**ЛАБОРАТОРНАЯ РАБОТА 1.**

**ИНДЕКСЫ**

*Цель работы: получение навыков использования языка SQL для создания индексов.*

Основная задача индекса – обеспечение быстрого доступа к данным по некоторому коду.

Реляционная таблица может одновременно обладать несколькими индексами:

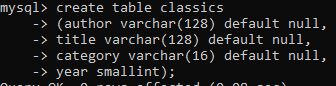
* индексом первичного ключа;
* несколькими индексами, обслуживающими внешние ключи;
* произвольным числом пользовательских индексов.

Индекс первичного ключа создается автоматически в момент назначения первичного ключа таблицы. Перед ним стоит одна-единственная задача – первичный индекс должен гарантировать, что во входящих в состав первичного ключа столбцах будут храниться уникальные данные.

Индексы внешнего ключа также создаются автоматически в момент назначения внешнего ключа. Основная задача индекса внешнего ключа – быстрый поиск записей во внешних таблицах для соединения с этими таблицами.

Пользовательские индексы создаются руками программиста. Они описывают дополнительные способы упорядочивания строк таблицы, ускоряют поиск данных и могут проверять данные на уникальность.

1. Создайте базу данных publications. Создайте в ней таблицу classics (author, title, category, year). Первичный ключ не задан. Внесите 5 строк данных.



В данный момент у нас есть таблица classics, в которой можно будет без труда, пользуясь средствами MySQL, отыскать нужную информацию. Но все так просто лишь до тех пор, пока она не разрастется до пары сотен строк. Тогда с каждой добавленной строкой доступ к базе данных будет становиться все медленнее и медленнее, поскольку MySQL при обработке запроса придется вести поиск в каждой строке. Это похоже на поиск нужной информации в каждой книге, имеющейся в библиотеке.

Разумеется, вам не придется вести поиск в библиотеках подобным образом, поскольку в них есть либо обычная картотека, либо, что более вероятно, собственная база данных. То же самое относится и к MySQL, поскольку ценой небольших затрат оперативной памяти и дискового пространства можно создать «картотеку» для таблицы, которая будет использоваться MySQL для выполнения мгновенного поиска.

***Создание индекса***

Возможности быстрого поиска можно добиться путем добавления индекса либо при создании таблицы, либо в любое время впоследствии. Существуют различные типы индексов, такие как INDEX, PRIMARY KEY или FULLTEXT. Кроме того, необходимо решить, каким столбцам нужен индекс, для этого нужно спрогнозировать, по каким данным каждого из столбцов будет осуществляться поиск. Индексы можно также усложнять, комбинируя в одном индексе данные из нескольких столбцов. И даже когда вы все это поймете, у вас будет возможность сократить размер индекса за счет ограничения объема данных каждого индексируемого столбца.

Если представить себе поисковые операции применительно к таблице classics, становится ясно, что поиск может осуществляться во всех столбцах. Но если бы был столбец pages, то он, наверное, не понадобился бы для индекса, поскольку большинство людей вряд ли стали бы искать книги по количеству страниц.

1. Добавьте индекс к каждому столбцу, воспользовавшись командами, приведенными в примере.

*ALTER TABLE classics ADD INDEX(author(20));*

*ALTER TABLE classics ADD INDEX(title(20));*

*ALTER TABLE classics ADD INDEX(category(4));*

*ALTER TABLE classics ADD INDEX(year);*

Первые две команды создают индексы для столбцов авторов и названий — author и title, ограничивая каждый индекс только первыми 20 символами. Например, когда MySQL индексирует название:

*The Adventures of Tom Sawyer*

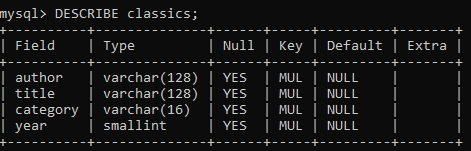
на самом деле в индексе будут сохранены только первые 20 символов:

*The Adventures of То*

Это делается для сокращения размера индекса и для оптимизации скорости доступа к базе данных. Мы выбрали 20 символов, поскольку их должно быть достаточно для обеспечения уникальности большинства строк, встречающихся в данных столбцах. Если MySQL обнаружит два индекса с одинаковым содержимым, ей нужно будет понапрасну потратить время на обращение к самой таблице и на проверку проиндексированного столбца, для того чтобы определить, какая именно строка действительно соответствует условиям поиска.

Что касается столбца категории — category, то на данный момент, чтобы идентифицировать уникальность строки, достаточно четырех символов, что дае возможность вводить такие категории, у которых первые три символа могут быть одинаковыми. Если позже набор категорий усложнится еще больше, этот столбец можно будет переиндексировать. И наконец, мы не стали задавать ограничения на индекс столбца года издания — year, поскольку он имеет четко определенную длину в четыре символа.

Результат ввода этих команд можно увидеть через команду describe, которая показывает наличие ключа MUL для каждого столбца. Это означает, что в этом столбце может многократно присутствовать одно и то же значение, что, собственно, нам и нужно, поскольку имена авторов могут встречаться многократно, одни и те же названия книг могут использоваться множеством авторов и т. д.



***Использование команды CREATE INDEX***

Индекс можно добавить не только командой ALTER TABLE, но и командой CREATE INDEX. Эти две команды являются равнозначными, за исключением того, что CREATE INDEX не может использоваться для создания индекса типа первичного ключа – PRIMARY KEY.

С полным синтаксисом инструкции create index можно ознакомиться на официальном сайте MySQL.



Упрощенный формат этой команды показан во второй строке примера. Обе команды в данном случае являются синонимами.

*ALTER TABLE classics ADD INDEX(author(20));*

*CREATE INDEX author ON classics (author(20));*

***Добавление индексов при создании таблиц***

Чтобы добавить индекс, не нужно выжидать какое-то время после создания таблицы. Это может отнять много времени, поскольку добавление индекса к большой таблице — длительный процесс. Поэтому рассмотрим команду, создающую таблицу classics с уже имеющимися индексами.

*CREATE TABLE classics\_2 ( author VARCHAR(128), title VARCHAR(128), category VARCHAR(16), year SMALLINT,*

*INDEX(author(20)),*

*INDEX(title(20)),*

*INDEX(category(4)),*

*INDEX(year));*

1. Выполните создание таблицы с индексами по приведенному примеру.

***Первичные ключи***

В данный момент у нас создана таблица classics и за счет добавления индексов обеспечен быстрый поиск, но кое-что все же упущено. Можно вести поиск по всем имеющимся в таблице изданиям, но нет единого уникального ключа для каждого издания, обеспечивающего мгновенный доступ к строке. Важность наличия ключа с уникальным значением для каждой строки проявится, когда мы станем комбинировать данные из разных таблиц.

Пусть первичным ключом будет признанный во всем мире номер ISBN. Продолжим работу с таблицей и создадим новый столбец для этого ключа. Теперь, помня о том, что номер ISBN состоит из 13 символов, можно решить, что с задачей справится следующая команда:

*ALTER TABLE classics ADD isbn CHAR(13) PRIMARY KEY;*

Но это не так. Если запустить эту команду на выполнение, будет получено сообщение об ошибке, связанной с дубликатом записи для ключа: «Duplicate entry». Причина в том, что таблица уже заполнена данными, а эта команда пытается добавить столбец со значением NULL к каждой строке, что запрещено, поскольку все столбцы, использующие первичный ключ, должны иметь уникальное значение.

Но если бы таблица была пуста, то эта команда была бы выполнена без проблем, как и при добавлении первичного ключа сразу же после создания таблицы.

В сложившейся ситуации нужно немного схитрить: создать новый столбец без индекса заполнить его данными, а затем добавить индекс ретроспективно, воспользоваться командой из ранее приведенного примера с alter table … .

1. Добавьте в таблицу столбец ISBN и заполните данными. После задайте его в качестве первичного ключа.

*UPDATE classics SET isbn='9781598184891' WHERE year='1999';*

*UPDATE classics SET isbn='9780582506206’ WHERE year=’1811';*

*UPDATE classics SET isbn=’9780517123201’ WHERE year=’1856';*

*UPDATE classics SET isbn=’9780099533474’ WHERE year='1841';*

*UPDATE classics SET isbn=’9780192814968’ WHERE year=’1594’;*

*ALTER TABLE classics ADD PRIMARY KEY(isbn);*

*DESCRIBE classics;*

Обратите внимание на то, что в синтаксисе команды ALTER TABLE ключевое слово INDEX заменено ключевыми словами PRIMARY KEY.

1. Создайте таблицу classics\_3 с первичным ключом.

***Создание индекса FULLTEXT***

В отличие от обычного индекса, имеющийся в MySQL индекс FULLTEXT позволяет осуществлять сверхбыстрый поиск целых столбцов текста. Он сохраняет каждое слово каждой строки данных в специальном индексе, в котором можно вести поиск, используя «естественный язык» наподобие того, что применяется в поисковом механизме.

Вообще-то утверждение о том, что система MySQL хранит все слова в индексе FULLTEXT, не вполне соответствует действительности, поскольку в ней имеется встроенный список более чем из 500 слов, которые она предпочитает игнорировать в силу их широкой распространенности и практической бесполезности при любом поиске. Этот список, называемый стоповыми словами — stopwords, включает слова the, as, is, of и т. д. Список помогает MySQL работать при FULLTEXT-поиске намного быстрее и не раздувать размеры базы данных.

Рассмотрим особенности индексов FULLTEXT, о которых нужно знать.

* Индексы FULLTEXT могут создаваться только для столбцов с типами данных CHAR, VARCHAR и TEXT.
* Определение индекса FULLTEXT может быть дано в инструкции CREATE TABLE при создании таблицы или добавлено позже с использованием инструкции ALTER TABLE (или CREATE INDEX).
* Намного быстрее будет загрузить большие наборы данных в таблицу, не име ющую индекса FULLTEXT, а затем создать индекс, чем загружать их в таблицу, у которой уже имеется индекс FULLTEXT.

Чтобы создать индекс FULLTEXT, примените его к одной или нескольким записям как в примере, в котором индекс FULLTEXT добавляется к двум столбцам - author и title, принадлежащим таблице classics (этот индекс является дополнением к тем, что уже были созданы, и не влияет на их работу).

1. Добавьте индекс в таблицу classics по примеру.

*ALTER TABLE classics ADD FULLTEXT (author,title);*

Теперь в этой паре столбцов можно вести поиск с использованием индекса FULLTEXT. Такая возможность могла бы проявиться в полную силу, если бы вы могли теперь ввести весь текст этих книг в базу данных (учитывая, что они не защищены авторскими правами), и тогда они были бы полностью доступны для поиска. Поисковые операции с использованием индекса FULLTEXT рассмотрены далее в пункте «MATCH...AGAINST» подраздела «Создание запросов к базе данных MySQL» далее.

Если система MySQL станет при доступе к вашей базе данных работать медленнее, чем вы от нее ожидали, то проблема скорее всего заключается в ваших индексах. Либо у вас нет индекса там, где он нужен, либо индексы составлены неоптимальным образом. Зачастую данная проблема решается за счет тонкой настройки индексов таблиц.

***MATCH...AGAINST***

Конструкция MATCH … AGAINST может быть применена к столбцу, для которого был создан индекс FULLTEXT. Используя эту конструкцию, можно вести поиск, применяя в качестве критерия элементы обычного языка, как при работе с поисковыми механизмами Интернета. В отличие от конструкций WHERE … или WHERE ... LIKE, конструкция MATCH … AGAINST позволяет вводить в поисковый запрос несколько слов и проверять на их наличие все слова в столбцах, имеющих Индекс FULLTEXT. Индексы FULLTEXT нечувствительны к регистру букв, поэтому неважно, какой именно регистр используется в ваших запросах.

Предположим, что вы добавили индекс FULLTEXT к столбцам author и title и три запроса, показанные в примере. Первый из них требует вернуть любые строки, в которых содержится слово and. Поскольку and является стоповым сло вом, MySQL его проигнорирует и запрос всегда будет возвращать пустой набор независимо от того, что хранится в столбцах. Второй запрос требует вернуть любые строки, содержащие в любом месте и в любом порядке оба слова: curiosity и shop Третий запрос применяет тот же вид поиска для слов tom и sawyer.

1. Выполните поиск строк для classics по примеру.

*SELECT author,title FROM classics WHERE MATCH(author,title) AGAINST('and');*

*SELECT author,title FROM classics WHERE MATCH(author,title) AGAIST('curiosity shop');*

*SELECT author,title FROM classics WHERE MATCH(author,title) AGAINST('tom sawyer');*

***MATCH...AGAINST в булевом режиме***

При желании придать своим запросам с конструкцией MATCH... AGAINST более широкие возможности нужно воспользоваться булевым режимом. Это изменение выражается в том, что стандартный запрос по индексу FULLTEXT ведет поиск любой комбинации искомых слов, не требуя наличия всех этих слов в тексте, Наличие отдельного слова в столбце приводит к тому, что поисковая операция возвращает строку.

Булев резким позволяет также ставить впереди искомых слов знак + или - чтобы показать, что они должны быть включены или исключены. Если обычный булев режим требует «искать присутствие любого из этих слов», то знак «плюс» означает, что «это слово обязательно должно присутствовать, иначе строку возвращать не нужно». Знак «минус» означает, что «этого слова быть не должно, а если оно присутствует, то строку возвращать не нужно».

В примере показаны два запроса, использующие булев режим. Первый запрос требует вернуть все строки, в которых содержится слово Charles и нет слова species. Во втором запросе используются двойные кавычки, чтобы потребовать вернуть все строки, включающие в себя фразу origin of.

1. Выполните пример использования MATCH...AGAINST в булевом режиме

SELECT author,title FROM classics WHERE MATCH(author,title)

AGAINST('+charles -species' IN BOOLEAN MODE);

SELECT author,title FROM classics WHERE MATCH(author,title)

AGAINST('"origin of"' IN BOOLEAN MODE);

Как, наверное, и ожидалось, первый запрос вернет только запись о книге The Old Curiosity Shop Чарльза Диккенса. Запись о книге Чарльза Дарвина игнорируется, поскольку из результата должна быть исключена любая строка, содержащая слово species.

1. Создайте в БД proezdnye несколько пользовательских индексов. Выполните выборку с их использованием. Все действия и их результаты внесите в отчет.

**Контрольные вопросы**

1. Для чего нужен индекс в MySQL?
2. Какие преимущества дает индекс FULLTEXT?
3. Что такое стоповое слово?