ОПТИМИЗАЦИЯ ЗАПРОСОВ SQL (MySQL)

Эдуард ЦУРКАН

Департамент Программная Инженерия и Автоматика, группа TI-195, Факультет Вычислительной Техники, Информатики и Микроэлектроники, Технический Университет Молдовы, Кишинев, Республика Молдова

Автор корреспондент: Эдуард ЦУРКАН, e-mail: eduard.turcan@isa.utm.md

Научный руководитель: Дориан САРАНЧУК, DISA, FCIM, UTM

Аннотация: *Статья посвящена оптимизации запросов sql (MySQL). Целью является дать ответ на вопрос - как увеличить производительность системы, которая использует базу данных с (медленными) не оптимизированными запросами. Как, благодаря простым правилам уменьшить время ожидания и ускорить работу всей системы, чтобы работа с ней приносила удовольствие конечным пользователям.*

Ключевые слова: *оптимизация, индексы, SELECT, WHERE, LIKE, LIMIT, PRIMARY KEY, снижение нагрузки, DISTINCT.*

Введение

Оптимизация – это изменение системы с целью повышения ее быстродействия. Она состоит из 3-х типов: оптимизация запросов, оптимизация структуры и оптимизация сервера. В данной статья будет рассмотрен 1-ый тип оптимизаций БД. К примеру, если рассматривать обычный сайт, у которого посещаемость пользователей мала и который не так нагружен информацией, то не будет понятно, оптимизированы запросы к БД или нет. А вот для популярных, больший рабочих сайтов, серверов под большой нагрузкой (большая посещаемость пользователей), то разница между оптимизированным и неоптимизированным SQL является огромной, и во время выполнения они могут значительно влиять на поведение и надежность сервисов (медленно обрабатываются, большая задержка, видная пользователям).

1. Индексы

Индексы используются в MySQL для поиска кортежей с указанными значениями атрибутов, в паре с WHERE, где в качестве параметра задается условие поиска. Без индексов, MySQL начинает проходится по отношению с первого кортежа, в поиске указанных ранее значений. Чем больше отношение, тем больше понадобится времени на работу запроса [1]. Если у отношения, атрибуты имеют индексы, то они будут использоваться в запросе для поиска, тем самым MySQL быстрее найдет нужный кортеж, без просмотра всего отношения.

2. SELECT

2.1 SELECT* - вывод всех атрибутов из отношения. Как известно, чем больше данных используется в запросе для считывается из отношения, тем медленнее данный запрос будет выполнятся. SELECT* - увеличивает время работы с хранилищем данных [2]. Использования этого метода может привести к большой задержке при передаче данных по сети, если веб-приложение использует БД с такими запросами. Самое простое решение этой проблемы – это прописать вручную, какие именно столбцы из запроса нужны.

2.2 Оптимизация WHERE

Используя несколько условий в WHERE, разделенные ключевым словом AND, необходимо расставить их в правильном порядке. Первыми условиями должны идти те, которые с большей вероятностью могут вернуть FALSE. Это делается для того, чтобы запрос меньше обрабатывался в случае ошибки, как следствие скорость выполнения увеличивается. Что касается той же ситуации, но с использованием ключевого слова OR, то тут немного иначе. Условия должны располагаться в порядке уменьшения вероятности истинности

данного условия. Это означает, что чем быстрее одно из условий вернет TRUE, тем меньше условий еще будет обработано, тем самым увеличься производительность запроса.

Если необходим запрос, в котором проверяется несколько конкретных условий, по типу: нужна информация о пассажирах 1 и 3-ого авиарейсов, то лучше использовать операцию IN, взамен OR. В данном случае скорость выполнения запроса возрастет, так как операция IN изначально работает быстрее, чем операция OR.

Прошлое замечание уместно и для следующих операций. Если нужно узнать, существует ли кортеж, который соответствует каким-то условиям, то используйте EXISTS вместо COUNT> 0 в своих подзапросах. Это непременно уменьшит время выполнения всего запроса. Операцию LIKE, также, лучше использовать реже, взамен предлагается использовать поиск, основанный на FULL-TEXT индексах. Аргумент – прост, банально быстрее.

2.3 Группировка с помощью SELECT

Чтобы оптимизировать операцию группировки следует использовать как можно меньше атрибутов для группировки. Лучше использовать оператор WHERE вместо HAVING, потому что в этом случае уменьшается количество строк для группировки на ранней стадии [3]. Использование команды DISTINCT ускорит работу группировки, если не используются функций (COUNT (), MIN (), MAX и т.д.). Для более быстрого выполнения запросов с использованием DISTINCT стоит добавить команду LIMIT. Эта делается из-за того, что команда DISTINCT требует повышенного времени обработки.

2.4 Сортировка в SELECT

Одной из самых ресурсоемких задач в MY SQL является сортировка строк(кортежей). Чтобы снизить нагрузку этой операции необходимо, при инициализации атрибутов устанавливать точный размер, столько сколько нужно, и не выделять лишние байты про запас. Как пример, для атрибута “фамилия” не выделять больше места (VARCHAR (100)), а выделить минимально возможный размер (VARCHAR (30)).

3. Создание атрибута ID для каждого отношения

Для нормальной работы БД, в каждой таблице желательно иметь поле ID, которое будет PRIMARY KEY или одним из атрибутов составного ключа с типом INT [4]. Так же неплохо сделать это поле - UNSIGNED, потому что отрицательные значения у него никогда не будут. К примеру, если существует отношение, в котором есть уникальное поле username (имя пользователя), это еще не значит, что оно может являться ключем. Использование поля с типом VARCHAR, как PRIMARY KEY - очень медлительно. Читабельнее будет, если каждый пользователь, помимо имени будет обладать уникальным номером (ID), так как в этом отношении могут быть кортежи с одинаковыми значениями в поле username(Тески).

4. Использование NOT NULL

Если есть необходимость в использовании NULL, то его можно без особых проблем для запроса использовать, но, если надобности нет, то лучше использовать NOT NULL. В сравнении с NOT NULL, NULL занимает больше места в память и усложняет работу запроса, использующего этот атрибут. В подтверждении этого - Документация MySQL: «Столбцы NULL занимают больше места в записи, из-за необходимости отмечать, что это NULL значение [5]. К примеру, для таблиц MyISAM (одна из основных систем хранения данных в СУБД MySQL.), каждое поле с NULL занимает 1 дополнительный бит, который округляется до ближайшего байта».

5. Использование LIMIT 1, при поиске одного-единственного кортежа

Создавая запрос, где результатом будет один-единственный кортеж желательно использовать LIMIT 1. Это могут быть примеры, по типу: необходимо найти один уникальный кортеж или же, просто проверить существование записей, которые удовлетворяют запросу WHERE.

Во всех этих случаях, использование LIMIT 1 в запросе будет ускорять его работу. Следовательно, база данных остановит выборку кортежей, после нахождения первого соответствия, вместо того, чтобы выбирать всю таблицу или индекс какого-то атрибута.

6. Использование статичных отношений.

Статичным отношением, можно назвать таблицу, которая имеет атрибуты фиксированного размера, их еще называют «таблицы фиксированного размера». VARCHAR (50) и CHAR (50) с первого взгляда кажутся похожими, но дело в том, что запись (CHAR (50)) будет занимать 50 байтов, вне зависимости от ее реального содержания, а (VARCHAR (50)) не обязательно все 50 байтов (это замечание очень помогает при восстановление данных, при их потере). К примеру, VARCHAR, TEXT, BLOB являются типами данных для которых размер не известен заранее и может меняться (не фиксированный размер). Если добавить в отношение такой атрибут, то это отношение уже не будет статичным, соответственно, оно будет обрабатываться MySQL по-другому.

Использование таблиц с фиксированным размером увеличит эффективность базы данных. Зная размер кортежа, MySQL сможет очень быстро вычислить его позицию в базе данных. Если же, размер кортежа окажется не фиксирован, тогда MySQL будет обрабатывать это действие по-другому, через индексы. К преимуществам использования статичных отношений еще можно отнести: простое кэширование данных, и соответственно восстановление после различных ошибок (“падение БД”).

7. Заключение

В данной статье были затронуты лишь некоторые возможные варианты оптимизации запросов. Если придерживаться данным советам, то ваши SQL запросы будут выполняться быстрее, тем самым вы увеличите производительность своей БД [6]. Все советы, данные в этой статье были основаны исходя из синтаксиса языка запросов SQL и при помощи официальной документации MySQL, которая настоятельно рекомендует придерживаться всем вышеперечисленным стандартам работы с запросами в БД MySQL.

Библиография

1. Оптимизация запросов в MySQL – [Электронный ресурс]. – [дата обращения 02.03.2023], Доступно: http://www.php.su/articles/?cat=phpdb&page=005
2. Оптимизация MySQL запросов – [Электронный ресурс]. – [дата обращения 02.03.2023], Доступно: https://habr.com/ru/post/41968/
3. Оптимизация SQL – [Электронный ресурс]. – [дата обращения 02.03.2023], Доступно: https://falcon.web-automation.ru/list/sqlserver/opt
4. Оптимизация MySQL запросов – [Электронный ресурс]. – [дата обращения 02.03.2023], Доступно: https://handyhost.ru/help/poleznyie-stati/optimizacziya-mysql-zaprosov.html
5. MySQL 8.0 Reference Manual / Optimization – [Электронный ресурс]. – [дата обращения 02.03.2023], Доступно: https://dev.mysql.com/doc/refman/8.0/en/optimization.html
6. MySQL Documentation – [Электронный ресурс]. – [дата обращения 02.03.2023], Доступно: https://dev.mysql.com/doc/