Базы данных

Лекция 02 – Berkeley DB. Введение

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, 2024

2024.09.02

Оглавление

Berkeley DB....3
Службы доступа к данным....5
Хэш-таблицы....5
B-деревья....5
Номера записей....6
Очереди....6
Услуги по управлению данными....7
Чем Berkeley DB не является....8
Berkeley DB не является реляционной базой данных....9
Терминология и права....11
Где взять BDB?....12
Методы доступа....14
Выбор методов доступа....16
Выбор между BTree и Hash....17
Выбор между Queue и Recno....18
Окружение (environment)....19
Возвращение ошибок....21

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 (около мегабайта 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 иерархической 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.

1) 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 по ключу;
- обновить уже найденную запись.

2) 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.

3) 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 существенно повысить производительность базы данных и понизить на нее нагрузку. 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 все это программист реализует в приложении самостоятельно.

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 операторов SQL.

Она использует Berkeley DB для уровня хранения и предоставляет инструменты семантики и доступа.

Терминология и права

Что такое база данных Berkeley DB?
Это то же самое, что таблица реляционной базы данных (SQL)?

ORACLE: Да, концептуально база данных Berkeley DB представляет собой отдельную таблицу реляционной базы данных. Также вы можете рассматривать каждую пару ключ/значение как одну строку в таблице, где столбцы представляют собой данные, закодированные приложением либо в ключе, либо в значении.

Лицензирование

«Starting with release 12.1.6.0.x, we have moved under the standard AGPL license.»

Стандартная общественная лицензия GNU Афферо[1] – модифицированная версия обычной GNU GPL версии 3. В ней добавлено одно требование: если вы выполняете измененную программу на сервере и даете другим пользователям общаться на нем с этой программой, ваш сервер должен также позволять им получить исходный текст, соответствующий той модифицированной версии программы, которая выполняется на сервере.

Создана специально для таких программ, как веб-приложения, что бы пользователи, использующие изменённую программу по сети сеть, могли получить её исходный код.

1) GNU Affero General Public License

Где взять BDB?

1) В дистрибутивах Linux (5.3.28).

libdb	The Berkeley DB database library for C
libdb-cxx	The Berkeley DB database library for C++
libdb-utils	Command line tools for managing Berkeley DB databases
libdb-devel	C development files for the Berkeley DB library
libdb-cxx-devel	C++ development files for the Berkeley DB library
libdb-devel-static	
libdb-devel-doc	
libdb-sql	Development files for using the Berkeley DB with sql
libdb-tcl	The Berkeley DB database library for C

2) Собрать из исходников (`https://edelivery.oracle.com/akam/otn/berkeley-db`)

> **401 Oracle Export Error**
> In compliance with U.S. and applicable Export laws we are unable to process your request to https://edelivery.oracle.com/akam/otn/berkeley-db/db-18.1.32.tar.gz. Please contact RPLS-Ops_ww@oracle.com if you believe you are receiving this notice in error.

`ftp://student@lsi.bas-net.by/software/BerkeleyDB`
db-18.1.32.tar.g
db-18.1.40.tar.gz – Ошибка при установке. Решение: скопировать из 32 недостающее в дистрибутив.

Устанавливается по умолчанию в `/usr/local/BerkeleyDB.18.1`

```
./bin
./docs
./include
./lib
```

Соответственно, нужно самостоятельно прописывать пути к библиотекам и утилитам при использовании BDB[2].

2) Некоторые дистрибутивы включают каталоги /usr/local в пути поиска по умолчанию

Методы доступа

1) 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.

2) Записи в Berkeley DB представляют собой пары (ключ, значение).

3) 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.

4) И ключи, и значения могут быть произвольными строками байтов фиксированной или переменной длины. Соответственно, программисты могут помещать в базу структуры данных на любом языке программирования, не занимаясь преобразованием их в/из записей чужого формата.

5) 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.

6) Ключи, и значения могут иметь длину до четырех гигабайт (32 бита).

Метод доступа можно выбрать только при создании базы данных. После этого выбора фактическое использование API, как правило, идентично для всех методов доступа. То есть, хотя существуют некоторые исключения, в принципе, взаимодействие с библиотекой одинаково, независимо от того, какой метод доступа выбран.

Метод доступа, который предстоит выбрать, зависит, во-первых, от того, что необходимо использовать в качестве ключа, а во-вторых, от производительности, которая обеспечивается для данного метода доступа.

Btree

Данные хранятся в отсортированной сбалансированной древовидной структуре.

И ключ, и данные для записей BTree могут быть произвольно сложными.

Они могут содержать отдельные значения, такие как целое число или строка, или сложные типы, такие как структура. Кроме того есть возможность (это не поведение по умолчанию) двум записям использовать ключи, сравниваемые как равные.

Когда это происходит, записи считаются дубликатами друг друга.

Hash

Данные хранятся в расширенной линейной хеш-таблице.

Как и в случае с BTree, ключ и данные, используемые для хеш-записей, могут представлять собой произвольно сложные данные. Также, как и в BTree, опционально поддерживаются дублирующиеся записи.

Queue

Данные хранятся в очереди в виде записей фиксированной длины.

Каждая запись в качестве своего ключа использует логический номер записи. Этот метод доступа предназначен для быстрой вставки в хвост очереди и имеет специальную операцию, удаляющую и возвращающую запись из головы очереди.

Этот метод доступа необычен тем, что предусматривает блокировку на уровне записи. Это может улучшить производительность приложений, которым требуется одновременный доступ к очереди.

Recno

Данные хранятся в записях фиксированной или переменной длины.

Как и в **Queue**, в записях типа **Recno** в качестве ключей используются номера логических записей.

Выбор методов доступа

Чтобы выбрать метод доступа, необходимо сначала решить, что использовать в качестве ключа для записей базы данных.

1) если необходимо использовать произвольные данные (включая строки), следует использовать **`BTree`** или **`Hash`**.

2) если нужно использовать логические номера записей (по существу, это целые числа), следует использовать **`Queue`** или **`Recno`**.

После этого нужно выбрать между **`BTree`** или **`Hash`**, **`Queue`** или **`Recno`**.

Выбор между BTree и Hash

Для небольших рабочих наборов данных, которые полностью помещаются в память, никакой разницы между BTree и Hash нет – оба будут работать одинаково хорошо.

В такой ситуации рекомендуется использовать BTree, хотя бы по той причине, что большинство приложений БД используют именно BTree.

Основной отправной точкой выбора является именно ***рабочий набор данных***, а не весь набор данных. Многие приложения поддерживают большие объемы информации, но им требуется доступ только к небольшой части этих данных. Поэтому нужно принимать во внимание те данные, которые будут регулярно использоваться, а не общую массу всех данных, которыми управляет приложение.

Объемы данных имеют тенденцию неуправляемо расти и в конце концов могут не умещаться в памяти целиком, поэтому нужно быть более осторожным при выборе метода доступа.

В частности, имеет смысл выбирать:

BTree, если ключи имеют определенную ссылочную локальность.

То есть, если предполагается их сортировка и можно ожидать, что запрос для данного ключа, вероятно, будет сопровождаться запросом для одного из его соседей.

Hash, если набор данных очень большой.

Для любого заданного метода доступа БД должна поддерживать определенный объем внутренней информации. Однако объем информации, которую БД должна хранить для **`BTree`**, намного больше, чем для **`Hash`**. В результате по мере роста набора данных эта внутренняя информация может доминировать в кэше до такой степени, что для данных приложения остается относительно мало места.

В результате **`BTree`** может быть вынужден выполнять дисковый ввод-вывод гораздо чаще, чем **`Hash`** при том же объеме данных.

Если ваш набор данных станет настолько большим, что БД почти наверняка придется выполнять дисковый ввод-вывод для удовлетворения случайного запроса, тогда **`Hash`** по производительности превзойдет **`BTree`**, потому что у него меньше внутренних записей для обеспечения поиска.

Выбор между Queue и Recno

Queue или **Recno** используются, когда в качестве первичного ключа базы данных приложение хочет использовать номера логических записей.

Логические номера записей — это, по существу, целые числа, которые однозначно идентифицируют запись базы данных. Они могут быть как изменяемыми, так и фиксированными.

Изменяемый номер записи — это номер, который может меняться при сохранении или удалении записей базы данных.

Фиксированные номера логических записей никогда не меняются независимо от выполняемых операций с базой данных.

При выборе между **Queue** и **Recno** следует иметь ввиду:

Queue, если приложение требует высокой степени параллелизма.

Очередь обеспечивает блокировку на уровне записи (в отличие от блокировки на уровне страницы, которую используют другие методы доступа), и это может привести к значительному увеличению производительности для приложений с высокой степенью параллелизма.

Однако, **Queue** поддерживает только записи фиксированной длины. Поэтому, если размер данных, которые необходимо хранить, сильно различается от записи к записи, скорее всего лучше выбрать метод доступа, отличный от **Queue**.

Recno, если нужны изменяемые номера записей.

Окружение (environment)

Окружение — это, по сути, инкапсуляция одной или нескольких баз данных.

Сначала открывается окружение, а затем базы данных в этом окружении. При этом базы данных создаются/располагаются в локациях относительно домашнего каталога.

Окружения предлагают очень много функций, которые не может предложить автономная база данных BDB:

Окружения обычно не используются приложениями, работающими во встроенных средах, где важен каждый байт. Однако, если приложение требует чего-либо, кроме минимальной функциональности, без них не обойтись.

1) Файлы с несколькими базами данных

В BDB возможно содержать несколько баз данных в одном физическом файле на диске.

Это желательно для тех приложений, которые открывают несколько баз данных. Однако для того, чтобы в одном физическом файле содержалось более одной базы данных, приложение должно использовать *окружение*.

2) Поддержка многопоточности и многопроцессности

При использовании окружения такие ресурсы, как кэши в памяти и блокировки, могут совместно использоваться всеми базами данных, открытыми в данном окружении.

Окружение дает возможность использовать подсистемы, предназначенные для предоставления нескольким потокам и/или процессам доступа к базам данных BDB.

Например, можно использовать окружения, чтобы предоставить возможность реализовать параллельное хранилище данных (CDS — Concurrent Data Store), подсистемы блокировок и/или пулы буферов общей памяти.

3) Транзакционная обработка

BDB предлагает транзакционную подсистему, которая обеспечивает полную ACID-защиту базы данных при записи.

Для включения транзакционной подсистемы и получения идентификаторов транзакций требуется использовать окружения.

4) Поддержка высокой доступности (репликации)

BDB предлагает подсистему репликации, которая обеспечивает репликацию базы данных с одним мастер-набором данных с несколькими копиями реплицируемых данных только для чтения. Чтобы подключить эту подсистему и впоследствии ею управлять, требуется использовать окружения.

5) Подсистема протоколирования (регистрации)

BDB предлагает ведение журнала с опережающей записью тем приложениям, которые хотят получить высокую степень восстанавливаемости в случае сбоя как приложения, так и системы.

После подключения подсистема ведения журнала позволяет приложению выполнять два вида восстановления, «обычное» и «катастрофическое», с использованием информации, содержащейся в файлах журнала.

Возвращение ошибок

1) Интерфейсы БД всегда возвращают значение 0 в случае успеха.

2) Если операция по какой-либо причине не удалась, возвращаемое значение будет ненулевым.

3) Если произошла системная ошибка (например, в БД закончилось место на диске, или было отказано в доступе к файлу, или для одного из интерфейсов был указан недопустимый аргумент), БД возвращает значение **errno**.

Все возможные значения errno больше 0.

4) Если операция завершилась не из-за системной ошибки, но и не была успешной, БД возвращает специальное значение ошибки.

Например, при попытке получить данные из базы, а искомая запись не существует, DB вернет **DB_NOTFOUND**, специальное значение ошибки, означающее, что запрошенный ключ не отображается (map) в базе данных.

Все возможные значения таких специальных ошибок меньше 0.

DB предлагает программную поддержку для отображения возвращаемых значений ошибок.

1) Функция **db_strerror()** возвращает указатель на сообщение об ошибке, соответствующее возвращенному коду ошибки БД, аналогично C-функции **strerror()**. Она может обрабатывать кроме возвращаемых значений, специфичных для БД, также возвраты и системных ошибок.

2) Есть две функции ошибок: **DB->err()** и **DB->errx()**. Эти функции работают так же, как C-функция **printf()**, принимая строку формата в стиле **printf()** и список аргументов. Также она может добавить стандартную строку ошибки к сообщению, составленному из строки формата и других аргументов.

Документация не содержит man'ов.