Важным является вопрос минимизации накладных расходов в виде избыточного сетевого трафика, создаваемого при работе системы. Главным критерием здесь является минимальное количество сетевых обменов между клиентом и сервером и минимизация размера передаваемых данных.

Второй подзадачей является вопрос выбора режима взаимодействия:

– с сохранением состояния и установкой соединения;

– без установки соединения и без сохранения состояния.

Для первого варианта характерной является большая нагрузка на сеть, так как необходимо отслеживать состояние соединения и более сложная логика серверной части, так как при перезагрузке компьютера из одной операционной системы в другую будет требоваться повторная установка соединения и, возможно, повторный опрос для получения данных.

В зависимости от периодичности опроса должен решаться вопрос кэширования извлеченной из локальной операционной системы информации о входе/выходе пользователей системы. При высокой частоте опроса и больших объемах файлов протоколов (например, в отсутствие ротации log-файлов) клиентский модуль может порождать нежелательную нагрузку на клиентский узел за счет повторного разбора данных из источника.

Заключение. Данный этап работ предусматривал проектирование и экспериментальную реализацию клиентского модуля для операционных систем семейства GNU/Linux, а также проектирование сетевой подсистемы для взаимодействия с сервером, предоставляющим данные для обработки модулю аналитики. В настоящее время выполняется разработка нескольких вариантов реализации клиентского модуля на базе различных технологии и языков программирования и оценка эффективности подходов к построению сетевой подсистемы коммуникации между клиентами и серверным модулем. Результаты удовлетворяют заявленным требованиям к проекту. Далее планируется тестовая эксплуатация и отладка сетевой подсистемы в режиме реальной эксплуатации.

1. Новый, В.В. Об учете времени использования компьютерной техники в учебных лабораториях / В.В. Новый // Наука – образованию, производству, экономике: материалы 74-й Региональной научно-практической конференции преподавателей, научных сотрудников и аспирантов, Витебск, 18 февраля 2022 г. – Витебск: ВГУ имени П.М. Машерова, 2022. – с. 33–34. – URL: https://rep.vsu.by/handle/123456789/31552 (дата обращения: 31.01.2024).

ОБ ОПТИМИЗАЦИИ РАБОТЫ С БАЗАМИ ДАННЫХ ПРИ ИСПОЛЬЗОВАНИИ ORM НА ПРИМЕРЕ ENTITY FRAMEWORK

М.Г. Семёнов
Витебск, ВГУ имени П.М. Машерова

Современные приложения повсеместно используют базы данных для хранения информации. Для доступа приложения к базе данных классически используются такие шаблоны проектирования, как Repository, Unit of Work, DAO, ORM и другие. Однако если раньше было целесообразно разрабатывать свою реализацию шаблона для конкретного решения, то современные приложения используют готовые реализации. Одной из таких реализаций, использующейся при разработке на стеке технологий .NET, является Entity Framework.

При работе с БД в целом и с Entity Framework в частности, разработчики программного обеспечения сталкиваются с неизбежной задачей оптимизации SQL запросов для улучшения производительности. *Целью* данной работы является систематизация и анализ методов оптимизации SQL запросов, направленных на уменьшение нагрузки на базу данных и повышение общей эффективности приложения, при работе современных ORM на примере Entity Framework.

Материал и методы. Материалом для исследования являются документация ASP.NET, Entity Framework и полученные ранее опыт в данном направлении. Методы исследования: анализ источников, изучение и обобщение сведений, сравнительный анализ.

Результаты и их обсуждение. *Первым* направлением оптимизации запроса является четкое ограничение количества связных данных, которое выгружается из базы в память приложения. Базы данных представляют собой множество (десятки, сотни, а иногда и тысячи) связных таблиц. Если попытаться выгрузить больше информации чем реально нужно, то время работы запроса будет значительно увеличиваться с каждой дополнительной таблицей. В тоже время, если в запросе мы получим не все необходимые данные, то в последующем мы можем попасть в ситуацию, когда данные будут выгружаться дополнительными запросами и, возможно, такие запросы будут отправляться для каждой записи из выборки. Такой подход приведет к еще большой нагрузке на базу данных.

В этом направлении в Entity Framework реализовано несколько подходов:

1) «жадная» загрузка (Eager loading)

2) ленивая загрузка (Lazy loading)

3) явная загрузка (Explicit loading)

При использовании Eager Loading следует использовать метод Include, чтобы явно указать, какие связанные данные следует загрузить вместе с основными. Этот подход зарекомендовал себя лучше всего, поскольку позволяет явно указать список связных данных (таблиц) и получить эти данные для всех необходимых записей.

При использовании Lazy loading мы не указываем список данных заранее, но эти данные неявно подгружаются по мере их необходимости. Однако, при неверном использовании данного подхода количество запросов и нагрузка на базу данных может существенно вырасти.

Explicit loading, в свою очередь, представляет нечто среднее между двумя предыдущими. С одной стороны мы не получаем все данные сразу, с другой мы не уменьшаем риски значительного увеличения количества запросов за счет явного контроля дополнительных данных.

Вторым направлением является уменьшение количества запросов за счет отслеживания состояний (state tracking) сущностей на уровне приложения. При изменении нескольких из них, мы можем поместить все такие изменения в один запрос, что создаст значительно меньшую нагрузку, чем изменение данных «поштучно». Применение данного подхода в Entity Framework является поведением по умолчанию. Стоит отметить, что это создает некоторую избыточность потребления ресурсов в сценариях, когда мы выгружаем данные только для чтения. Для оптимизации таких сценариев рекомендуется явно отключать отслеживание состояний (например, использовать метод AsNoTracking).

Третьим направлением является рациональное применения фильтрации и проекций. В случае если в отдельной таблице находится достаточно много полей, однако для текущего варианта использования нашего приложения необходимо получить лишь незначительную часть из них, можно применить проекцию верхнего уровня и получить из базы данных только то что нужно. Например,

```
var specificOrders = dbContext.Orders
    .Select(o => new { o.OrderId, o.OrderDate })
    .ToList();
```

Четвертым направлением является применение классических SQL запросов и хранимых процедур. Хотя современные версии Entity Framework достаточно хорошо оптимизируют запросы внутренними средствами, иногда (стоит отметить, что с каждой

новой версией все реже) встречаются сценарии в которых SQL запрос написанный вручную имеет значительно белее высокую производительность. Нужно понимать, что при использовании данного подхода мы лишаемся оптимизаций EntityFramework запросов, которые совершенствуются с каждой новой версии. Кроме того, применение такого подхода значительно уменьшает читаемость кода и увеличивает затраты на его сопровождение. Стоит отмечать места применения классических SQL запросов и хранимых процедур для потенциального рефакторинга в будущем, а также применять их только в случае значительного выигрыша в производительности в тех местах где это действительно необходимо.

Пятым подходом является индексирование в базе данных. Данный подход не зависит от применяемой технологии доступа к данным, однако является классическим и показывает хорошие результаты, поэтому достоин упоминания в контексте данной работы.

Заключение. В зависимости от конкретных требований и структуры базы данных, эти методы оптимизации могут быть адаптированы для достижения наилучших результатов. Следует помнить, что оптимизация SQL запросов – это непрерывный процесс, и регулярный мониторинг производительности приложения поможет выявить новые возможности для улучшений.

1. Entity Framework Docs [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/ef/core/ – Дата доступа: 27.01.2024.

2. Rojansky, S. Efficient Querying [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/en-us/ef/core/performance/efficient-querying. – Дата доступа: 28.01.2024.

О ПРЕПОДАВАНИИ КУРСА «МАТЕМАТИЧЕСКИЙ АНАЛИЗ» ДЛЯ СПЕЦИАЛЬНОСТИ «ФИЗИКО-МАТЕМАТИЧЕСКОЕ ОБРАЗОВАНИЕ (МАТЕМАТИКА И ИНФОРМАТИКА)»

Т.Л. Сурин, Ж.В. Иванова
Витебск, ВГУ имени П.М. Машерова

В настоящее время любая страна не может обойтись без грамотных, квалифицированных специалистов, от деятельности которых зависит экономика и производство, а, следовательно, и благосостояние ее граждан. Поэтому так необходима четко функционирующая система образования, где работают компетентные, знающие свой предмет педагоги. Как следствие, одной из наиболее важных государственных задач является подготовка кадров для данной системы. Формирование профессиональных компетенций, которыми должен обладать будущий педагог, является сложной задачей и должно осуществляться в ходе всего учебного процесса, в частности при изучении специальных дисциплин, например, таких как математический анализ. Все это в полной мере относится к подготовке будущих учителей математики и информатики.

Цель исследования – рассмотреть некоторые проблемы, которые возникают при чтении курса «Математический анализ» студентам специальности «Математика и информатика» («Физико-математическое образование (математика и информатика)»).

Материал и методы. Материалом исследования являются образовательные стандарты высшего образования, учебные планы и учебные программы для специальности 1-02 05 01 «Математика и информатика» (6-05-0113-04 «Физико-математическое образование (математика и информатика)»).

В качестве методов исследования использованы общенаучные методы: анализ, обобщение, систематизация.

Результаты и их обсуждение. В школах Витебска и Витебской области в последние годы ощущается нехватка учителей математики и информатики. Подготовка спе-