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*

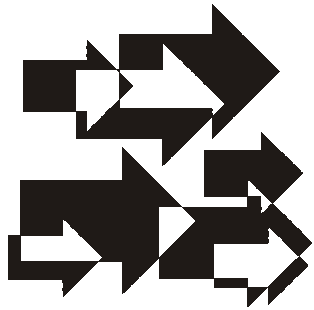
Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.01.06.
Формат 70×100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 19,35.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

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 1

Введение

Эффективное использование баз данных является исключительно важным фактором, существенно влияющим на качество прикладной информационной системы, поскольку практически любая такая система использует какую-либо систему управления базами данных для организации хранения информации.

Этот факт, очевидный для специалистов в области баз данных и тех, кто занимается сопровождением работающих систем, часто недооценивается разработчиками (как приложений, так и инструментальных программных средств).

1.1. Как возникают проблемы производительности?

Проблемы производительности возникают, прежде всего, потому, что проектирование и разработка сколько-нибудь сложных информационных систем всегда оказывается довольно трудоемким делом. Обычно проектировщикам требуется значительное время для того, чтобы освоить предметную область, собрать и проанализировать требования к системе и так далее в соответствии с принятой для проекта методологией проектирования и разработки. Точное выполнение функциональных требований занимает значительную часть всего времени, выделенного на проект, и поэтому требования по надежности и производительности часто рассматриваются как менее приоритетные.

Хорошо известно, что проекты такого типа часто затягиваются. Это может приводить к тому, что неизбежные компромиссы между качеством системы и временем, затрачиваемым на ее разработку, смещаются в пользу более быстро реализуемых решений.

Стремительный рост мощности вычислительной техники создает иллюзию неограниченных возможностей аппаратуры. Конечно, профессионалы, которым адресована эта книга, хорошо знают, что сложность алгоритмов не всегда линейная и поэтому далеко не всегда рост объемов данных может быть компенсирован увеличением производительности аппаратуры, но этот факт обычно трудно учитывать, когда приближаются сроки сдачи проекта. Более того, в начальный период эксплуатации системы объемы данных, скорее всего, невелики и поэтому неэффективность реализации никак не проявляется.

Одна из основных причин, по которым производительность системы может оказаться неудовлетворительной, связана с тем, что современные языки программирования используют способ представления алгоритмов, принципиально отличающийся от того, на котором основаны языки запросов баз данных. Эта проблема, известная под метафорическим названием "несоответствие импеданса" (impedance mismatch), обсуждается в *главе 4*. Разработчики могут недооценивать или просто быть не в состоянии учесть сложности, вызванные этим несоответствием.

Наряду с этими субъективными факторами имеются и объективные.

Технологии баз данных развиты в такой степени, что для небольших по размеру и сложности баз данных удовлетворительная производительность достигается автоматически, без каких-либо дополнительных усилий со стороны разработчиков, в особенности при наличии избыточной мощности аппаратных вычислительных средств. Иначе говоря, даже не очень эффективное проектирование и реализация взаимодействия с базой данных не приводят к существенному ухудшению эксплуатационных характеристик информационной системы, и с ростом производительности вычислительных систем абсолютные размеры небольшой системы постепенно растут.

Более 15 лет назад ведущие исследователи и разработчики систем управления базами данных утверждали: "Программистам придется принять тот факт, что оптимизатор СУБД может запрограммировать и выполнить запрос лучше, чем они, точно также, как компилятор языка программирования вырабатывает лучший код" [12]. Время оптимизаторов высокого качества наступило: сегодня мы имеем возможность использовать СУБД, способные выполнять большинство запросов весьма эффективно и, безусловно, лучше чем это может сделать программист среднего класса.

Тем не менее, производительность прикладных информационных систем очень часто оказывается неудовлетворительной и, несмотря на все достижения технологий проектирования приложений и технологий систем управления базами данных, настройка, т. е. вмешательство с целью улучшения характеристик производительности зачастую оказывается необходимой.

1.2. Что такое настройка?

Мы называем настройкой прикладной системы совокупность разнообразных мер, направленных на приведение системы в соответствие с требованиями к производительности без изменения функциональности системы.

Как правило, термин "настройка" применяется по отношению к системам управления базами данных. Это вызвано следующими факторами:

- система управления базами данных всегда является одним из основных компонентов прикладной системы и обычно несет основную нагрузку по выполнению функций этой системы;
- управление сервером базы данных реализуется с помощью высокоуровневых средств, как правило, только косвенно влияющих на его поведение.

Мы используем термин "настройка" в более широком смысле, чем только настройка сервера базы данных, потому что во многих случаях получить удовлетворительные результаты, ограничиваясь только настройкой базы данных, не удается, и требуется внесение изменений или учет требований производительности на уровне различных компонентов системы.

Методы настройки можно условно разделить на два класса: глобальные и локальные. Цель глобальной настройки — улучшение эксплуатационных характеристик системы в целом, в то время как целью локальной настройки является улучшение характеристик отдельных запросов или классов запросов. При этом важно отметить, что в том и другом случае эффект от настройки достигается небольшими (локальными) изменениями. Переписывание всей системы заново не может считаться настройкой как по смыслу самого слова "настройка", так и потому, что полное переписывание, скорее всего, не приведет к достижению целей настройки.

Никакие действия по настройке не должны изменять функциональные свойства или надежность настраиваемой системы. Например, если для ускорения работы системы разработчики отказываются от создания резервных копий или ведения журналов, то такое действие не может считаться элементом настройки, потому что оно снижает надежность системы. Если система оказывается не в состоянии обеспечить приемлемое время ответа при немедленном обновлении и поэтому в системе применяется отложенное обновление, то это также не может считаться элементом настройки, потому что в этом случае пользователь может видеть устаревшие данные, что является изменением функциональности.

Сказанное не означает, конечно, что подобные решения не должны применяться. Компромиссы при создании любой системы неизбежны. Функциональность системы должна быть сбалансирована с возможностями аппаратных средств, а требования к производительности должны быть

реализуемыми, при этом функциональность неизбежно ограничивается. Однако изменение функциональности не может рассматриваться как элемент настройки системы.

По причинам, упомянутым выше, настройка обычно необходима только для достаточно больших или сложных систем. Более того, во многих случаях значительная часть запросов выполняется достаточно эффективно в результате работы оптимизатора запросов СУБД и поэтому тоже не требует каких-либо усилий по настройке. Иначе говоря, как правило, требуется локальная настройка, а не глобальная.

Приведенное выше определение предполагает, что некоторые действия по настройке должны выполняться, начиная с самых ранних фаз создания системы и далее на всех этапах ее проектирования и разработки. Эта точка зрения не совпадает с широко распространенным представлением о том, что настройка может выполняться только в период эксплуатации системы, и отражает идеальную картину, а не реальность. Часто бывает трудно заранее убедить заказчика в том, что настройка является частью нормального процесса создания прикладной системы. Изменить это положение вряд ли возможно, однако в *главе 3* обсуждаются действия по настройке системы на различных фазах проектирования и разработки.

Конечно, необходимость настройки в период эксплуатации готовой системы совсем не обязательно является признаком низкого качества или ошибок при ее создании. Необходимость настройки может возникать и по причинам, которые не могли быть известны до начала эксплуатации.

Хотя мы не ограничиваем настройку только уровнем базы данных, центральное место в этой книге занимают приемы локальной настройки серверов баз данных и отдельных запросов. Авторам приходилось применять подобные методы для настройки систем, уже находящихся в эксплуатации, однако во многих случаях знание этих приемов может существенно улучшить характеристики системы еще на фазе разработки.

1.3. Чем может помочь эта книга?

Несмотря на то, что количество публикаций по системам управления базами данных огромно, вопросы настройки обсуждаются очень редко и недостаточно полно. Возможно, одной из причин этого является то, что работа по настройке баз данных весьма трудоемка и требует очень высокой квалификации. Кроме этого, довольно часто необходимость настройки может быть выявлена только в период эксплуатации системы, когда база данных выросла до достаточно больших объемов. Дорогостоящая настройка системы в этот период плохо вписывается в стандартные жизненные циклы разработки.

Среди немногих публикаций, уделяющих серьезное внимание настройке систем управления базами данных, необходимо выделить книгу [11], в которой на большом экспериментальном материале анализируются и проверяются различные методы настройки, в основном глобальные, хотя и достигаемые локальными изменениями конфигурации системы.

Большинство руководств по системам управления базами данных также уделяет основное внимание глобальной настройке. Возможно, причина этого состоит в том, что для настройки сервера базы данных не требуется внесения каких-либо изменений в программный код приложения. Возможности такой настройки, однако, ограничены и обычно их воздействие недостаточно локально. Поэтому необходимость изменения кода запросов возникает достаточно часто, а иногда появляется необходимость и в изменении программного кода приложения.

Как и в любой быстро развивающейся области знаний и технологий, в программировании вообще, и в настройке баз данных в частности, существует огромное количество мифов, т. е. распространенных мнений, принимаемых большинством разработчиков на веру, без какого-либо анализа их справедливости и применимости в конкретных условиях. Довольно часто носители подобных мифов склонны защищать их с религиозным фанатизмом. Некоторые вопросы вызывали, как известно, даже "религиозные войны", к счастью, без применения оружия.

Механизмы возникновения подобных мифов достаточно просты: ярко сформулированные идеи легко запоминаются и передаются, однако при передаче утрачивается контекст, в котором эти идеи были первоначально предложены, обоснованы и привели к получению хороших результатов. После многократной передачи и фольклорных изменений восстановить исходный контекст без кропотливого исследования литературы уже не представляется возможным. Утраченное понимание технических аргументов подменяется ссылками на авторитет рынка. В результате хорошая идея превращается в миф, истинность которого в отрыве от контекста оказывается весьма сомнительной.

Довольно часто мифы попадают в книги, и, таким образом, вера в них поддерживается априорным авторитетом печатного слова. Одним из примеров таких изданий является книга [14], представляющая методологию настройки, основанную почти исключительно на мифах, большая часть которых неверна или неприменима в контексте современных высокопроизводительных СУБД. Несмотря на это, книга [14], безусловно, понравится читателю, который надеется освоить материал, не затрачивая много времени на его изучение и глубокий анализ.

Авторы надеются, что приведенные в этой книге рекомендации не станут основой для новых мифов. При обсуждении приемов настройки мы показываем, почему эти приемы могут оказаться полезными, и в каком контексте

они хорошо себя показали в нашей практике. Это, безусловно, не означает, что те же приемы окажутся полезными в других ситуациях. Весьма вероятно, что в иных контекстах понадобятся другие приемы. Любой миф превращается в точное знание, если известны условия его применимости, обоснование и способ проверки.

В отличие от [14], данная книга ориентирована на подготовленного читателя. Она не является руководством по использованию баз данных. В основном мы обсуждаем вопросы настройки, хотя по необходимости затрагиваем и некоторые другие темы, в частности, проектирование схем баз данных.

Некоторые из необходимых предварительных сведений в сжатом виде изложены в *главе 2*. Материал этой главы уточняет определения основных понятий и терминологию, используемую в данной книге, но ни в коем случае не может заменить изучение серьезного руководства по системам управления базами данных. Читателю, не знакомому с этими темами, мы рекомендуем прочитать, как минимум, любые две книги из [17, 18, 21]. Каждая из этих книг содержит достаточно серьезное изложение материала, однако взгляды авторов на многие вопросы различаются, и для того, чтобы не оказаться в плену религиозного фанатизма, целесообразно ознакомиться и с альтернативными точками зрения.

Материал *главы 3* предполагает некоторое знакомство с процессом создания программных систем. Полезно, но не обязательно знакомство с какой-либо конкретной методологией и соответствующими инструментальными средствами и компонентами.

Для понимания материала *главы 4* необходима готовность читать тексты, записанные на (вообще говоря, незнакомом) языке программирования, и желательно практическое знание какого-либо из распространенных языков программирования.

Для детального понимания материала *глав 6* и *7* необходимо знание языка запросов SQL. Кроме уже упомянутых книг по системам управления базами данных для изучения SQL можно использовать [19].

Многие черты современных систем управления базами данных являются воплощением идей, обсуждавшихся исследователями в 80-х годах прошлого века и в концентрированном виде представленных в работе [12], которую мы уже упоминали, и [1]. В первую очередь это относится к объектным расширениям модели данных.

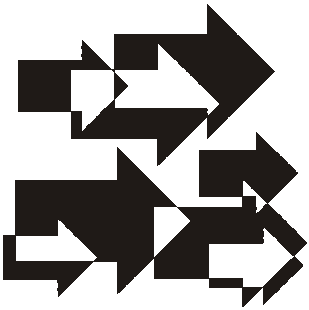
В реальности, однако, эти расширения используются относительно редко, поэтому мы также ограничиваемся в основном обсуждением реляционных средств языка SQL. Мы не затрагиваем также возможности СУБД, связанные с обработкой структур XML, потому что их использование пока не стало общепринятой практикой.

Примеры, включенные в данную книгу, подготовлены на основе задач, возникавших при настройке реальных прикладных систем, в которых применялись различные версии нескольких коммерческих СУБД, включая как наиболее высокопроизводительные, так и небольшие системы.

Во многих случаях в процессе настройки приходится учитывать особенности языка запросов и расширения, специфические для используемой системы управления базами данных. Несмотря на существование стандарта SQL, практически все производители предоставляют свои собственные диалекты. Даже в тех случаях, когда реализуется одна и та же версия стандарта, небольшие различия (например, в именах встроенных функций) позволяют легко определить, для какой именно системы написан запрос.

Авторы не могли, конечно, сделать изложение полностью независимым от специфики расширений. Тем не менее, читатель не найдет в данной книге рекомендаций по настройке каких-либо конкретных систем.

Принципы настройки мало зависят как от особенностей языка, так и от особенностей реализации конкретных систем управления базами данных, потому что для выполнения запросов во всех системах используются одни и те же алгоритмы, сложность которых не может зависеть от производителей СУБД. Вооружившись принципами и методами, изложенными в данной книге, читатель легко сможет найти необходимые инструменты и средства в документации по конкретной системе, с которой ему приходится работать.



Глава 2

Предварительные сведения

2.1. Основные понятия

В данном разделе кратко определяются основные понятия, связанные с применением баз данных, описываются главные функции баз данных и уточняется терминология, используемая в этой книге.

2.1.1. Базы данных и приложения

Прикладной системой будем называть совокупность всех программных компонентов и средств, совместно использующих некоторые данные или обеспечивающих их использование, а также выполняющих какие-либо полезные для потребителя функции. Система может считаться действительно прикладной, если ее пользователи, как правило, не являются профессиональными программистами (или проектировщиками программных систем).

В состав прикладной системы входит все, что нужно для того, чтобы она работала: операционная система (или системы), обязательно система управления базами данных (СУБД), возможно, Web-сервер и другие виды серверов, специализированные пакеты (например, средства анализа данных, генераторы отчетов), и, конечно, программный код, написанный непосредственно для данной прикладной системы. Мы не включаем в состав прикладной системы средства проектирования, разработки и тестирования, хотя без всего этого она работать не сможет.

Системой управления базами данных (СУБД) называется программный продукт, предназначенный для централизованного хранения данных. Функции систем управления базами данных обсуждаются более детально в *разд. 2.1.2*, сейчас важно отметить, что в состав прикладной системы всегда

включается готовая СУБД, т. е. СУБД никогда не входит в состав программного кода, написанного специально для конкретной прикладной системы.

Будем называть базой данных совокупность всех данных, доступ к которым в прикладной системе осуществляется через программные средства, предоставляемые системой управления базами данных. Возможно, в прикладной системе используются и файлы данных, не находящиеся под контролем СУБД, такие данные не будут больше упоминаться в данной книге.

Таким образом, база данных — это совокупность всех постоянно хранимых данных, используемых приложением, независимо от того, каким образом и на каком уровне абстракции они описаны: файлы операционной системы, в которых хранятся базы данных, хранимые данные (таблицы, кластеры, индексы и т. п.), логические структуры данных, в терминах которых происходит обработка (таблицы и представления), или концептуальная модель данных. В старых книгах писали, что база данных описывает всю вселенную для прикладной системы. В некоторых СУБД термин "база данных" используется в более узком смысле, обозначая крупную структурную единицу хранения данных.

Будем называть экземпляром сервера базы данных (или просто сервером базы данных) совокупность компонентов СУБД, находящихся в состоянии выполнения и способных принимать запросы на обработку данных. В некоторых СУБД понятие сервера базы данных обозначается термином instance.

Приложением базы данных, в отличие от прикладной системы, называется любой программный компонент, использующий услуги сервера базы данных, т. е. передающий запросы на сервер базы данных и получающий результаты их выполнения. Чаще всего в роли приложения выступает программный код, написанный специально для прикладной системы, однако с этой точки зрения приложениями являются, например, генераторы отчетов или другие специализированные пакеты для обработки данных. Иногда взаимодействие происходит опосредованно через промежуточные компоненты прикладной системы. Для сервера баз данных это не имеет значения: в роли приложения в этих случаях выступают промежуточные компоненты (например, сервер приложений). В данной книге мы будем называть приложения баз данных просто приложениями.

Фактически с сервером базы данных взаимодействует экземпляр приложения, находящийся в состоянии выполнения. Иногда нам нужно будет различать разные выполняемые экземпляры одного и того же кода приложения (например, запущенные для обслуживания нескольких пользователей) и экземпляры разных приложений (например, интерактивные средства и генератор отчетов).

Любое взаимодействие приложения с сервером базы данных происходит в рамках сеанса (session). С точки зрения сервера базы данных, приложение

начинает работу тогда, когда создает сеанс, регистрируя себя на сервере базы данных. После этого сервер может принимать и выполнять запросы на обработку данных от этого приложения. Работа приложения заканчивается, опять же с точки зрения сервера базы данных, когда приложение закрывает сеанс (или он прекращается по другим причинам). Обычно сервер баз данных может поддерживать несколько сеансов с одним экземпляром приложения, но при этом запросы, передаваемые в разных сеансах, с точки зрения сервера базы данных, приходят от разных приложений.

Заметим, что сеанс работы с сервером базы данных совсем не обязательно связан с сеансом работы пользователя прикладной системы. Эти понятия взаимосвязаны только для некоторых типов архитектур прикладной системы, которые мы коротко обсудим ниже в этой главе. В некоторых случаях сеанс работы пользователя может включать несколько сеансов сервера базы данных, или, наоборот, приложение в рамках одного сеанса может обслуживать несколько пользовательских сеансов последовательно или чередуя запросы от разных пользователей.

Вся работа, выполняемая приложением, для сервера базы данных представляет собой последовательность запросов. На самом деле запросы группируются в транзакции, но определения, связанные с понятием транзакции, мы рассмотрим отдельно в *главе 8*.

Запросы к серверу базы данных записываются на языке запросов СУБД (обычно это диалект SQL). Поскольку само приложение пишется на другом языке программирования, возникает необходимость совмещения текстов на разных языках. Существуют различные способы такого совмещения, которые можно классифицировать как встроенный, статический и динамический SQL.

Синтаксические различия могут быть очень значительными, однако по существу различие состоит в том, как соотносятся время компиляции приложения и время компиляции запроса, а также в том, каким образом запросы параметризуются. Хотя в современных интерфейсах СУБД способ передачи запросов выглядит обычно динамическим, однако сервер базы данных обрабатывает динамические запросы, преобразуя их в статические, поэтому различие между динамическим и статическим SQL не очень важно для настройки. (Однако важно, изменяется ли текст запроса при повторном выполнении.)

Для выполнения каждого запроса создается объект, который называется курсором. Понимание роли курсоров исключительно важно для всего процесса настройки прикладной системы. Детально курсоры обсуждаются в *главе 4*, сейчас мы только подчеркнем, что курсор создается для любого запроса к серверу базы данных, независимо от того, какой тип интерфейса (встроенный, статический или динамический) используется. Важность понятия курсора для настройки определяется тем, что как создание, так и вы-

полнение курсора является относительно дорогостоящей операцией и эффективность работы приложения может зависеть от количества создаваемых и используемых в нем курсоров.

2.1.2. Функции СУБД

Правильное понимание роли и функций СУБД очень важно для создания высокоэффективных систем. Многие проблемы производительности возникают вследствие упрощенного понимания этой роли.

Перечислим основные функции систем управления базами данных, как они были определены в [22] для СУБД, позже названных системами первого поколения:

- независимость данных и приложений;
- защита целостности данных;
- поддержка согласованности данных;
- высокоуровневый язык запросов;
- восстановление после отказов;
- защита от несанкционированного доступа;
- разграничение доступа.

В современных системах значение этих функций существенно изменилось. Некоторые из функций полностью или частично перешли к другим типам компонентов прикладных систем, важность и наличие других зависит от архитектуры прикладной системы и от типа СУБД.

Обеспечение независимости данных и приложений рассматривалось как важнейший элемент поддержки эволюции систем. Независимость данных и программ означает, что допускается изменение структуры данных для поддержки новых задач или в целях настройки, при этом старые приложения остаются работоспособными. Средством достижения независимости является наличие языка описания данных и словаря данных, который представляет структуру данных, хранимых под управлением СУБД. Словари данных и языки описания данных в тои или иной форме включены во все современные системы.

Кроме структуры данных, система обеспечивает выполнение некоторых соотношений, которым должны удовлетворять хранимые данные. Эти соотношения называются ограничениями целостности (integrity). Функция защиты целостности состоит в том, что СУБД не допускает нарушения этих ограничений ни при каких обстоятельствах. Операции, выполнение которых привело бы к нарушению ограничений целостности, отвергаются. Наиболее широко известными типами ограничений целостности являются ограниче-

ния ссылочной целостности, уникальности значений, запрет на неопределенные значения и условия на значения атрибутов.

Понятие согласованности (consistency) в русскоязычной литературе часто тоже обозначают термином "'целостность", что приводит к некоторой путанице, поэтому остановимся на значении этого понятия подробнее. Некоторые из целостных состояний базы данных называются согласованными, СУБД допускает выполнение операций, приводящих базу данных в несогласованное состояние, однако приложение обязано выполнить действия, возвращающие базу данных в другое согласованное состояние.

Более точно набор операций, выполняемых приложением и переводящих базу данных из одного согласованного состояния в, вообще говоря, другое согласованное состояние, называется транзакцией. Функция поддержки согласованности СУБД означает, что после завершения работы приложения база данных будет в согласованном состоянии, т. е. все транзакции приложения будут либо выполнены полностью, либо не оставят никаких следов. Это требование называется атомарностью транзакций.

Поддержка согласованности существенно усложняется, если сервер базы данных допускает конкурентное выполнение нескольких транзакций (вообще говоря, из разных сеансов), потому что в этом случае необходимо исключить взаимные помехи, которые могут возникать при конкурентном выполнении. Сколько-нибудь детальное обсуждение понятий и механизмов, связанных с управлением транзакциями, далеко выходит за рамки данной книги. Некоторые дополнительные сведения приведены в *главе 8*. Детальное изложение теории и методов, применяемых для поддержки согласованности, можно найти в книгах [16, 2, 6].

Необходимо пояснить термин "конкурентное выполнение", использованный в предыдущем абзаце. Мы говорим о конкурентном выполнении, когда нам важно, что операции, выполняемые в разных транзакциях, могут чередоваться или выполняться в каком-то смысле одновременно, но механизм, с помощью которого эта одновременность достигается, не имеет значения. Выполнение может быть конкурентным, если операции выполняются параллельно на многопроцессорном сервере, но параллельность вовсе не требуется. Точно также операции могут выполняться псевдопараллельно в режиме мультипрограммирования (в разных процессах или нитях операционной системы), но это также не требуется, конкурентное выполнение в принципе возможно, даже если сервер базы данных использует всего одну нить в одном процессе операционной системы.

Наличие высокоуровневого языка запросов является, пожалуй, наиболее важной отличительной особенностью СУБД. Значительная часть данной книги посвящена методам выполнения и настройки запросов.

Функция восстановления после отказов часто рассматривается как часть функции поддержки согласованности и также остается одной из важных функций СУБД.

Наконец, функции защиты данных от несанкционированного доступа и разграничение доступа чаще всего не используются или используются в ограниченном объеме, даже если они реализованы в СУБД. Способ реализации этих функций в прикладной системе не оказывает непосредственного влияния на производительность, и поэтому мы не будем уделять этим вопросам много внимания.

2.2. Метрики производительности

Целью настройки любой прикладной системы является улучшение ее работы по каким-либо параметрам. Исходя из этого, первая задача, которую требуется решить, приступая к настройке, — определить, как будет измеряться качество работы системы, и каким образом предполагается его улучшить в процессе настройки.

Таким образом, нас интересует производительность системы и способы ее оценки. Наиболее часто используемые для оценки производительности характеристики системы — *время ответа* и *пропускная способность*. Временем ответа называется время ожидания и время выполнения запроса, а пропускной способностью — количество запросов, обрабатываемых системой за определенный промежуток времени.

Пока мы не делаем различия между этими характеристиками для системы в целом и для базы данных. Сейчас для нас важны особенности и разновидности этих характеристик, а не система, к которой они относятся. Однако это различие становится важным, когда определяются конкретные цели настройки, поскольку время ответа, которое видит пользователь, может быть значительно большим, чем время ответа системы управления базами данных.

Говоря о времени ответа системы в целом, можно рассматривать как среднее время ответа (или время ожидания) пользователя, так и максимальное время ответа. В зависимости от особенностей конкретной системы в качестве цели настройки может выступать уменьшение как среднего, так и максимального времени ответа. Однако хорошо известно, что ограничения на предельное (максимальное) время ответа крайне трудно обеспечить в СУБД. Поэтому такие критерии используются весьма редко, обычно только в системах реального времени, в которых предсказуемость времени ответа является обязательным требованием. Практически во всех остальных классах систем можно ограничивать время выполнения для "почти всех" случаев выполнения запроса. В таком ослабленном виде ограничения на максимально допустимое время ответа становятся реализуемыми.

Аналогичные варианты определения времени ответа системы применимы, конечно, не только к прикладной системе в целом, но и к серверу базы данных.

Следует отметить, что говорить о среднем или максимальном времени ожидания имеет смысл только для запросов одного класса. В промышленных системах разные запросы и разные пользователи могут иметь разные приоритеты в обслуживании, поэтому настройка системы может быть направлена в первую очередь на минимизацию времени обработки конкретных типов запросов.

На первый взгляд может показаться, что увеличение пропускной способности системы приводит к улучшению времени ответа и наоборот, поскольку чем больше запросов может обработать система за определенный промежуток времени, тем меньше времени будет выполняться каждый отдельный запрос. Однако реально такой прямой зависимости не существует, поскольку обычно несколько запросов выполняются одновременно, и, соответственно, пропускная способность системы может быть разной при том же времени ответа.

Иными словами, высокая пропускная способность достигается при высокой степени загрузки системы (ресурсы систем не простаивают), а малое время ответа — при малой загруженности системы (необходимые ресурсы всегда доступны).

Разумной комбинацией требований является, по-видимому, достижение максимально возможной пропускной способности при условии выполнения ограничения на время ответа.

Такие характеристики системы, как надежность и доступность, не являются характеристиками производительности, однако они должны учитываться при настройке, потому что меры, улучшающие эти характеристики, могут приводить к некоторому ухудшению характеристик производительности.

Наряду с абсолютными критериями производительности во многих случаях важны относительные, как правило, объединяемые термином масштабируемость. Масштабируемость — это функция, описывающая зависимость некоторой характеристики производительности от размеров системы (количества оборудования, объема хранимых данных и количества поступающих запросов).

Например, масштабируемость по пропускной способности показывает, как изменяется пропускная способность системы при одновременном увеличении количества оборудования и объема хранимых данных. Очевидно, рост пропускной способности не может превышать увеличение количества оборудования, поэтому идеальная масштабируемость по пропускной способности описывается линейной функцией (с коэффициентом наклона, равным 1).

Часто рассматривается масштабируемость по числу пользователей, однако эта характеристика почти эквивалентна масштабируемости по пропускной способности, но, в отличие от пропускной способности, не поддается количественным измерениям в тестовой обстановке.

Для масштабируемости по времени ответа, наоборот, идеальная масштабируемость описывается постоянной функцией (время ответа не возрастает, несмотря на увеличение объема данных и нагрузки на систему).

Конечно, идеальная масштабируемость никогда не может быть достигнута на практике, так как некоторая часть ресурсов используется на организацию взаимодействия между компонентами системы. Есть и другая причина, по которой идеальная масштабируемость может оказаться недостижимой: запросы разных пользователей и приложений могут использовать и обновлять общие данные.

Не следует переоценивать значение масштабируемости. Далеко не всегда масштабируемость системы необходима, и практически всегда, когда она действительно нужна, она необходима в некоторых ограниченных пределах. Рост многих систем ограничен естественными факторами, такими как число сотрудников организации, или количеством реальных объектов, данные о которых хранятся в системе. Конечно, эти ограничения могут существенно измениться, если изменяется функциональность системы.

Более подробно характеристики масштабируемости обсуждаются в *главе 9*.

Перечисленные выше характеристики применяются для определения целей и оценки результатов настройки системы. В самом процессе настройки важны и другие характеристики производительности, на значения которых можно влиять только на фазе первоначального проектирования, когда планируется покупка оборудования, но которые важны как для оценки реалистичности требований, так и для выбора стратегии настройки. Речь, конечно, идет о характеристиках производительности оборудования.

Традиционно для систем обработки больших объемов данных наиболее важной характеристикой оборудования является быстродействие внешних устройств хранения данных, т. е. дисков. Среди многочисленных характеристик дисковых устройств (минимальное, среднее, максимальное время подвода головок, скорость вращения и др.) наиболее важными, по-видимому, являются среднее и максимальное время доступа в случайном порядке и скорость передачи данных при последовательном доступе к соседним участкам.

Как это ни удивительно, но для многих современных систем производительность процессора (или процессоров) также оказывается важной — многие операции в базах данных интенсивно используют вычислительную мощность процессора. То же самое можно сказать и о некоторых типичных функциях, реализуемых в приложениях.

Как для процессоров, так и для дисков существенное влияние на производительность оказывает наличие и размеры буферной (кэш) памяти. Ни в том, ни в другом случае проектировщики и разработчики не могут влиять на использование этой памяти непосредственно, однако косвенное управление возможно, так как возможности использования буферной памяти устройств зависят от размещения данных.

Наконец, для систем, в работе которых участвует больше одного компьютера, важна пропускная способность вычислительной сети. Обычно, особенно в разговорной речи, эта характеристика называется скоростью, что не совсем точно отражает ее смысл. Конечно, большие сообщения передаются по широкополосной сети значительно быстрее, чем по сети с низкой пропускной способностью, но это определяется именно пропускной способностью. А время передачи небольших сообщений определяется, прежде всего, количеством промежуточных узлов сети (маршрутизаторов и т. п.), а также загруженностью этих узлов.

2.3. Базы данных в архитектурах прикладных систем

В данном разделе обсуждаются различные классы архитектур приложений, использующих базы данных, и влияние особенностей каждого класса архитектур на производительность системы.

Изложение ни в какой степени не является исчерпывающим, мы предполагаем, что читатель знаком с этими вопросами из других источников, и рассматриваем только аспекты, которые важны в контексте данной книги.

Границы между различными классами архитектур, рассматриваемых в этом разделе, являются размытыми. Некоторые системы, которые следует отнести к одному из классов, могут обладать свойствами, более характерными для других классов.

Прежде чем обсуждать особенности отдельных архитектур, необходимо подчеркнуть, что эти особенности нельзя рассматривать как достоинства или недостатки. Каждая из обсуждаемых характеристик может оказывать как положительное, так и отрицательное влияние не только на качество (производительность) получаемой системы, но и на стоимость ее разработки, тиражирования и сопровождения. Не существует архитектуры или метода построения системы, которые превосходили бы другие абсолютно во всех случаях.

Кроме этого, обычно разработчик не может произвольно выбирать ни архитектуру, ни используемые компоненты — то и другое может быть навязано (или выбрано) на основе иных соображений, не имеющих прямого отношения к техническим характеристикам программных средств.

2.3.1. Встроенные СУБД

Авторы относят к этому классу системы, в которых компоненты, реализующие функции системы управления базами данных, выполняются в рамках того же процесса или той же задачи операционной системы, что и приложение.

Можно сказать, что компоненты СУБД встраиваются в код, реализующий функции приложения. Мы намеренно используем неточный термин "встраивается", потому что детали способа объединения компонентов в исполняемый объект для нас несущественны. Компоненты, реализующие функции СУБД, могут быть выполнены в виде библиотеки подпрограмм, статически или динамически подключаемой к коду приложения. В некоторых системах код приложения может интерпретироваться оболочкой, предоставляющей доступ к СУБД (в этом случае, скорее, приложение встраивается в СУБД, чем наоборот).

Системы этого класса получили очень широкое распространение, начиная с конца 80-х гг. прошлого века в связи с появлением и массовым применением персональных компьютеров, достаточно производительных для обработки небольших баз данных.

Для нас важны следующие особенности этого типа архитектур.

- Функциональность приложения и СУБД реализуются в рамках одного процесса операционной системы и поэтому взаимодействие между СУБД и приложением оказывается весьма эффективным (отсутствуют затраты на организацию взаимодействия процессов и задержки, связанные с передачей сообщений по сети).
- Как правило, базы данных невелики по объему, поэтому обычно в таких СУБД используются очень простые механизмы выполнения запросов, зачастую отсутствует оптимизатор запросов. Нередко в таких системах реализуется только некоторое подмножество языка запросов SQL.
- Обычно подобные СУБД работают в однопользовательском режиме, т. е. во время работы приложения другие программы не могут получать доступ к тем же данным. Иногда совместное использование базы данных допускается, но СУБД при этом не обеспечивает поддержку согласованности.
- Как правило, база данных используется только одним приложением.
- Производительность приложения обычно быстро падает с ростом объема данных, в особенности при выполнении сложных запросов.

Можно сказать, что из функций СУБД, перечисленных в предыдущем разделе, для встроенных СУБД наибольшее значение имеют язык запроса, средства описания данных и поддержка целостности. Согласованность

обычно реализуется в ограниченном виде, а остальные функции, как правило, не требуются совсем.

В системах, для которых применение этого класса СУБД целесообразно, единственно важным критерием производительности является время ответа, а пропускная способность и масштабируемость не имеют большого значения.

Еще одна особенность таких систем — наличие дружественного интерфейса и простота использования. Иногда пользователь, не занимающийся программированием профессионально, отчаявшийся получить хоть что-нибудь полезное от программистов, успешно осваивает такую систему сам.

В силу перечисленных особенностей такие системы практически никогда не требуют какой-либо настройки, и поэтому не будут рассматриваться в этой книге сколько-нибудь детально.

Мы упоминаем эти системы потому, что для огромного большинства специалистов, занимающихся созданием приложений, системы такого класса стали первыми, с которыми они столкнулись в своей работе, и опыт, приобретенный при использовании этих систем, существенно повлиял на формирование их навыков.

Одна из целей данной книги — показать, почему техника работы с базами данных, хорошо зарекомендовавшая себя на небольших системах, чаще всего оказывается непригодной при разработке приложений, использующих средние и большие базы данных.

Поскольку системы этого класса не обладают мощными исполнителями запросов, и, с другой стороны, накладные расходы, связанные с передачей запросов из приложения в ядро СУБД, невелики, как правило, наиболее целесообразно использование небольших (по сложности и количеству обрабатываемых данных) запросов. С другой стороны, использование сложных запросов в системах этого класса может приводить к неконтролируемому росту времени ответа.

Даже при выполнении вычислений, на которые объективно требуется много времени, разбиение задачи на небольшие запросы дает возможность информировать пользователя о ходе вычислений или выводить частичные результаты.

Небольшие объемы данных дают возможность использовать логические структуры хранения, обеспечивающие высокую степень гибкости и простоту разработки приложения, но которые были бы крайне неэффективны для больших баз данных.

В последнее время широкое распространение получает другой класс встроенных СУБД, предназначенный для работы в составе распределенных систем. Как правило, эти системы работают на мобильных устройствах и обеспечивают ограниченные возможности обработки данных в периоды, когда

по тем или иным причинам устройство используется в автономном режиме, т. е. без соединения с основной базой данных, размещенной на сервере.

Объемы данных, хранимых в таких системах, обычно совсем невелики и поэтому базы данных сами по себе практически никогда не требуют настройки в традиционном понимании. Узким местом в таких системах является, как правило, взаимодействие и обмен данными с серверными системами.

2.3.2. Серверы баз данных

В данном разделе рассматривается архитектура систем, известная с середины 80-х годов XX века под названием "клиент-сервер". Поскольку термины "клиент" и "сервер" используются в различных контекстах и в довольно разном смысле, необходимо уточнить, что для нас эти понятия характеризуют способ взаимодействия программных компонентов, а не оборудования, и определяют роли этих программных компонентов во взаимодействии, а не сами эти компоненты. В некоторых случаях один и тот же компонент может одновременно выполнять функции как клиента, так и сервера (например, в распределенных СУБД или в многослойных архитектурах, обсуждаемых в *разд. 2.3.3*).

Фактически все большие системы управления базами данных реализовывались как системы типа "клиент-сервер" с момента возникновения концепции СУБД в конце 60-х годов, значительно раньше, чем эти термины стали общепринятыми. Это вызвано тем, что по существу одна из основных функций СУБД — централизация управления данными, используемыми различными приложениями.

Для того чтобы обеспечить централизацию, необходимо выделить компонент (ядро СУБД), который бы постоянно находился в состоянии выполнения, принимал бы запросы приложений (клиентов), выполнял эти запросы и возвращал результаты выполнения, т. е. был бы сервером базы данных в смысле, определенном ранее в этой главе.

Системами класса "клиент-сервер" принято называть такие, в которых программный код приложения взаимодействует с СУБД непосредственно, но выполняется в рамках другого процесса операционной системы и обычно на другом компьютере. Именно для систем класса "клиент-сервер" особенно важно, чтобы СУБД поддерживала все основные функции, перечисленные в предыдущем разделе, в наибольшем объеме.

Во всех современных системах управления базами данных взаимодействие клиента с сервером всегда осуществляется с использованием некоторого сетевого протокола общего назначения (чаще всего TCP/IP, но как раз это для нас несущественно), над которым надстраивается протокол сервера СУБД.

Этот протокол непосредственно используется клиентом СУБД для передачи запросов и получения результатов их выполнения. Сетевые протоколы СУБД могут быть специфическими для конкретной СУБД (например, Oracle SQL*Net) или поддерживаться несколькими (или многими) СУБД различных производителей. Иногда такие обобщенные протоколы надстраиваются над специализированными протоколами конкретных СУБД, при этом возможности использования особенностей конкретной СУБД ограничиваются. Широко используемыми протоколами такого типа являются ODBC (Open DataBase Connectivity, открытый интерфейс доступа к базам данных), и JDBC (Java DataBase Connectivity, средство организации доступа Java-приложений к базам данных) и, вероятно, не существует сколько-нибудь широко распространенных альтернатив.

Любой из упомянутых в предыдущем абзаце протоколов обеспечивает взаимодействие приложения и сервера базы данных на уровне языка запросов. Иногда между приложением и протоколом взаимодействия с сервером базы данных используются дополнительные программные компоненты, изолирующие программиста приложений от языка запросов. Неправильное использование таких компонентов может очень существенно ухудшить производительность всей прикладной системы. Более детально мы рассмотрим эту проблему в *главе 4*.

Для наших целей важны не сами протоколы, а тот факт, что механизм передачи запросов (и результатов) реализуется довольно сложной "башней" протоколов, поэтому время, необходимое для этого взаимодействия, обычно оказывается довольно большим.

Мы уже отмечали, что взаимодействие каждого клиента с сервером баз данных осуществляется в рамках сеанса (подключения, сессии) с базой данных. Информация о сеансе сохраняется как в клиенте, так и на сервере, иными словами, протоколы взаимодействия с СУБД являются протоколами с состоянием. Во многих СУБД процесс установления сеанса занимает относительно много времени, поэтому часто программа-клиент использует один и тот же сеанс работы с базой данных на протяжении всего (интерактивного) сеанса работы с конечным пользователем.

Наиболее существенно для нас то, что даже при использовании сетей с очень высокой пропускной способностью время, необходимое для получения ответа на сообщение (round trip), может быть значительным, даже если обработка этого сообщения не занимает сколько-нибудь значительное время. С другой стороны, передача относительно больших сообщений на широкополосных сетях может выполняться очень быстро.

Важно отметить, что во всех современных СУБД сетевые протоколы используются даже в том случае, если клиент и сервер выполняются на одном и том же компьютере.

Обработка любого запроса к СУБД включает в себя несколько операций, требующих синхронизации клиента и сервера, т. е. несколько ожиданий ответа, поэтому выполнение любого запроса требует некоторого минимального времени, независимо от сложности запроса. С другой стороны, при передаче результатов выполнения запроса пересылка данных может осуществляться в какой-то мере асинхронно. Поэтому дополнительное время, вызванное задержками сетевых протоколов, весьма существенно для коротких запросов, быстро выполняемых на сервере и возвращающих небольшое количество данных. Однако с ростом сложности запроса или объема возвращаемых данных относительная доля сетевых задержек в общем времени ответа очень быстро уменьшается, в особенности в сетях с высокой пропускной способностью.

В *главе 4* обсуждается, каким образом эти особенности архитектуры можно учесть при разработке приложений.

2.3.3. Многоуровневые архитектуры

Системы типа "клиент-сервер" оказываются неэффективными или слишком ограничительными для некоторых распространенных классов приложений. Наиболее серьезные проблемы возникают в тех случаях, когда количество (конечных) пользователей, работающих с системой, очень велико (десятки тысяч и более). В этом случае поддержка сеансов работы с системой становится не менее важной, чем поддержка целостности и согласованности данных.

Так, глобальная система резервирования авиабилетов не может быть остановлена (или перезапущена) не только из-за потенциальной потери части транзакций, но и потому, что восстановление всех активных сеансов с системой резервирования заняло бы несколько часов и привело бы к катастрофическим убыткам [8]. Конечно, системы такой сложности и с такими требованиями к надежности уникальны, однако сходные проблемы возникают и в меньших по масштабу системах.

Системы, ориентированные на обслуживание очень большого количества пользователей, обычно используют мониторы транзакций или (в современной терминологии) серверы приложений. Основное отличие мониторов транзакций от серверов приложений состоит в том, что первые, как правило, используют специализированные сетевые протоколы, а последние — стандартные протоколы WWW. Детальное обсуждение относительных преимуществ и недостатков протоколов такого типа выходит за рамки этой книги.

Для нас существенно то, что сервер приложений, выступая в роли сервера по отношению к клиентской программе, принимает запросы на выполнение определенных приложений и организует их выполнение, в том числе их

взаимодействие с базой данных, выступая по отношению к серверу базы данных в роли клиента.

При этом сервер приложений может брать на себя часть функций СУБД (частично дублируя их). В частности, сервер приложений берет на себя функции защиты от несанкционированного доступа, выполняя идентификацию и проверку полномочий пользователей. Взаимодействие с базой данных осуществляется от имени одного пользователя базы данных для всех пользователей прикладной системы. При этом функция разграничения доступа может перекладываться на приложение.

Для сокращения времени на установление сеансов (которые в этом контексте обычно называются соединениями) с базой данных используется пул соединений (может поддерживаться как СУБД, так и сервером приложений). При использовании пула соединений сервер приложений не создает новое соединение для каждого выполнения приложения, а выделяет одно из свободных, а по окончании работы приложения возвращает это соединение в пул. В данной книге термин "соединение" будет использоваться в другом смысле (как русское название реляционной операции `JOIN`).

Преимущества использования серверов приложений определяются следующими факторами:

- при взаимодействии с клиентом (сервера приложений) может использоваться протокол без состояния;
- при использовании пула соединений, вообще говоря, экономится время на установление соединений;
- взаимодействие сервера приложений с клиентом может быть организовано на основе протокола без состояний;
- как правило, сервер приложений находится в одной локальной сети с сервером базы данных, что обеспечивает более эффективное взаимодействие приложения (выполняемого на сервере приложений, а не на клиентской машине) с базой данных.

Рассмотрим особенности этой архитектуры, важные с точки зрения настройки производительности системы.

Прежде всего, выполнение запросов на сервере баз данных не зависит от того, используется сервер приложений или нет, поэтому основные принципы настройки одинаковы для систем "клиент-сервер" и систем, включающих сервер приложений.

Имеется, однако, существенная разница в критериях оптимизации запросов. Многие СУБД могут оптимизировать запросы по-разному в зависимости от того, требуется быстрое получение первых строк ответа или результата выполнения запроса полностью. В системах "клиент-сервер" получение первых строк создает у пользователя иллюзию более быстрого ответа системы и

поэтому может быть целесообразно. В системах же с сервером приложений для формирования ответа пользователю необходимо, как правило, получить результат выполнения запроса полностью.

В связи с этим возникает проблема результатов большого объема. Если большой по объему результат выполнения запроса возвращать полностью, то непомерно возрастает нагрузка на сервер приложений и увеличивается время передачи результата клиенту, а если результат фрагментируется приложением, то неоправданно увеличивается нагрузка на сервер базы данных и в меньшей степени на сервер приложений.

Предположим, ответ на запрос пользователя содержит несколько сотен тысяч строк. Такой объем данных может быть передан из СУБД в приложение за единицы секунд, однако на формирование страницы HTML такого размера и, в особенности, на передачу такой страницы клиенту потребуется намного больше времени.

Если же вывод будет сегментироваться, потребуется многократное выполнение запроса сервером базы данных (если сервер приложений используется как сервер без состояний).

Наличие сервера приложений само по себе мало влияет на низкую эффективность слишком мелких запросов к базе данных. В некоторых случаях время, затрачиваемое на взаимодействие с базой данных, может в десятки раз превышать время выполнения самого приложения.

В зависимости от особенностей приложения эта проблема может решаться двумя различными способами:

- сохранением результатов выполнения запросов к базе данных в памяти сервера приложений (кэшированием);
- перераспределением функциональности приложения между компонентами системы.

Кэширование результатов на сервере приложений основано на предположении о том, что количество различных запросов, выполняемых пользователями, во много раз меньше количества пользователей, поэтому вероятность того, что результаты выполнения запроса вскоре понадобятся снова, достаточно велика.

В некоторых системах в кэш сервера приложений копируется вся база данных, что фактически означает отказ от использования процессора запросов базы данных.

Перераспределение функциональности приложения состоит в том, что часть кода приложения выполняется непосредственно на сервере баз данных (в форме хранимых процедур и триггеров). Такое решение дает возможность в полной мере использовать мощные возможности СУБД, в том числе объектные расширения языка запросов. Подобная архитектура, однако, может