает адекватно их смысловое значение, и данная модель иногда может выдавать некорректные результаты.

SELECT \* FROM articles WHERE MATCH (title,body) AGAINST ('MySQL');

Empty set (0.00 sec)

Поиск по слову «MySQL» в предыдущем примере не приводит к каким-либо результатам, так как это слово присутствует более чем в половине строк. По существу, данное слово целесообразно трактовать как стоп-слово (т.е. слово с нулевой смысловой ценностью). Это наиболее приемлемое решение - запрос на естественном языке не должен возвращать каждую вторую строку из таблицы размером 1Гб.

Маловероятно, что слово, встречающееся в половине строк таблицы, определяет местонахождение релевантных документов. На самом деле, наиболее вероятно, что будет найдено много не относящихся к делу документов. Общеизвестно, что такое случается слишком часто при попытке найти что-либо в Интернет с помощью поисковых машин. Именно на этом основании подобным строкам должно быть назначено низкое смысловое значение в данном конкретном наборе данных.

В MySQL возможен полнотекстовый поиск также и в логическом режиме с использованием модификатора IN BOOLEAN MODE.

**Задание 6.**

Выполните следующее:

SELECT \* FROM articles WHERE MATCH (title,body)

AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);

+----+------------------------------+-------------------------------------+

| id | title | body |

+----+------------------------------+-------------------------------------+

| 1 | MySQL Tutorial | DBMS stands for DataBase ... |

| 2 | How To Use MySQL Efficiently | After you went through a ... |

| 3 | Optimising MySQL | In this tutorial we will show ... |

| 4 | 1001 MySQL Trick | 1. Never run mysqld as root. 2. ... |

| 6 | MySQL Security | When configured properly, MySQL ... |

+----+------------------------------+-------------------------------------+

Данный запрос вывел все строки, содержащие слово «MySQL» (заметьте, 50-процентная пороговая величина здесь не используется), но эти строки не содержат слова «YourSQL». Следует отметить, что логический режим поиска не сортирует автоматически строки в порядке уменьшения релевантности. Это видно по результату предыдущего запроса, где строка с наиболее высокой релевантностью (содержащая слово «MySQL» дважды) помещена последней, а не первой. Логический режим полнотекстового поиска может работать даже без индекса FULLTEXT, хотя и очень медленно.

В логическом режиме полнотекстового поиска поддерживаются следующие операторы:

+

Предшествующий слову знак «плюс» показывает, что это слово должно присутствовать в каждой возвращенной строке.

-

Предшествующий слову знак «минус» означает, что это слово не должно присутствовать в какой-либо возвращенной строке.

По умолчанию (если ни плюс, ни минус не указаны) данное слово является не обязательным, но содержащие его строки будут оцениваться более высоко. Это имитирует поведение команды MATCH() ... AGAINST() без модификатора IN BOOLEAN MODE.

< >

Эти два оператора используются для того, чтобы изменить вклад слова в величину релевантности, которое приписывается строке. Оператор < уменьшает этот вклад, а оператор > - увеличивает его. См. пример ниже.

( )

Круглые скобки группируют слова в подвыражения.

~

Предшествующий слову знак «тильда» воздействует как оператор отрицания, обуславливая негативный вклад данного слова в релевантность строки. Им отмечают нежелательные слова. Строка, содержащая такое слово, будет оценена ниже других, но не будет исключена совершенно, как в случае оператора – «минус».

\*

Звездочка является оператором усечения. В отличие от остальных операторов, она должна добавляться в конце слова, а не в начале.

"

Фраза, заключенная в двойные кавычки, соответствует только строкам, содержащим эту фразу, написанную буквально.

Ниже приведен ряд примеров:

apple banana

находит строки, содержащие по меньшей мере одно из этих слов.

+apple +juice

... оба слова.

+apple macintosh

... слово «apple», но ранг строки выше, если она также содержит слово ``macintosh''.

+apple -macintosh

... слово «apple», но не «macintosh».

+apple +(>pie <strudel)

... «apple» и «pie», или «apple» и «strudel» (в любом порядке), но ранг «apple pie» выше, чем «apple strudel».

apple\*

... «apple», «apples», «applesauce», и «applet».

"some words"

... «some words of wisdom», но не «some noise words».

Ограничения для полнотекстового поиска

Все параметры функции MATCH() должны быть столбцами одной и той же таблицы, т.е. частью одного и того же индекса FULLTEXT, за исключением работы MATCH() в режиме IN BOOLEAN MODE.

Список столбцов в команде MATCH() должен точно соответствовать списку столбцов в определении индекса FULLTEXT для этой таблицы, за исключением работы данной функции MATCH() в режиме IN BOOLEAN MODE.

Аргумент в выражении AGAINST() должен быть неизменяемой строкой.

**Контрольные вопросы:**

1. Синтаксис оператора CREATE INDEX.

2. Операторы, поддерживаемые в логическом режиме полнотекстового поиска.

3. Полнотекстовый поиск в MySQL.