Базы данных

Лекция 11 – Berkeley DB

Преподаватель: Поденок Леонид Петрович, 505a-5

+375 17 293 8039 (505a-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2023

2023.11.03

Оглавление

Коротко о типах БД

Группировку БД по типам можно выполнять на основе общих меток, которые наносят на БД их поставщики. Модные словечки:

- сетевой;
- реляционный;
- объектно-ориентированный;
- встроенный

плюс некоторое кросс-опыление — объектно-реляционный, встроенная сеть...

Понимание этих модных словечек важно – каждое из них имеет не только некоторое теоретическое обоснование, но и превратилось в практический ярлык для классификации систем, которые работают определенным образом. Все системы баз данных, независимо от модных словечек, которые к ним относятся, предоставляют несколько общих услуг. По сути, системы баз данных предоставляют две услуги — доступ к данным и управление данными

Первая услуга — доступ к данным

Доступ к данным означает:

- добавление новых данных в базу данных (вставка);
- поиск интересующих данных (поиск);
- изменение уже сохраненных данных (обновление);
- удаление данных из базы данных (удаление).

Все базы данных предоставляют эти услуги. Обычно записи базы данных имеют структуру, отличную от структур или экземпляров, поддерживаемых языками программирования, на которых пишут приложения. В результате работа с записями БД может означать:

1) использование операций с базой данных, таких как поиск и обновление записей;

2) преобразование между структурами языка программирования и типами записей базы данных в приложении.

Вторая услуга — управление данными

Управление данными сложнее, чем доступ к данным. Предоставление качественных услуг по управлению данными — сложная часть построения системы баз данных.

При выборе системы базы данных для использования в создаваемом приложении, важно убедиться, что СУБД поддерживает необходимые службы управления данными.

Услуги управления данными включают:

1) возможность одновременной работы нескольких пользователей с базой данных (параллелизм);
2) возможность мгновенного изменения нескольких записей (транзакции);
3) защиту от сбоев приложений и системы (восстановление).

Различные системы баз данных предлагают различные услуги по управлению данными.

Службы управления данными полностью независимы от служб доступа к данным.

Параллелизм означает, что несколько пользователей могут работать с базой данных одновременно. Поддержка параллелизма варьируется от нулевой (только однопользовательский доступ) до полной (много программ чтения и записи работают одновременно).

Транзакции — теория реляционных баз данных не требует, чтобы система поддерживала транзакции, но это требуется для большинства коммерческих реляционных систем.

Транзакции позволяют пользователям ***осуществлять сразу несколько изменений***. Например, перевод средств между банковскими счетами должен быть транзакцией, поскольку остаток на одном счете уменьшается, а остаток на другом увеличивается. Обе эти операции должны происходить одновременно (атомарно).

Свойства транзакций ACID

В системах баз данных транзакции имеют четко определенные свойства:

- они **атомарны** (atomic), так что изменения происходят все сразу или не происходят вообще;
- они **непротиворечивы** (consistent), так что база данных находится в допустимом состоянии, и когда транзакция начинается и когда она заканчивается;
- обычно они **изолированы** (isolated), что означает, что любые другие пользователи в базе данных не могут вмешаться в работу транзакций, пока они выполняются;
- они **долговечны** (durable), так что если система или приложение рухнет после завершения транзакции, изменения потеряны не будут.

Эти свойства атомарности, непротиворечивости, изоляции и устойчивости известны как свойства ACID.

Поддержка транзакций зависит от базы данных.

Некоторые предлагают атомарность, не давая гарантий долговечности.

Некоторые игнорируют изолируемость, особенно в однопользовательских системах, поскольку нет необходимости изолировать других пользователей от последствий изменений, в том случае, когда этих других пользователей нет.

Восстановление

Другой важной услугой по управлению данными является восстановление.

Восстановление — это процедура, которую система выполняет при запуске.

Цель восстановления — гарантировать, что база данных находится в надлежащем состоянии и ее можно использовать. Наиболее важно это после сбоя системы или приложения, когда база данных может быть повреждена. Процесс восстановления гарантирует, что внутренняя структура базы данных исправна. Восстановление обычно означает, что все завершенные транзакции проверяются, а любые потерянные изменения повторно применяются к базе данных. В конце процесса восстановления приложения могут использовать базу данных, как если бы не было перерыва в обслуживании.

Копирование данных

Наконец, существует ряд служб управления данными, позволяющих копировать данные.

Например, большинство систем баз данных могут импортировать данные из других источников и экспортировать их для использования в другом месте.

В большинстве систем предусмотрены способы ***резервного копирования баз данных и восстановления*** в случае сбоя системы, который повреждает базу данных.

Многие коммерческие системы допускают «горячее» резервное копирование, так что пользователи могут создавать резервные копии баз данных во время их использования, поскольку многие приложения должны работать без перерыва и не могут быть остановлены для резервного копирования.

Другие услуги

Конкретная система базы данных может предоставлять другие услуги по управлению данными.

- браузеры, отображающие структуру и содержимое базы данных;
- инструменты, обеспечивающие соблюдение правил целостности данных, таких, как правило, согласно которому ни у одного сотрудника не может быть отрицательной зарплаты.

Тем не менее, эти службы управления данными не являются общими для всех систем.

Службы управления данными, поддерживаемые большинством поставщиков баз данных:

- параллелизм;
- восстановление;
- транзакции.

Реляционные базы данных

Реляционные базы данных основаны на такой области математики, как теория множеств.

Реляционные базы данных работают с кортежами или записями, состоящими из значений нескольких различных типов данных, включая целые числа, строки символов и другие. Операции включают поиск записей, значения которых удовлетворяют некоторым критериям, обновление записей и т.д. Практически все реляционные базы данных используют какой-либо язык структурированных запросов, например, SQL.

SQL является основным практическим преимуществом систем реляционных баз данных — вместо того, чтобы писать компьютерную программу для поиска интересующих записей, пользователь реляционной системы может просто ввести запрос в простом синтаксисе, а движок (engine) сделает всю работу. Это дает пользователям огромную гибкость — им не нужно заранее решать, какой поиск они хотят выполнить, и им не нужны дорогостоящие программисты для поиска нужных данных.

Изучение SQL требует некоторых усилий, но для большинства целей это гораздо проще, чем полноценный высокоуровневый язык программирования.

Объектно-ориентированные базы данных

Объектно-ориентированные базы данных менее распространены, чем реляционные системы, но все же распространены широко. Большинство объектно-ориентированных баз данных изначально задумывались как постоянные системы хранения, тесно связанные с конкретными языками программирования высокого уровня, такими как C++ и, позже, Java.

Большинство из них теперь поддерживают более одного языка программирования, но объектно-ориентированные системы баз данных в основном предоставляют те же абстракции классов и методов, что и объектно-ориентированные языки программирования.

Объектно-ориентированные системы позволяют приложениям работать с объектами единообразно, независимо от того, находятся ли они в памяти или на диске. Эти системы создают иллюзию того, что все объекты постоянно находятся в памяти.

Объектно-ориентированные базы данных не так широко распространены, как реляционные системы. Чтобы привлечь разработчиков, разбирающихся в реляционных системах, во многие объектно-ориентированные системы добавлена поддержка языков запросов, очень похожих на SQL.

На практике объектно-ориентированные базы данных в основном используются для постоянного хранения объектов в программах на C++ и Java.

Сетевые базы данных

«Сетевая модель» — довольно старый метод управления данными приложения и навигации по ним. Сетевые базы данных предназначены для очень быстрого обхода по указателям.

Каждая запись, хранящаяся в сетевой базе данных, может содержать указатели на другие записи. Эти указатели, как правило, являются физическими адресами, поэтому выборка записи, на которую он ссылается, означает просто чтение ее с диска по ее дисковому адресу.

Системы сетевых баз данных обычно позволяют записям содержать:

- целые числа;
- числа с плавающей запятой;
- символьные строки;
- ссылки на другие записи.

Приложение, используя ссылки, может найти интересующие записи. После извлечения какой-либо записи приложение может быстро получить любую другую запись, на которую оно ссылается.

Обход по указателям выполняется быстро, поскольку большинство сетевых систем используют в качестве указателей физические дисковые адреса и для чтения нужной записи требуется только единственный доступ к диску, в отличие от других систем, которые, чтобы найти конкретную запись, должны выполнить более одного чтения с диска.

Ключевое преимущество сетевой модели является одновременно и ее основным недостатком — с одной стороны, приложения будут работать хорошо, с другой стороны, хранение указателей по всей базе данных очень затрудняет реорганизацию базы данных.

Клиенты и серверы

У поставщиков баз данных есть два варианта архитектуры системы:

- сервер, к которому подключаются удаленные клиенты, и все управление базой данных выполнятся внутри сервера;
- модуль, который напрямую подключается к приложению и выполняет все управление базой данных локально.

В любом случае разработчику приложения нужен какой-то способ связи с базой данных. Как правило, это интерфейс прикладного программирования (API), который работает в процессе или связывается с сервером для выполнения работы.

Почти все коммерческие продукты баз данных реализованы как серверы, а приложения подключаются к ним как клиенты. У серверов есть несколько особенностей, которые делают их привлекательными.

Во-первых, поскольку всеми данными управляет отдельный процесс и, возможно, на отдельной машине, легко изолировать сервер базы данных от ошибок и сбоев в приложении.

Во-вторых, поскольку некоторые продукты баз данных (в частности, реляционные механизмы) довольно велики, разделение их на отдельные серверные процессы позволяет приложениям оставаться небольшими, что требует меньше дискового пространства и памяти. Реляционные механизмы содержат код для разбора операторов SQL, их анализа и создания планов выполнения, их оптимизации и выполнения.

Наконец, хранение всех данных в одном месте и управление ими с помощью одного сервера упрощает организациям резервное копирование, защиту и настройку политик для своих баз данных.

Однако в некоторых случаях централизованное администрирование может быть недостатком. В частности, если необходимо создать приложение, использующее базу данных для хранения важной информации, то поставка (shipping) и поддержка приложения будет намного сложнее. Конечный пользователь должен установить и администрировать отдельный сервер базы данных, а программист должен поддерживать не один продукт, а два.

Berkeley DB

1) Berkeley DB — это встроенная библиотека баз данных, которая предоставляет приложениям масштабируемые, высокопроизводительные, защищенные транзакциями сервисы управления данными.

2) Berkeley DB является «***встроенной***», поскольку она напрямую связана с приложением. Она работает в том же адресном пространстве, что и приложение. В результате для операций с базой данных не требуется никаких межпроцессных взаимодействий ни по сети, ни между процессами на одном компьютере.

3) Berkeley DB предоставляет простой API вызова функций для доступа к данным и управления ими для ряда языков программирования, включая C, C++, Java, Perl, Tcl, Python и PHP.

4) Все операции с базой данных происходят внутри библиотеки. Несколько процессов или несколько потоков в одном процессе могут использовать базу данных одновременно, поскольку каждый из них использует библиотеку Berkeley DB.

5) Все низкоуровневое обслуживание, такое как блокировки, журналирование транзакций, управление общими буферами, управление памятью и т. д., библиотекой обрабатываются прозрачно.

6) Библиотека Berkeley DB чрезвычайно переносима. Он работает практически под всеми вариантами UNIX и Linux, Windows и многими встраиваемыми операционными системами реального времени.

7) Она работает как на 32-битных, так и на 64-битных системах.

8)Может быть развернута на высокопроизводительных интернет-серверах, настольных компьютерах, а также на карманных компьютерах, телеприставках, сетевых коммутаторах и в других местах.

9) Berkeley DB хорошо масштабируется. Сама библиотека базы данных довольно компактна (менее 300 килобайт памяти, занимаемой исполняемым кодом на обычных архитектурах), но при этом может использовать гигабайты памяти и терабайты дискового пространства, если дисковое оборудование это позволяет.

10) Каждый из файлов базы данных Berkeley DB может содержать до **256 терабайт данных** при условии, что базовая файловая система способна поддерживать файлы такого размера.

Приложения Berkeley DB часто используют несколько файлов базы данных и объем данных, которым может управлять приложение Berkeley DB, ограничен только ограничениями, накладываемыми операционной системой, файловой системой и физическим оборудованием.

11) Berkeley DB также поддерживает **высокий уровень параллелизма**, что позволяет тысячам пользователей одновременно работать с одними и теми же файлами базы данных.

Berkeley DB превосходит реляционные и объектно-ориентированные системы баз данных во встроенных приложениях по нескольким причинам:

1) поскольку библиотека работает в том же адресном пространстве, для операций с базой данных не требуется межпроцессное взаимодействие. Стоимость связи между процессами на одной машине или между машинами в сети намного выше, чем стоимость вызова функции.

2) поскольку Berkeley DB использует простой интерфейс вызова функций для всех операций, нет языка запросов для разбора и плана выполнения.

Службы доступа к данным

Приложения Berkeley DB могут выбирать структуру хранения, которая лучше всего подходит для приложения. Berkeley DB поддерживает:

- хеш-таблицы;
- B-деревья;
- простое хранилище на основе номеров записей;
- постоянные очереди (persistent queue).

Программисты могут создавать таблицы, используя любую из этих структур хранения, и могут смешивать операции с разными типами таблиц в одном приложении.

Хэш-таблицы

Хэш-таблицы, как правило, хороши для очень больших баз данных, которым требуется предсказуемое время поиска и обновления для записей произвольного доступа.

Хэш-таблицы позволяют пользователям спрашивать: «Существует ли этот ключ?» или получить запись с известным ключом.

Хэш-таблицы не позволяют пользователям запрашивать записи с ключами, близкими к известному ключу.

B-деревья

B-деревья лучше подходят для поиска на основе диапазона, например, когда приложению необходимо найти все записи с ключами между некоторым начальным и конечным значением.

B-деревья также лучше используют *окрестность* ссылки. Если приложение может одновременно работать с ключами, расположенными рядом друг с другом, B-деревья работают очень эффективно.

Древовидная структура хранит ключи, которые находятся рядом друг с другом в памяти, поэтому для извлечения ближайших значений обычно не требуется доступ к диску.

Номера записей

Хранение на основе номеров записей естественно для приложений, которым необходимо хранить и извлекать записи, но у которых нет простого способа генерировать собственные ключи — ключом для записи является номер записи в таблице номеров записей. Berkeley DB автоматически генерирует эти номера.

Очереди

Очереди хорошо подходят для приложений, которые создают записи, а затем должны обрабатывать эти записи в порядке их создания. Хорошим примером являются системы онлайн-покупок. Заказы могут поступать в систему в любое время, но, как правило, они должны выполняться в том порядке, в котором они были размещены.

Услуги по управлению данными

Berkeley DB предлагает важные услуги по управлению данными, включая:
- параллелизм;
- транзакции;
- восстановление.

Все эти услуги работают на всех структурах хранения.

Несколько пользователей могут работать с одной и той же базой данных одновременно.

Berkeley DB прозрачно обрабатывает блокировку, гарантируя, что два пользователя, работающие с одной и той же записью, не будут мешать друг другу.

По умолчанию библиотека обеспечивает строгую семантику транзакций ACID:
- атомарность (Atomicity);
- непротиворечивость (Consistentsy);
- изолированность (isolation);
- долговечность (durability).

Тем не менее, приложения могут ослаблять изоляцию, которую гарантирует система баз данных.

Несколько операций могут быть сгруппированы в одну транзакцию и могут быть зафиксированы или отменены атомарно.

Приложение при запуске может попросить Berkeley DB запустить восстановление.

Восстановление восстанавливает базу данных до согласованного состояния со всеми зафиксированными изменениями даже после сбоя. База данных гарантированно непротиворечива, и все зафиксированные изменения гарантированно будут присутствовать после завершения восстановления.

Чем Berkeley DB не является

Berkeley DB не является реляционной базой данных.
Berkeley DB не является объектно-ориентированной базой данных.
Berkeley DB не является сетевой базой данных.
Berkeley DB не является сервером базы данных.
В отличие от большинства других систем баз данных, Berkeley DB предоставляет относительно простые службы доступа к данным.

Записи в Berkeley DB представляют собой пары (ключ, значение). Berkeley DB поддерживает только несколько логических операций с записями:

- вставить запись в таблицу;
- удалить запись из таблицы;
- найти запись в таблице, поискав ее ключ;
- обновить уже найденную запись.

Berkeley DB никогда не работает со значением записи.

Значения — это просто полезная нагрузка, которая хранится вместе с ключами и надежно доставляется обратно в приложение по запросу.

И ключи, и значения могут быть произвольными строками байтов фиксированной или переменной длины.

В результате программисты могут помещать структуры данных на их любимом языке программирования в базу данных, не преобразовывая их сначала в формат чужой записи.

Хранение и извлечение очень просты, но **приложению необходимо заранее знать структуру ключа и структуру значения** — приложение не может запросить эту информацию у Berkeley DB, потому что Berkeley DB ее не знает.

Ключи, и значения могут иметь длину до четырех гигабайт, поэтому в одной записи могут храниться изображения, аудиопотоки или другие большие значения данных.

Berkeley DB не является реляционной базой данных.

Доступ к данным, хранящимся в Berkeley DB, осуществляется с помощью традиционных API-интерфейсов Berkeley DB.

Традиционные API-интерфейсы Berkeley DB — это способ, которым большинство пользователей Berkeley DB будут использовать Berkeley DB.

Хотя интерфейсы довольно просты, они нестандартны в том смысле, что не поддерживают операторы SQL.

Поддержка SQL — это палка о двух концах.

Одним из больших преимуществ реляционных баз данных является то, что они позволяют пользователям писать *простые декларативные запросы на языке высокого уровня*. Система базы данных знает все о данных и может выполнить команду. Соответственно, не требуется программирование, а также легко искать данные новыми способами и задавать новые вопросы базе данных.

С другой стороны, если программист может заранее предсказать, как приложение будет обращаться к данным, то написание низкоуровневой программы для получения и хранения записей может быть быстрее. Это устраняет накладные расходы на синтаксический анализ, оптимизацию и выполнение запросов. Программист должен понимать представление данных и должен написать код для выполнения работы, но как только это будет сделано, приложение может работать очень быстро.

В Berkeley DB нет понятия схемы и типов данных, как в реляционных системах. Схема — это структура записей в таблицах и отношения между таблицами в базе данных. Например, в реляционной системе программист может создать запись из фиксированного набора типов данных. Поскольку типы записей объявляются в системе, реляционная машина может проникать внутрь записей и проверять в них отдельные значения. Кроме того, программисты могут использовать SQL для объявления связей между таблицами и для создания индексов для таблиц. Реляционные механизмы обычно поддерживают эти связи и индексы автоматически.

В Berkeley DB ключ и значение в записи непрозрачны для Berkeley DB.

Они могут иметь сложную внутреннюю структуру, но библиотека об этом не знает. В результате Berkeley DB не может разложить часть значения записи на составные части и не может использовать эти части для поиска интересующих значений. Это может сделать только приложение, которое знает структуру данных.

Berkeley DB поддерживает индексы для таблиц и автоматически поддерживает эти индексы по мере изменения связанных с ними таблиц.

Berkeley DB не является реляционной системой. Системы реляционных баз данных семантически богаты и предлагают высокоуровневый доступ к базе данных. По сравнению с такими системами Berkeley DB представляет собой высокопроизводительную транзакционную библиотеку для хранения записей.

Поверх Berkeley DB можно построить реляционную систему.

Фактически, популярная реляционная система MySQL использует Berkeley DB для управления таблицами с защитой транзакций и берет на себя весь анализ и выполнение SQL. Он использует Berkeley DB для уровня хранения и предоставляет инструменты семантики и доступа.