УДК 004.65

Нурматова Е.В.
канд. техн. наук, доцент
МИРЭА – Российский технологический университет
г. Москва, Россия
Завгородний Н.И.
магистрант
Технологический институт университета «Дубна»
г. Дубна, Россия

ГЕНЕРАЦИЯ ЗАПРОСОВ ДЛЯ НАГРУЗОЧНОГО ТЕСТИРОВАНИЯ СЕРВЕРА БАЗ ДАННЫХ

Аннотация. В процессе разработки клиентской части систем часто сталкиваемся с ситуациями, когда тестирование программного кода отнимает много времени, но обычно не хватает достаточного времени на тестирование базы данных, или это выполняется по остаточному принципу. В результате работа с базой данных может стать узким местом в производительности приложения. Целью данной работы является исследование производительности запросов и разработка приложения для генерации тестовых запросов и тестирования производительности сервера баз данных.

Ключевые слова: тестирование; SQL-запрос; генерация; производительность сервера; оптимизация; нагрузка, приложение.

Настройка производительности запросов может быть достаточно сложной задачей, особенно при работе с большими данными, где даже самое незначительное изменение может резко повлиять на производительность запроса. Когда разработчик имеет дело с огромными таблицами, оптимизация является ключевым фактором. Неэффективный запрос может создать нагрузку на ресурсы базы данных и привести к снижению производительности или потере обслуживания для других пользователей, если в запросе содержатся ошибки [6].

Рассмотрим факторы, по которым можно определить необходимость оптимизации или настройки системы управления данными, направленной на повышение скорости её работы или сокращение объёма используемой памяти [1]:

– *Увеличение скорости обработки и выполнения запросов.* Скорость работы сайта прямо пропорционально зависит от времени обработки и выполнения SQL-запроса к базе данных, которое должно быть минимальным.

– *Предотвращение перегрузки сервера.* При перегрузке сервера работа web-ресурса или приложения будет нестабильной. Host-ер может заблокировать ресурс, чтобы последний не нарушал работу всего сервера, на котором также работают и другие сайты.

– *Уменьшение времени ожидания загрузки web-страницы.* Когда идет огромное количество sql-запросов к базе данных, происходит существенное замедление работы сайта, что недопустимо для коммерческого или представительского интернет-ресурса.

– *Экономия ресурсов хостинга.* Если система управления данными не оптимизирована, то происходит значительный перерасход использования ресурсов сервера (процессорного

времени, оперативной памяти). На этом основании host-ep имеет право заблокировать работу ресурса.

– *Возможность масштабировать ресурс*. При расширении сайта будет невозможно обеспечить хорошее качество его работы.

Тестирование запросов с помощью приложения позволяет оптимизировать или найти более подходящий вариант для требуемых задач и сэкономить средства [3]. Существует много инструментов для качественной генерации тестовых запросов с различными встроенными функциями, наиболее востребованными из которых являются DTM Data Generator, Databene Benerator. EMS data generotor MySql и многие другие [2].

Цель разработки приложения, описываемого в работе, состоит в проверке сервера баз данных большим количеством одновременных транзакций. Предполагается, что таким образом моделируется многопользовательская работа. При помощи приложения поверяем среднее время отклика при обработке параллельных транзакций, производительность привилегированных, либо часто вызываемых транзакций, и далее, стандартных и менее важных транзакций, время, требуемое на обработку запросов записи в базу. Другими словами, производится моделирование нагрузочного тестирования сервера баз данных.

На рисунке 1 показана схема процесса генерации тестовых запросов, при помощи которого создаётся сценарий, поддерживающий необходимый синтаксис написания тестов.

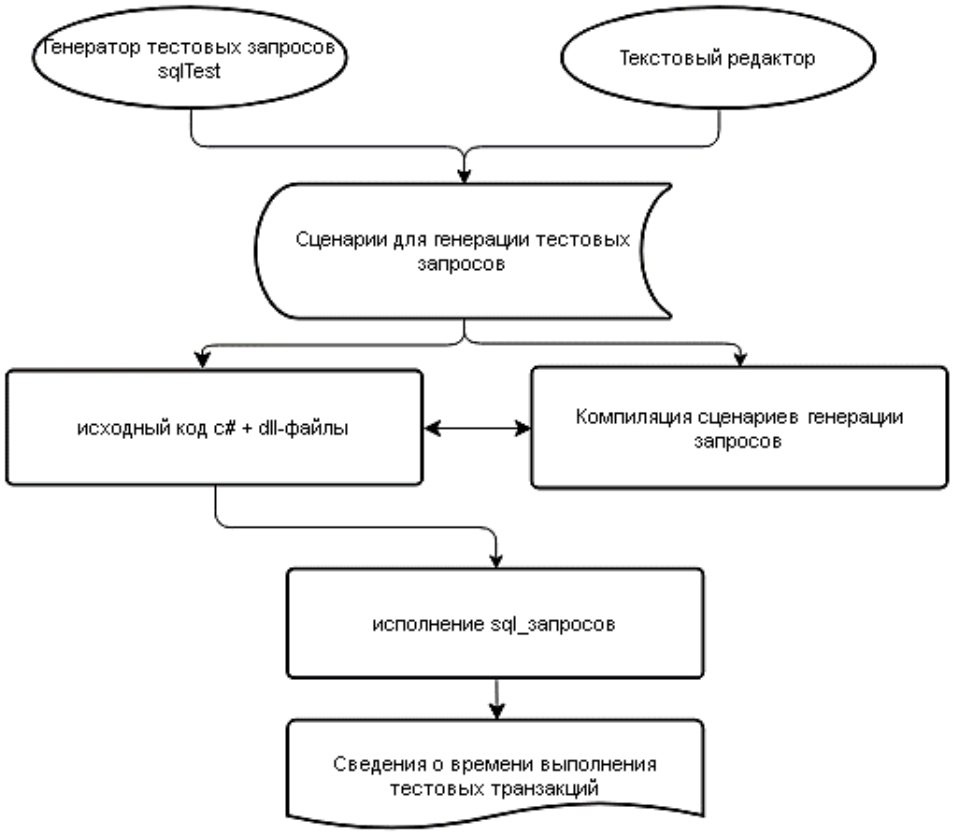

Рис. 1. Схема генерации тестовых запросов

Рассмотрим реализацию приложения для быстрой генерации тестовых запросов в базу данных с последующим сохранением/ редактированием написанных сценариев. На рисунке 2

приведен общий интерфейс генератора с элементами управления интерфейсом приложения: 1 – выбор префикса для работы с БД; 2 – выбор теста; 3 – данные для подключения; 4 – кнопки проверочного подключения и отключения; 5 – кнопка начала теста; 6 – log-текст бокс; 7 – кнопка очистки log-а.

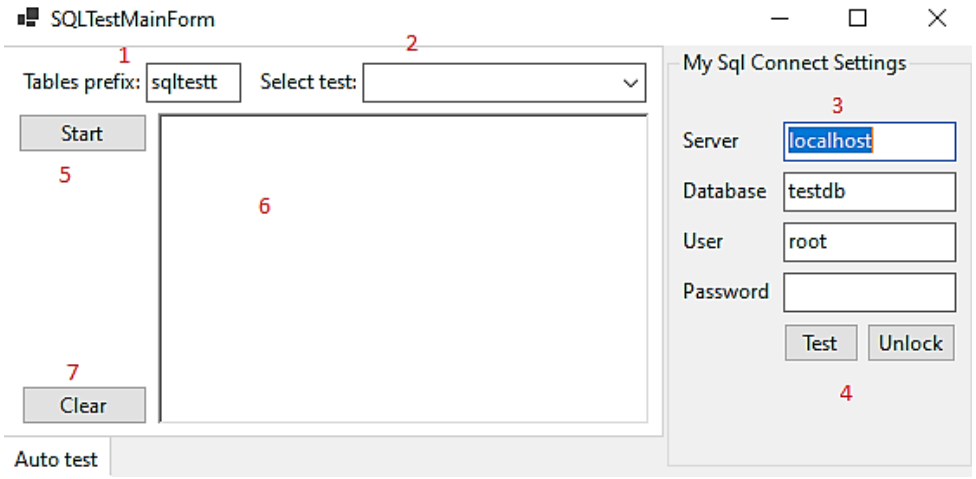

Рис. 2. Основное элементы управления в программе

Справа указываются данные для подключения к host-ерам сервера, параметры тестируемой базы данных, пользователя, имеющего соответствующие права. При любой ошибке во время подключения или во время тестового запроса будет выведено соответствующее сообщение на экран пользователя: «access denied for user 'root2'@'localhost' (using password: NO)».

Тесты подгружаются динамически в программу при её запуске, все компоненты тестов содержатся в папке data\autotests (рис. 3).

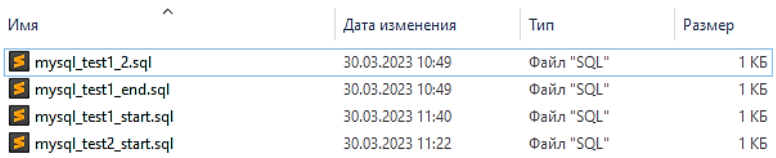

Рис. 3. Содержание папки autotests

Каждый тест содержит в себе основной «начальный» файл, он должен называться по маске “mysql_*_start.sql”.

На основном окне программы можно увидеть поле для ввода префикса баз, это «заполнитель», который будет подставляться в файлы тестов на место !@prefix@!, так же добавляется символ _, используется для использования уникальных имён во время тестов.

Синтаксис последней строки определяет дальнейшее действие программы после выполнения. Синтаксис для выполнения следующего файла имеет следующий вид:

Next: mysql_test1_2.sql : 200000

где, mysql_test1_2.sql – название следующего файла, а 200000 – количество раз для выполнения этого файла. Далее в следующем файле будет содержаться такая же строка, которая продолжает выполнение. Последний файл должен содержать в последней строке тэг

корректного завершения теста: ## End ##. Этот тег позволяет определить, что последний файл точно выполнен и произошло корректное завершение теста.

При необходимости заполнения случайных данных их можно сгенерировать средствами SQL, это так же вынесет нагрузку на SQL сервер, собственно, что мы и тестируем. В случае неуспешного завершения теста, возвращается сообщение об ошибке и место прерывания (см. рис. 5).

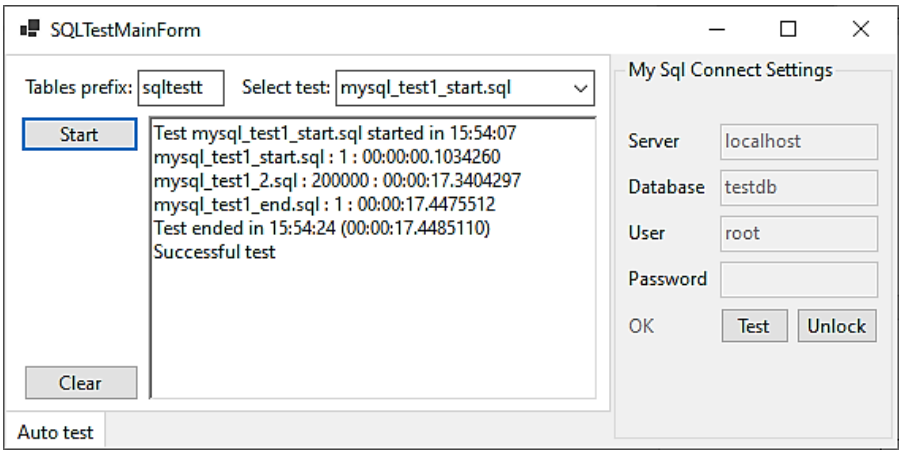

Рис. 4. Пример успешного выполнения теста

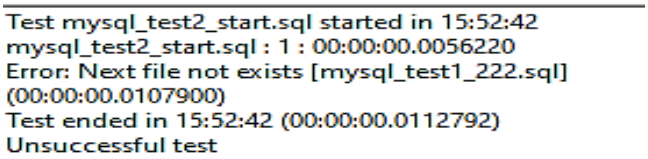

Рис. 5. Сообщение об ошибке, в случае не успешного выполнения теста

Общий подход построения тестов позволяет вынести тесты из кода программы и обновлять их быстрее и независимо от программы. Первый предложенный тест для примера создаёт пустую таблицу, делает 200000 select-запросов и удаляет базу таблицу. Однако предложенный синтаксис продолжения теста позволяет протестировать абсолютно любой сценарий неограниченной сложности, благодаря этому можно имитировать работу приложений и проверять сценарии работы с базой данных на разных серверах для сравнения общей производительности.

Так же благодаря этому подходу вся сложность выполнения и генерации (при необходимости) переносит нагрузку на SQL сервер, делая минимальную погрешность в тесте самой программы. Сам тест выполняет в отдельном потоке и не мешает работу интерфейсу программы, время теста так же замеряется самой программой.

Для написания тестов со случайными значениями предлагается использовать, например,

CROSS APPLY (SELECT TOP(CAST(RAND(N) * 10 AS INT)) txt FROM (VALUES(N'Сергей'),(N'Олег'), (N'Николай')) t(txt)) t

И аналогичные запросы.

Благодаря вышеизложенным инструкциям можно сгенерировать комбинацию запросов, которые образуют тест любой сложности. Тестирование на разных серверах одного и того же

теста будет отображать производительность тестируемого кейса в общем и на каждом этапе теста конкретно, это необходимо для сравнения и дальнейшей оптимизации сервера базы данных или изменения в запросах приложениях. Этому так же способствует возможность быстрого переключения между тестами и серверами в приложении.

В завершение, можно отметить, что, перенося базу данных в производственную среду, нужно позаботиться о том, чтобы производительность базы данных была адекватной. Если к проекту прилагается формальная спецификация требований, проверить, указываются ли в ней какие-либо ограничения производительности. Для приложений, работающих с базами данных, нередко задается максимальное время выполнения запросов. Продолжительность времени между вводом инструкции и получением результатов запроса зависит от многих факторов. Необходимо заранее учесть те факторы, которые впоследствии нельзя будет контролировать.

Если обнаруживается, что система требует оптимизации, в первую очередь подумать об обновлении аппаратной части. Это может оказаться самым дешевым вариантом. В 1965 г. Гордон Мур (Gordon Moore) установил, что вычислительные мощности удваиваются каждые 18 месяцев. Данное правило называют законом Мура [4; 5]. Но, несмотря на столь стремительный рост производительности, удельная стоимость вычислительных средств неуклонно снижается. Например, центральные процессоры за полтора года удвоят тактовую частоту, хотя стоить будут так же, как и полтора года назад. Таким образом, обновление компьютера может обойтись дешевле, чем оптимизация проекта. Во вторую очередь стоит подумать об обновлении программного обеспечения. Основной программный компонент– это операционная система. Известно, что Linux и BSD UNIX позволяют повысить производительность старых компьютеров, превосходя в этом отношении коммерческие операционные системы, такие, как Windows, особенно если бессбойная работа сервера очень важна.

Обновляется и сама система управления данными. Когда появится новая версия, ее производительность будет повышена в сравнении с текущей. Но и в текущую версию регулярно вносят мелкие исправления, так что желательно идти в ногу со временем. Основная причина оптимизации – желание сэкономить деньги. Не стоит забывать об этом в попытках повысить производительность программы. Нет смысла затрачивать на оптимизацию больше денег, чем она способна принести. Стоит потрудиться над такой программой, с которой работает множество людей, особенно если это коммерческое приложение.

Литература

1. Борчук Л.Е., Кузьмин А.А. Оценка времени выполнения запроса в реляционной СУБД на основе асимптотических моделей затрат ресурсов. Наукоемкие технологии. 2008. №4. С. 61-64.

2. Иванов О. Машинное обучение для планирования запросов // Открытые системы. СУБД. 2016. №1. С. 22-25.

3. Нурматова Е.В. Анализ процедурного плана sql-запроса // Опорный образовательный центр. Казань: Иннополис, 2021. Т.2. С. 116-121.

4. Пашинин О.В. Оптимизация запросов к базам данных. М.: Математические структуры и моделирование. 2017. Вып. 17. С. 100-107.

5. Хайлан А.М., Польщиков К.А., Алгазали С.М.М. Обнаружение ресурсоемких запросов к базам данных на основе применения самоорганизующихся карт и нечеткого вывода // Экономика. Информатика. 2021. Т. 48. №3. С. 578-593. https://doi/org/10.52575/2687-0932-2021-48-3-578-593.

6. Wang S., Rundensteiner E.A., Mani M. Optimization of nested xquery expressions with orderby clauses // Data & Knowledge Engineering. 2007. Т. 60. №2. С. 303-325. https://doi/org/10.1016/j.datak.2006.03.004