Оглавление

О курсе

На кого ориентирован курс

Курс рассчитан на студентов младших курсов (бакалавриата) классических и технических университетов, а также других вузов, имеющих базовую подготовку по программированию и продолжающих специализироваться в областях, близких к программированию.

Какие знания будут получены

В курсе подробно рассматриваются основные понятия, устройство и принципы работы СУБД, а также технологии (архитектура, алгоритмы, структуры данных), лежащие в их основе.

Прослушавшие курс получат уверенные знания и практические навыки по следующим вопросам:

- устройство и принципы работы СУБД;
- проектирование баз данных;
- работа с SQL — составление и оптимизация запросов;
- разработка серверных приложений;
- использование различных типов индексов;
- обработка транзакций и одновременный доступ;
- основы эксплуатации баз данных;
- обеспечение надежности хранения, отказоустойчивости и высокой доступности;
- принципы организации и работы параллельных и распределенных СУБД;
- работа со слабоструктурированными данными (JSON, XML).

Такая подготовка позволит на старших курсах (в магистратуре) специализироваться на разработке и настройке приложений баз данных либо в областях проектирования и разработки СУБД.

Структура курса

Курс состоит из двух частей.

В первой части, представленной в этой книге, рассматриваются основные сведения о базах данных и системах управления базами данных: реляционная модель данных, язык SQL, обработка транзакций.

Во второй части подробно рассматриваются технологии, лежащие в основе функционирования СУБД, а также современные направления и тенденции развития СУБД, основные аспекты их практического применения. При этом некоторые темы, рассмотренные в первой части, изучаются повторно на более глубоком уровне.

Курс в основном касается классических реляционных и объектно-реляционных СУБД, но затрагивает также тематику неклассических СУБД.

Практические занятия не только помогают закрепить пройденный на лекциях материал. Они содержат много дополнительной информации, закрепляющей и расширяющей знания, изложенные в теоретической части. В качестве СУБД для практических занятий используется PostgreSQL.

Как первая, так и вторая части курса могут быть выделены в самостоятельные курсы. Отдельные разделы курса могут быть скомбинированы так, чтобы получить более практическую или более фундаментальную направленность либо адаптировать курс к конкретному учебному плану вуза.

Программные средства, используемые в курсе

Для эффективного освоения материала курса и для выполнения упражнений необходимо установить на компьютере ряд программных продуктов. Набор этих продуктов может зависеть от используемой операционной системы и от других обстоятельств, но в любом случае понадобятся:

- система управления базами данных PostgreSQL;

- демонстрационная база данных, которая используется в большинстве примеров;
- текстовый редактор для подготовки запросов на языке SQL.

Установка PostgreSQL и демобазы рассматривается в главе 3.

Для выполнения упражнений по созданию и редактированию моделей данных может потребоваться инструмент для редактирования диаграмм.

Для разработки приложений на императивных языках программирования (C, C++, Java, Python и др.) потребуются соответствующие среды разработки, однако такие упражнения не включены в состав этого курса.

При работе с PostgreSQL можно использовать ряд приложений, предоставляющих графические интерфейсы для работы с базами данных. Многие из этих приложений могут работать с различными СУБД. Как правило, в таких системах ограничены возможности работы с особенностями любой конкретной СУБД. При освоении материала этого курса более целесообразно использовать непосредственно средства системы PostgreSQL или другие программы, спроектированные специально для работы с PostgreSQL.

Благодарности

Подготовка этого курса была бы невозможна без активной поддержки со стороны компании Postgres Professional и ее руководства, в частности О. Бартунова и И. Панченко. Качество материала существенно улучшилось благодаря усилиям Е. Рогова, взявшего на себя огромный труд по редактированию курса.

Главы 7 и 9 написаны совместно Е. Горшковой и Б. Новиковым. В подготовке упражнений принимали участие В. Бусаров, К. Секереш, Г. Шалыгина и Е. Михайлова.

Глава 1

Введение

Компьютеры используются повсеместно: невозможно найти предприятие или учреждение, которые не использовали бы их для решения производственных или управленческих задач. Подобные высказывания не слишком заметны в средствах массовой информации потому, что они уже давно не являются новостью. Профессионалы, однако, понимают, что на самом деле важны не компьютеры, а информационные системы, которые на них работают, а в центре любой информационной системы находятся данные.

Эта книга о том, как хранить данные, обеспечивать их корректность и сохранность и как их обрабатывать эффективно.

1.1. Базы данных и СУБД

Появление и относительно широкое распространение в начале 60-х годов XX века запоминающих устройств достаточно большой емкости с возможностью доступа к произвольным участкам памяти — магнитных дисков, — открыло широкие возможности для создания сложных структур долговременно хранимых данных. Высокая скорость обновления небольших объемов данных (доли секунды) создала условия для создания приложений, способных функционировать в режиме оперативной работы (on-line). В отличие от систем предшествующих поколений, время ответа стало измеряться не сутками, а секундами или долями секунды.

Эти возможности, однако, привели к существенному усложнению кода приложений и, как следствие, к удорожанию их разработки и снижению надежности. В связи с этим появилась идея централизации функций управления данными, которая привела к появлению систем, предоставляющих приложениям услуги по обработке данных. Такие системы получили название систем управления базами данных (СУБД).

Поскольку СУБД используются многими приложениями, ожидается, что они могут обеспечивать более высокие значения эксплуатационных характеристик, таких как надежность хранения и эффективность обработки, недостижимые при индивидуальной разработке средств управления данными для каждого приложения.

Важно отметить, что многие особенности и характеристики, присущие ранним системам управления базами данных, связаны с требованиями тех областей применения и классов приложений, которые были наиболее актуальны в то время. В первую очередь это приложения, работающие в режиме оперативной обработки (on-line transaction processing, OLTP) в банковской и финансовой сферах.

Прежде чем обсуждать, каким образом эти области применения повлияли на характеристики СУБД, уточним значение некоторых терминов, которые будут использоваться в дальнейшем.

Система управления базами данных (СУБД) — это программный комплекс, обеспечивающий централизованное хранение данных и предоставляющий приложениям услуги по обработке данных.

Совокупность данных, хранимых под управлением СУБД, называется *базой данных*. В оригинальном английском варианте словосочетание data base означает «основание, состоящее из данных». Этот смысл несколько искажается в русском словосочетании «база данных». На самом деле это — фундамент, на котором строятся приложения и который состоит из данных. Действительно, данные (а следовательно, база данных) являются очень существенной частью практически любой информационной системы.

Система управления базами данных, находящаяся в фазе выполнения, связанная с некоторой конкретной базой данных и готовая выполнять запросы на обработку этой базы данных, называется *экземпляром* (instance) или *сервером базы данных*. На самом деле экземпляр и сервер — разные понятия: один сервер баз данных может управлять несколькими экземплярами баз данных, однако это различие станет важным только начиная с главы 5.

1.2. Требования к СУБД

Ранние системы управления данными очень сильно различались как по своей внутренней организации, так и по предоставляемым возможностям. Потребо-

валось несколько лет, для того чтобы определить, каковы основные функции систем управления базами данных и какие требования следует предъявлять к таким системам.

Основные требования к системам управления базами данных были сформулированы в документе, опубликованном в 1971 году комитетом по системам и языкам обработки данных (CODASYL) [13], русский перевод которого издан в 1975 году [17]. Основой для этих требований послужил анализ систем, применявшихся в период подготовки отчета, и особенностей прикладных областей, в которых эти системы использовались.

В дальнейшем круг областей применения СУБД непрерывно расширялся, появлялись новые системы и уходили старые, однако многие из этих требований остались актуальными и сегодня, и большинство современных СУБД в той или иной форме реализует значительную часть этих требований. Однако далеко не все классы приложений, в которых используются современные системы, предъявляют те же требования к обработке данных, поэтому и системы реализуются иначе.

Приложения, относящиеся к классу оперативной обработки (OLTP), характеризуются тем, что:

- каждое выполнение приложения занимает мало времени (в идеале — не больше долей секунды);
- данные используются совместно многими приложениями;
- при каждом выполнении приложение использует ничтожную долю общего объема хранимых данных, и обычно количество используемых данных не зависит от общего объема базы.

Важно также отметить, что процессы обработки данных и структуры данных в тех областях, в которых использовались ранние СУБД, фактически были формализованы задолго до появления электронных вычислительных систем. Так, правила бухгалтерского учета в основном сложились в XIV веке и мало изменились в последующем. Возможно, это привело к тому, что СУБД, как правило, ориентированы на обработку структурированных данных.

Перечислим основные требования к системам управления базами данных.

Разделение программ и данных. Описание структуры данных должно быть отделено от кода приложений, и система должна допускать независимое

изменение структуры данных и кода приложения. В документе [17] использовался термин «независимость» (independence), однако «разделение» лучше отражает существо этого требования.

Высокоуровневый язык запросов. Система должна предоставлять средства для обработки данных, не включенные в какое-либо приложение.

Целостность. Система должна предотвращать запись данных, нарушающих заранее специфицированные ограничения.

Согласованность. Система должна предотвращать появление некорректностей в данных вследствие параллельной или псевдопараллельной работы нескольких приложений.

Отказоустойчивость. СУБД не должна допускать потери данных даже в случае отказов оборудования.

Защита и разграничение доступа. Система должна предотвращать несанкционированный доступ к данным и предоставлять каждому пользователю (или приложению) доступ только к части данных в соответствии с правами этого пользователя.

Заметим, что в ранние годы существования СУБД предполагалось, что все данные, необходимые для информационных систем предприятия, будут храниться в единой базе данных. Безусловно, это очень хорошая идея, поскольку при этом создаются возможности для исключения избыточности данных, проверки их корректности и предотвращается рассогласованность данных, используемых различными подразделениями предприятия. Почти все учебники по базам данных написаны с учетом этого предположения.

Однако на практике это никогда не было реализовано: как правило, для каждого приложения или небольшой группы приложений создается отдельная база данных. Основная причина, по-видимому, состоит в том, что структуры данных, необходимые для обеспечения информационных потребностей предприятия, слишком сложны для того, чтобы их можно было описать в рамках одной базы данных. Это обстоятельство существенно влияет на значение и использование как перечисленных требований, так и других особенностей систем управления базами данных.

Далее в этой главе данные требования и возможности СУБД обсуждаются более детально, а в последующих главах показано, как эти возможности реализуются в современных системах и, в частности, в системе PostgreSQL.

1.3. Разделение данных и программ

Каждая система управления данными создает некоторый уровень абстракции для других программных компонент, которые используют ее услуги. Для того чтобы реализовать определенный уровень абстракции данных, необходимо, чтобы система могла работать с описанием данных, соответствующих требуемому уровню абстракции.

Так, современные файловые системы представляют файлы как последовательности байтов. Соответственно, операции над содержимым файлов выражаются в терминах позиций байтов внутри файла. Никакие более сложные операции не могут быть определены, потому что файловая система не имеет информации о структуре данных внутри файла.

Поскольку ожидается, что системы управления базами данных предоставляют операции над данными сложной структуры, необходимо, чтобы описание этой структуры было доступно СУБД и было бы общим для всех программ (приложений), использующих эти данные. Это приводит к идее отделения описания структуры данных от программ. Такое описание хранится в самой базе данных и называется *схемой базы данных*.

Для определения схем используются языки описания данных. Чем больше возможностей у такого языка, тем больше услуг может предоставить система управления базами данных. В то же время необходимость подготовки детализированного описания данных на фазе проектирования прикладной системы может в некоторых случаях оказаться чрезмерно трудоемкой.

Само по себе отделение описания данных от приложений не дает достаточной гибкости. Для того чтобы в полной мере реализовать идею разделения данных и программ, в 1975 году (тем же комитетом CODASYL) была подготовлена обобщенная модель языка описания данных. Описание этой модели стало известно под названием «модель данных ANSI/SPARC», так как предполагалось, что эта модель будет иметь статус стандарта. Схематически основные компоненты этой модели представлены на рис. 1.3.1.

Модель включает:

внешнюю схему, содержащую описание данных в таком виде, в котором они будут использоваться приложением (отдельно для каждого приложения), а также отображение логической структуры данных во внешнюю схему;

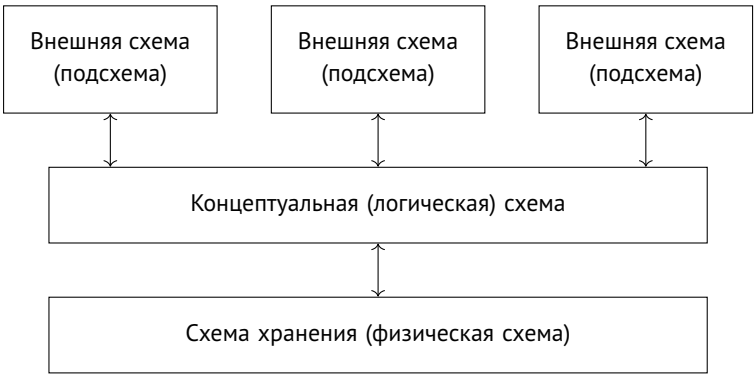

Рис. 1.3.1. Трехуровневая модель данных ANSI/SPARC

концептуальную схему, содержащую полное описание логической структуры данных, доступное СУБД (этот уровень описания было бы правильнее называть логической схемой базы данных);

схему хранения, описывающую, как организовано хранение логических структур данных.

Фактически эта модель никогда не была полностью реализована ни в одной системе, однако ее удобно использовать для того, чтобы определять назначение тех или иных составляющих языка описания данных.

В идеале, трехуровневая модель обеспечивает возможности относительно независимой эволюции приложений и системы в целом. Так, при появлении новых приложений, использующих те же данные, достаточно определить новую внешнюю схему. В результате внедрение нового приложения не повлияет на работу других приложений.

Если новая версия приложения использует дополнительные элементы или структуры данных, достаточно определить новую внешнюю схему. Тогда старая и новая версии приложения смогут сосуществовать, что существенно упрощает постепенный и безопасный переход на новую версию.

Если для работы нового приложения требуются дополнительные структуры данных, эти структуры могут быть добавлены в концептуальную схему, что теоретически не повлияет на работу других приложений, так как их внешние схемы не будут содержать новых элементов данных.

Конечно, такая идеальная картина не всегда может реализоваться. Далеко не все изменения логической схемы могут остаться незаметными даже для приложений, которым не нужны измененные части, потому что изменения могут влиять не только на элементы данных, но и на взаимосвязи между ними.

В реальности эти потенциальные возможности могут реализоваться только в том случае, если разработчики как базы данных, так и приложений их тщательно учитывают. Эволюция базы данных имеет смысл в тех случаях, когда ценность накопленных данных высока. В других случаях создание новой базы данных может оказаться более оправданным решением. Далее в данном курсе будут рассмотрены примеры, иллюстрирующие оба подхода.

Развитые СУБД, в том числе PostgreSQL, предоставляют большое разнообразие методов хранения и поиска данных. Выбор этих методов влияет на производительность приложений, но не влияет на логическую структуру и на результаты выполняемых операций. Поэтому изменение структуры хранения можно использовать для того, чтобы повлиять на характеристики производительности отдельных приложений и системы в целом. Процесс внесения изменений, направленных на улучшение характеристик производительности и не затрагивающих логику работы приложений, называется *настройкой*.

В последние годы значение разделения описаний данных и программ зачастую недооценивается. Многие методологии разработки приложений предполагают генерацию схемы базы данных на основе объектной модели приложения. В частности, это характерно для методологий быстрой разработки прототипов, не предполагающих проведения тщательного предварительного анализа предметной области. Конечно, сложность при этом не исчезает, и задачи, связанные с проектированием схемы базы данных, все равно приходится решать, даже если они не выделены как отдельная единица работы.

Особенности проектирования схемы базы данных существенно зависят от применяемой модели данных. Различные типы моделей обсуждаются в главе 2. Здесь мы только отметим, что сложность проектирования может по-разному распределяться между базой данных и приложением: чем беднее модель данных, тем больше требуется делать на уровне приложения.

Наконец, заметим, что в некоторых подходах описание данных не оформляется как отдельная единица хранения (схема). Вместо этого описание данных может храниться вместе с самими данными (например, при использовании XML или JSON).

1.4. Языки запросов

Наличие описания логической структуры данных открывает возможности для выполнения достаточно сложных операций манипулирования данными внутри СУБД. Такие операции записываются на *языке запросов*.

Входящие в состав современных СУБД языки запросов являются декларативными, то есть позволяют описывать требуемый результат вычислений, но не способ выполнения этих вычислений. Благодаря этому СУБД может выбрать наиболее эффективные (по некоторому критерию производительности) алгоритмы получения результата. Это оказывается наиболее полезным при массовой обработке данных, так как, с одной стороны, позволяет исключить передачу промежуточных результатов между сервером базы данных и приложением и, с другой стороны, выбрать оптимальный способ выполнения вычислений с учетом характеристик фактически хранимых данных, что недостижимо при программировании эквивалентных операций в коде приложения.

Мощные средства обработки декларативных запросов, предоставляемые современными СУБД, в том числе PostgreSQL, зачастую используются недостаточно. Это происходит по многим причинам, среди которых можно выделить плохую совместимость декларативных средств языков запросов с императивными средствами массово применяемых языков программирования.

Практика разработки приложений без использования возможностей языков запросов привела к появлению ряда систем, не предоставляющих такие средства (так называемые NoSQL-системы). При использовании подобных систем для хранения данных, очевидно, часть функций СУБД переносится в приложение. Потенциально это может приводить к увеличению сложности приложения и стоимости его разработки либо к снижению качества — что во многих случаях оказывается допустимым.

1.5. Целостность и согласованность

В состав логической схемы базы данных могут включаться не только описания структур данных и зависимостей между ними, но и дополнительные условия, которым хранимые данные обязательно должны удовлетворять. Такие условия называются *ограничениями целостности* (integrity constraints). Система управления базами данных проверяет ограничения целостности при выполнении

любых изменений хранимых данных и не допускает выполнения операций, нарушающих эти ограничения.

Поддержка ограничений целостности на уровне СУБД позволяет существенно упростить разработку приложений и одновременно улучшить их качество, так как исключает необходимость обработки некорректных данных. Такие данные просто не могут быть записаны в базу данных и, следовательно, никогда не будут выданы в качестве ответа на запрос приложения.

В качестве ограничений целостности можно задавать только условия, которые не могут нарушаться ни при каких обстоятельствах. Существуют, однако, условия корректности другого типа, обычно связывающие значения нескольких элементов данных. В качестве примера таких условий чаще всего приводится правило, согласно которому суммарный баланс при переводе средств с одного бухгалтерского счета на другой не может измениться. Такие условия могут нарушаться для отдельных операций, однако удовлетворяются для набора из нескольких операций.

Состояния базы данных, в которых выполняются подобные условия, называются *согласованными* (consistent), а сами правила — условиями согласованности. Конечный набор операций, который переводит согласованное состояние в другое согласованное, называется *транзакцией*. Несмотря на то что каждое приложение обеспечивает согласованность при выполнении своих транзакций, при неконтролируемом параллельном или псевдопараллельном выполнении нескольких транзакций согласованность может нарушаться. Одной из важных функций СУБД является предотвращение нарушений согласованности при одновременной работе многих приложений (или одного и того же приложения от имени разных пользователей).

Различие между целостностью и согласованностью можно пояснить следующим образом. Ограничения целостности описываются условиями в базе данных, и СУБД отвечает за то, чтобы эти ограничения выполнялись. Условия согласованности определяются приложением и не могут быть проверены СУБД, однако она гарантирует, что результаты выполнения приложения (транзакции) не будут зависеть от факторов, находящихся вне контроля приложения, в том числе от работы других приложений, сбоев и отказов вычислительной системы.

Литература по базам данных изобилует упрощенными примерами из финансовой области приложений, однако ни в коем случае не следует отождествлять понятия транзакции в базах данных с банковскими транзакциями или другими транзакциями в смысле прикладных предметных областей. В реальности

даже самые простые банковские транзакции реализуются в информационных системах как комбинации из нескольких транзакций в базах данных.

В русскоязычной литературе зачастую термин *consistency* переводится как «целостность», что приводит к путанице, так как термин *integrity* переводится точно так же. В этой книге слово *целостность* всегда относится к ограничениям целостности (integrity constraints), а термин *согласованность* обозначает понятие, выражаемое термином *consistency*.

1.6. Отказоустойчивость

В наши дни информационные системы используются повсеместно: едва ли найдется предприятие или организация, в которой они бы не применялись. Во многих случаях работа информационной системы стала жизненно важной для основных производственных процессов, то есть отказы информационной системы приводят к остановке бизнес-процессов, а потеря данных приводит к катастрофическим последствиям (зачастую не только для производственных функций, но и для жизни людей или состояния окружающей среды).

Поэтому при разработке систем управления базами данных с самого начала очень большое внимание уделялось средствам, обеспечивающим отказоустойчивость данных и их выживаемость. В результате длительного развития технологий, связанных с базами данных, эта цель была достигнута.

Современные системы при соответствующей конфигурации могут гарантировать полную сохранность данных и восстановление после отказов оборудования в корректном (согласованном) состоянии. При необходимости система может быть организована таким образом, чтобы восстановление занимало доли секунды. Другими словами, СУБД способны обеспечить значительно более высокую надежность и доступность данных, чем надежность или доступность оборудования, на котором эти данные хранятся и на котором работают эти системы.

Однако создание высоконадежных систем неизбежно оказывается весьма дорогостоящим. Это связано с необходимостью многократного дублирования используемых средств на всех уровнях, начиная от оборудования, что приводит к существенному усложнению системы. В случае использования внешних сервисов, например при размещении данных в облачной среде, необходимо иметь запасные ресурсы, способные обеспечить выживание при отказе этой среды,

даже если она позиционируется как высоконадежная. Поэтому при проектировании системы следует выбрать такой уровень защищенности от отказов, который для нее действительно необходим.

Технологических ограничений, которые не позволяли бы в достаточной мере защитить данные, не существует; все случаи, когда потеря данных происходила, связаны с недооценкой рисков при проектировании системы.

1.7. Безопасность и разграничение доступа

Данные нуждаются в защите не только от отказов оборудования и природных явлений, но и от несанкционированного доступа. Все развитые системы содержат средства как для предотвращения доступа к базе данных от имени лиц, не имеющих на это права, так и для разграничения доступа к данным тех, кто такое право имеет. Допускается обработка (чтение или модификация) только тех данных, для которых соответствующие операции разрешены лицу, от имени которого они выполняются.

Подобные средства защиты реализуются не только на уровне СУБД: защитой приходится заниматься практически на всех уровнях и во всех компонентах информационной системы. Во многих простых системах средства защиты, предоставляемые СУБД, вообще не используются. Однако по-настоящему надежная защита должна быть многоуровневой, а некоторые виды защиты вообще невозможно реализовать без использования СУБД.

1.8. Производительность

Сравнение различных систем управления базами данных и оценка их применимости невозможны без учета их производительности. Для того чтобы такой учет был по возможности объективным, необходимы количественные метрики для измерения производительности.

Наиболее важными интегральными (то есть учитывающими несколько различных факторов) метриками являются *пропускная способность* и *время отклика* системы. Обе характеристики измеряются на определенной нагрузке системы. Обе метрики имеют смысл для очень широкого класса систем; уточнение

того, какие нагрузки имеет смысл рассматривать, зависит, конечно, от класса системы и от требований к ней. Для систем управления базами данных это может быть некоторая смесь запросов или других действий разной сложности. Когда такая нагрузка определена, можно измерить среднее количество подобных действий, выполняемых за единицу времени (пропускная способность), или среднее время выполнения одного действия (время отклика). Во многих случаях имеет смысл оценивать время отклика отдельно для каждого класса действий (в зависимости от их сложности для системы).

Необходимо подчеркнуть, что для достижения высокой пропускной способности может потребоваться конфигурация системы, отличающаяся от конфигурации, необходимой для достижения низкого времени отклика, и улучшение одной из этих характеристик совсем не обязательно приводит к улучшению другой. Этот факт можно пояснить следующим образом. Для получения высокой пропускной способности следует максимально использовать имеющиеся вычислительные ресурсы, что может приводить к росту очередей заданий, ожидающих выполнения, и, следовательно, к увеличению времени отклика за счет ожидания в очереди. С другой стороны, для сокращения времени отклика необходимо сократить время ожидания, в том числе в очередях, поэтому вычислительные ресурсы должны работать с неполной нагрузкой.

Для времени отклика важно также различать измерение на стороне клиента и на стороне сервера. Время выполнения не очень сложного запроса на сервере может оказаться значительно меньше времени пересылки текста запроса и возврата результата по вычислительной сети. В связи с этим может быть важно оценивать время отклика не для отдельных запросов или других операций с базой данных, а для каких-либо функций приложения в целом. Например, для веб-приложений наиболее важным вариантом времени отклика является время, необходимое для генерации HTML-страницы в ответ на запрос пользователя.

Для параллельных систем наиболее важной характеристикой является *масштабируемость* (scalability). В действительности масштабируемость не является отдельной характеристикой, а показывает, как изменяется другая характеристика при изменении нагрузки и размеров вычислительной системы. Можно говорить о масштабируемости пропускной способности или масштабируемости времени отклика, но не о масштабируемости вообще. Для того чтобы оценить масштабируемость, необходимо оценить какую-либо характеристику для системы, состоящей из одного вычислителя, на определенном объеме базы данных и определенной нагрузке и ту же самую характеристику для системы,

в которой количество вычислителей, количество данных и нагрузка (число запросов, например) все увеличены в N раз. Тогда масштабируемость выражается отношением второго значения к первому. Поведение масштабируемости по пропускной способности показано на рис. 1.8.1.

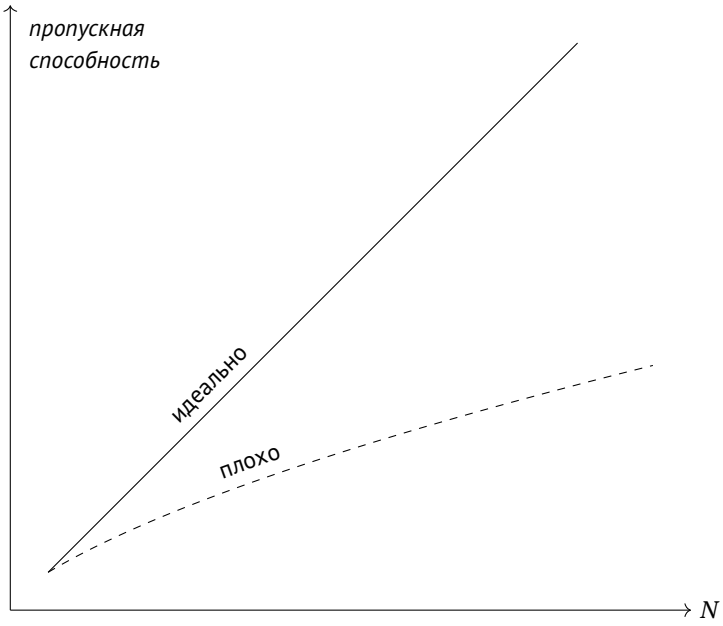

Рис. 1.8.1. Масштабируемость пропускной способности

Идеальное поведение масштабируемости по пропускной способности может быть представлено линейной зависимостью от N: система, содержащая в N раз больше оборудования и данных, способна обрабатывать в N раз больше запросов. В реальности такая масштабируемость недостижима, так как некоторая часть вычислительных ресурсов необходима для синхронизации работы параллельных вычислителей.

Для масштабируемости по времени отклика идеальная зависимость представляется константой: при увеличении количества вычислителей, объема данных и потока запросов время отклика не возрастает. Так же как и для пропускной способности, идеальная масштабируемость по времени отклика практически недостижима. Поведение масштабируемости по времени отклика иллюстрируется рис. 1.8.2.

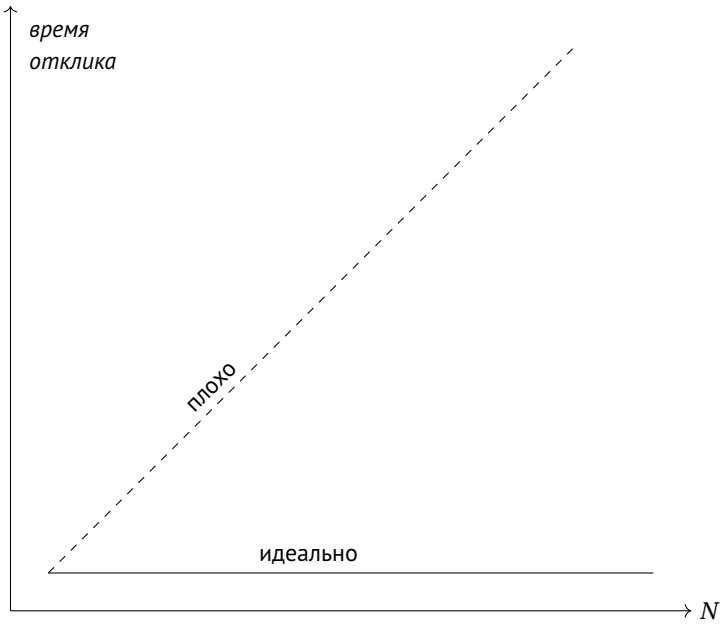

Рис. 1.8.2. Масштабируемость времени отклика

Еще одной характеристикой параллельных систем является *ускорение*, которое измеряется как отношение времени отклика на системе с одним вычислителем ко времени отклика на системе из N вычислителей.

Кроме характеристик производительности, имеется ряд других характеристик, из которых наиболее важной является *доступность* (availability). Доступность определяется как отношение времени нормальной работы системы к длине интервала времени, в течение которого измеряется доступность. Одновременное достижение высоких значений всех характеристик обычно требует значительного усложнения и удорожания системы.

Интегральные метрики, рассмотренные выше, полезны для оценки работы системы в целом, однако они мало полезны для оценки эффективности отдельных операций или запросов приложения. Поэтому в самой СУБД применяются метрики, значения которых можно предсказывать на основе информации об алгоритмах выполнения операций и статистических характеристик хранимых данных. Значением таких метрик является количество некоторых вычислительных ресурсов, необходимых для выполнения операции, например процессорное время или количество операций обмена с внешними устройствами,

а в случае параллельных или распределенных систем — еще и нагрузка на вычислительную сеть, которая может выражаться количеством сообщений, объемом передаваемых данных, количеством синхронных сообщений (с ожиданием ответа) и т. п.

Во многих случаях запросы приложений, сформулированные на высокоуровневом декларативном языке, могут быть выполнены многими различными способами, эквивалентными по результату, но использующими разное количество вычислительных ресурсов. Высокопроизводительные СУБД в таких случаях выбирают способ выполнения (план), для которого необходимое количество ресурсов минимально.

1.9. Создание приложений, взаимодействующих с базой данных

Современные СУБД поставляются с инструментами для создания и ведения баз данных. Эти инструменты предназначены для администраторов баз данных и помогают решать типовые задачи, такие как создание и изменение таблиц, редактирование записей, разграничение доступа и управление резервными копиями. Несмотря на то что эти программы обладают графическим интерфейсом, большинство задач удобнее решать, используя команды языка SQL.

Разумеется, такой интерфейс не подходит для бизнес-приложений. Система, ориентированная на массового пользователя, должна быть настолько понятна, чтобы пользователь, хорошо разбирающийся в предметной области, мог без всякого обучения начать с ней работать. Такие системы скрывают доступ к данным за графическим интерфейсом, который позволяет эффективно решить задачу и максимально ограждает пользователя от ошибок.

Для создания бизнес-приложений используется клиент-серверная архитектура. Ядро СУБД работает на сервере, а прикладная программа — на клиенте, причем сам клиент может быть сложным и состоять из нескольких уровней. Бизнес-логика может быть реализована как на сервере в виде хранимых процедур, так и на клиенте с использованием высокоуровневого языка программирования. Поскольку данные на клиенте и на сервере представлены по-разному, возникает проблема *потери соответствия* (impedance mismatch).

В случае объектно-ориентированных языков программирования и реляционных баз данных эта проблема называется объектно-реляционной потерей со-

ответствия. Реляционные базы данных не поддерживают основных концепций объектно-ориентированной парадигмы. Наибольшие затруднения при трансформации объектов в таблицы вызывает несоответствие системы типов, отображение наследования и многозначных связей, а также поддержка навигации между объектами.

Объектно-ориентированные языки пытаются решить проблемы при помощи каркасов объектно-реляционных отображений (object-relational mapping frameworks). Такие каркасы представляют собой библиотеки, написанные на объектно-ориентированном языке программирования. Разработчик приложения работает с привычными объектными моделями, которые автоматически преобразуются в таблицы и наоборот. Например, при операции сохранения каркас получает объект, отображает его в соответствующие таблицы и формирует SQL-запрос для вставки записи. При операции чтения каркас получает идентификатор объекта, формирует SQL-запрос для выборки данных, отображает найденные записи в объекты и возвращает их приложению.

Использование каркасов объектно-реляционных отображений ускоряет разработку, поскольку программисту не требуется глубоко знать ни SQL, ни реляционную теорию. Однако такие каркасы практически не используют специфические особенности конкретной СУБД, из-за чего тонкая настройка запросов становится невозможна. По-видимому, хорошим решением является использование каркасов для стандартных операций и чистого SQL для сложных запросов.

Попытка решить проблему несоответствия стала одной из причин создания баз данных NoSQL, которые представляют собой альтернативу реляционным СУБД. Такие базы данных не имеют структурированной схемы и работают напрямую с объектами. Преимущества и недостатки баз данных NoSQL рассматриваются в главе 9.

1.10. Итоги главы

В этой главе определены основные понятия, связанные с системами управления базами данных, обсуждены основные требования к таким системам, как они были определены исторически и как они эволюционировали на протяжении десятилетий существования СУБД. Более детально многие из этих тем разбираются в последующих главах.

1.11. Контрольные вопросы

Вопрос 1.1. Какие основные требования предъявляются к системам управления базами данных?

Вопрос 1.2. Какие основные компоненты содержит обобщенная трехуровневая модель данных ANSI/SPARC?

Вопрос 1.3. Каковы основные характеристики языков запросов в современных СУБД?

Вопрос 1.4. Что означает термин «независимость данных»?

Вопрос 1.5. Какие преимущества возникают при использовании независимости данных?

Вопрос 1.6. Что означает термин «согласованность данных»?

Вопрос 1.7. Что понимается под ограничением целостности в системах управления базами данных?

Вопрос 1.8. Как трактуются понятия безопасности и разграничения доступа в современных системах управления данными?

Вопрос 1.9. Какие основные метрики используются для оценки производительности?

Вопрос 1.10. Что такое архитектура клиент-сервер? Как распределяются программные компоненты?

Вопрос 1.11. Что называют объектно-реляционной потерей соответствия?