НАСТРОЙКА ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

Б. А. НОВИКОВ, Г. Р. ДОМБРОВСКАЯ

Функциональность и настройка системы

Модель приложения и схема базы данных: логическая структура данных и производительность, выбор структур хранения

Короткие запросы: использование индексов, выбор критериев селекции

Настройка больших запросов: управление полным просмотром

Транзакции и производительность

Параллельные системы: критерии качества, уровни параллелизма, размещение данных

Настройка сервера баз данных

bhv

Б. А. Новиков
Г. Р. Домбровская

НАСТРОЙКА ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

Рекомендовано УМО по образованию в области инновационных междисциплинарных образовательных программ в качестве учебного пособия для студентов вузов, обучающихся по специальности "Математическое обеспечение и администрирование информационных систем"

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06(075.8)
ББК 32.973.26-018.2я73
Н73

Новиков Б. А., Домбровская Г. Р.

Н73 Настройка приложений баз данных. — СПб.: БХВ-Петербург, 2006. — 240 с.: ил.

ISBN 5-94157-840-7

Описываются методы настройки баз данных и приложений, применяемые при создании высокоэффективных систем. Показывается, как решения, принимаемые на различных этапах создания системы, начиная с определения требований и выбора архитектуры, влияют на итоговый продукт. Детально обсуждаются методы и приемы, обеспечивающие получение высокой эффективности приложения и базы данных, разработки запросов, а также действия по настройке базы данных во время эксплуатации системы.

Для системных аналитиков, проектировщиков и разработчиков прикладных информационных систем, студентов старших курсов технических вузов

УДК 681.3.06(075.8)
ББК 32.973.26-018.2я73

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Татьяна Лапина*
Зав. редакцией	*Григорий Добин*
Редактор	*Наталья Сержантова*
Компьютерная верстка	*Татьяны Олоновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Инны Тачиной*
Зав. производством	*Николай Тверских*

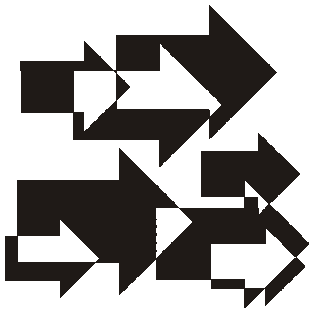
Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.01.06.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 19,35.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-840-7

Оглавление



Глава 3

Когда настраивать БД?

В данной главе мы уточняем, что такое настройка базы данных и какова ее роль в процессе разработки и эксплуатации системы. Основная цель этой главы — показать, что настройку необходимо предусматривать, начиная с самых ранних фаз проекта. Мы продемонстрируем, каким образом решения, принимаемые на различных этапах процесса разработки и эксплуатации системы, влияют на ее характеристики и ограничивают возможности последующей настройки.

Можно сравнить настройку системы с ее отладкой. Программный код, полученный в результате разработки проекта, может выполнять требуемые функции неправильно, и цель отладки состоит в том, чтобы добиться (по возможности) правильного выполнения функциональных требований к системе. Точно также цель настройки — приведение системы в соответствие с требованиями к производительности.

Одна из основных функций современных методологий, языков и инструментов разработки программ — предотвращение ошибок. Правильное использование этих средств существенно сокращает затраты времени и ресурсов на отладку и в то же время улучшает качество продукта. Точно также правильный учет требований к производительности на всех стадиях проектирования и разработки может значительно упростить настройку и в то же время сделать ее более результативной.

3.1. Функциональность и настройка системы

Любое хорошее руководство по разработке баз данных, а также документация по основным СУБД подчеркивает важность правильной настройки СУБД и программного приложения. Указывается, что для обеспечения эффективной работы системы необходимо выполнять определенные действия

по настройке, начиная с самых ранних фаз проектирования системы. Настройка в период эксплуатации, как правило, не может быть в такой же степени результативной.

Однако на практике довольно часто менеджер (не очень опытный) формулирует задачу специалисту по настройке примерно таким образом: "Система, которую мы сдали заказчику несколько месяцев назад, работает очень медленно, заказчики жалуются на недопустимо большие задержки при работе с базой данных. Пожалуйста, посмотрите, какие надо построить индексы". В этой формулировке содержится сразу несколько грубых ошибок:

- во-первых, скорее всего, необходимость настройки не была осознана при проектировании системы, и никаких действий по настройке во время разработки не выполнялось;
- во-вторых, проблема не была сформулирована конкретно;
- в-третьих, процесс настройки далеко не ограничивается созданием индексов. Инструменты, которыми пользуется специалист по настройке, многочисленны и разнообразны, и построение индексов далеко не всегда может дать необходимый результат.

Как объяснить такое несоответствие между рекомендациями учебников и документации по СУБД с одной стороны, и реальной практикой с другой?

Конечно, дело вовсе не в том, что квалификация менеджеров или разработчиков недостаточна. Можно утверждать, что довольно часто программисты недостаточно хорошо знакомы с тем, как следует использовать базы данных, но это обстоятельство не является определяющим.

Каждая программная система создается в рамках некоторого проекта, имеющего вполне конкретные сроки и ресурсы (более или менее точно определенные). Как руководители проекта, так и разработчики ограничены в своих действиях бюджетом времени и ресурсов проекта, а также другими условиями, и поэтому вынуждены принимать различные компромиссные решения.

Любой проект начинается с определения его целей. По отношению к проекту разработки программной системы мы можем условно разделить эти цели на две категории. Первую категорию составляют функциональные требования, т. е. описание того, что система должна делать. Ко второй категории мы относим требования к производительности системы. Чем сложнее требования (не важно, функциональные или к производительности), тем больше ресурсов необходимо для их реализации. Иначе говоря, создание высокопроизводительных систем требует значительно больших ресурсов, чем систем средней или малой производительности.

Хорошо известно, что обсуждать производительность системы, которая не выполняет необходимые функции, бессмысленно. Поэтому чаще всего основные усилия разработчиков в первую очередь направлены на реализацию

функциональности системы. Для проверки функциональности удобно использовать наборы данных небольшого размера. Выявить проблемы, связанные с производительностью, на таких наборах данных невозможно. Если участники проекта попадают под "пресс времени", то возникает опасность стихийного компромисса, сводящегося к полному игнорированию или частичному ослаблению требований к производительности.

Как правило, в любом проекте используются стандартные компоненты, блоки, типовые решения и т. п., которые существенно сокращают время разработки, но не всегда обеспечивают наивысшую производительность для проектируемой системы. Такие решения могут определяться специализацией или стандартами компании, в этом случае компромиссы принимаются еще до начала работы над проектом.

Мощность современной вычислительной техники такова, что во многих случаях даже не оптимально построенная система способна обеспечивать удовлетворительные характеристики производительности. В таких случаях настройка действительно совсем не нужна.

Тем не менее, представляется важным учитывать требования к производительности, начиная с самых ранних фаз проекта. Далее в этой главе мы обсуждаем, какие решения могут приниматься на различных фазах проекта и каким образом эти решения могут влиять на производительность системы и возможности ее настройки.

Существуют различные методологии проектирования и разработки программных систем. Иногда некоторые из обсуждаемых в этой главе фаз развития проекта могут не выделяться как отдельные этапы или, возможно, проектирование выполняется в другом порядке. Возможно, некоторые фазы сливаются вместе, все это не имеет для наших целей большого значения. Обсуждаемые здесь решения необходимо принимать в большинстве проектов приложений, использующих базы данных.

3.2. Определение и анализ требований к системе

Некоторые действия, связанные с процессом настройки, выполняются еще на фазе определения требований к системе. Эти действия состоят в том, что в состав требований включаются требования к характеристикам производительности. На последующих фазах проекта эти требования становятся основой для формулирования целей настройки.

Эти требования могут быть сформулированы с различной степенью точности и детализации, например:

- время ответа системы не должно превышать 5 с для 95% запросов при одновременной работе 200 пользователей;
- система должна обрабатывать не менее 50 000 запросов в сутки;

- сводные отчеты должны вырабатываться за разумное время;
- система должна быть высокоэффективной.

Конечно, неточно сформулированные требования значительно менее полезны, однако далеко не всегда на начальном этапе проекта могут быть сформулированы реалистические и достижимые требования с большой точностью.

Не менее важным для последующей настройки является и оценка размеров и других количественных характеристик данных, которые будут обрабатываться в системе. Очень важно максимально полно определить способы предполагаемого использования данных, а именно: какие запросы предполагается выполнять, с какой частотой, какие данные будут с наибольшей вероятностью использоваться в комбинации, каким образом будет происходить ввод данных, предполагаемая скорость роста объемов данных, обновляемость и т. п. Все это невозможно определить абсолютно точно, но чем реалистичнее будут сделаны предварительные оценки, тем эффективнее будет последующая настройка.

Такие характеристики, как примерное количество объектов каждого типа, количество и суммарный размер атрибутов, количество различных значений атрибутов, частота обновления и другие, могут быть очень важны для правильного проектирования структуры базы данных, проектирования запросов и создания возможностей для настройки системы во время эксплуатации.

Оценки ожидаемой скорости роста объемов данных и нагрузки на систему очень важны для прогнозирования того, насколько важна масштабируемость системы.

Конечно, степень детализации этих требований зависит от размеров и сложности проекта, однако даже решение о том, что никакая настройка разрабатываемой системы никогда не потребуется, и все требования по производительности будут удовлетворены автоматически, должно приниматься явным образом, с учетом степени риска.

3.3. Высокоуровневое проектирование

3.3.1. Выбор типа архитектуры

Практически для всех систем (кроме уникальных по размерам или сложности) задача выбора архитектуры очень проста, поскольку фактически надо выбирать один из двух вариантов: использование встроенной СУБД или сочетание многоуровневой архитектуры с архитектурой типа "клиент-сервер". Тем не менее, этот выбор, очевидно, очень существенно определяет не только возможную функциональность системы, но и в значительно большей мере — характеристики производительности и возможности настройки.

Фактически в большинстве случаев выбор навязывается не особенностями требований приложения, а возможностями и опытом коллектива разработчиков. Это не обязательно означает, что коллектив "подгоняет" задачу под свои возможности, скорее разработчики берутся за реализацию таких задач, которые они лучше умеют решать.

Приложения, построенные на основе встроенных СУБД, обладают рядом привлекательных свойств, в первую очередь — относительно низкой стоимостью. Тем не менее, выбирая такую архитектуру, разработчик должен понимать, что система вряд ли будет обладать какими-либо свойствами масштабируемости.

Альтернативой, как уже отмечено, является комбинация многослойной архитектуры с архитектурой "клиент-сервер". Несмотря на то, что обычно эти две архитектуры противопоставляются, практически в любой прикладной системе с развитой функциональностью приходится применять компоненты, работающие с базой данных непосредственно, минуя сервер приложений. Это вызвано тем, что многослойная архитектура, в особенности в случае использования протоколов без состояний, накладывает довольно серьезные ограничения на класс запросов, который может выполняться эффективно. В частности, это относится к функциям, выполнение которых требует значительного времени (десятки минут или часы), например, получение отчетов на основании данных большого объема.

Выбор конкретной СУБД имеет, с технической точки зрения, существенно меньшее значение. Конечно, СУБД различных производителей очень значительно отличаются по набору предоставляемых средств, организации структур хранения данных, производительности и по многим другим характеристикам, однако принципы и основные методы настройки остаются примерно одинаковыми для всех больших промышленных СУБД (разумеется, для реализации этих принципов и методов надо использовать разные средства, зависящие от конкретной СУБД).

Поскольку многие коллективы разработчиков специализируются на использовании СУБД какого-либо одного производителя, можно сказать, что выбор СУБД в большей мере определяется организационными и коммерческими причинами, чем техническими характеристиками систем.

3.3.2. Распределение функций в системе

После того как выбран тип архитектуры, который будет использован для создания приложения, необходимо определить, каким образом выполнение функции системы будет распределено между уровнями и компонентами.

Сложность этого этапа в том, что принятые решения практически невозможно изменить на более поздних фазах выполнения проекта. Поскольку никаких объективных оценок производительности на этой фазе еще нет,

решения должны приниматься исключительно на основе таких субъективных факторов, как предпочтения и опыт команды разработчиков, а также доминирующих типовых решений (продвигаемых на рынок производителями базовых программных продуктов). Как и на других ранних фазах, решение может быть навязано внешними факторами, и тогда у разработчиков просто нет выбора.

Иногда рекламная информация о продуктах принимается за объективную.

Наиболее широко распространенное типовое распределение функций может быть кратко охарактеризовано формулировкой: "Интерфейс на клиенте, логика на сервере приложений, таблицы в базе данных". Основное достоинство этой стратегии — хорошо разработанные возможности взаимодействия с объектными методологиями разработки программного обеспечения. Однако бездумное и слишком прямолинейное применение может приводить к нежелательным эффектам. Прежде всего, логическая структура базы данных полностью подчиняется особенностям кода приложения, часто игнорируются такие возможности базы данных, как проверка ограничений целостности, что приводит к снижению качества приложения в целом.

Довольно часто приложения, построенные таким способом, используют возможности СУБД крайне неэффективно. Мы подробнее обсуждаем эту проблему в *главе 4*. Можно сказать, что во многих случаях системы обеспечивают удовлетворительные характеристики производительности только вследствие избыточной мощности оборудования.

Альтернативой является архитектура, в которой значительная часть функций приложения реализована на сервере базы данных с использованием объектных возможностей современных СУБД (хранимых процедур, функций, пакетов, объектов, типов и т. п.). Достоинства таких архитектур — высокая степень логической целостности данных, идентичность представления данных для всех приложений и возможность высокоэффективной реализации. Основной недостаток — требуется более высокая квалификация команды разработчиков и необходимость более тщательной разработки структуры базы данных с учетом потребностей всех приложений, а не только интерактивных. (Как мы отмечали, обычно любая достаточно большая система включает компоненты, работающие по схеме "клиент-сервер", даже если основные функции реализованы в многоуровневой архитектуре.)

В действительности охарактеризованные выше альтернативные решения можно рассматривать как крайние решения. Фактическая реализация почти всегда занимает некоторое промежуточное положение между этими крайностями. Например, возможно использование высокоэффективных средств обработки запросов, предоставляемых СУБД, даже если хранимые процедуры и другие объектные возможности СУБД не используются. И, конечно, неадекватный стиль программирования может сделать крайне неэффективными не только

компоненты, выполняемые на сервере приложений, но и хранимые объекты в базе данных.

Иногда довольно простые на вид требования к системе приводят к существенному усложнению архитектуры. Например, требование обеспечить возможность автономной работы клиентов почти неизбежно приводит к необходимости реализовать распределенную неоднородную базу данных, сочетающую многоуровневую архитектуру на сервере и встроенные СУБД (или их заменители) на клиентах.

3.4. Детальное проектирование и разработка

Результатом детального проектирования является модель системы, формально описывающая ее структуру и выполняемые функции. Существуют различные методы, языки и методологии для построения таких моделей, обсуждение этих инструментов выходит за рамки этой книги.

Для нас, тем не менее, важно то, что разные участники процесса проектирования (или разные роли участников) могут использовать разные языки и средства для представления этой модели.

Например, разработчик программного кода приложения может использовать язык моделирования UML (Unified Modeling Language, унифицированный язык моделирования), а результатом детального проектирования может в этом случае быть диаграмма классов. В то же время для разработчика схемы базы данных более естественно строить модель в терминах диаграмм "сущность-связь".

Каждая из этих моделей отражает некоторые важные свойства системы, которые чаще всего невозможно учесть в рамках другой модели. Даже в тех случаях, когда одна из моделей генерируется на основе другой (например, диаграмма "сущность-связь" генерируется на основе диаграммы классов), обычно необходима доработка полученной модели.

Согласование моделей разного типа является одной из самых сложных задач на фазе детального проектирования. Участники проектирования должны быть готовы к поиску и принятию компромиссных решений.

Рассмотрим более детально различные задачи, решаемые на фазе проектирования.

3.4.1. Проектирование логической структуры базы данных

Общеизвестно, что логическая модель данных должна определять представление в базе данных сущностей, взаимосвязей между ними, ограничений целостности и, может быть, еще каких-то свойств данных таким образом,

чтобы обеспечить их корректность и соответствие функциональным требованиям к системе.

Наша задача — показать, что ошибки в логической структуре базы данных могут приводить не только к некорректностям в поведении системы и усложнению кода приложений, но и вызывать очень существенное ухудшение характеристик производительности.

В особенности сильное влияние на производительность могут иметь:

- выбор между естественными и суррогатными первичными ключами для сущностей;
- выбор составных или суррогатных ключей для связей;
- выбор между статическими и динамическими атрибутами;
- ограничения целостности по ссылкам (foreign key) и уникальности;
- нормализация хранимых таблиц.

Более подробно эти вопросы обсуждаются в *главе 5*.

3.4.2. Разработка приложения и проектирование запросов

Результатом разработки является программный код, реализующий требуемые функции создаваемой системы. По своему влиянию на характеристики производительности эта фаза является наиболее важной. Неправильное программирование может уничтожить любые удачные решения, принятые на предыдущих фазах проекта.

Для того чтобы построить эффективно работающее приложение, необходимо:

- правильно распределить работу между кодом приложения, написанным на языке программирования, и запросами, передаваемыми в базу данных на языке запросов СУБД (обычно SQL);
- подготовить запросы таким образом, чтобы они могли эффективно выполняться сервером базы данных.

Соотношение между языками программирования и языками запросов обсуждается в *главе 4*, а методы эффективного программирования запросов — в *главах 6* и *7*.

3.4.3. Проектирование структур хранения базы данных

Один из основных принципов технологии баз данных — независимость логической структуры данных и структуры хранения. Это позволяет изменять структуры хранения, не затрагивая ни код приложения, ни семантику выполнения запросов.

По этой причине проектирование структуры хранения является относительно изолированной задачей. Необходимость учитывать структуры хранения при проектировании запросов возникает в относительно редких случаях, например, при явном использовании материализованных представлений.

В процессе проектирования структуры хранения определяются:

- размещение данных на дисках (т. е. привязка объектов логической структуры, таких как таблицы, к табличным пространствам, файлам и т. п.);
- структуры хранения отдельных таблиц (неупорядоченные, кластеризованные и т. п.);
- атрибуты или группы атрибутов, для которых должны быть построены индексы;
- материализованные представления;
- другие структуры хранения, возможно, зависящие от СУБД.

Хотя проектирование физической структуры, безусловно, важно для достижения высокой производительности, мы не рассматриваем его детально. Вместо этого мы обсуждаем структуры хранения при освещении других тем, главным образом — проектирования запросов.

3.4.4. Тестирование

Цель тестирования — выявление несоответствий результата разработки тем требованиям, которые были установлены в начале проекта.

Если на фазе определения требований были указаны не только функциональные возможности системы, но и характеристики производительности, то, очевидно, при тестировании необходимо проверять и эти требования.

Наиболее важная особенность тестирования производительности (в отличие от функциональности) состоит в том, что для такого тестирования необходимы наборы данных реалистического размера и с близкими статистическими характеристиками. Создание таких тестовых наборов данных часто становится достаточно непростой и дорогостоящей задачей.

Сложность подготовки тестовых баз данных вызвана тем, что предсказать поведение пользователей будущей системы, а, следовательно, и статистические характеристики реальной смеси запросов оказывается практически невозможно.

По этой причине методы типа "стресс-тестов", возможно, полезные для выявления слабых мест в системе, чаще всего не помогают правильно настроить систему для работы в реальных условиях.

3.5. Эксплуатация системы

Как правило, любая успешно используемая информационная система (начиная с некоторого порога сложности и размеров базы данных) нуждается в настройке в период эксплуатации.

Мы уже говорили о том, что вопреки распространенному мнению, согласно которому настройка может быть необходима только в период эксплуатации системы, она должна начинаться с определения требований и продолжаться на всех этапах проектирования и разработки. Однако в данном разделе мы будем использовать слово "настройка" в ограниченном смысле, т. е. только на фазе эксплуатации готовой системы.

Необходимость в дополнительной настройке в процессе эксплуатации системы может быть вызвана, например, следующими факторами:

- изменение или уточнение требований к системе;
- изменение объема данных;
- изменение частоты сбора статистики;
- изменяющаяся частота выполнение запросов;
- неточности исходного планирования.

Часто необходимость настройки может возникать вследствие недостаточного внимания к требованиям по производительности в период разработки. Однако важно подчеркнуть, что меры, перечисленные в предыдущих разделах этой главы, не могут полностью исключить необходимость настройки в период эксплуатации.

Можно сказать, что тщательный учет требований к характеристикам производительности создает предпосылки для того, чтобы последующая настройка была результативной, и наоборот, никакая настройка не может компенсировать грубые ошибки, допущенные при разработке системы.

Можно рассматривать процесс настройки как отдельный проект (или подпроект), поэтому процесс настройки всегда должен начинаться с формулировки и анализа требований. Чем точнее сформулированы требования, тем лучше будут результаты настройки. Приведем несколько примеров требований.

- Необходимо, чтобы форумы были быстрее.
- Нас не устраивает время ответа 45 с, оно должно быть не больше 20 с.
- Поиск проектов по населенному пункту выполняется за 5—8 с, это приемлемое время ответа, но поиск по региону занимает 1.5—2 мин, это неприемлемо.
- Пользователи жалуются на неоправданные задержки при работе с системой, иногда до нескольких минут на операциях, которые обычно выполняются за 2—3 с.

Очевидно, требование первого типа нуждается в уточнении.

После того как требования определены, обычно необходим анализ того, что именно является причиной неудовлетворительной производительности. В современных системах со сложной многоуровневой архитектурой, в которых одновременно могут выполняться запросы десятков и более пользователей, найти узкие места обычно бывает достаточно сложно. Довольно часто причиной низкой производительности оказывается не какой-либо отдельный компонент системы, а их взаимодействие. Мы подробно рассмотрим примеры таких ситуаций в следующих главах, но уже сейчас можно отметить, что наиболее частой причиной неудовлетворительной производительности становится чрезмерное количество очень простых запросов к СУБД.

Поскольку система обычно состоит из разнородных компонентов, работающих в неоднородной среде, характеристики производительности, полученные с помощью средств профилирования и трассировки отдельных компонентов, могут давать результаты, уводящие от правильного решения. Например, тот факт, что программа на сервере приложений 99% времени ожидает результатов выполнения запросов в СУБД, не обязательно означает, что необходима настройка именно СУБД. С другой стороны, относительно высокая загрузка сервера приложений и низкая 2—3% загрузка процессора на сервере базы данных не обязательно означает, что база данных не требует настройки!

Для того чтобы правильно проанализировать причины низкой производительности, кроме реального и процессорного времени, необходимо оценивать загруженность систем ввода-вывода, загруженность сети, размеры очередей и т. п.

В некоторых случаях неудовлетворительное поведение системы не удается воспроизвести на тестовой копии и приходится выполнять измерения непосредственно на работающей системе. В подобной ситуации самым важным требованием к процессу измерений и анализа становится их неразрушающий характер: никакие действия во время анализа не должны мешать нормальной работе пользователей системы.

Тем не менее, основная сложность анализа состоит не в том, чтобы собрать необходимые данные, а в том, чтобы на основании этих данных выявить места, нуждающиеся в настройке, и доказать (на основе собранных данных), что узкие места выявлены правильно.

Другая задача, решаемая во время анализа системы, — определение того, какие значения характеристик производительности являются достижимыми при заданной конфигурации системы и нагрузке на нее. Иными словами, нужно убедиться в том, что требования к настройке реализуемы хотя бы теоретически.

В этой книге нас в основном интересуют ситуации, в которых узким местом является именно база данных, т. е. в результате анализа удалось выявить запросы или группы запросов, которые выполняются недопустимо долго.

После того как такие запросы выявлены, необходимо снова уточнить требования по настройке. Дело в том, что время ответа, которое видит пользователь системы, может быть значительно больше, чем время ответа СУБД. Необходимо также проанализировать технические причины неудовлетворительной производительности (узкие места): обмен данными с диском, процессорное время, слишком частые состояния ожидания, пропускная способность сети и т. п.

Можно выделить три класса методов воздействия на систему, позволяющих улучшить ее производительность.

- Настройка параметров СУБД, таких как размер буферного пула, число конкурентно выполняемых транзакций, количество процессов, параметры оптимизатора запросов и т. п.

  Изменения этого класса оказывают влияние на все без исключения запросы, выполняемые в СУБД, при этом улучшение характеристик для одного класса запросов может привести к ухудшению характеристик других классов.

- Изменения схемы хранения (создание или удаление индексов, материализованных представлений, кластеризации таблиц и т. п.).

  Влияние таких изменений более локально по сравнению с изменениями параметров системы: характеристики запросов, которые не используют модифицированные структуры данных, скорее всего, не изменятся.

- Модификация SQL-кода отдельных запросов, в том числе использование подсказок оптимизатору и эквивалентные преобразования кода запроса.

  Такие изменения всегда являются локальными, поскольку затрагивают только отдельные запросы.

При выполнении настройки необходимо соблюдать некоторые технологические требования, обеспечивающие воспроизводимость полученных результатов. Все изменения должны протоколироваться вместе с количественными характеристиками результата этих изменений. Необходимо проверять, что улучшения в одних частях системы не привели к ухудшению производительности других.

Наконец, настройка должна быть прекращена, когда либо установленные требования достигнуты, либо установлено, что на имеющейся конфигурации оборудования получить требуемые характеристики невозможно. (Например, получаемый объем результата не может быть передан за требуемое время при имеющейся пропускной способности сети.)