Оптимизация в БД

"Сложная система, спроектированная наспех,
никогда не работает,
и исправить её, чтобы заставить работать,
невозможно".

SQL – декларативный язык

- Мы говорим, *что* надо получить в результате запроса, но не говорим, *как.*
- Аналогично с таблицами – мы определяем атрибуты и связи между объектами, но не управляем физическим хранением.

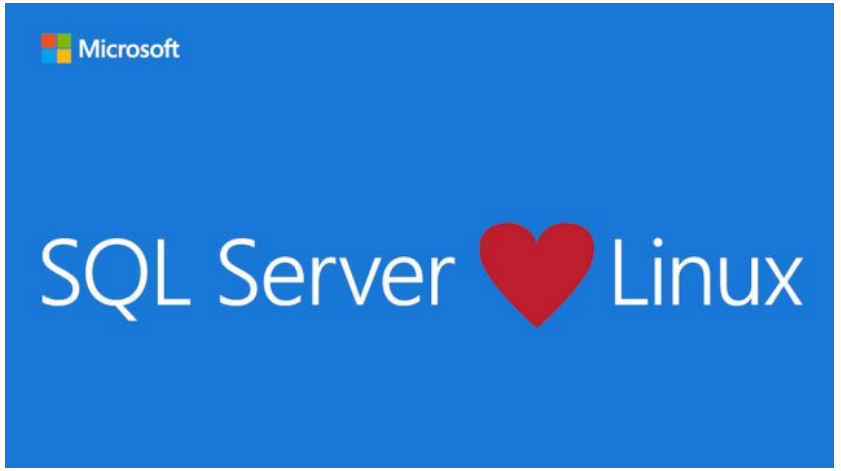
Что влияет на выполнение запроса?

- Внутренний оптимизатор СУБД, который в определяет наиболее эффективный способ выполнения реляционных запросов.
- Использование таких реляционных запросов, для которых СУБД могла бы использовать более эффективные способы нахождения данных.

Оптимизация производительности сервера

- Конфигурация памяти
- Конфигурация ввода и вывода
- Настройка параметров Windows

  с 2017 г.



Конфигурация памяти

- Компонент управления памятью при запуске SQL Server динамически определяет объема выделяемой для него памяти.
- Выделяемый объем памяти определяется исходя из объема, уже используемого операционной системой и другими приложениями.
- При изменении загрузки компьютера и SQL Server меняется и объем выделенной памяти.

Настройки памяти

- min server memory
- max server memory
- max worker threads (255 по умолчанию) максимальное число рабочих потоков SQL Server.
- index create memory управляет объемом памяти, используемой операциями сортировки при создании индексов.
- min memory per query минимальный объем памяти для выполнения запроса. Может повысить производительность выполнения ресурсоемких запросов. В этом случае выполнение запроса задерживается до момента освобождения необходимого количества памяти, либо истечения времени ожидания, указанного в параметре конфигурации query wait.

Оптимизация производительности сервера с помощью параметров конфигурации ввода и вывода

- Параметр **recovery interval** используется, чтобы установить максимальное количество минут для каждой базы данных, необходимое Microsoft SQL Server для их восстановления.
- SQL Server оценивает, какое количество изменений данных он может накатить в течение интервала восстановления. Обычно SQL Server делает контрольную точку в базе данных, когда количество сделанных в ней с момента создания последней контрольной точки изменений достигает количества, для которого SQL Server предположительно может произвести накат за интервал восстановления.

Оптимизация производительности сервера с помощью параметров Windows

- Максимальное увеличение пропускной способности
- Настройка управления задачами на сервере
- Настройка виртуальной памяти (важно для Full-Text Search)
  - задайте объем виртуальной памяти, как минимум в 3 раза превышающий объем физической памяти компьютера;
  - Присвойте параметру настройки SQL Server max server memory значение, в полтора раза превышающее объем физической памяти компьютера (вдвое меньшее, чем объем виртуальной памяти).

Выполнение запросов

- Язык SQL является декларативным языком. В его командах отсутствует информация о том, как выполнить запрос, какие методы доступа к данным использовать. А почти каждая команда SQL из подмножества языка манипулирования данными (DML) может быть выполнена разными способами.
- Критерий оценки стоимости выполнения запроса - число обменов с устройствами внешней памяти, которые потребуются при выполнении плана запроса.

План выполнения запроса

- Задача:
  - по декларативной формулировке запроса построить программу — план выполнения запроса, которая выполнялась бы максимально эффективно и выдавала бы результаты, соответствующие указанным в запросе свойствам.
- Уточним задачу:
  - построить все возможные программы, результаты которых соответствуют указанным свойствам
  - выбрать из множества этих программ (найти в пространстве планов выполнения запроса) такую программу, выполнение которой было бы наиболее эффективным.

Шаги при выборе оптимального плана

- Прежде всего, необходимо обнаружить все корректные планы выполнения запроса или, по крайней мере, не упустить какой-либо план, который является наиболее эффективным.
- Сократить пространство корректных планов, оставив только те планы, которые претендуют на максимальную эффективность.
- Найти в пространстве планов выполнения запроса единственный план, в соответствии с которым запрос будет реально выполнен. Здесь уже требуются формальные критерии отбора.

- Теперь, что касается самонастраивающихся оптимизаторов запросов. Эта идея (как и большинство идей вообще) не нова. В конце 70-х — начале 80-х годов много писалось о так называемой «глобальной» оптимизации запросов, под которой, главным образом, понимался механизм автоматического поддержания набора индексов, обеспечивающих возможность оптимального выполнения запросов данной рабочей нагрузки СУБД. В то время результаты исследований не нашли практического применения.

Пример № 1 БД

Сотрудники 4-го отдела не старше 25 лет с «чистой» ЗП >= 30000 рублей".

```
SELECT * FROM Emp
WHERE depNo=4 AND born>'1985/01/01'
AND salary*0.87>=30000;
```

```
SELECT * FROM Emp
WHERE depNo=4 AND born>'1985/01/01'
AND salary*0.87>=30000;
```

- Если есть индексы, то способы выполнения этого запроса могут быть:

1. Найти по индексу INDEX(depNo) записи, удовлетворяющие первому условию, и проверить для найденных записей 2-е и 3-е условия.
2. Найти по индексу INDEX(born) записи, удовлетворяющие второму условию, и проверить для найденных записей 1-е и 3-е условие.
3. Последовательно считать все записи таблицы Emp и проверить для каждой записи все условия.

- Индексом по полю salary система воспользоваться не может, т.к. это поле находится внутри выражения.

Пример № 2 БД

employees (enr, ename, status, city)

papers (enr, title, year)

departments (dname, city, street address)

courses (cnr, cname, abstract)

lectures (cnr, dname, enr, daytime)

Считаем операции чтения (r) и записи (w)

Найти названия отделов, расположенных в Нью-Йорке и предлагающих курсы по управлению базами данных

- Имеется 100 отделов, 5 из которых размещаются в Нью-Йорке. В физическом блоке может поместиться 5 записей об отделах или 50 значений dname.
- Имеется 500 курсов, 20 из которых посвящены управлению базами данных. В физическом блоке помещается 10 записей.
- Имеется 2000 лекций, три сотни из них - про управление базами данных, 100 проходят в отделах в Нью-Йорке и 20 (из трех отделов) удовлетворяют обоим условиям. В физический блок помещаются 10 записей.
- Предположим также, что время сортировки составляет N * log(2)N, где N - размер файла в блоках, и что имеется буфер из одного блока для каждого отношения.
- Отношения физически упорядочены по возрастанию значений ключа.

Стратегия 1

- Сформировать декартово произведение отношений "courses", "lectures" и "departments"

  (r: 200000)
- Оставить столбец dname из тех записей "departments", для которых значения столбцов cnr в "courses" и "lectures" совпадают, и значения столбцов dname из "lectures" и "departments" совпадают, и cname = 'database management' and city = 'New York'.

  (w: 1)
- итого: приблизительно 200000 обращений.

Стратегия 2

- Выполнить слияние отношений "courses" и "lectures"
  (r: 50 + 200; w: 400)
- Отсортировать результат по dnames
  (r + w: 400*1og(2)400)
- Выполнить слияние результата с отношением "departments"
  (r: 400 + 20; w: 400 + 400)
- Выбрать комбинации с cname = 'database management' and city = 'New York'
  (r: 800)
- Оставить только столбец dname.
  (w: 1)
- Итого: Приблизительно 6000 обращений.

Стратегия 3

- Выполнить слияние отношений "courses" и "lectures"
  (r: 50 + 200)
- Оставить только dnames из комбинаций cname = 'database management'
  (w: 2)
- Отсортировать сгенерированный список dname
  (r + w: 2)
- Выполнить слияние результата с отношением "departments"
  (r: 2 + 20)
- Оставить только те dnames, для которых city = 'New York'
  (w: 1)
- Итого: 277 обращений

Построение плана выполнения запроса

- Цель СУБД – выполнить запрос, причём сделать это как можно более эффективным способом.
- План выполнения запроса состоит из последовательности шагов, каждый из которых либо физически извлекает данные из памяти, либо делает подготовительную работу.
- Построением этого плана занимается оптимизатор – специальная компонента СУБД.

Квазиоптимальный процедурный план

- План называется квазиоптимальным, т.к. система не гарантирует, что она для любого запроса выберет оптимальный план.
- Для гарантированного выбора оптимального плана необходимо рассмотреть все возможные планы и сравнить их, а это может потребовать больше времени, чем выполнение самого запроса.
- СУБД выбирает и анализирует лишь несколько планов выполнения каждого запроса, и среди них может не оказаться оптимального плана.

Работа оптимизатора состоит из 5 стадий

1. На первой фазе запрос, представленный на языке запросов, подвергается лексическому и синтаксическому анализу.
2. На второй фазе запрос в своем внутреннем представлении подвергается логической оптимизации.
3. Третий этап обработки запроса состоит в выборе на основе информации, которой располагает оптимизатор, набора альтернативных процедурных планов выполнения данного запроса
4. На четвертом этапе по внутреннему представлению наиболее оптимального плана выполнения запроса формируется процедурное представление плана.
5. Наконец, на последнем, пятом этапе обработки запроса происходит его реальное выполнение в соответствии с выполняемым планом запроса.

Оптимизация запросов

- Синтаксическая проверка
- Привязка указанных таблиц столбцов к физическим объектам
- Алгебраический анализатор
- Оптимизатор запроса
- План выполн
- Выполнение

1. Лексический и синтаксический анализ

- Лексический анализатор разбивает запрос на лексические единицы – лексемы (наименования полей и таблиц, константы, знаки операций и т.д.).
- Синтаксический анализатор проверяет синтаксическую правильность запроса.
- В результате вырабатывается внутреннее представление запроса. Оно отражает структуру запроса и содержит информацию, которая характеризует объекты базы данных, упомянутые в запросе (таблицы, поля, константы). Информация об объектах базы данных выбирается из словаря-справочника данных.

2. Логическая оптимизация

- различные преобразования, "улучшающие" начальное представление запроса. Среди этих преобразований могут быть эквивалентные преобразования. После проведения эквивалентных преобразований получается внутреннее представление, семантически эквивалентное начальному запросу.
- могут использовать ограничения целостности, существующие в БД. Такие преобразования являются семантическими, т.е. они основаны на семантике (смысле) предметной области. В этом случае получаемое представление не является семантически эквивалентным начальному запросу. Но система гарантирует, что результат выполнения преобразованного запроса совпадает с результатом запроса в начальной форме при соблюдении ограничений целостности, существующих в базе данных.

Пример № 1 БД

- Например, если для таблицы Emp определено такое ограничение целостности:

(CHECK (salary>9500 AND salary<80000),

SELECT * FROM Emp

WHERE depNo=4 AND born>'1985/01/01'

AND salary*0.87>=30000;

Пример № 1 БД

- Например, если для таблицы Emp определено такое ограничение целостности:

(CHECK (salary>9500 AND salary<80000),

SELECT * FROM Emp

WHERE depNo=4 AND born>'1985/01/01'

AND salary*0.87>=30000 AND salary>9500 AND salary<80000;

Подходы:

- Преобразовывать формулировку запроса к такому виду, чтобы ограничения индивидуальных таблиц производились до их соединения.
- Преобразовать SQL-запросы, в разделе FROM которых присутствуют подзапросы, в запросы с соединениями. Важность этих результатов в том, что: (1) SQL стимулирует использование запросов с вложенными подзапросами; (2) в большинстве оптимизаторов запросов для реализации таких запросов используется некоторая фиксированная стратегия генерации планов (в основном, вложенные циклы); (3) альтернативные формулировки запросов с соединениями допускают порождения большего числа планов, среди которых могут находиться наиболее эффективные.

3. Процедурные планы выполнения запроса

- Основой является информация о существующих путях доступа к данным.
- Единственный путь доступа, который возможен в любом случае, – это последовательное чтение.
- Возможность использования других путей доступа зависит от способов размещения данных в памяти (например, кластеризация данных), наличия индексов и формулировки самого запроса.
- На этом же этапе для каждого плана оценивается предполагаемая стоимость выполнения запроса по этому плану.
- При оценках используется либо доступная оптимизатору статистическая информация о состоянии базы данных, либо информация о механизмах реализации различных путей доступа.
- Из полученных альтернативных планов выбирается наиболее оптимальный с точки зрения некоторого (заранее выбранного или заданного) критерия.

Преобразования операций реляционной алгебры

- Операндами операций РА являются отношения, т.е. неупорядоченные множества кортежей. Для выполнения операций необходимо просмотреть все кортежи исходного отношения (или отношений). Следствием этого является большая размерность операций РА. Уменьшения размерности можно достичь, изменяя последовательность выполняемых операций.
- $|R_1|=1000$, и $|R_2|=1000$, $|s_F(R_1)|=10$, $|s_F(R_2)|=10$
- $s_F(R_1 \cup R_2)$ и $s_F(R_1) \cup s_F(R_2)$
- объединение выполняется путем сортировки данных для удаления одинаковых кортежей и промежуточный результат надо хранить

Преобразования операций реляционной алгебры

- Оптимизация выполнения запросов реляционной алгебры основана на понятии эквивалентности реляционных выражений.
- Операндами выражений являются переменные-отношения R_i и константы. Каждое выражение реляционной алгебры определяет отображение кортежей переменных-отношений R_i (i=1,…,n) в кортежи единственного отношения, которое получается в результате подстановки кортежей каждого R_i и выполнения всех определяемых выражением вычислений.
- Два выражения реляционной алгебры считаются эквивалентными, если они описывают одно и то же отображение.

Законы для эквивалентных преобразований выражений реляционной алгебры (1):

1. Закон коммутативности для декартовых произведений: $R_1 \times R_2 = R_2 \times R_1$
2. Закон коммутативности для соединений (F – условие соединения): $R_1 \bowtie_F R_2 = R_2 \bowtie_F R_1$
3. Закон ассоциативности для декартовых произведений: $(R_1 \times R_2) \times R_3 = R_1 \times (R_2 \times R_3)$
4. Закон ассоциативности для соединений: $(R_1 \bowtie_{F1} R_2) \bowtie_{F2} R_3 = R_1 \bowtie_{F1} (R_2 \bowtie_{F2} R_3)$
5. Комбинация селекций (каскад селекций): $\sigma_{F1}(\sigma_{F2}(R)) = \sigma_{F1 \vee F2}(R)$
6. Комбинация проекций (каскад проекций):

   $\pi_{A1,A2,\ldots Am}(\pi_{B1,B2,\ldots Bn}(R)) = \pi_{A1,A2,\ldots Am}(R)$, где $\{Am\} \subset \{Bn\}$
7. Перестановка селекции и проекции:

$\sigma_F \pi_{A1,A2,\ldots,Am}(R) = \pi_{A1,A2,\ldots,Am}(\sigma_F(R))$

Законы для эквивалентных преобразований выражений реляционной алгебры (2):

8. Перестановка селекции с объединением: $s_F(R_1 U R_2)=s_F(R_1)Us_F(R_2)$
9. Перестановка селекции с декартовым произведением: $s_F(R_1 \times R_2)= (s_{F1}(R_1))\times(s_{F2}(R_2))$
10. Перестановка селекции с разностью: $s_F(R_1 - R_2)=s_F(R_1)-s_F(R_2)$
11. Перестановка проекции с декартовым произведением: $p_{A1,A2,\ldots,Am}(R_1 \times R_2)= (p_{B1,B2,\ldots,Bn}(R_1))\times(p_{C1,C2,\ldots,Cr}(R_2))$
12. Перестановка селекции с пересечением: $s_F(R_1 \cup R_2)=s_F(R_1) \cup s_F(R_2)$

Порядок выполнения операторов

- Операторы выполняются слева направо – идет запрос к данным.

Виды соединения таблиц

- **Соединение вложенных циклов** Nested Loops
  - Сложность: O(NlogM)
  - Используется, если хотя бы одна таблица достаточно маленькая
  - Бо’льшая таблица имеет индекс по ключу соденининия
- **Соединение слиянием** Merge Join
  - Сложность : O(N+M)
  - Обе таблицы отсортированы по столбцу слияния
  - Слияние происходит по равенству
  - Хорошо для больших таблиц
- **Хэш-соединение** Hash Match
  - Сложность : $O(N*h_c+M*h_m+J)$
  - Используется в крайнем случае
  - Использует таблицу хэширования и динамическую хэш-функцию к строкам

Выбор процедурного плана

- Для выбора альтернативных процедурных планов выполнения запроса в соответствии с его внутренним представлением, оптимизатор использует информацию из словаря-справочника данных о существующих путях доступа к данным. Путь доступа, который возможен в любом случае, – это последовательное чтение (FULL SCAN). Возможность использования других путей доступа зависит от способов размещения данных в памяти (например, кластеризация или хеширование данных), от наличия индексов и формулировки самого запроса.

два основных вида оптимизаторов.

- Оптимизатор, основанный на анализе заданных правил (rule-based optimizer).
- Оптимизатор, основанный на анализе затрат (cost-based optimizer).

rule-based optimizer (Oracle)

- Этот оптимизатор выбирает методы доступа на основе предположения о статичности СУБД
- Такой оптимизатор учитывает иерархическое старшинство операций.
- Если для какой-либо операции существует более одного пути ее выполнения, то выбирается тот путь, чей ранг выше, т.к. в большинстве случаев он выполняется быстрее, чем путь с более низким рангом.
- План выполнения запроса формируется из выбранных путей доступа с максимальными рангами.

Ранжирование методов доступа в Oracle

Ра	Метод доступа
1	Одна строка по ее идентификатору
2	Одна строка по объединению кластеров
3	Одна строка по хэш-ключу кластера с уникальным или первичным ключом
4	Одна строка по уникальному или первичному ключу
5	Объединение кластеров
6	Кэш-ключ кластера
7	Индекс кластера
8	Составной индекс
9	Индекс на основе одного столбца
10	Ограниченный диапазон поиска по индексированным столбцам
11	Неограниченный диапазон поиска по индексированным столбцам
12	Объединение с сортировкой и слиянием
13	Поиск максимального или минимального значения по индексированным столбцам
14	Упорядочение по индексированным столбцам
15	Полное сканирование таблицы

Стоимость выполнения

- Стоимость (затраты)– это оценка ожидаемого времени выполнения запроса с использованием конкретного плана выполнения.
- Оптимизатор может учитывать количество необходимых ресурсов памяти, стоимость операций ввода-вывода, времени процессора и оперативной памяти, необходимой для выполнения плана.

Оценка стоимости

- Для каждого из выбранных планов оценивается предполагаемая стоимость выполнения запроса по этому плану.
- При оценках используется либо доступная оптимизатору статистическая информация о распределении данных, либо информация о механизмах реализации путей доступа. Из альтернативных планов выбирается наиболее оптимальный с точки зрения некоторого (заранее выбранного или заданного) критерия.

Оптимизация выполнения запроса осуществляется в следующем порядке:

1. Вычисление выражений и условий, содержащих константы.
2. Преобразование сложной команды в эквивалентную ей с использованием соединения (проводится не всегда).
3. Если команда выполняется над представлением, то оптимизатор обычно объединяет запрос на создание представления и запрос к этому представлению в одну команду.
4. Выбор метода оптимизации.
5. Выбор путей доступа к таблицам, к которым обращается запрос.
6. Выбор порядка соединения (если в запросе соединяются несколько таблиц, то оптимизатор определяет, какие две таблицы будут соединяться первыми, какая таблица следующей будет подключаться в результату и т. д.).
7. Выбор операции соединения для каждой команды соединения.

Статистика – основа для оценки стоимости

- Во время выполнения запросы оптимизатор пытается найти наиболее оптимальный план (наиболее, но не самый!)
- Стоимость плана выполнения запроса определяется на основании сведений о распределении данных в таблицах, к которым обращается команда, и связанных с ними кластеров и индексов. Эти сведения о распределении значений данных называются статистикой и хранятся в словаре-справочнике данных.

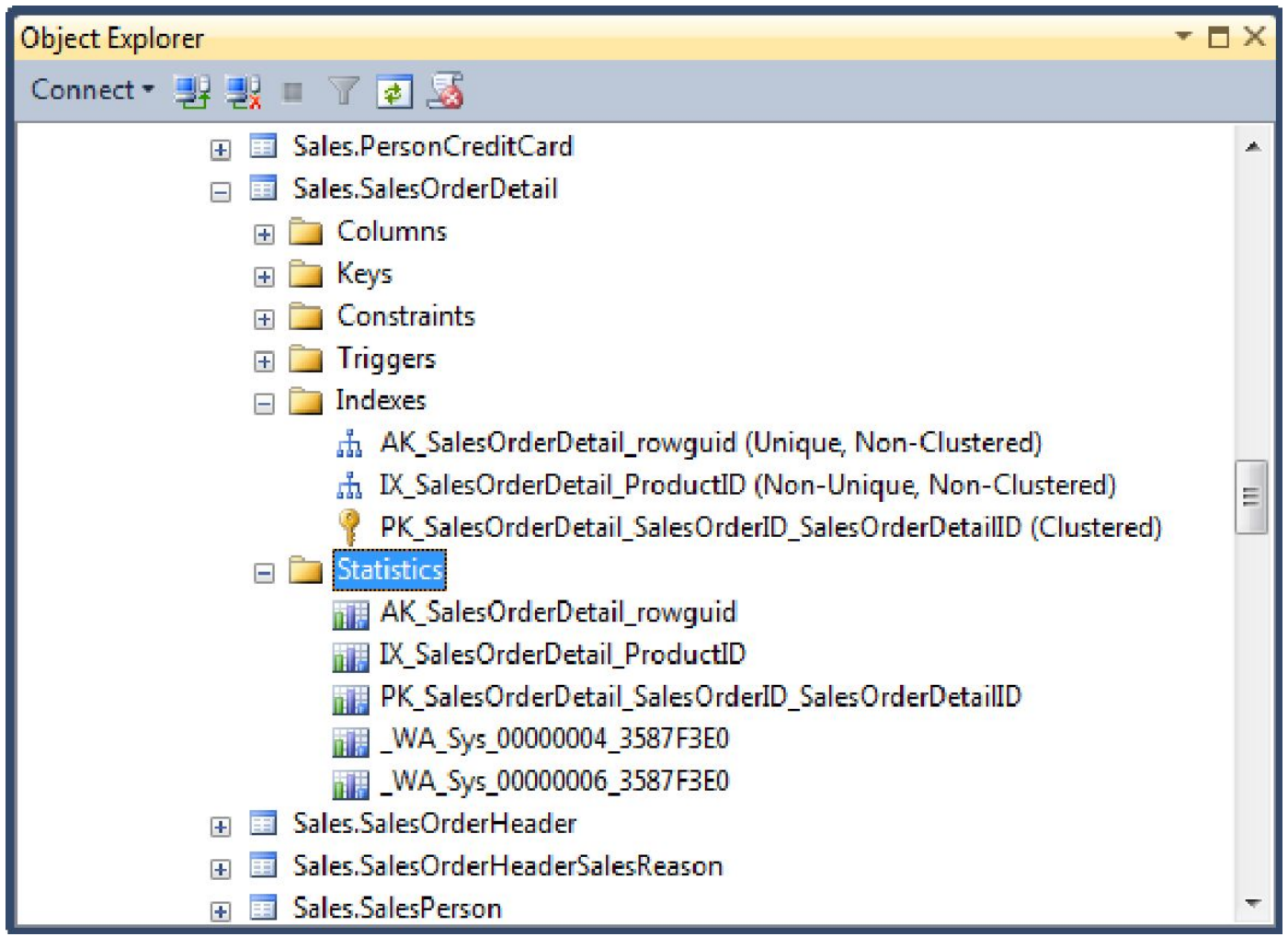
Статистика по таблице

- общее количество блоков данных (страниц памяти), выделенных таблице;
- количество пустых блоков данных (страниц памяти);
- количество записей в таблице;
- среднюю длину записи в таблице;
- среднее количество записей на блок (страницу) памяти;
- какие индексы построены для таблицы.

Статистика по индексам

- общее количество проиндексированных записей (оно может быть меньше, чем количество записей в таблице);
- минимальное и максимальное индексированные значения;
- количество различных индексированных значений.
- Распределение значений в столбце может быть отражено с помощью гистограммы, которая также входит в статистику.

Как увидеть статистику?



Статистика

- Статистика влияет на метод выборки данных:
- Сканирование таблицы
- Поиск по индексу

- Перед выполнением запроса оптимизатор пытается построить статистику, если ее до этого не было

Опции базы данных по статистике

- Создание статистики
- Обновление статистики
- Удаление статистики

- Create statistics

FULLSCAN, SAMPLE number (present|rows)

- Update statistics

FULLSCAN, SAMPLE number (present|rows)

- Drop statistics

Пример оптимизации на уровне выражений

- 1. SELECT ИД FROM ПРОДАВЦЫ WHERE ДОЛЖНОСТЬ='МЕНЕДЖЕР'
- 2. SELECT ИД FROM ПРОДАВЦЫ WHERE ДОЛЖНОСТЬ='ПРОДАВЕЦ‘
- Для торговой организации с 10 менеджерами, 1000 продавцов и общим числом сотрудников — около 6000

Индекс – использовать или нет?

- при использовании оптимизации, основанной на анализе затрат, знание некоторых характеристик распределения данных (например, того, что строки с данными о менеджерах составляют 1/600 часть всех строк) позволяет применять неуникальный индекс для запроса 1.
- Однако для выполнения запроса 2 будет уместно и эффективно полное сканирование таблицы.

Полное сканирование или индекс?

- При необходимости доступа к значительной части строк какой-либо таблицы полное сканирование является более эффективным, чем индексное.
- Дело в том, что для сканирования индекса и извлечения строки требуются, по крайней мере, две операции чтения для каждой строки, а в некоторых случаях и больше — в зависимости от количества уникальных данных в индексе.
- А при полном сканировании таблицы для извлечения строки требуется только одна операция чтения.
- При доступе к большому количеству строк — как, например, в запросе 2 — становится очевидной неэффективность использования индекса по сравнению с полным сканированием таблицы, при котором строки считываются непосредственно из таблицы.

Процедурное представление плана

- Выполняемое представление плана хранится в процедурном кэше. Процедура «с плохими параметрами».
- На последнем этапе обработки запроса происходит его реальное выполнение в соответствии с процедурным планом выполнения запроса. Результат помещается в специальную область ОП (курсор).

Оптимизация приложений

- В ОП хранятся все результаты ранее выполненных запросов до тех пор, пока эта память не потребуется для записи результатов последующих запросов.
- Подготовленные к исполнению SQL-операторы обычно помещаются в разделяемую SQL-область.
- Перед началом выполнения запроса система проверяет, есть ли в этой области аналогичный запрос: если есть, то он отправляется на выполнение минуя стадию предварительной обработки (компиляции).
- Составляя запросы таким образом, чтобы они совпадали в уже имеющимися в SQL-области, можно исключить предобработку запроса, что является важным моментом оптимизации приложений.

Рекомендации по оптимизации

- Пишите так, чтобы помочь оптимизатору запросов.
- Можно сильно затруднить работу оптимизатора, написав запрос, который ему «тяжело» будет разбирать, например, содержащий вложенные представления – когда одно представление получает данные из другого, а то из третьего – и так далее.

1. Раздел WHERE является критическим.

- Для следующих примеров раздела WHERE индексный путь доступа не будет использоваться, даже если индекс существует (COL1 и COL2 - столбцы одной таблицы, и создан индекс на COL1):
- COL1 > COL2
- COL1 < COL2
- COL1 >= COL2
- COL1 <= COL2
- COL1 IS NOT NULL
- COL1 NOT IN (value1, value2)
- COL1 != expression
- COL1 LIKE '%patern'
- NOT EXISTS subquery

1.1. Не использовать выражения от индексных столбцов

- Любые выражения, функции и вычисления, включающие индексированные столбцы, препятствуют использованию индекса.
- Например, в следующем примере наличие функции UPPER не дает возможность использовать сканирование по индексу, и будет применен полный просмотр таблицы:
- SELECT DEPT_NAME FROM DEPARTMENT WHERE UPPER(DEPT_NAME) like 'SALES%');

2. Для фильтрации записей используйте WHERE, а не HAVING.

Если для таблицы EMP существует индекс на столбце DEPTID, в при выполнении следующего запроса этот индекс использоваться не будет:

```
SELECT DEPTID, SUM(SALARY)
FROM EMP
GROUP BY DEPTID
HAVING DEPTID = 100;
```

Однако этот запрос можно переписать так, чтобы индекс применялся:

```
SELECT DEPTID, SUM(SALARY)
FROM EMP
WHERE DEPTID = 100
GROUP BY DEPTID;
```

3. Указывайте в разделе WHERE начальные столбцы ключа индекса.

- Для следующего запроса может быть применен составной индекс на столбцах PART_NUM и PRODUCT_ID, образованный в связи с ограничением первичного ключа:

```
SELECT * FROM  PARTS
WHERE PART_NUM = 100;
```

- то время как в приводимом ниже запросе составной индекс использоваться не может:

```
SELECT * FROM PARTS
WHERE PRODUCT_ID = 5555;
```

Как заставить использовать индекс?

- Последний запрос можно переписать так, чтобы индекс можно было применить. В этом запросе предполагается, что столбец PART_NUM будет всегда содержать положительные значения:

```
SELECT * FROM PARTS
WHERE PART_NUM > 0
AND PRODUCT_ID = 5555;
```

4. Сравните сканирование через индекс с полным просмотром таблицы.

- При выборе из таблицы более 15 процентов строк полный просмотр таблицы обычно выполняется быстрее, чем сканирование через индекс.
- Когда использование индекса приносит больше вреда, чем пользы, можно применять методы, чтобы воспрепятствовать использованию индекса.

- SELECT * FROM EMP
- WHERE SALARY+0 = 50000;

5. Используйте ORDER BY для индексного сканирования.

- Оптимизатор будет использовать индексное сканирование, если запрос содержит раздел ORDER BY с указанием индексированного столбца.
- Для выполнения следующего запроса будет использован индекс на столбце EMPID, даже если этот столбец не используется в условиях раздела WHERE.

- SELECT SALARY FROM EMP
- ORDER BY EMPID;

6. Минимизируйте число просмотров таблиц

- Таблица STUDENT содержит четыре столбца с именами NAME, STATUS, PARENT_INCOME и SELF_INCOME.
- Форма запроса предполагает два просмотра таблицы STUDENT, создание временной таблицы для последующей обработки и сортировку для устранения дубликатов:

```
SELECT NAME, PARENT_INCOME
FROM STUDENT WHERE STATUS = 1
UNION
SELECT NAME, SELF_INCOME
FROM STUDENT WHERE STATUS = 0;
```

- Тот же самый результат будет получен при выполнении запроса с одним просмотром таблицы:
- SELECT NAME, PARENT_INCOME * STATUS + SELF_INCOME * (1 - STATUS) FROM STUDENT;

7. Соединяйте таблицы в правильном порядке.

- Всегда следует выполнять сначала максимально ограничивающий поиск, чтобы отфильтровать как можно большее число строк на ранних фазах выполнения запроса с соединениями.
- Тогда на следующих фазах соединения оптимизатору придется иметь дело с меньшим числом строк, что повысит эффективность.
- Следует убедиться, что главная таблица (просматриваемая во внешнем цикле соединения на основе вложенных циклов) содержит наименьшее число строк.

8. При возможности используйте только поиск через индексы.

- Оптимизатор будет использовать только поиск в индексе, если вся информация, необходимая для выполнения запроса, содержится в самом индексе.
- Если для таблицы EMP существует составной индекс на столбцах LNAME и FNAME, то при выполнении следующего запроса будет использован только поиск в индексе:

  SELECT FNAME FROM EMP WHERE LNAME = 'SMITH';

- В то же время при выполнении запроса

  SELECT FNAME, SALARY FROM EMP WHERE LNAME = 'SMITH';

- будет производиться индексное сканирование таблицы с доступом к ее строкам по ROWID

9. Старайтесь писать как можно более простые («тупые») операторы SQL.

- Много мелких запросов в цикле лучше объединить в один большой.
- Если запрос имеет множество уровней вложенности, то внутреннию(ие) части надо вынести в отдельную выборку, заполнить временную таблицу, построить индексы, а потом использовать ее в основной выборке. Скорость работы будет значительно выше.

10. Варьируйте использование UNION или OR в зависимости от наличия индекса.

- Например, список пациентов палат №3 и 8 при наличии индекса должен быть таким:

```
select * from patients
where room=3
union all
select * from patients
where room=8;
```

- а если индекса нет, то таким:

```
select * from patients
where room=3 or room=8;
```

11. Если после слияния таблиц отбираются поля только из одной таблицы, то вместо операции join надо использовать операцию in

- Исходный запрос:

```
select emp.name
from emp, empjob
where emp.no = empjob.emp
and empjob.salary > 900;
```

- Оптимизированный запрос:

```
select name from emp
```

11. **IN, EXISTS или JOIN?**

- Если в основной выборке много строк, а в подзапросе мало, то лучше *IN,* т.к. в этом случае запрос в in выполнится один раз и сразу ограничит большую основную таблицу.
- Если в подзапросе сложный запрос, а в основной выборке относительно мало строк, то лучше *EXISTS*. В этом случае сложный запрос выполнится не так часто.
- Если и там и там сложно, надо использовать *JOIN*.

12. Группировка

- Если после группировки надо отсортировать результат, то желательно, чтобы поля сортировки и поля группировки перечислялись в одном порядке.

13. CTE не имеют индексов

- Значительно облегчает жизнь, если запрос в with необходимо использовать несколько раз в основной выборке или если число строк в подзапросе не значительно.
- В других случаях необходимо использовать прямые подзапросы в from или в заранее подготовленную таблицу с нужными индексами и данными из WITH.
  Причина плохой работы WITH в том, что при его джойне не используются ни какие индексы и если данных в нем много, то все встанет. Вторая причина в том, что оптимизатор не знает сколько данных нам вернет with и оптимизатор не может построить правильный план запроса.

14. Используем кеш

- В ОП хранятся все результаты ранее выполненных запросов до тех пор, пока эта память не потребуется для записи результатов последующих запросов.
- Подготовленные к исполнению SQL-операторы обычно помещаются в разделяемую SQL-область.
- Перед началом выполнения запроса система проверяет, есть ли в этой области аналогичный запрос: если есть, то он отправляется на выполнение минуя стадию предварительной обработки (компиляции).
- Составляя запросы таким образом, чтобы они совпадали в уже имеющимися в SQL-области, можно исключить предобработку запроса, что является важным моментом оптимизации приложений.

15. Избегайте неявного преобразования типов

- Использование неправильных типов данных
- Когда происходит неявное преобразование типа для столбца в выражении WHERE или же в условии соединения – сканирование таблицы (table scan).

Использование Multi-statement UDF

- *Multi-statement UDF в русской редакции msdn переводится примерно как «Функции, определяемые пользователем, состоящие из нескольких инструкций, но звучит это, на мой взгляд, как-то странно, поэтому в заголовке и дальше по тексту я старался избегать перевода этого термина — прим. переводчика*

  По сути, они загоняют вас в ловушку. На первый взгляд, этот чудесный механизм позволяет нам использовать T-SQL как настоящий язык программирования. Вы можете создавать эти функции и вызывать их одну из другой и код можно будет использовать повторно, не то что эти старые хранимые процедуры. Это восхитительно. До тех пор пока вы не попробуете запустить этот код на большом объеме данных.

  Проблема с этими функциями заключается в том, что они строятся на табличных переменных. Табличные переменные – это очень крутая штука, если вы используете их по назначению. У них есть одно явное отличие от временных таблиц – по ним не строится статистика. Это отличие может быть очень полезным, а может … убить вас. Если у вас нет статистики, оптимизатор предполагает, что любой запрос, выполняющийся к табличной переменной или UDF, возвратит всего одну строку. Одну (1) строку. Это хорошо, если они действительно возвращают несколько строк. Но, однажды они возвратят сотни или тысячи строк и вы решите соединить одну UDF с другой… Производительность упадет очень-очень быстро и очень-очень сильно.

Необоснованное использование хинтов в запросах

- Люди слишком поспешно принимают решение об использовании хинтов. Наиболее часто встречающаяся ситуация – это когда хинт помогает решить одну, очень редко встречающуюся проблему, на одном из запросов. Но, когда люди видят значительный прирост производительности на этом запросе … они немедленно начинают совать его вообще везде.

- Например, множество людей считает, что LOOP JOIN – это лучший способ соединения таблиц. Они приходят к такому выводу, поскольку он наиболее часто встречается в небольших и быстрых запросах. Поэтому они решают принудительно заставить SQL Server использовать именно LOOP JOIN. Это совсем не сложно:

```
SELECT  s.[Name] AS StoreName,
        p.LastName + ', ' + p.FirstName
FROM    Sales.Store AS s
        JOIN sales.SalesPerson AS sp
        ON s.SalesPersonID = sp.BusinessEntityID
        JOIN HumanResources.Employee AS e
        ON sp.BusinessEntityID = e.BusinessEntityID
        JOIN Person.Person AS p
        ON e.BusinessEntityID = p.BusinessEntityID
OPTION  (LOOP JOIN);
```

Необоснованное использование вложенных представлений

- Представления, ссылающиеся на представления, соединяющиеся с представлениями, ссылающимися на другие представления, соединяющиеся с представлениями… Представление – это всего лишь запрос. Но, поскольку с ними можно обращаться как с таблицами, люди могут начать думать о них как о таблицах. А зря. Что происходит, когда вы соединяете одно представление с другим, ссылающееся на третье представление и так далее? Вы всего лишь создаете чертовски сложный план выполнения запроса. Оптимизатор попробует упростить его. Он будет пробовать планы, в которых используются не все таблицы, но, время на работу по выбору плана ограничено и чем более сложный план он получит, тем меньше вероятность того, что в итоге у него получится достаточно простой план выполнения. И проблемы с производительностью будут практически неизбежны.

Улучшение оценки количества элементов для переменных и функций

- Если в предикате запроса используется локальная переменная, рекомендуется переписать запрос так, чтобы вместо локальной переменной в нем использовался параметр. Значение локальной переменной неизвестно в момент, когда оптимизатор запросов создает план выполнения запросов. Если в запросе используется параметр, то оптимизатор запросов использует оценку количества элементов для первого фактического значения параметра, передаваемого хранимой процедуре.
- Если хранимая процедура содержит запрос, в котором используется переданный параметр, не следует изменять значение параметра в рамках хранимой процедуры до того, как он будет использоваться в запросе. Оценка количества элементов для запроса основывается на значение переданного параметра, а не на обновленном значении. Чтобы исключить изменение значения параметра, можно переписать запрос так, чтобы использовать две хранимые процедуры.

Функции с табличным значением

- Для хранения результатов функции с табличным значением с несколькими инструкциями рекомендуется использовать стандартную или временную таблицу. Оптимизатор запросов не создает статистику для функций с табличным значением с несколькими инструкциями. Такой подход позволяет оптимизатору запросов создавать статистику по столбцам таблицы и использовать их для создания улучшенного плана запроса.

Используйте временные таблицы вместо табличных переменных

- Вместо табличных переменных рекомендуется использовать стандартную или временную таблицу. Оптимизатор запросов не создает статистику для табличных переменных.
- При выборе между временной таблицей и табличной переменной следует учитывать, что табличные переменные, используемые в хранимых процедурах, вызывают меньше перекомпиляций хранимой процедуры, чем временные таблицы. В зависимости от приложения использование временной таблицы вместо табличной переменной не обязательно приведет к повышению производительности.

План параметризованных процедур

- Создаем процедуру с параметром
  create procedure proc1 (@user_name varchar(10)) as …
- При первом вызове этой процедуры создастся план, актуальный для входного параметра, который и будет помещён в кэш
- При следующих вызовах не компилируется новый план, процедура берёт его из кэша, при этом он может быть не самым оптимальным для нового входного значения.

Пример распределения значений ключа

	RANGE_HI_KEY	RANGE_ROWS	EQ_ROWS	DISTINCT_RANGE_ROWS	AVG_RANGE_ROWS
1	User01	0	1000	0	1
2	User03	1	1	1	1
3	User05	1	1	1	1
4	User07	1	1	1	1
5	User09	1	1	1	1

Способы решения

- Постоянная перекомпиляция (RECOMPILE)
- Использование нескольких процедур
- Динамические запросы
- Опция OPTIMIZE FOR

Постоянная перекомпиляция (RECOMPILE)

- Создание хранимой процедуры с параметром WITH RECOMPILE в определении указывает, что SQL Server не будет кэшировать план этой процедуры.
- Параметр WITH RECOMPILE полезен в том случае, когда хранимая процедура принимает параметры, значения которых сильно меняются между выполнениями, что приводит к созданию каждый раз новых планов выполнения.
- Этот параметр используется редко и замедляет выполнение хранимых процедур, так как они должны перекомпилироваться при каждом запуске.
- create procedure dbo.p_user_activity_log (@user_name varchar(10)) with recompile as

Использование нескольких процедур

- Две процедуры:
  - для пользователя User01 (99% данных в таблице)
  - для всех остальных
- create procedure dbo.p_user_activity_log (@user_name varchar(10))
  as
  if @user_name = 'User01' exec proc1 @user_name
  else exec dproc2 @user_name
- Для 2-х процедур два разных плана в кэше, которые будут оптимальными для наших входных параметров.
- Но: трудно предсказать, как наши данные в исходных таблицах будут меняться с течением времени или сценариев >> двух.

Динамические запросы

- create procedure dbo.p_user_activity_log (@user_name varchar(10)) as
  begin
  declare @str nvarchar(max);
  set @str = 'select * from dbo.user_activity_log ' + 'where user_name = ' +
  quotename(@user_name, '''');
  exec sp_executesql @str;
  end;
- При каждом вызове процедуры в кэш будет помещаться план для конкретного запроса с конкретным параметром(или браться из кэша, если такой запрос там уже есть):

- exec dbo.p_user_activity_log @user_name = 'User01';
- exec dbo.p_user_activity_log @user_name = 'User02';

Опция OPTIMIZE FOR

- Подсказка OPTIMIZE FOR может использоваться для отмены определения параметров по первому вызову при создании структуры плана.
- Предположим, что как правило, процедура будет вызываться с параметром @user_name = 'User01' и мы создадим её с подсказкой OPTIMIZE FOR:
- create procedure dbo.p_user_activity_log (@user_name varchar(10)) as
  begin
  select * from dbo.user_activity_log
  where user_name = @user_name option (optimize for (@user_name = 'User01'));
  end;
- Вызовем эту процедуру с параметром @user_name = 'User02' – индекс не использован
- Благодаря подсказке optimize for, не смотря, на входной параметр, в кэш попал план, в котором оптимизатор использует сканирование всей таблицы, как нам и нужно.

Опция OPTIMIZE FOR UNKNOWN

- Наши данные регулярно меняются и мы не можем передать в качестве подсказки конкретное значение.
- В качестве решения этой проблемы мы можем использовать подсказку, - OPTIMIZE FOR UNKNOWN. Эта подсказка предписывает оптимизатору запросов использовать статистические данные вместо начальных значений для всех локальных переменных.
- create procedure dbo.p_user_activity_log (@user_name varchar(10)) as
  select * from dbo.user_activity_log
    where user_name = @user_name
  option (optimize for (@user_name unknown));

Улучшение оценки количества элементов с помощью указаний запросов

- Чтобы улучшить оценку количества элементов для локальных переменных, можно использовать указания запросов OPTIMIZE FOR и OPTIMIZE FOR UNKNOWN с параметром RECOMPILE.
- Для некоторых приложений повторная компиляция запроса при каждом выполнении может занять слишком продолжительное время. Указание запроса OPTIMIZE FOR может повысить производительность даже в случае, когда параметр RECOMPILE не используется.

Улучшение оценки количества элементов с помощью структур плана

- Для некоторых приложений рекомендации по конструированию запросов могут не действовать, поскольку запрос невозможно изменить или указание запроса RECOMPILE может вызвать слишком много повторных компиляций. С помощью структур плана можно задавать другие указания, такие как USE PLAN, чтобы управлять работой запроса, пока идет согласование изменений приложения с поставщиком приложения.