МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инновационного непрерывного образования

Кафедра электронных вычислительных машин

Дисциплина: Системное программирование обеспечение вычислительных машин

ПОЯНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
на тему:
ВСТРАИВАЕМАЯ СУБД

Студент Д.А. Мисан

Руководитель Л.П. Поденок

МИНСК 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данное приложение является библиотекой-оберткой. Библиотека служит для встраивания в сложные системы для работы с БД.

В приложении используется не SQL база, а база данных с индексно-последовательными прямыми методами доступа (Berkeley DB).

На ее базе сделана библиотека-обертка (wrapper-library) на си, реализующая несколько простых функций:

- Add;
- Delete;
- Set;
- Get;
- Put;
- Next;
- Prev.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Теоретические сведения

Berkeley DB (BDB) — высокопроизводительная встраиваемая система управления базами данных, реализованная в виде библиотеки. BDB является нереляционной базой данных — она хранит пары «ключ — значение» как массивы байтов и поддерживает множество значений для одного ключа. BDB может обслуживать тысячи процессов или потоков, одновременно манипулирующих базами данных размером в 256 терабайт, на разнообразном оборудовании под различными операционными системами, включая большинство UNIX-подобных систем и Windows, а также на операционных системах реального времени.

Первая версия Berkeley DB была разработана в Университете Беркли во время разработки BSD версии 4.3 (июнь 1986 года). Netscape попросила авторов Berkeley DB улучшить и расширить библиотеку — в то время версию 1.85, — чтобы она удовлетворяла их требованиям к использованию в сервере LDAP и в браузере Netscape. Этот запрос привёл к созданию Sleepycat Software (купленной корпорацией Oracle в феврале 2006 года). Berkeley DB распространяется под лицензией Sleepycat Public License (англ.), которая была одобрена OSI и FSF. Программа поставляется с полным исходным кодом, средствами сборки, инструментами тестирования и документацией. Качество кода и практичность вместе со свободной лицензией привели к использованию Berkeley DB во многих свободных и открытых программах. В рамках техники двойного лицензирования Oracle также распространяет проприетарную лицензию на использование библиотеки в закрытых проектах.

Berkeley DB примечательна своей простой архитектурой в сравнении с другими системами баз данных, такими как, например Microsoft SQL Server и Oracle Database. Например, в ней отсутствует сетевой доступ — программы используют базу данных через вызовы внутрипроцессного API. Она поддерживает SQL в качестве одного из интерфейсов, начиная с версии 5.0, хотя и не поддерживает столбцы в таблицах в традиционном понимании на уровне внутренней архитектуры. Berkeley DB предполагает работу с парами ключ-значение, где ключ и значение могут иметь фиксированную или переменную длину, а функция сравнения ключей может быть написана и назначена прикладным программистом. Программа, которая использует БД, сама решает, как данные сохраняются в записи; БД не налагает ограничений на данные, хранимые в записях. Запись и её ключ оба могут иметь размер до четырёх гигабайт.

Berkeley DB поддерживает необходимые возможности баз данных, такие как ACID-транзакции, детальные блокировки, интерфейс распределённых

транзакций XA, горячее резервное копирование и репликацию. Berkeley DB может использоваться как средство для построения хранимых индексов, так и в качестве хранилища данных.

Oracle предлагает BDB в трёх вариантах:

- Berkeley DB — собственно библиотека на языке «C»;
- Berkeley DB Java — библиотека, переписанная на Java (поддержка Google Android, Apache Maven);
- Berkeley DB XML — библиотека на Си, реализующая XML-СУБД на основе Berkeley DB со средствами работы с XML (Xerces, XPath, XQuery, XQilla).

Berkeley DB входит в состав большинства дистрибутивов Linux. Существуют средства для работы с Berkeley DB на языках Perl, Python и других.

Индексно-последовательный метод доступа.

Первичный ключ – это атрибут физической записи с уникальным значением, по которому запись может быть однозначно идентифицирована.

Индексно-последовательный метод требует, чтобы файл данных был упорядочен по первичному ключу. Записи делятся на блоки. Далее создается индексный файл, который содержит ссылки на блоки записей. Обычно в него заносится либо наибольшее, либо наименьшее значение в блоке. Индексный файл также упорядочен, поэтому и для него можно создать другой индексный файл. При добавлении новой записи выполняется один из двух алгоритмов:

1. Запись сохраняется в отдельной области (область переполнения) и связывается с блоком, которому логически принадлежит.
2. Если запись невозможно добавить в некоторый блок, то блок делится пополам. В индексный файл записывается новый ключ.

Эффективность доступа может варьировать в зависимости от размера блока и числа индексных файлов. Эффективность хранения зависит от количества свободных физических блоков, отведенных на область переполнения, и размера блока.

Последовательности

Для работы с последовательностями необходимо ознакомиться с небольшим набором элементов языка: командами CREATE SEQUENCE, ALTER SEQUENCE и DROP SEQUENCE, которые используются для создания, изменения и удаления последовательностей, соответственно; функцией NEXT VALUE FOR, служащей для того, чтобы извлекать следующее значение в последовательности; процедурой sp_sequence_get_range, применяемой для

защиты непрерывного диапазона значений последовательности, и представлением sys.sequences, которое используется для запроса информации о существующих последовательностях. Начнем с основных принципов последовательностей, а затем перейдем к более близкому знакомству с их свойствами.

Последовательность — независимый объект в базе данных, в отличие от IDENTITY, свойства, привязанного к определенному столбцу в конкретной таблице. Основная и типичная форма определения последовательности — указать тип данных последовательности, начальное значение и приращение

1.2. Постановка задачи

На базе Berkeley DB создать библиотеку-обертку (wrapper-library), реализующую несколько простых функций:

- add – добавляет запись в БД;
- delete – удаляет запись из базы данных;
- set – устанавливает позицию в ключе;
- get – получает запись по значению ключа;
- put – сохраняет запись по значению ключа;
- next/prev – получает следующую/предыдущую запись в последовательности ключа.

2. ОПИСАНИЕ РЕАЛИЗАЦИИ

2.1. Выбор и обоснование средств реализации

Для разработки приложения используем официальную среду разработки - Microsoft Visual Studio.

Visual Studio более просторен в плане языков программирования, на которых можно писать приложения. Так же у Visual Studio есть просто огромный плюс в отладчике – панель, отображающая состояния объектов, значения переменных и т.д. в текущий момент выполнения программы.

Для реализации приложения была выбрана система программирования Microsoft Visual Studio от Microsoft, так как она предоставляет наиболее широкие возможности для программирования приложений под управлением операционной системы Windows.

Среда VS относится к классу инструментов ускоренной разработки программ и позволяет быстро и эффективно разработать приложение. Это достигается за счет двух характерных свойств VS: визуального конструирования форм и широкого использования библиотеки визуальных компонентов. Система программирования VS рассчитана на программирование различных приложений и предоставляет большое количество компонентов для этого.

Визуальное конструирование форм избавляет программиста от многих аспектов разработки интерфейса программы, так как VS автоматически готовит необходимые программные заготовки и соответствующий файл ресурсов.

Ключевой особенностью VS является возможность быстро создавать качественные приложения, не создавая при этом код большого размера. Таким образом, ускоряется работа по созданию приложения и в короткие сроки можно создать достаточно серьезные приложения.

Важной особенностью Visual Studio является возможность быстро создавать качественные приложения, не создавая при этом код большого размера. Таким образом, ускоряется работа по созданию приложения и в короткие сроки можно создать достаточно серьезные приложения.

Обёртка библиотеки является промежуточным слоем между прикладной программой и другой библиотекой или интерфейсом программирования приложений (API).

Целью написания обёртки библиотеки может быть обеспечение работоспособности библиотеки (API) в каком-либо (чаще скриптовом) языке, в котором прямой вызов функций этой библиотеки API затруднителен или невозможен.

Другой целью может быть обеспечение дополнительного удобства для прикладного программиста, например адаптация библиотеки к объектно-ориентированному стилю программирования, компенсация неудобного дизайна библиотеки и т.п.

Существуют также кроссплатформенные обёртки библиотек, скрывающие реализацию для разных операционных систем, например wxWidgets.

2.2. Алгоритмы функций

Функции приложения:

1. добавить запись в базу данных;

Самый простой способ добавить элементы в базу данных. Функция принимает пять аргументов:

1) db – дескриптор базы данных, возвращенный db_create;
2) txnid – дескриптор транзакции. В нашем простом случае мы не ожидаем восстановления базы данных после сбоя приложения или системы, поэтому мы не используем транзакции и оставим этот аргумент NULL;
3) key – ключевой элемент для пары ключ / данные, который мы хотим добавить в базу данных;
4) data – элемент данных для пары ключ / данные, который мы хотим добавить в базу данных;
5) flags – необязательные флаги, изменяющие базовое поведение интерфейса.

2. удалить запись из базы данных;

Метод DB-> del удаляет пары ключ / данные из базы данных. Пара ключ / данные, связанная с указанным ключом, отбрасывается из базы данных. При наличии дублированных значений ключа все записи, связанные с назначенным ключом, будут отброшены.

При вызове базы данных, которая была преобразована во вторичный индекс с использованием метода DB-> associate, метод DB-> del удаляет пару ключ / данные из первичной базы данных и всех вторичных индексов.

Если операция должна быть защищена транзакцией (кроме указания флага DB_AUTO_COMMIT), параметр txnid является дескриптором транзакции, возвращенным из DB_ENV-> txn_begin ; в противном случае NULL.

Значение флагов должно быть установлено в 0 или следующее значение:

I. DB_AUTO_COMMIT
   a. Заключите вызов DB-> del в транзакцию. Если вызов завершится успешно, изменения, внесенные операцией, будут восстановлены. Если вызов не удался, операция не внесла никаких изменений.

Если указанный ключ отсутствует в базе данных, метод DB-> del вернет DB_NOTFOUND. В противном случае метод DB-> del возвращает ненулевое значение ошибки при ошибке и 0 при успехе.

3. установить позицию в ключе;

Установка, в какую позицию будет делаться запись. Сделано это для удобства пользования.

4. получить запись по значению ключа;

В DB->get()паре метод извлекает ключ / данные из базы данных. Адрес и длина данных, связанных с указанным ключом, возвращаются в структуре, к которой относятся данные .

При наличии дублированных значений ключа, DB->get()вернется первый элемент данных для назначенного ключа. Дубликаты сортируются по:

- Их порядок сортировки, если была задана дублирующая функция сортировки;
- Любой явный курсор обозначает вставку;
- По порядку вставки. Это поведение по умолчанию.

Параметры

data

Данные DBT оперированы.

flags

Параметр flags должен быть равен 0 или одному из следующих значений:

- DB_CONSUME
- DB_CONSUME_WAIT
- DB_GET_BOTH
- DB_SET_RECNO

Кроме того, следующие флаги могут быть установлены путем побитового включения ИЛИ в параметре flags:

- DB_IGNORE_LEASE
- DB_MULTIPLE
- DB_READ_COMMITTED
- DB_READ_UNCOMMITTED
- DB_RMW

key

Ключ DBT.

pkey

Параметр pkey - это ключ возврата из первичной базы данных.

txnid

Если операция является частью указанной приложением транзакции, параметр txnid является дескриптором транзакции, возвращенным из DB_ENV-> txn_begin () ; если операция является частью группы параллельных хранилищ данных Berkeley DB, параметр txnid является дескриптором, возвращаемым из DB_ENV-> cdsgroup_begin () ; в противном случае NULL. Если дескриптор транзакции не указан, но операция выполняется в транзакционной базе данных, операция будет неявно защищена транзакцией.

5. сохранить запись по значению ключа;

Метод DB-> put сохраняет пары ключ / данные в базе данных. Поведение функции DB-> put по умолчанию заключается в вводе новой пары ключ / данные, замене любого ранее существующего ключа, если дубликаты запрещены, или добавлении дублирующего элемента данных, если дубликаты разрешены. Если база данных поддерживает дубликаты, метод DB-> put добавляет новое значение данных в конце набора дубликатов. Если база данных поддерживает отсортированные дубликаты, новое значение данных вставляется в правильное отсортированное местоположение.

Если операция должна быть защищена транзакцией (кроме указания флага DB_AUTO_COMMIT), параметр txnid является дескриптором транзакции, возвращенным из DB_ENV-> txn_begin ; в противном случае NULL.

Значение флагов должно быть равно 0 или одному из следующих значений:

DB_APPEND

Добавьте пару ключ / данные в конец базы данных. Для указания флага DB_APPEND основной базой данных должна быть база данных Queue или Recno. Номер записи, назначенный записи, возвращается в указанном ключе.

Существует небольшое поведенческое различие между методами доступа Recno и Queue для флага DB_APPEND. Если транзакция, включающая операцию DB-> put с флагом DB_APPEND, прерывается, номер записи может быть уменьшен (а затем перераспределен последующей операцией DB_APPEND) методом доступа Recno, но не будет уменьшен или перераспределен методом доступа Queue.

DB_NODUPDATA

В случае методов доступа Btree и Hash вводите новую пару ключ / данные, только если она еще не появилась в базе данных. Если пара ключ / данные уже присутствует в базе данных, возвращается DB_KEYEXIST . Флаг DB_NODUPDATA может быть указан только в том случае, если базовая база данных настроена для поддержки отсортированных дубликатов.

Флаг DB_NODUPDATA не может быть указан для методов доступа Queue или Recno.

DB_NOOVERWRITE

Введите новую пару ключ / данные, только если ключ еще не появился в базе данных. Если ключ уже появляется в базе данных, возвращается DB_KEYEXIST . Даже если база данных допускает дублирование, вызов DB-> put с установленным флагом DB_NOOVERWRITE завершится неудачей, если ключ уже существует в базе данных.

Кроме того, следующий флаг может быть установлен побитовым включением ИЛИ в параметре flags:

DB_AUTO_COMMIT

Заключите DB-> Put в транзакцию. Если вызов завершится успешно, изменения, внесенные операцией, будут восстановлены. Если вызов не удался, операция не внесла никаких изменений.

В противном случае метод DB-> put возвращает ненулевое значение ошибки при ошибке и 0 при успехе.

6. получить следующую запись в последовательности ключа;

Метод DBcursor-> c_get извлекает пары ключ / данные из базы данных. Адрес и длина ключа возвращаются в объекте, к которому относится ключ (за исключением случая флага DB_SET, в котором объект ключа не изменяется), а адрес и длина данных возвращаются в объекте, к которому относится ключ. данные относятся.

При вызове курсора, открытого в базе данных, которая была преобразована во вторичный индекс с использованием метода DB-> associate, методы DBcursor-> c_get и DBcursor-> c_pget возвращают ключ из вторичного индекса и элемент данных из первичного база данных. Кроме того, метод DBcursor-> c_pget возвращает ключ из первичной базы данных. В базах

данных, которые не являются вторичными индексами, интерфейс DBcursor-> c_pget всегда будет давать сбой и возвращать EINVAL.

Изменения в базе данных во время последовательного сканирования будут отражены в сканировании; то есть записи, вставленные за курсором, не будут возвращены, а записи, вставленные перед курсором, будут возвращены.

В базах данных Queue и Recno пропущенные записи (то есть записи, которые никогда не создавались явно или которые были созданы и затем удалены) будут пропущены во время последовательного сканирования.

Значение флага должно быть установлено на DB_NEXT

- Если курсор еще не инициализирован, DB_NEXT (DB_PREV) идентичен DB_FIRST (DB_LAST). В противном случае курсор перемещается на следующую (предыдущую) пару ключ / данные базы данных, и эта пара возвращается. При наличии дублированных значений ключа значение ключа может не измениться.
- Если база данных является базой данных Queue или Recno, DBcursor-> c_get с использованием флага DB_NEXT (DB_PREV) пропустит все ключи, которые существуют, но никогда не были явно созданы приложением, или те, которые были созданы и впоследствии удалены.
- Если курсор уже находится на последней (первой) записи в базе данных, метод DBcursor-> c_get вернет DB_NOTFOUND.

7. prev – получить предыдущую запись в последовательности ключа.

Метод DBcursor-> c_get извлекает пары ключ / данные из базы данных. Адрес и длина ключа возвращаются в объекте, к которому относится ключ (за исключением случая флага DB_SET, в котором объект ключа не изменяется), а адрес и длина данных возвращаются в объекте, к которому относится ключ. данные относятся.

При вызове курсора, открытого в базе данных, которая была преобразована во вторичный индекс с использованием метода DB-> associate, методы DBcursor-> c_get и DBcursor-> c_pget возвращают ключ из вторичного индекса и элемент данных из первичного база данных. Кроме того, метод DBcursor-> c_pget возвращает ключ из первичной базы данных. В базах данных, которые не являются вторичными индексами, интерфейс DBcursor-> c_pget всегда будет давать сбой и возвращать EINVAL.

Изменения в базе данных во время последовательного сканирования будут отражены в сканировании; то есть записи, вставленные за курсором, не будут возвращены, а записи, вставленные перед курсором, будут возвращены.

В базах данных Queue и Recno пропущенные записи (то есть записи, которые никогда не создавались явно или которые были созданы и затем удалены) будут пропущены во время последовательного сканирования.

Значение флага должно быть установлено на DB_PREV

- Если курсор еще не инициализирован, DB_PREV идентичен DB_FIRST (DB_LAST). В противном случае курсор перемещается на следующую (предыдущую) пару ключ / данные базы данных, и эта пара возвращается. При наличии дублированных значений ключа значение ключа может не измениться.
- Если база данных является базой данных Queue или Recno, DBcursor-> c_get с использованием флага DB_PREV пропустит все ключи, которые существуют, но никогда не были явно созданы приложением, или те, которые были созданы и впоследствии удалены.
- Если курсор уже находится на последней (первой) записи в базе данных, метод DBcursor-> c_get вернет DB_NOTFOUND.

3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для использования библиотеки необходимо подключить заголовочный файл, который находится рядом с библиотекой. Далее в своем проекте необходимо вызывать функции из подключённого заголовочного файла.

Примерный вариант использования библиотеки представлен ниже (Рис 1.).

```
#include <iostream>
#include "MathLibrary.h"

using namespace Database;
int main()
{
  db db;
  db.init();
  std::cout << "add elem 1-9" << std::endl;
  db.add("1");
  db.add("2");
  db.add("3");
  db.add("4");
  db.add("5");
  db.add("6");
  db.add("7");
  db.add("8");
  db.add("9");
  db.view();
  std::cout << "delete first elem" << std::endl;
  db.del();
  db.view();
  std::cout << "delete elem pos 4" << std::endl;
  db.set(4);
  db.del();
  db.view();
  std::cout << "next elem after pos 4" << std::endl;
  db.next();
  std::cout << "next elem after pos 5" << std::endl;
  db.next();
  db.set(3);
  std::cout << "prev elem before pos 3" << std::endl;
  db.prev();
  std::cout << "prev elem before pos 2" << std::endl;
  db.prev();
  std::cout << "set pos 5 and put elem 10" << std::endl;
  db.set(5);
  db.put("10");
  db.view();
  db.set(1);

  system("PAUSE");

  return 0;
}
```

Рис. 1. Пример кода для использования библиотеки

– Функция add – добавляет запись в БД;
– Функция delete – удаляет запись из базы данных;
– Функция set – устанавливает позицию в ключе;
– Функция get – получает запись по значению ключа;
– Функция put – сохраняет запись по значению ключа;
– Функция next/prev – получает следующую/предыдущую запись в последовательности ключа;
– Функция view выводит на экран таблицу базы данных.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было разработано мобильное приложение для платформы Android, позволяющее осуществлять обмен сообщениями в режиме реального времени. В качестве хранилища информации была использована облачная база данных, сконфигурировано и осуществлено подключение к API для интеграции. В приложении реализованы функции загрузки мультимедиа-файлов, а именно изображений в облачное хранилище, совместно с процессом сохранения нового сообщения в облачную базу данных. Производится обработка исключительных ситуаций, используются функции сортировки для сообщений таким образом, чтобы новые сообщения отображались в начале списка. В процессе работы программы активно используются возможности мобильного устройства к подключению к сети Интернет для загрузки и обновления информации, а также возможность отключения и подключения к сети в рамках мобильного приложения, посредством использования разрешений на доступ к системным функциям.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1 С. Макконелл «Совершенный код» / —Москва: Русская редакция, 2013.

2 Официальная документация Berkeley DB [Электронный ресурс]. – Режим доступа:https://docs.oracle.com/cd/E17276_01/html/programmer_reference/index.html. – Дата доступа: 22.05.2020

3 The C Book [Электронный ресурс]. – Режим доступа: https://publications.gbdirect.co.uk//c_book/. – Дата доступа: 22.05.2020