Краткое содержание

Оглавление

Питеру Хинтьенсу, от которого я получил свою первую подписанную книгу: вдохновляющему программисту распределенных систем, автору, философу и другу

Предисловие к русскому изданию

С детства я говорил на украинском и русском языках. Но ради того, чтобы как можно больше людей узнало о предмете настолько важном, как базы данных, я решил написать эту книгу на английском. Можете представить себе мой восторг, когда я узнал, что издательство «Питер», технические книги которого я помню еще с университетских лет, занимается переводом моей книги на русский.

Мы приложили все усилия для того, чтобы вы не только познакомились с устройством баз данных, но и могли использовать русский перевод книги как основу для дальнейшего изучения этой темы, и снабдили русские термины их английскими эквивалентами для того, чтобы помочь русскоязычному сообществу разработчиков баз данных стандартизировать терминологию.

Предисловие

Распределенные системы управления базами данных являются неотъемлемой частью большинства предприятий и подавляющего большинства программных приложений. Приложения отвечают за логику и взаимодействие с пользователем, в то время как системы управления базами данных (СУБД) обеспечивают целостность, согласованность и избыточность данных.

В 2000 году вы могли использовать лишь несколько разновидностей баз данных, большинство из которых были реляционными и в силу этого мало чем отличались друг от друга. Конечно, это не означает, что эти базы данных были совершенно одинаковыми, однако они имели много сходства в плане функциональности и сценариев использования.

Некоторые из этих баз данных ориентировались на *горизонтальное масштабирование* — повышение производительности и увеличение емкости за счет запуска нескольких экземпляров базы данных, действующих как единый логический блок: Gamma Database Machine Project, Teradata, Greenplum, Parallel DB2 и многие другие. Горизонтальное масштабирование и сегодня остается одним из самых востребованных свойств баз данных, что объясняется растущей популярностью облачных сервисов. Зачастую проще развернуть новый экземпляр и добавить его в кластер, чем производить *вертикальное масштабирование*, перенося базу данных на более крупную и мощную машину. Миграция часто требует больших затрат времени и усилий, что, в свою очередь, может приводить к простоям.

Примерно в 2010 году начал формироваться новый класс баз данных *с конечной согласованностью* и стали набирать популярность такие термины, как *NoSQL* и *большие данные*. За последние 15 лет сообщество разработчиков ПО с открытым исходным кодом, крупные интернет-компании и поставщики баз данных создали так много баз данных и инструментов, что можно легко запутаться, пытаясь разобраться в их специфических особенностях и сценариях использования.

В 2007 году разработчики компании Amazon опубликовали статью с описанием системы хранения данных Dynamo [DECANDIA07], которая произвела столь сильное впечатление на сообщество разработчиков баз данных, что за короткое время на свет появилось множество вариантов и реализаций этой системы. Наиболее известными среди них были такие системы хранения, как Apache Cassandra, разработанная компанией Facebook, Project Voldemort от компании LinkedIn и Riak от бывших разработчиков компании Akamai.

Сегодня ситуация в области систем хранения данных снова меняется: на смену хранилищам типа «ключ–значение», NoSQL и системам с конечной согласованностью стали приходить более масштабируемые и производительные базы данных,

способные выполнять сложные поисковые запросы с более высокой гарантией согласованности.

Кому предназначена эта книга

На конференциях мне часто задают один и тот же вопрос: «Как мне узнать больше о внутреннем устройстве баз данных? Я даже не знаю, с чего начать». Авторы большинства книг по СУБД не вдаются в детали реализации подсистем хранения данных, лишь в общих чертах описывая такие методы доступа, как использование B-деревьев. Поскольку лишь очень немногие книги подробно освещают такие актуальные концепции, как различные сценарии использования B-деревьев и журналированные подсистемы хранения, я обычно рекомендую читать статьи.

Каждый, кто пробовал читать статьи, знает, что это не так просто: вам часто не хватает контекста, формулировки могут быть неоднозначными, между статьями мало или вообще нет связи и их трудно найти. Эта книга содержит краткое описание важных концепций систем управления базами данных и может служить в качестве справочника для тех, кто хотел бы копнуть глубже, или в качестве шпаргалки для тех, кто уже знаком с этими концепциями.

Хотя далеко не все хотят стать разработчиками баз данных, эта книга поможет всем, кто создает программное обеспечение, использующее СУБД: разработчикам, специалистам по обеспечению надежности, архитекторам и руководителям проектных работ.

Если ваша компания использует какой-либо компонент инфраструктуры, будь то база данных, очередь сообщений, контейнерная платформа или планировщик задач, то вам следует читать журналы изменений проекта и списки рассылки, чтобы оставаться на связи с сообществом и знать о последних изменениях. Понимание терминологии и знание внутреннего устройства позволят вам извлекать больше информации из этих источников и более продуктивно использовать свои инструменты для поиска и устранения неполадок, выявления и предотвращения рисков и узких мест. Общее понимание принципов работы СУБД будет существенным подспорьем в том случае, если что-то пойдет не так. Используя эти знания, вы сможете выдвинуть некоторую гипотезу, проверить ее, найти первопричину проблемы и предоставить ее описание другим участникам проекта.

Эта книга также предназначена пытливым умам — людей, которые любят изучать что-то без непосредственной необходимости, тем, кто любит проводить время, взламывая что-то из чистого интереса, создавая компиляторы, собственные операционные системы, текстовые редакторы и компьютерные игры, изучая языки программирования и впитывая новую информацию.

Предполагается, что читатель имеет некоторый опыт разработки серверных систем и работы с СУБД в качестве пользователя. Хорошим подспорьем в усвоении материала книги также будет наличие некоторых знаний о различных структурах данных.

Зачем мне читать эту книгу?

СУБД часто описываются в терминах тех концепций и алгоритмов, которые они реализуют, т. е. указывается, что СУБД «использует gossip-протокол для распространения членства» (см. главу 12), «реализует Dynamo» или «соответствует описанию, предоставленному в статье, посвященной протоколу Spanner» (см. главу 13). Если же разговор идет об алгоритмах и структурах данных, часто можно услышать что-то вроде: «У ZAB и Raft есть много общего» (см. главу 14), «Bw-деревья — это что-то наподобие B-деревьев, реализованных поверх журналированной подсистемы хранения» (см. главу 6), или «они используют одноуровневые указатели, как в B^{link}-деревьях» (см. главу 5).

Чтобы говорить о сложных понятиях, нужны абстракции. Кроме того, невозможно каждый раз заново обсуждать терминологию, начиная какой-либо разговор. Наличие удобного инструмента в виде общего языка помогает переключить внимание на проблемы более высокого уровня.

Одно из преимуществ изучения фундаментальных концепций, доказательств и алгоритмов состоит в том, что они никогда не устаревают. Конечно, всегда будут появляться новые алгоритмы, однако это часто происходит как результат выявления недостатков или возможностей для улучшения в классическом алгоритме. Знание истории помогает лучше понять различия новых алгоритмов и мотивы их создания.

Изучение таких вещей вдохновляет. Вы видите разнообразие алгоритмов, видите, как последовательно решались проблемы в сфере баз данных, и начинаете ценить эту работу. Еще один положительный эффект состоит в том, что разрозненные фрагменты головоломки начинают складываться у вас в голове в цельную картину, которой вы всегда сможете поделиться с другими.

Рассматриваемые темы

Эта книга не посвящена реляционным или нереляционным (NoSQL) СУБД; она посвящена алгоритмам и концепциям, используемым во всех типах СУБД, с акцентом на *подсистеме хранения данных* и на компонентах, отвечающих за распределение.

Некоторые из рассматриваемых здесь концепций, включая, в частности, планирование и оптимизацию поисковых запросов, диспетчеризацию, реляционную модель и т. д., уже освещены в ряде отличных пособий по СУБД. Некоторые из этих концепций обычно описываются с точки зрения пользователя, в то время как в этой книге акцентируется внимание на их внутреннем устройстве. Вы сможете найти ссылки на полезную литературу в конце части II и в заключительных разделах каждой главы. В этих книгах вы найдете ответы на многие из имеющихся у вас вопросов относительно баз данных.

В этой книге не рассматриваются языки запросов, поскольку не существует общего языка, который можно было бы использовать для всех рассматриваемых здесь СУБД.

Чтобы собрать материал для этой книги, я изучил более 15 книг, более 300 статей и бесчисленное количество записей в блогах, файлы исходного кода и документацию по нескольким СУБД с открытым исходным кодом. Главным критерием при отборе рассматриваемых концепций был вопрос о том, говорят ли о той или иной концепции специалисты и ученые в области баз данных? Если ответ был утвердительным, я включал соответствующую концепцию в длинный список рассматриваемых тем.

Структура книги

Есть несколько примеров расширяемых баз данных с подключаемыми компонентами (например, [SCHWARZ86]), но они довольно редки. В то же время существует множество примеров использования базами данных подключаемого хранилища. Точно так же поставщики баз данных редко говорят о выполнении запросов, но всегда готовы поговорить о том, как поддерживается согласованность в предлагаемых ими базах данных.

Наиболее существенные различия между разными СУБД касаются используемых методов *хранения* и методов *распределения* данных. (Иногда играют важную роль и различия в других подсистемах, но мы не будем их рассматривать в этой книге.) Книга разбита на две части, в которых соответственно обсуждаются подсистемы и компоненты, отвечающие за *хранение* (часть I) и *распределение* (часть II).

В **части I** рассматриваются процессы внутри узлов и основное внимание уделяется подсистеме хранения данных — центральному компоненту СУБД и одному из наиболее важных отличительных аспектов. Мы начнем с архитектуры СУБД и рассмотрим несколько способов классификации СУБД в зависимости от среды хранения данных и структуры хранилища.

Затем мы рассмотрим структуры хранения данных и попытаемся понять, чем размещаемые на диске структуры отличаются от структур, размещаемых в оперативной памяти. Мы познакомимся с концепцией B-деревьев и изучим алгоритмы эффективной работы со структурами B-деревьев на диске, включая сериализацию, постраничную компоновку и представление данных на диске. Далее мы обсудим различные варианты концепции B-деревьев, чтобы наглядно увидеть ее мощь и оценить разнообразие структур данных, созданных под ее влиянием.

Наконец, мы обсудим несколько вариантов журналированного хранилища, широко используемых для реализации файловой системы и системы хранения, включая мотивы и причины их использования.

Часть II посвящена объединению нескольких узлов в кластер баз данных. Сначала мы узнаем, почему важно понимать теоретические принципы создания отказоустойчивых распределенных систем, чем распределенные системы отличаются от одноузловых приложений и с какими проблемами, ограничениями и сложностями приходится сталкиваться в распределенной среде.

После этого мы глубоко погрузимся в распределенные алгоритмы. Начнем с алгоритмов обнаружения сбоев, которые помогают повысить производительность и стабильность, выявляя сбои и сообщая о них, а также действуя в обход отказавших узлов. Поскольку для понимания многих алгоритмов, рассматриваемых далее в книге, необходимо понимание концепции «лидера», мы изучим несколько алгоритмов выбора лидера и поговорим о том, в каких случаях их лучше использовать.

Так как одной из самых сложных задач в распределенных системах является обеспечение согласованности данных, мы последовательно рассмотрим такие концепции, как репликация, модели согласованности, возможные расхождения между репликами и конечная согласованность. Поскольку в системах с конечной согласованностью часто применяется механизм антиэнтропии для обеспечения конвергенции, а также gossip-протокол для распространения данных, мы обсудим несколько подходов к их использованию. Завершит эту часть книги рассмотрение логической согласованности в контексте транзакций базы данных и алгоритмов консенсуса.

Было бы невозможно написать эту книгу, не опираясь на исследования и публикации других авторов. По ходу чтения вы встретите множество ссылок на статьи и публикации — они будут заключены в квадратные скобки: например: [DECANDIA07]. Вы можете использовать эти ссылки, чтобы изучить соответствующие концепции более подробно.

В конце каждой главы вы найдете раздел «Итоги», содержащий список дополнительной литературы, имеющей отношение к содержанию главы.

Условные обозначения

В этой книге используются следующие типографские условные обозначения.

Курсивный шрифт

Применяется для выделения новых терминов.

`Моноширинный шрифт`

Используется для записи листингов программ, а также для выделения в тексте таких элементов, как имена переменных и функций, базы данных, типы данных, переменные среды, операторы и ключевые слова.

Так обозначаются подсказки и советы.

Так обозначаются примечания общего характера.

Так обозначаются предупреждения и предостережения.

Благодарности

Это издание не состоялось бы без сотен людей, упорно работавших над научными трудами и книгами, которые стали источниками идей, вдохновения и служили справочниками.

Я хотел бы поблагодарить всех людей, которые рецензировали рукописи и оставляли отзывы, которые помогали убедиться, что материал в этой книге является корректным, а формулировки — точными: Дмитрий Алимов, Тимур Сафин, Питер Альваро (Peter Alvaro), Карлос Бакеро (Carlos Baquero), Джейсон Браун (Jason Brown), Блейк Эгглстон (Blake Eggleston), Маркус Эрикссон (Marcus Eriksson), Франсиско Фернандес Кастаньо (Francisco Fernandez Castano), Хайди Говард (Heidi Howard), Вайдехи Джоши (Vaidehi Joshi), Максимилиан Карась (Maximilian Karasz), Стас Кельвич (Stas Kelvich), Михаил Клишин, Предраг Кнежевич (Predrag Knežević), Джоэл Найтон (Joel Knighton), Евгений Лазин, Нейт Макколл (Nate McCall), Кристофер Мейклджон (Christopher Meiklejohn), Тайлер Нили (Tyler Neely), Максим Неверов, Марина Петрова, Стефан Подковинский (Stefan Podkowinski), Эдвард Рибьеро (Edward Ribiero), Денис Рысцов, Кир Шатров, Алекс Сорокоумов, Массимилиано Томасси (Massimiliano Tomassi) и Ариэль Вайсберг (Ariel Weisberg).

И конечно, эта книга не появилась бы на свет без поддержки моей семьи: жены Марины и дочери Александры, которые поддерживали меня на каждом шагу этого пути.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Пока что не вся терминология, связанная со сферой распределенных данных, устоялась в русскоязычном сообществе. Поэтому часть терминов в этой книге сопровождена их английскими вариантами.

ЧАСТЬ I

Подсистема хранения данных

Основной задачей любой СУБД является надежное хранение данных и обеспечение доступа к ним со стороны пользователей. Мы используем базы данных в качестве основного источника данных, помогающего обмениваться данными между различными частями приложений. Вместо того чтобы искать способ хранения и извлечения информации и изобретать новый способ организации данных при создании каждого нового приложения, мы используем базы данных. Таким образом, мы можем сосредоточиться на логике приложений, а не на инфраструктуре.

Для большей компактности вместо громоздкого термина «система управления базами данных» на протяжении этой книги мы будем в основном употреблять соответствующую аббревиатуру (СУБД) или просто выражение «база данных», имея в виду то же самое.

База данных представляет собой модульную систему, включающую в себя несколько составных частей: транспортный уровень, который принимает запросы, обработчик запросов, который выбирает наиболее эффективный способ выполнения запросов, подсистему выполнения, которая производит выполнение операций, и подсистему хранения (см. раздел «Архитектура СУБД» на с. 25).

Подсистема хранения данных (или ядро СУБД) — это программный компонент СУБД, отвечающий за хранение, извлечение и управление данными в памяти и на диске, предназначенный для работы с постоянной, долговременной памятью каждого узла [REED78]. В то время как базы данных могут отвечать на сложные запросы, подсистемы хранения рассматривают данные более детально и предлагают простой API для манипуляций с данными, позволяющий пользователям создавать, обновлять, удалять и извлекать записи. СУБД можно рассматривать как приложение, которое надстроено поверх подсистемы хранения и предлагает некоторую схему данных, язык запросов, индексацию, транзакции и много других полезных функций.

Для обеспечения гибкости и ключи, и значения могут быть произвольными последовательностями байтов без заданной формы. Их классификация и семантика представления определяются в подсистемах более высокого уровня. Например, вы можете использовать `int32` (32-разрядное целое число) в качестве ключа в одной из

таблиц и `ascii` (строку в кодировке ASCII) — в другой; с точки зрения подсистемы хранения оба ключа являются просто сериализованными записями.

Такие подсистемы хранения, как BerkeleyDB (https://databass.dev/links/92), LevelDB (https://databass.dev/links/93) и ее потомок RocksDB (https://databass.dev/links/94), LMDB (https://databass.dev/links/95) и ее потомок libmdbx (https://databass.dev/links/96), Sophia (https://databass.dev/links/97), HaloDB (https://databass.dev/links/98) и многие другие, были разработаны независимо от тех СУБД, в которые они теперь встроены. Использование существующих подключаемых подсистем хранения позволило разработчикам СУБД обойти этап их создания и сосредоточиться на других подсистемах.

В то же время четкое разделение между компонентами СУБД открывает возможность переключаться между различными подсистемами, потенциально более подходящими для конкретных сценариев использования. Например, MySQL, популярная система управления базами данных, имеет несколько подсистем хранения (https://databass.dev/links/99), включая InnoDB, MyISAM и RocksDB (https://databass.dev/links/100) (в составе MyRocks (https://databass.dev/links/101)). MongoDB позволяет переключаться между подсистемами хранения WiredTiger (https://databass.dev/links/102), In-Memory и (теперь уже устаревшей) MMAPv1 (https://databass.dev/links/103).

Сравнение баз данных

Выбор той или иной СУБД может иметь долгосрочные последствия. Если есть вероятность того, что база данных не подойдет из-за проблем с производительностью, обеспечением согласованности или выполнением операций, лучше выяснить это на раннем этапе цикла разработки, так как миграция на другую систему может оказаться нетривиальной. В некоторых случаях для этого потребуются существенные изменения в коде приложения.

У каждой СУБД есть свои сильные и слабые стороны. Для снижения риска дорогостоящей миграции можно потратить некоторое время перед принятием решения о выборе конкретной базы данных, чтобы быть уверенным в ее соответствии потребностям вашего приложения.

Попытка сравнить базы данных исходя из их компонентов (например, какую подсистему хранения они используют, каким образом данные совместно используются, реплицируются и распределяются и т. д.), их рейтинга (субъективная оценка, сделанная консалтинговыми агентствами, такими как ThoughtWorks (https://www.thoughtworks.com/de/radar), или сайтами со сравнением баз данных, такими как DB-Engine (https://db-engines.com/de/ranking) или Database of Databases (https://dbdb.io/)) или языка реализации (C++, Java или Go и т. д.) может привести к неверным и поспешным выводам. Такие методы можно использовать только для предварительного сравнения, и они могут быть такими же грубыми, как выбор между HBase и SQLite, поэтому даже поверхностное понимание того, как работает каждая база данных и что находится внутри нее, поможет принять более взвешенное решение.

Каждое сравнение должно начинаться с четкого определения цели, потому что даже малейшая предвзятость может полностью свести на нет все исследование. Если вам нужно, чтобы база данных хорошо подходила для имеющихся у вас (или ожидаемых) рабочих нагрузок, то лучше всего будет смоделировать эти рабочие нагрузки для различных СУБД, измерить важные для вас показатели производительности и сравнить результаты. Некоторые проблемы, особенно когда речь заходит о производительности и масштабируемости, начинают проявляться только через некоторое время или по мере роста емкости. Для выявления потенциальных проблем лучше всего провести длительные тесты в среде, максимально точно имитирующей реальное рабочее окружение.

Моделирование реальных рабочих нагрузок не только помогает понять, как работает база данных, но и позволяет научиться ее использовать и отлаживать, а также выяснить, насколько дружелюбно и полезно сложившееся вокруг нее сообщество. Выбор базы данных всегда определяется сочетанием этих факторов, и производительность часто оказывается не самым важным аспектом: лучше использовать базу данных, которая медленно сохраняет данные, а не быстро их теряет.

Чтобы сравнить базы данных, полезно тщательно изучить сценарий использования и определить текущие и ожидаемые переменные:

- размеры схемы данных и записей;
- количество клиентов;
- типы запросов и паттерны доступа;
- скорость выполнения запросов на чтение и запись;
- ожидаемые изменения в любой из этих переменных.

Знание этих переменных может помочь ответить на следующие вопросы:

- Поддерживает ли база данных необходимые запросы?
- Способна ли база данных обрабатывать тот объем данных, который мы планируем хранить?
- Сколько операций чтения и записи может обрабатывать один узел?
- Сколько узлов должна включать в себя система?
- Как расширить кластер с учетом ожидаемых темпов роста?
- В чем заключается процесс обслуживания?

Получив ответы на эти вопросы, вы сможете построить тестовый кластер и смоделировать свои рабочие нагрузки. Для большинства баз данных уже есть инструменты стресс-анализа, которые можно применить для воспроизведения конкретных сценариев использования. Если в экосистеме базы данных нет стандартного инструментария стресс-анализа для создания реалистичных рандомизированных рабочих нагрузок, это может стать настораживающим знаком. Если что-то мешает вам использовать инструменты, поставляемые вместе с самой базой, можно попро-

бовать один из существующих инструментов общего назначения или реализовать его с нуля.

Если тесты показывают положительные результаты, часто полезно ознакомиться с кодом базы данных. При изучении кода обычно лучше сначала выявить составные части базы данных, понять, как найти код различных компонентов, а затем пройтись по ним. Имея даже приблизительное представление о кодовой базе СУБД, вы сможете лучше разбираться в создаваемых ею записях журнала и ее параметрах конфигурации, а также выявлять проблемы в использующем ее приложении и даже в самом коде СУБД.

Было бы здорово, если бы мы могли использовать базы данных как черные ящики и никогда не заглядывать в них, но практика показывает, что рано или поздно возникает ошибка, сбой, регрессия производительности или какая-то другая проблема и лучше быть готовым к этому. Если вы знаете и понимаете внутренние компоненты базы данных, то можете снизить бизнес-риски и повысить шансы на быстрое восстановление.

Одним из популярных инструментов для тестирования, оценки производительности и сравнения является Yahoo! Cloud Serving Benchmark (YCSB) (https://databass.dev/links/104). YCSB предлагает инфраструктуру и общий набор рабочих нагрузок, которые могут быть применены к различным хранилищам данных. Как и все средства общего назначения, этот инструмент следует использовать с осторожностью, так как можно с легкостью прийти к неправильным выводам. Чтобы провести справедливое сравнение и принять обоснованное решение, необходимо потратить достаточно времени, чтобы понять реальные условия, в которых должна функционировать база данных, и соответствующим образом адаптировать сравнительные тесты.

СРАВНИТЕЛЬНЫЙ ТЕСТ TPC-C

Совет по оценке производительности обработки транзакций (Transaction Processing Performance Council, TPC) предлагает набор сравнительных тестов, которые поставщики баз данных используют для сравнения и рекламирования производительности своих продуктов. TPC-C — это тест обработки транзакций в реальном времени (Online Transaction Processing, OLTP), представляющий собой смесь транзакций только для чтения и обновления, которые имитируют распространенные рабочие нагрузки приложений.

TPC-C тестирует производительность и корректность выполняемых конкурентных транзакций. Основным показателем производительности является пропускная способность: количество транзакций, которые СУБД может обрабатывать в минуту. Выполняемые транзакции должны сохранять ACID-свойства и соответствовать (не противоречить) набору свойств, определяемых самим тестом.

Этот тест не относится к какому-либо конкретному бизнес-сегменту, но предоставляет абстрактный набор действий, важных для большинства приложений, для которых подходят базы данных OLTP. Он включает в себя несколько таблиц и объектов, таких как склады, товарные запасы, заказчики и заказы, и определяет макеты таблиц, сведения о транзакциях, которые могут быть выполнены с этими таблицами, минимальное количество строк в таблице и ограничения на уровень устойчивости данных.

Это не означает, что сравнительные тесты можно использовать *только* для сравнения баз данных. Сравнительные тесты могут быть полезны для определения и проверки положений соглашения об уровне обслуживания[1], определения системных требований, планирования производительности и многого другого. Чем больше вы узнаете о базе данных до ее использования, тем меньше времени придется тратить при эксплуатации.

Выбор базы данных — это долгосрочное решение, и потому желательно отслеживать выход новых версий, понимать, что именно изменилось и почему, и иметь некоторую стратегию обновления. Новые выпуски обычно содержат улучшения и исправления ошибок и проблем безопасности, но могут содержать и новые ошибки, вести к снижению производительности или неожиданному поведению, поэтому новые версии тоже важно тестировать перед их развертыванием. Проверка того, как разработчики базы данных проводили обновления ранее, может дать вам хорошее представление о том, чего ожидать в будущем. Если предыдущие обновления проходили гладко, это еще не гарантирует, что также будет и в дальнейшем, однако если они вызывали затруднения, это может быть признаком того, что будущие обновления тоже дадутся нелегко.

Понимание преимуществ и недостатков

В качестве пользователей мы можем видеть, как базы данных ведут себя в различных условиях, но при разработке баз данных мы должны принимать решения, оказывающие непосредственное влияние на это поведение.

Проектирование подсистемы хранения, безусловно, сложнее, чем просто реализация структуры данных из учебника: здесь имеется много деталей и пограничных случаев, с которыми обычно не удается справиться с первого раза. Нужно спроектировать физический макет данных и организовать указатели, принять решение о формате сериализации, понять, как данные будут удаляться при сборке мусора, как подсистема хранения вписывается в семантику СУБД в целом, выяснить, как заставить ее работать в конкурентном окружении, и, наконец, убедиться, что мы не будем терять данные ни при каких обстоятельствах.

Помимо того что вам нужно принять решения в отношении множества различных вещей, ситуация осложняется еще и тем, что в большинстве случаев эти решения подразумевают нахождение некоторого компромисса. Например, если мы будем сохранять записи в том же порядке, в котором они вносятся в базу данных, их можно будет сохранять быстрее, но если при этом их нужно будет извлекать в лексикографическом порядке, то их потребуется пересортировывать перед возвращением

[1] Соглашение об уровне обслуживания (service-level agreement, SLA) — это обязательство поставщика услуг относительно качества предоставляемых услуг. Помимо прочего, такое соглашение может включать информацию о задержке, пропускной способности, дрожании, а также количестве и частоте отказов.

результатов клиенту. Как вы увидите на протяжении этой книги, существует много различных подходов к разработке подсистемы хранения и каждая реализация имеет свои собственные плюсы и минусы.

При рассмотрении различных подсистем хранения мы будем обсуждать все их преимущества и недостатки. Если бы существовала подсистема хранения, идеальным образом подходящая для каждого возможного сценария использования, то все бы просто использовали его. Но поскольку такой подсистемы хранения не существует, мы должны подходить к выбору очень тщательно, принимая в расчет ожидаемые рабочие нагрузки и сценарии использования.

Существует множество различных подсистем хранения, использующих разные структуры данных и реализованных на разных языках, начиная с таких низкоуровневых языков, как C, и заканчивая такими высокоуровневыми языками, как Java. В то же время все подсистемы хранения сталкиваются с одинаковыми проблемами и ограничениями. Это можно сравнить с планированием городской застройки — планируя застройку под конкретное количество людей, вы можете увеличивать город в вертикальном или горизонтальном направлении. Хотя в обоих случаях город сможет принять одинаковое количество людей, эти подходы подразумевают совершенно разный образ жизни. При вертикальном росте застройки люди будут жить в квартирах и высокая плотность населения, вероятно, приведет к высокой интенсивности транспортного движения. С другой стороны, при менее плотном размещении люди, вероятно, будут жить в домах, но дальше от места работы.

Аналогичным образом проектные решения, принимаемые разработчиками подсистем хранения, делают их более подходящими для различных целей: одни из них обеспечивают минимальное время чтения и записи, другие — максимальную плотность (количество хранимых данных, приходящееся на каждый узел), а третьи — максимальную простоту эксплуатации.

В разделе «Итоги» в конце каждой главы вы найдете ссылки на полное описание применяемых в реализации алгоритмов и другую полезную информацию. Чтение этой книги должно хорошо подготовить вас для эффективного использования этих источников и дать прочное понимание того, какие имеются альтернативы для описываемых в них концепций.

ГЛАВА 1

Введение и обзор

СУБД служат различным целям: некоторые используются в основном для временных «горячих» данных, некоторые служат в качестве долговременного хранилища «холодных» данных, некоторые подходят для сложных аналитических запросов, некоторые позволяют получать доступ к значениям только по ключу, некоторые оптимизированы для хранения данных временных рядов, а некоторые эффективно хранят большие двоичные объекты. Сначала мы рассмотрим и обозначим различия, начав с краткой классификации и обзора, поскольку это поможет оценить масштаб дальнейшей дискуссии.

Терминология иногда может быть неоднозначной и трудной для понимания без принятия во внимание полного контекста. Примером могут служить различия между *колоночными* хранилищами и хранилищами *с широкими столбцами*, которые имеют очень мало общего между собой, или взаимосвязь *кластеризованных* или *некластеризованных индексов* и *индексированных таблиц.* Цель данной главы состоит в том, чтобы дать однозначное и точное определение таких терминов.

Мы начнем с обзора архитектуры СУБД (см. раздел «Архитектура СУБД» на с. 25) и обсудим отдельные ее компоненты и их функции. После этого мы осветим различия между различными СУБД с точки зрения используемой среды хранения (см. раздел «Резидентные и дисковые СУБД» на с. 27) и структуры (см. раздел «Колоночные и строчные СУБД» на с. 30).

Эти два способа разделения являются далеко не единственными вариантами классификации СУБД. Например, СУБД часто разделяют на три основные категории:

Базы данных для обработки транзакций в реальном времени (*online transaction processing, OLTP*)

Они обрабатывают большое количество поступающих со стороны пользователя запросов и транзакций. Запросы часто бывают предопределенными и с коротким жизненным циклом.

Базы данных для аналитической обработки данных в реальном времени (*online analytical processing, OLAP*)

Они обрабатывают сложные агрегаты данных. OLAP-системы часто используются для аналитики и построения хранилищ данных и способны обрабатывать сложные произвольные специальные запросы с длительным жизненным циклом.