УДК 004.02

СПОСОБЫ ОПТИМИЗАЦИИ БАЗ ДАННЫХ ПРИ РАБОТЕ С MYSQL

А.П. ГАЙДЕЛЬ
(Представлено: Д.В. ПЯТКИН)

Приведены практические советы использования MySQL и проектирования баз данных. Проблемы при использовании MySQL можно разделить на три группы : использование индексов, структура базы данных, SQL запросы.

Работа с базой данных занимает большую часть работы практически любого сайта. И именно работа с БД чаще всего является узким местом веб-приложений.

Проблемы при использовании MySQL

Проблемы при использовании MySQL можно разделить на три группы (в порядке значимости):

1. Неиспользование или неправильное использование индексов.
2. Неправильная структура БД.
3. Неправильные \ неоптимальные SQL запросы.

Остановимся на каждой из этих групп подробнее.

Использование индексов

Неиспользование или неправильное использование индексов – это то, что чаще всего замедляет запросы.

***Советы по использованию индексов*:**

Не нужно индексировать все подряд. Довольно часто, не понимая смысла, многие просто индексируют все поля таблицы. Индексы ускоряют выборки, но замедляют вставки и обновления строк, поэтому выбор каждого индекса должен быть осмыслен.

• Один из основных параметров, характеризующий индекс — селективность(selectivity) — количество разных элементов в индексе. Нет смысла индексировать поле, в котором два-три возможных значения. Пользы от такого индекса будет мало.

• Выбор индексов должен начинаться с анализа всех запросов к данной таблице. Очень часто после такого анализа вместо трех-четырех индексов можно сделать один составной.

• При использовании составных индексов порядок полей в индексе имеет определяющее значение.

• Не забывайте про покрывающие(covering) индексы. Если все данные в запросе могут быть получены из индекса, то MySQL не будет обращаться непосредственно к таблице. Подобные запросы будут выполняться очень быстро. Например, для запроса SELECT name FROM user WHERE login=«test» при наличии индекса (login, name) обращения к таблице не потребуется. Порой имеет смысл добавить в составной индекс дополнительное поле, которое сделает индекс покрывающим и ускорит запросы.

• Для индексов по строкам часто достаточно индексировать лишь часть строки. Это может значительно уменьшить размер индекса.

• Если % стоит в начале LIKE(SELECT * FROM table WHERE field LIKE '%test') индексы использоваться не будут.

• FULLTEXT индекс используется только с синтаксисом MATCH … AGAINST.

Структура БД

Грамотно спроектированная БД – залог быстрой и эффективной работы с базой. С другой стороны, плохо продуманная БД – это всегда проблема для разработчиков.

Советы по проектированию БД:

Используйте минимально возможные типы данных. Чем больше тип данных, тем больше таблица, тем больше обращений к дискам нужно для получения данных. Используйте удобную процедуру: SELECT * FROM table_name PROCEDURE ANALYSE(); для определения минимально возможных типов данных.

• На этапе проектирования соблюдайте нормальные формы. Часто программисты прибегают к денормализации уже на этом этапе. Однако в большинстве случаев в начале проекта не ясно, чем это может закончиться. Денормализовать таблицу гораздо проще, чем страдать от неоптимально денормализованной. Да и JOIN порой работает быстрее, чем неверно денормализованные таблицы.

• Не используйте NULL столбцы кроме случаев, когда они вам осознанно нужны.

SQL запросы

Избегайте запросов в цикле. SQL – язык множеств, и к написанию запросов нужно подходить не языком функций, а языком множеств.

• Избегайте * (звездочки) в запросах. Перечислите именно те поля, которые вы выбираете. Это сократит количество выбираемых и пересылаемых данных. Кроме этого, не забывайте о покрывающих индексах. Даже если вы действительно выбираете все поля в таблице, лучше их перечислить. Во-первых, это повышает читабельность кода. При использовании звездочек невозможно узнать, какие поля есть в таблице, без заглядывания в нее. Во-вторых, сегодня в вашей таблице пять INT столбцов, а через месяц добавилось еще одно TEXT и BLOB, а звездочка как была, так и осталась.

• При постраничном выборе для получения общего количества записей используйте SQL_ CALC_ FOUND_ROWS и SELECT FOUND_ROWS(); При использовании SQL_CALC_FOUND_ROWS MySQL кеширует выбранное количество строк (до применения LIMIT) и при SELECT FOUND_ROWS() только отдает это закешированное значение без необходимости повторного выполнения запроса.

• Не забывайте, что у INSERT есть синтаксис для множественной вставки. Один запрос будет выполняться на порядок быстрее, чем множество запросов в цикле.

• Используйте LIMIT там, где вам не нужны все данные.

• Используйте INSERT… ON DUPLICATE KEY UPDATE… вместо выборки и INSERT или UPDATE после нее, а также часто вместо REPLACE.

• Не забывайте про замечательную функцию GROUP_CONCAT. Она может выручить при сложных запросах.

Ведется много споров на счет эффективности и надежности использования некоторых из описанных методов, но окончательное решение за вами. В любом случае, теперь есть варианты, над которыми стоит задуматься.

ЛИТЕРАТУРА

1. Оптимизация запросов СУБД [Электронный ресурс] / Википедия – Свободная энциклопедия. – Режим доступа: https://ru.wikipedia.org/wiki/Оптимизация_запросов_СУБД. – Дата доступа: 22.09.2017.
2. Оптимизация структуры базы данных [Электронный ресурс] / mysql.ru. – Режим доступа: http://www. mysql.ru/docs/mysql-man-4.0-ru/mysql-optimisation. html. – Дата доступа: 22.09.2017.
3. Оптимизация в MySQL [Электронный ресурс] / OpenNet. – Режим доступа: https://www.opennet.ru/ docs/RUS/mysqlcli/glava14.html. – Дата доступа: 23.09.2017.