TBASE 存储管理子系统的设计与实现

学　　院　计算机科学与技术

专　　业　计算机科学与技术

年　　级　2007 级

姓　　名　韩智攀

指导教师　张坤龙

2011 年 6 月 12 日

摘　要

TBase 是一个功能齐全的小型教学数据库，它主要由存储管理子系统、事务管理子系统和查询优化子系统组成。论文设计并实现了 TBase 的存储管理子系统。

存储管理子系统包含空间管理、缓冲管理、存取方法管理三个模块。空间管理模块实现了页管理、槽页管理、空闲空间管理和存储单元管理，为其他模块提供文件读写支持。缓冲管理模块实现了 Clock、LRU、MRU 三种缓冲调度算法，可根据需求进行调整，提高整个系统的读写性能。存储管理模块实现了最基本的堆文件，还实现了 B 树索引和哈希索引，能够快速响应多用户的并发访问，提高系统的整体性能。

论文对存储管理子系统的实现进行了测试。黑盒测试的结果表明，系统各项功能都能正确执行。

关键字：教学数据库；存储管理子系统；空间管理；缓冲管理；存取方法管理

ABSTRACT

TBase is a simple database used for teaching that is almost full-featured. It consist of storage manager, transaction manager and query optimizer. This paper design and implement the storage manager of TBase.

Storage manager is made up of space manager, buffer manager and accessing methods manager. Space manager provides the functions of page managing, slotted page managing, free space managing and storage container managing, and it provides interface for data reading and writing. Buffer manager implements three replacement algorithm: Clock replacement algorithm, LRU replacement algorithm, MRU replacement algorithm. It will improve the performance of data reading and writing. Access method manager implements based heap data structure. It also implements B-tree index and hash index. It gives fast response to multi-user accessing and improves the performance of the system.

This paper finally gives a test of the storage manager. Result of Black-box Test shows that the functions mentioned above performing correctly in accordance with the design principles.

Key words: database or teaching；storage management subsystem；space manager；buffer manager；accessing methods manager

目　　录

第一章　　绪论

随着当今社会人类文明的不断发展，人们所需要存储的信息量急速膨胀。数据库的管理已经与日常生活密不可分，数据库管理系统的应用也越发重要。

数据库管理系统[1]储管理子系统解决数据存储问题，它保证了存储数据的独立性和完整性，是数据库管理系统准确稳定快速运行的基本保障。

TBase 具有存储管理，事务处理和查询处理三大部分，本次毕业设计仅对存储管理子系统进行研究。

1.1　国内外发展状况

存储引擎，也称数据库引擎，是数据库管理系统[1]中的一个软件模块，负责插入、查找、更新、删除数据。

存储引擎已经被研究了多年，大致有如下两种：一种作为独立的产品，比如 Berkeley DB[2]；一种作为数据库管理系统中的一个组成部分。后一种又可被细分为两类，一类如SQL Server、Oracle的存储引擎组件，存储引擎作为一个固定的成员，内嵌在数据库管理系统中，另一类如Mysql的MyISAM、InnoDB、MERGE等[3]，用户可以根据不同的使用环境进行选择。

国外数据库行业发展比较迅速，其中比较主流的商业数据库产品有Oracle、Sybase、微软的SQL server。但由于它们不会为我们提供源码，所以对于教学研究的意义不大。国外主流的开源数据库有MySQL、PostgreSQL、Ingres、MaxDB、InterBase（即Firebird），这些数据库虽然提供源码，但是代码量大，并且相对较为复杂，不适于教学使用。国外还有许多教学用的数据库，代码量虽然小，但是可以让学生了解数据库内部的工作原理和实现方法。

因此，在国内数据库的学习和研究还有很多的事情可以去做，还有很大的研究发展空间。

存储管理子系统，是现代数据库的一个重要的组成部分。存储管理子系统所要实现的功能包括空间管理、缓冲管理和存取方法管理[3]。

存储管理子系统为数据库的其他部分提供了数据存储相关的支持，并且保证了良好的数据存储效率。数据库内部涉及存储相关的操作，都需要利用存储管理子系统提供的接口来完成。

1.2　TBase 的整体架构

TBase 数据库是基于传统的关系数据库的概念而设计的一个简易的教学数据库。它支持了传统数据库中的存储管理、事务管理、查询优化三部分。

TBase 数据库有如下特色：

1、 事务性：TBase 数据库支持 ACID 事务处理特性。

2、 多线程：TBase 数据库是多线程的，支持多个线程同时读写。

3、 先写日志：TBase 数据库实现了 WAL 恢复管理策略。

4、 基于加锁的并发控制：TBase 数据库实现了共享锁、互斥锁来管理并发访问。

5、 B 树索引：TBase 数据库实现了 B 树索引结构，它支持并发的读取、插入和删除数据。

6、 死锁检测：TBase 数据库支持死锁预防，设置了周期性检查锁表的后台线程，负责中断死锁事务。

TBase 数据库内部包含三个大的功能单元：存储管理、事务处理和查询优化。其体系结构如图 1-1 所示。

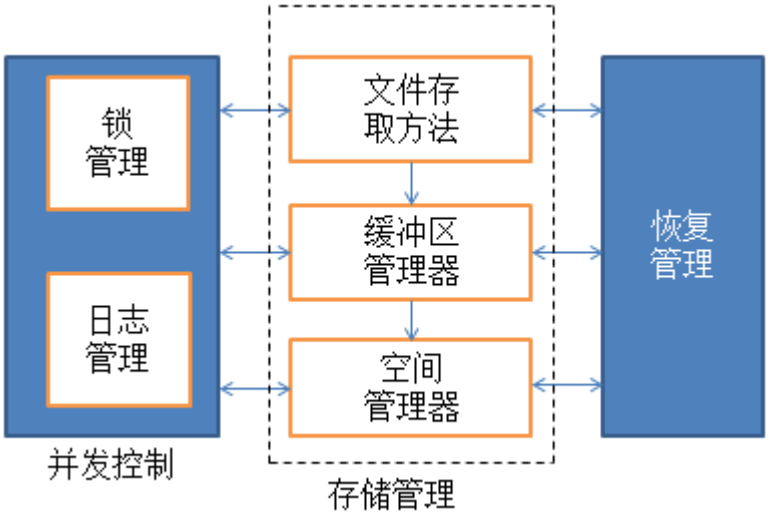

图 1-1 TBase 的体系结构

1.3 论文组织结构

第一章是绪论，概要地介绍本篇论文的研究背景，研究内容，以及所做的工作。

第二章是需求分析，包括系统功能需求，技术需求，模块划分，可行性分析，结构需求分析等。

第三章是空间管理模块，从设计原理，具体实现，正确性测试等几个方面，介绍存储管理子系统的空间管理模块。

第四章是缓冲管理模块，从设计原理，具体实现，正确性测试等几个方面，介绍存储管理子系统的缓冲管理模块。

第五章是存取方法管理模块，从设计原理，具体实现，正确性测试等几个方面，介绍存储管理子系统的存取方法管理模块。

第六章是总结与展望，总结本篇论文的成果，简要分析不足之处，并展望今后进一步的研究工作。

第二章　　需求分析

2.1　功能需求

存储管理子系统所要实现的基本功能就是能够为输入数据分配合适的存储空间，并能够保证存储的数据能被快速地访问[8]。在本论文中，存储管理子系统还必须能够支持事务处理，在多用户条件下，保证系统运行的正确性和高效性。

存储管理子系统需要实现的功能包括空间管理、缓冲管理、存取方法管理，每个模块的具体需求如下：

空间管理模块要能够根据不同的用户需求，分配合理的存储空间，维护整个系统的空间分配信息，保证用户需要存储的信息的准确性和持久性。并且，能够根据用户的需求，在合适的时间完成存储空间的整理。

缓冲管理模块要能够提供设计良好的缓冲调度策略，保证在大多数的情况下都能保持良好的性能，避免由于大量的缓冲页命中失效而引起的系统性能的下降。

存取方法管理模块需要能够提供高效的数据存取方法，保证用户能快速地存取数据，并且，在多用户并发访问的条件下，要保证能够正确、高效地完成数据访问。

2.2　开发环境与技术需求

存储管理子系统在开发过程中主要用到的工具和标准如表 2-1 所示。

表 2-1　开发环境

名称描述	具体内容
开发语言	C#
架构模型	C/S
开发平台	Microsoft Windows 7 Ultimate .Net Framework 4.0
IDE 工具	Visual Studio 2010

因为.net 平台上的教学数据库比较少见，而 C#的语言易于阅读，所以本系统采用 C#语言进行开发[7]。

系统运用了面向对象程序设计的方法[8]，并且灵活运用了 C#的安全性和错误处理机制，还采用了 C#的迭代器等库函数类来实现系统的基本功能[9]。

2.3　模块划分

存储管理子系统的功能包括空间管理、缓冲管理和存取方法管理。TBase 存储管理子系统的模块划分如图 2-1 所示。

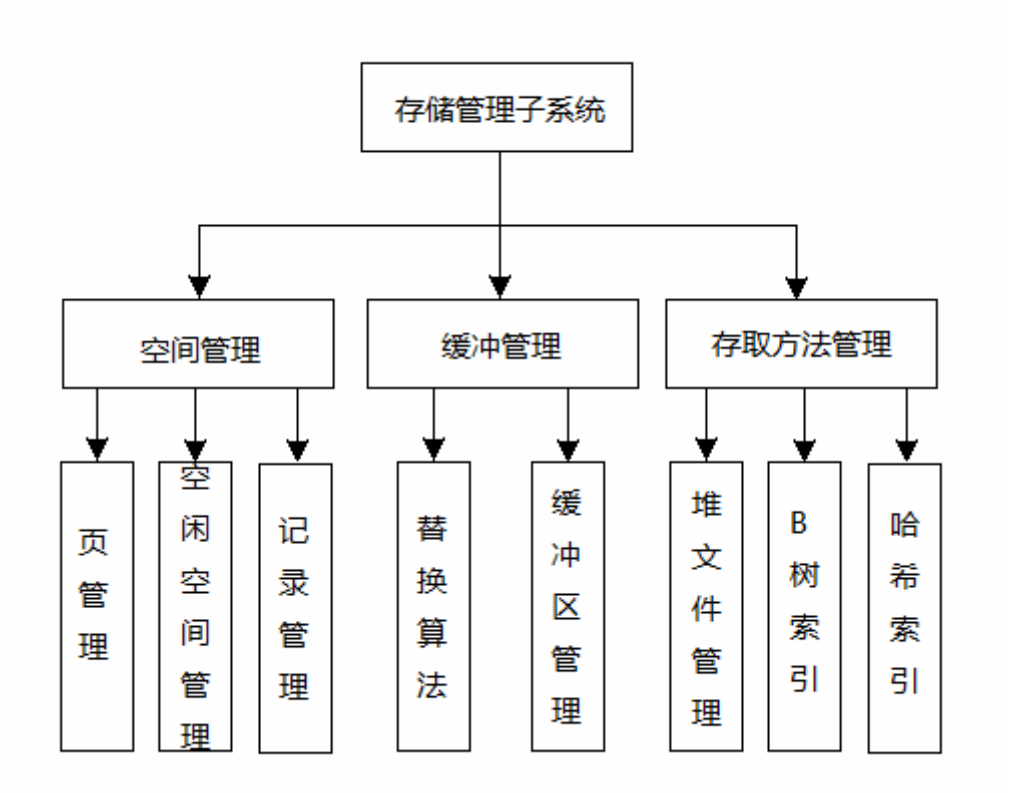

图 2-1 TBase 存储管理子系统的模块划分

2.4 接口需求分析

接口的需求分析分三部分来介绍。

空间管理模块对上层无接口提供，它主要是自行管理数据库的空间，不允许上层对其进行干扰，但是为存取方法层面提供了读取数据等一系列接口，为缓冲管理模块提供了最底层的 pin 和 unpin 操作等操作。

缓冲管理模块为事务处理部分提供了将页面装入缓冲区，pin 和 unpin 操作，其余操作均是封装在内部，不对外提供的。

存取方法为上层提供了读取记录、插入记录、删除记录等接口，将接口的实现封装起来，上层的应用变得很容易。

第三章　空间管理模块

3.1　设计原理

数据库中，系统为每一个表分配一个文件，文件中 1024 字节为一页，页中也分槽，每个槽存储一个记录。

3.1.1　页管理子模块

数据库系统存储的基本单位是叫做页，它是一个长度为 1024 的连续的字节数组。页管理子模块负责对系统的页进行管理。

TBase 数据库中，利用 C#中的文件读写来与操作系统中的文件进行对应，将文件划分为 1024 字节为一块，对应于数据库中的一页。

页管理子模块不需要管理页内部存储的内容，它只是维护页面最基本的信息，比如：页编号，页面 lsn，页类型等。这样，其他模块可以扩展基本的页类型，并且根据不同的需要增加附加特性[10]。

槽页结构提供一种增强版的页，它允许在页上的特定位置插入、更新、删除数据。槽页提供了一个内部允许分段，而整体上又统一的页。槽页内的每一条分段数据被称作一条记录。

通过这样一个基础的结构，其他模块能够在此之上实现更加复杂的功能。

页面内的每条记录会被分配一个 Slot Number，即槽号。槽号从 0 开始计数，比如：页面内的第一条记录可以使用 0 来访问。

另外，为了存储记录数据，每个槽被设计为要存储一组 short 型的标记值。可以通过这些标记值向客户端传递一些信息。

系统会为每个槽分配一个槽指针，槽管理子模块可以利用 Slot Position 在指定的位置插入、更新、删除记录。

3.1.2　空闲空间管理子模块设计

空闲空间管理[13]，实际上就是为新数据找到合适的位置存储。

论文使用空间映射页来支持空闲空间管理，系统会分配一个空间映射页[5]。系统启动时，映射页初始化为所有页面的映射位都置为空闲，当需要重新分配页面时，先将映射页对应的映射位置为使用，之后再为数据库分配新页。以此来进行空闲页面的管理[15]。

3.1.3　类图

空间管理模块的类图如图 3-1 所示。

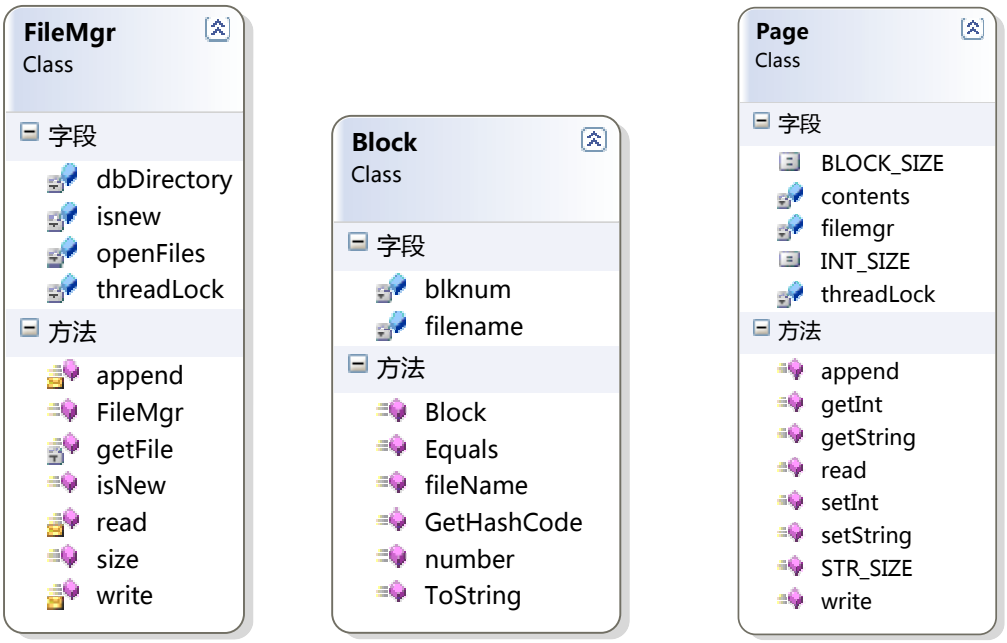

图 3-1 空间管理模块的类图

3.2 具体实现

根据空间管理模块的设计，我们可以轻易的完成对于空间管理模块的实现，具体代码包含三个类：Block、FileMgr、Page。

3.1.1 Block

概述：Block 类提供的是找到某一页的唯一标识，由文件名和页号构成。

成员变量：

```
private string filename;
private int blknum;
```

filename为string类型，表示这个页包含于哪个文件中，而blknum为int类型，表示该页在文件中的页号。运用这个类的实体可以唯一确定一个页面。

成员方法：

public Block(string filename, int blknum)

描述：构造函数，根据指定的文件名和指定的页号创建一个 Block 标识类。

入口参数：filename，指定的文件名；blknum，指定的页号。

实现：将参数列表中的参数值赋值给成员参数。

public string fileName()

描述：返回 Block 标识类中的文件名属性。

入口参数：无。

实现：将类中的 filename 的变量值作为返回值返回。

public int number()

描述：返回 Block 标识类中的页号属性。

入口参数：无。

实现：将类中的 blknum 的变量值作为返回值返回。

public override bool Equals(object obj)

描述：判断当前类与指定的类是否相等。

入口参数：obj，相比较的类。

实现：分别将两个类的 filename 和 blknum 做比较，若两属性值都相等，则表示两个类相等，返回 true，否则两个类不相等，返回 false。

public override string ToString()

描述：将当前类转换为字符串。

入口参数：无。

实现：分别将两个属性 filename 和 blknum 转换为字符串，添加到字符串“file @，block @”中的@位置。

3.1.2 FileMgr

概述：此类为 TBase 提供了最底层的文件操作，是 TBase 的文件管理器，例如文件的读、写等操作。

成员变量：

```
private string dbDirectory;
private bool isnew;
private Dictionary<string, FileStream> openFiles = new Dictionary<string, FileStream>();
private object threadLock = new object();
```

dbDirectory 为 String 类型，表示数据库存储的系统目录；isnew 为 bool 类型，表示文件是否为新建文件；openFiles 为字典类型，将打开的文件的文件流与其文件名相对应；threadLock 为 object 类，用于线程锁，无实际意义。

成员方法：

public FileMgr(DirectoryInfo di)

描述：构造函数，根据给出的目录信息来创建文件管理器。

入口参数：di，当前的目录信息。

实现：将di转换成字符串赋值给dbDirectory变量，如果di是新的，那么isnew就赋值为true，否则赋值为false。

internal void read(Block blk, byte[] bb)

描述：读数据的基本操作，将 blk 页的数据读入到 bb 数组中。

入口参数：blk，要读入的页标识。

出口参数：bb，保存读入数据的字节数组。

实现：先从字典openFiles中找到对应文件的文件流，应用该文件流来读取指定页中的数据，将数据赋值给bb数组，由bb数组传出数据。

internal void write(Block blk, byte[] bb)

描述：写数据的基本操作，将 bb 数组的内容写入到 blk 页。

入口参数：blk，要读入的页标识；bb，保存要写入数据的字节数组。

实现：先从字典openFiles中找到对应文件的文件流，应用该文件流来将bb字节数组的数据写入指定页中。

internal Block append(string filename, byte[] bb)

描述：写数据的基本操作，将 bb 数组的内容追加到 filename 文件的尾部。

入口参数：filename，指定的文件名；bb，保存要写入数据的字节数组。

实现：先在filename文件尾部新建一个块，然后调用write函数来将bb数组的内容写入到新建的块中。

public int size(string filename)

描述：返回指定文件 filename 的总页数。

入口参数：filename，指定的文件名。

实现：先根据filename找到对应的文件流，总页数为将文件流的长度与单个页大小的比值，将其返回。

public bool isNew()

描述：返回 isnew 变量值。

入口参数：无。

实现：将isnew的值返回。

private FileStream getFile(string filename)

描述：返回指定文件 filename 的文件流。

入口参数：filename，指定的文件名。

实现：根据 filename，从字典 openFiles 中找到对应的文件流，将其返回，设置写文件的方式。

3.1.3 Page

概述：此类定义了数据库中一个页的对象，大小为 512 字节。

成员变量：

```
private byte[] contents = new byte[BLOCK_SIZE];
private FileMgr filemgr = TBaseConfig.fileMgr();
private object threadLock = new object();
```

contents，字节数组，用来存储页中的数据；filemgr，FileMgr 类对象，用来对文件进行操作；threadLock，用来进行加锁，无实际意义。

成员方法：

public static int STR_SIZE(int n)

描述：返回指定长度的字符串所占的字节数。

入口参数：n，字符串的长度。

实现：因为存储的的字符串在第一位上保存整个字符串的长度，占一个整型的长度，所以总长度应该为 n 个字节加上一个整型的长度，将其结果进行返回。

public void read(Block blk)

描述：将指定页的内容读入到 contents 数组中。

入口参数：blk，指定的数据库的页。

实现：调用filemgr包含的read方法来实现。

public void write(Block blk)

描述：将 contents 中的内容写到指定的页中。

入口参数：blk，指定的数据库的页。

实现：调用filemgr包含的write方法来实现。

public Block append(string filename)

描述：将 contents 的内容追加到 filename 的尾部。

入口参数：filename，指定的文件名。

实现：调用filemgr包含的append方法来实现，并将新建的页返回。

public int getInt(int offset)

描述：在指定位置取出一个整型。

入口参数：offset，数据的偏移量。

实现：从contents的offset开始连续的四个字节取出合并便是这个整型数据，将offset后的字节依次左移八位，并且合并之后在强制转换成int类型。

public void setInt(int offset, int val)

描述：将指定位置的值设置为指定数值。

入口参数：offset，数据的偏移量；val，指定设置的值。

实现：从offset开始，每次将val后八位存入到一个字节中，并且将val右移八位。

public string getString(int offset)

描述：在指定位置取出一个字符串。

入口参数：offset，数据的偏移量。

实现：先从offset的位置取出一个整型，这个数表示字符串的长度，然后再按照长度依次将每个字节拷贝到新的字节数组中，复制之后用系统函数将其转换成为字符串类型。

public void setString(int offset, string val)

描述：将指定位置的值设置为指定的字符串。

入口参数：offset，数据的偏移量；val，指定设置的值。

实现：从offset开始，先计算val的长度，将长度用整型存入contents，再依次将字符串的每一个字节存入contents中。

3.3 功能测试

系统的登录界面和主界面如图3-2和图3-3所示。

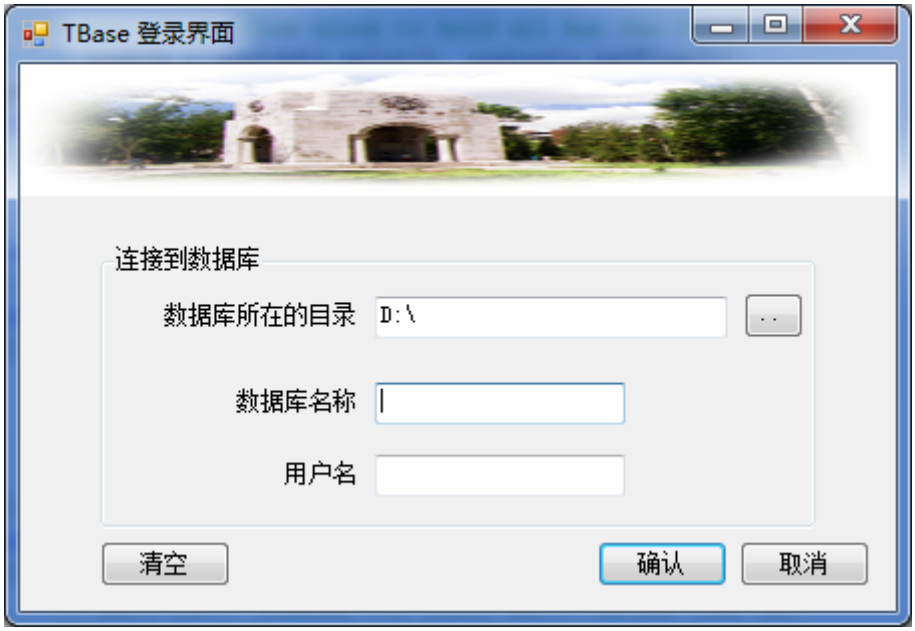

图3-2 系统的登录界面

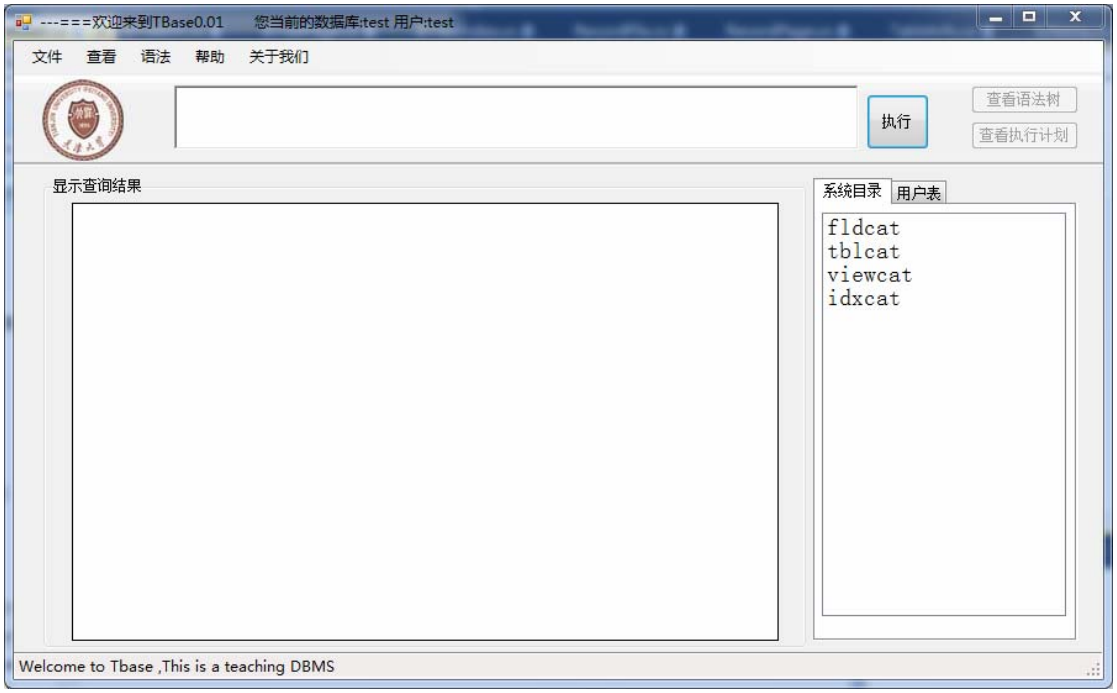

图3-3 系统的主界面

创建数据库之后，系统会创建自身所需要的一些属性表，如图3-4所示。

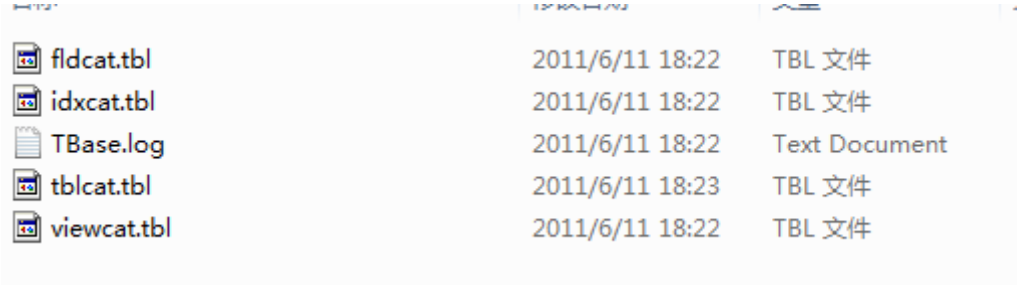

图3-4 系统的表

测试代码：

```
CREATE TABLE S (SSNO VARCHAR(3), SNAME VARCHAR(10), STATUS INT, SCITY VARCHAR(10));
CREATE TABLE P (PPNO VARCHAR(3), PNAME VARCHAR(10), COLOR VARCHAR(8), WEIGHT INT);
CREATE TABLE J (JJNO VARCHAR(3), JNAME VARCHAR(10), JCITY VARCHAR(10));
CREATE TABLE SPJ(SNO VARCHAR(3), PNO VARCHAR(3), JNO VARCHAR(3), QTY INT);
```

执行测试代码之后，数据库的文件如图3-5所示。

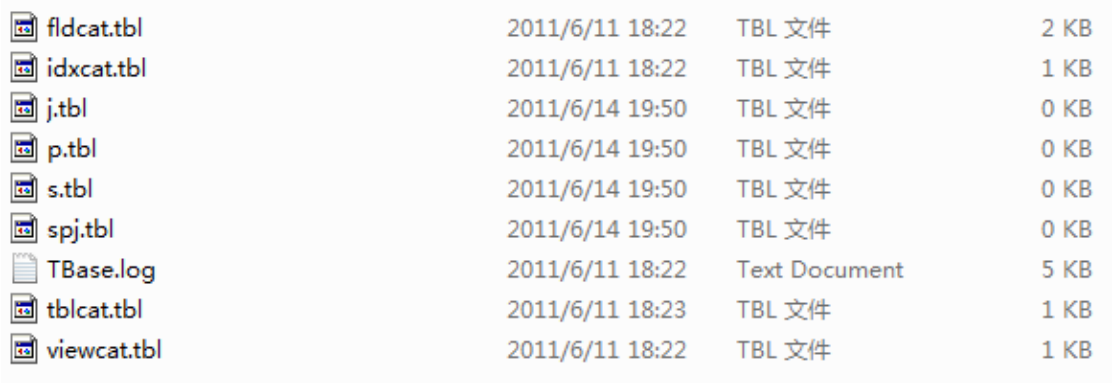

图3-5 执行代码之后的表文件列表

第四章　缓冲管理模块

4.1　设计原理

缓冲管理模块是存储管理子系统的核心模块之一，它的主要任务是将磁盘页缓存在内存中，减少数据访问过程中读写硬盘的次数，提高系统的效率。

论文为缓冲管理器分配了一个大小固定的缓冲池，缓冲池中的内容随着缓冲页的读入、写出而不停地变化[6]。为了实现方便，缓冲池用一个内存页的链表来实现。

缓冲区的作用是加快记录的读取速度，减少读取实际记录的代价。

系统将缓冲区设计为缓冲块的数组，缓冲区的块号就是数组的角标，对于每个缓冲块来说，都保存着页面的内容和一些附加信息，例如：pin_cnt。

缓冲区数据结构如图 4-1 所示。

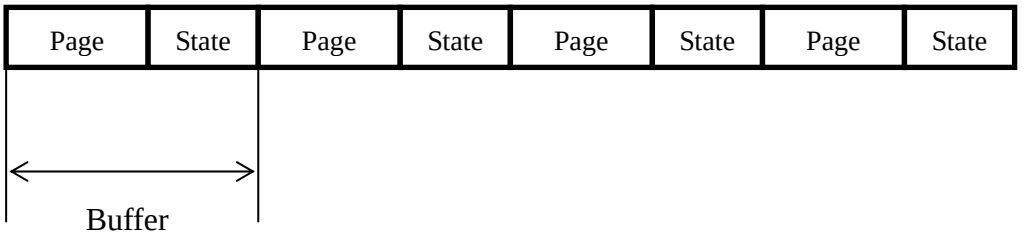

图 4-1 缓冲区结构

系统为缓冲区提供了三种替换算法[4]：Clock，LRU，MRU。为了完成替换算法，系统还需要维护一个状态数组，对应于缓冲区中的每个块。状态总共可分为三类：available、referenced 和 pinned。Clock 算法，即时钟算法，将缓冲区虚拟在一个圆盘表面，排列成为圆环形状，有一个指针指引扫描，当遇到的缓冲块状态位 available 时，挑选成为被替换的块，当遇到 pinned 块时，继续向下一个扫描，当遇到 referenced 块时，将该块状态改为 available，并继续向下扫描，直到挑选出被替换的块。LRU，即最近最少使用替换策略，系统增加维护一个缓冲块的队列，当某个缓冲块被使用时，将其块号移至队列的头部，每当要选取替换块的时候，挑选队尾的缓冲块即可。MRU，即最近最多使用替换策略，与 LRU 的替换策略类似。

4.1.1　类图

关于缓冲区管理操作的类图如图 4-2 所示。

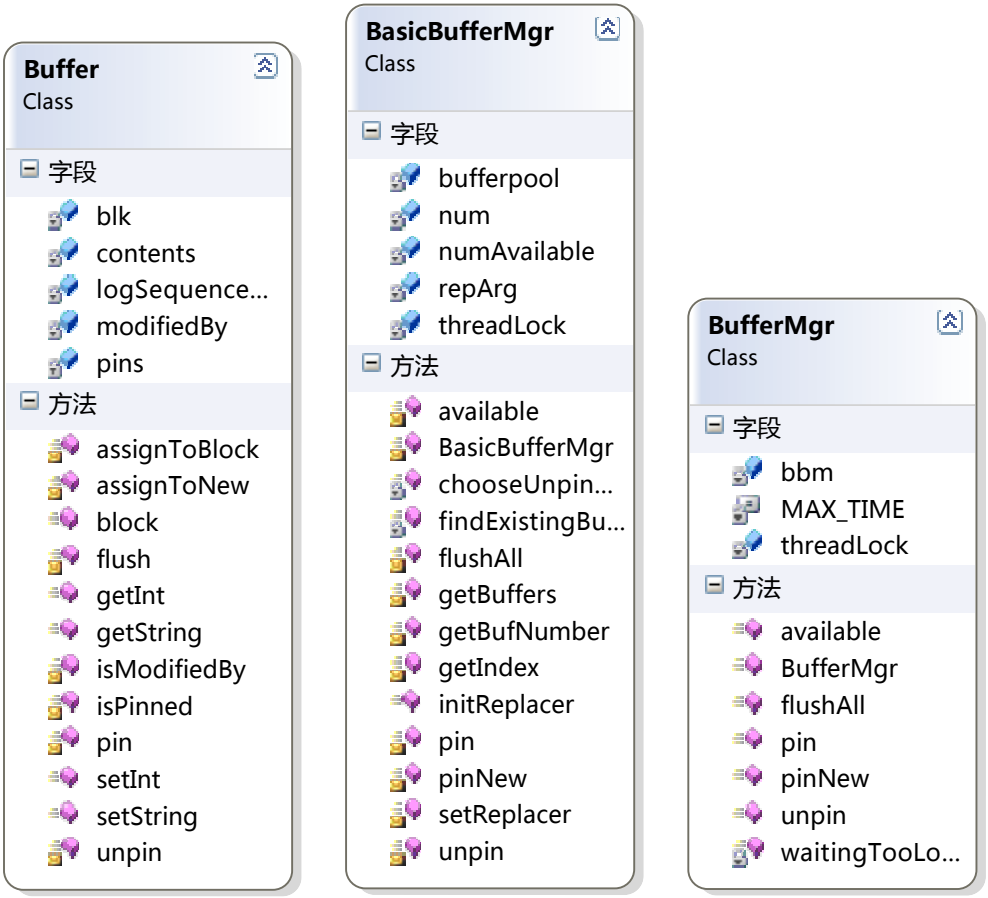

图 4-2 关于缓冲区管理的类图

关于页面替换算法的类图如图 4-3 所示。

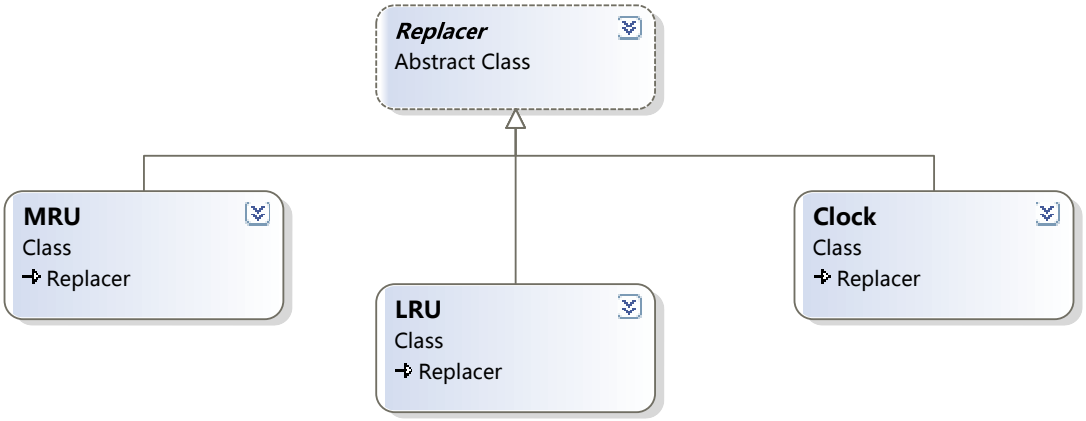

图 4-3 页面替换算法的类图

4.2 具体实现

缓冲区的实现，主要涉及的是替换算法的实现和对缓冲区数据结构的实现，还有就是对缓冲区的具体操作。

4.2.1 Buffer

概述：对应于缓冲区的一块，大小与数据库中一页的大小一样，也就是说数据库中的一页装入缓冲区后，占缓冲区的一块。

成员变量：

```
private Page contents = new Page();
private Block blk = null;
private int pins = 0;
private int modifiedBy = -1;
private int logSequenceNumber = -1;
```

contents，Page 类的对象，表示对应的数据库页的内容；blk，Block 类对象，对应的数据库页的页号；pins，整型类型，为正数时表示该缓冲块被 pin 住了，当为 0 时表示未被 pin；modifiedBy，整型类型，当为负一时表示没有被修改，当为正数时表示被修改的事务号；logSequenceNumber，整型类型，若为负一，则表示没有与之对应的日志记录，若不为负一，则表示对应的日志记录号。

成员方法：

public int getInt(int offset)

描述：将指定位置的值按照整型类型取出。

入口参数：offset，数据的偏移量。

实现：调用contents的getInt方法实现，并且将其返回值返回。

public string getString(int offset)

描述：将指定位置的值按照字符串类型取出。

入口参数：offset，数据的偏移量。

实现：调用contents的getString方法实现，并且将其返回值返回。

public void setInt(int offset, int val, int txnum, int lsn)

描述：将指定位置的值设置为指定的整型数。

入口参数：offset，数据的偏移量；val，指定设置的值；txnum，执行操作的事务号，lsn，对应的日志记录号。

实现：调用contents的setInt方法实现，并将isModifiedBy赋值为txnum，将logSequenceNumber赋值为lsn。

public void setString(int offset, string val, int txnum, int lsn)

描述：将指定位置的值设置为指定的字符串。

入口参数：offset，数据的偏移量；val，指定设置的值；txnum，执行操作的事务号，lsn，对应的日志记录号。

实现：调用contents的setString方法实现，并将isModifiedBy赋值为txnum，将logSequenceNumber赋值为lsn。

public Block block()

描述：返回当前页的标识类。

入口参数：无。

实现：将blk变量返回。

internal void flush()

描述：将缓冲块的内容写回磁盘中。

入口参数：无。

实现：先判断缓冲块的内容是否被修改，如果没有，则不用写回，如果被修改了，则调用contents的write方法将其写回到磁盘上，并将isModifiedBy变量置为-1。

internal void pin()

描述：将这个缓冲块钉住。

入口参数：无。

实现：pins变量加1。

internal void unpin()

描述：将这个缓冲块解钉。

入口参数：无。

实现：pins变量减1。

internal bool isPinned()

描述：判断这个缓冲块是否被钉住。

入口参数：无。

实现：判断pins变量是否大于0，如果大于0，则返回true，否则返回false。

internal bool isModifiedBy(int txnum)

描述：判断这个缓冲块的最近一次修改是否是 txnum 做的。

入口参数：txnum，指定的事务号。

实现：判断isModifiedBy变量是否等于txnum，若等于，返回true，若不等于，返回false。

internal void assignToBlock(Block b)

描述：将指定页读入到缓冲块中。

入口参数：b，指定的数据库页。

实现：先将这个缓冲块的内容刷回磁盘，然后将blk赋值为b，再将b的内容读入到缓冲块中，最后将pins变量设置为0。

internal void assignToNew(string filename, PageFormatter fmtr)

描述：新建一个页面并且将其读入缓冲块中。

入口参数：filename，文件名；fmtr，页面格式。

实现：先将这个缓冲块的内容刷回磁盘，然后将内容进行格式化，再调用contents的append方法实现新建页面，将其返回值赋值给blk，最后将pins变量设置为0。

4.2.2 Replacer

概述：替换算法的虚类。

成员变量：

```
protected internal BasicBufferMgr mgr;
public Dictionary<Buffer, STATE> state_bit = new Dictionary<Buffer, STATE>();
protected internal int head;
```

包含一个字典 state_bit<Buffer, STATE>，是由缓冲块和其对应的状态一一对应，状态分为三类：Pinned、Available 和 Referenced；head，整型类型，表示时钟替换检测时的头指针。

成员方法：

public virtual void Pin(Buffer buf)

描述：将指定的缓冲块钉住。

入口参数：buf，指定的缓冲块。

实现：调用buf的pin方法实现，并且将其状态设置为Pinned。

public bool Unpin(Buffer buf)

描述：将指定的缓冲块解钉。

入口参数：buf，指定的缓冲块。

实现：调用buf的unpin方法实现，并且将其状态设置为Referenced。

public void Free(Buffer buf)

描述：将指定的缓冲块从缓冲区中释放。

入口参数：buf，指定的缓冲块。

实现：调用buf的unpin方法实现，并且将其状态设置为Available。

public abstract int Pick_victim();

描述：虚函数，挑选将被替换的缓冲块的号码。

入口参数：无。

实现：由子类实现。

public abstract String Name();

描述：返回替换算法的名称。

入口参数：无。

实现：由子类实现。

public virtual void Info()

描述：输出当前情况下缓冲区的信息。

入口参数：无。

实现：依次扫描各个缓冲块，并将其号码和状态进行输出。

public int GetNumUnpinnedBuffers()

描述：计算未被钉住的缓冲块的数目。

入口参数：无。

实现：扫描整个缓冲区，若扫描到的缓冲块未被钉住，则计数器加1，扫描完毕后返回计数器结果。

protected internal Replacer(BasicBufferMgr minimgr)

描述：构造函数，根据给出的基础管理器来构造缓冲替换类

入口参数：minimgr，缓冲管理器。

实现：将minimgr赋值给mgr，将每一个缓冲块的状态进行初始化。

public virtual void SetBufferManager(BasicBufferMgr mgrArg)

描述：设置缓冲管理器。

入口参数：mgrArg，指定的缓冲管理器。

实现：将 mgrArg 赋值为 mgr。

4.2.3 Clock

概述：继承 Replacer 类，完成时钟替换算法的实现。

成员变量：无

成员方法：

public override int Pick_victim()

描述：重写父类的函数，挑选将被替换的缓冲块的号码。

入口参数：无。

实现：用head=(head+1)%numBuffers来实现时钟的效果，循环着扫描链表。Pick_victim()，挑选被替换或者被pin的缓冲块，如果扫描到的块状态为available，则选择此块；如果扫描到的块状态为referenced，则将其状态改为available，并扫描下一块；如果扫描到的块状态为pinned，则继续扫描下一块。

public sealed override String Name()

描述：重写父类的函数，返回当前替换算法的名称。

入口参数：无。

实现：返回“Clock”。

public override void Info()

描述：重写父类的函数，输出当前的缓冲区信息。

入口参数：无。

实现：调用父类的同名函数，并且加以输出当前Clock扫描臂head的位置。

4.2.4 LRU

概述：继承 Replacer 类，完成 LRU 替换算法的实现。采用 LRU 算法，即最近最少使用替换策略，算法的思想就是如果要从缓冲区中剔出一块，那么就挑选最近使用次数最少的一块。

成员变量：

```
private int[] frames;
private int nframes;
```

frames[]，来表示最近缓冲区中所有块的使用情况，还用一个变量来存储缓冲区中块的数目。nframes，使用的 frame 数目。

成员方法：

public override int Pick_victim()

描述：重写父类的函数，挑选将被替换的缓冲块的号码。

入口参数：无。

实现：分两种情况，如果缓冲区未满，那么就不需要替换，直接用未使用的块；如果缓冲区已满，那么就要进行替换，即将队列中的第一个未被pin的块挑选出来进行替换，如果队列中所有的块都被pin了，那么抛出异常。

public sealed override String Name()

描述：重写父类的函数，返回当前替换算法的名称。

入口参数：无。

实现：返回“LRU”。

public override void Info()

描述：重写父类的函数，输出当前的缓冲区信息。

入口参数：无。

实现：调用父类的同名函数，并且加以输出当前替换算法名称。

public override void Pin(Buffer buf)

描述：重写父类的函数，将指定的缓冲块钉住。

入口参数：buf，指定的缓冲块。

实现：调用父类的同名函数将指定块钉住，在找到该缓冲块的号码，调用 Update 函数调整 frame 数组。

public LRU(BasicBufferMgr mgrArg)

描述：构造函数，构造一个完整的 LRU 类。

入口参数：mgrArg，缓冲管理器。

实现：调用父类的构造函数函数，并且初始化 frame 数组。

private void Update(int frameNo)

描述：根据使用情况调整 frame 数组情况。

入口参数：frameNo，缓冲块的号码。

实现：因为 frameNo 块刚刚被使用过，那么就应该将队列中的这一块移到队列的末尾。

4.2.5 MRU

概述：继承 Replacer 类，完成 MRU 替换算法的实现。算法的思想就是如果要从缓冲区中剔出一块，那么就挑选最近使用次数最多的一块。

成员变量：

```
private int[] frames;
private int nframes;
```

frames[]，来表示最近缓冲区中所有块的使用情况，还用一个变量来存储缓冲区中块的数目。nframes，使用的 frame 数目。

成员方法：

public override int Pick_victim()

描述：重写父类的函数，挑选将被替换的缓冲块的号码。

入口参数：无。

实现：分两种情况，如果缓冲区未满，那么就不需要替换，直接用未使用的块；如果缓冲区已满，那么就要进行替换，即将队列中的第一个未被 pin 的块挑选出来进行替换，如果队列中所有的块都被 pin 了，那么抛出异常。

public sealed override String Name()

描述：重写父类的函数，返回当前替换算法的名称。

入口参数：无。

实现：返回“MRU”。

public override void Info()

描述：重写父类的函数，输出当前的缓冲区信息。

入口参数：无。

实现：调用父类的同名函数，并且加以输出当前替换算法名称。

public override void Pin(Buffer buf)

描述：重写父类的函数，将指定的缓冲块钉住。

入口参数：buf，指定的缓冲块。

实现：调用父类的同名函数将指定块钉住，在找到该缓冲块的号码，调用 Update 函数调整 frame 数组。

public LRU(BasicBufferMgr mgrArg)

描述：构造函数，构造一个完整的 LRU 类。

入口参数：mgrArg，缓冲管理器。

实现：调用父类的构造函数函数，并且初始化 frame 数组。

private void Update(int frameNo)

描述：根据使用情况调整 frame 数组情况。

入口参数：frameNo，缓冲块的号码。

实现：因为 frameNo 块刚刚被使用过，那么就应该将队列中的这一块移到队列的头部。

4.2.6 BasicBufferMgr

概述：对缓冲区进行的基本操作方法集合，为缓冲管理器提供基本的方法。

成员变量：

```
private Buffer[] bufferpool;
private int numAvailable;
private object threadLock = new object();
private int num;
private Replacer repArg;
```

bufferpool，Buffer 类数组，表示该系统的缓冲区；numAvailable，整型类型，表示系统缓冲区中状态位 Available 的缓冲块的数目；num，整型类型，表示缓冲区总的缓冲块数目，repArg，Replacer 类对象，表示系统当前采用的替换策略。

成员方法：

internal BasicBufferMgr(int numbuffs)

描述：构造函数，根据给出的 numbuffs 构造类。

入口参数：numbuffs，缓冲块的数目。

实现：将 numbuffs 赋值给 num 和 numAvailable，根据 numbuffs 创建缓冲区，再逐个将每个缓冲块实体化。

public void initReplacer(int numbuffs, string policy)

描述：初始化缓冲替换策略。

入口参数：numbuffs，缓冲块的数目；policy，要采取的缓冲替换算法的名称。

实现：根据 policy 的不同来对 repArg 进行实体化，再将每一个缓冲块的状态初始化为 Available。

internal void setReplacer(Replacer repArg)

描述：将缓冲替换算法设置为给定的替换算法。

入口参数：repArg，缓冲替换算法。

实现：将类中的 repArg 赋值为入口参数 repArg。

internal void flushAll(int txnum)

描述：将事务号为 txnum 的事务最近修改的缓冲块重新写回磁盘。

入口参数：txnum，事务号。

实现：对于缓冲区中的每个缓冲块，先调用 Buffer 的 isModifiedBy 函数来检测是否是 txnum 事务修改的，如果是，在调用 flush 函数将其刷回磁盘，如果不是 txnum 事务修改的，那么不写回磁盘。

internal Buffer pin(Block blk)

描述：将给定的页面装入的缓冲区，并且钉住。

入口参数：blk，给定的页面标识。

实现：先调用 findExistingBuffer 函数查找页面 blk 是否存在于缓冲区中，若在，直接将其钉住，若不在，调用 chooseUnpinnedBuffer 来依据缓冲替换算法选出的被替换的块，将 blk 装入该块，然后将其钉住。最后将这个缓冲块返回。

internal Buffer pinNew(string filename, PageFormatter fmtr)

描述：在指定文件末尾新建一个文件，然后装入缓冲区中。

入口参数：filename，指定的文件名；fmtr，文件格式。

实现：调用 chooseUnpinnedBuffer 来依据缓冲替换算法选出的被替换的块，然后调用 assignToNew 函数来将创建新页，并且装入挑选出来被替换的块，将其钉住，返回该缓冲块。

internal void unpin(Buffer buff)

描述：将指定的缓冲块解钉。

入口参数：buff，指定的缓冲块。

实现：调用 buff 自身的 unpin 方法进行解钉，并且将 numAvailable 加 1。

internal int available()

描述：返回缓冲区中未被 pin 的块的数目。

入口参数：无。

实现：返回变量 numAvailable 的值。

internal int getBufNumber()

描述：返回缓冲区缓冲块的总书目。

入口参数：无。

实现：返回变量 num 的值。

internal Buffer[] getBuffers()

描述：返回缓冲区的指针。

入口参数：无。

实现：直接返回 bufferpool。

internal int getIndex(Buffer buf)

描述：计算指定块在缓冲区数组中的序号。

入口参数：buf，指定缓冲块。

实现：依次扫描整个缓冲区，知道扫描到与指定块 buf 相等的缓冲块，返回它的序号。

private Buffer findExistingBuffer(Block blk)

描述：判断指定页是否装入到缓冲区中。

入口参数：blk，指定的页标识。

实现：依次扫描整个缓冲区，对每一个缓冲块对应的页标识和指定的页标识进行判等，如果找到相等的，返回该缓冲块，如果没有找到缓冲块，则返回 null。

private Buffer chooseUnpinnedBuffer()

描述：挑选出将要被替换的缓冲块。

入口参数：无。

实现：调用替换算法的 Pick_victim()来实现，若找到，返回那个缓冲块，若未找到，则返回 null。

4.2.7 BufferMgr

概述：整个系统的缓冲管理器，负责对整个系统的缓冲区进行管理操作，包括 pin 和 unpin 等基本的操作。

成员变量：

```
private BasicBufferMgr bbm;
private object threadLock = new object();
```

bbm，BasicBufferMgr 类型，负责缓冲区的基本操作；threadLock，用于线程加锁，不具有实际意义。

成员方法：

public BufferMgr(int numbuffers , string policy)

描述：构造函数，根据给定的缓冲块数目和替换策略构造缓冲管理器。

入口参数：numbuffers，缓冲块数目；policy，替换策略。

实现：用 numbuffers 来构建 bbm，在调用 bbm 的 initReplacer 方法来设置缓冲替换策略。

public Buffer pin(Block blk)

描述：将指定的页面装入缓冲区，并且将其钉住。

入口参数：blk，指定的页面标识。

实现：调用 bbm 的 pin 方法实现，并且进行死锁检测，如果等待时间过长，则放弃操作。

public Buffer pinNew(string filename, PageFormatter fmtr)

描述：将指定文件的尾部追加一页并装入缓冲区，并且将其钉住。

入口参数：filename，指定的文件名，fmtr，指定的页格式。

实现：调用bbm的pinNew方法实现，并且进行死锁检测，如果等待时间过长，则放弃操作。

public void unpin(Buffer buff)

描述：将指定的缓冲块解钉。

入口参数：buff，指定的缓冲块。

实现：调用 bbm 的 unpin 方法实现。

public void flushAll(int txnum)

描述：将缓冲区中的被指定事务修改过的脏页写回磁盘。

入口参数：txnum，指定的事务号。

实现：调用 bbm 的 flushAll 方法实现。

public int available()

描述：返回缓冲区可用缓冲块数目。

入口参数：无。

实现：调用 bbm 的 available 方法实现。

private bool waitingTooLong(long starttime)

描述：判断等待时间是否过长。

入口参数：starttime，开始时间。

实现：用现在的时间减去开始时间得到运行时间，与最大时间进行比较，如果大于最大时间返回 true，如果不大于，则返回 false。

4.3 功能测试

缓冲管理模块是存储管理子系统的核心模块，其主要的功能是将系统频繁访问的页面暂存在内存中，有效减少系统的磁盘读写次数，提高系统的性能。

根据缓冲管理器的工作特性，测试的重点是缓冲帧的读写，以及在多线程同时访问缓冲池的情况下，测试系统的执行结果。

测试的缓冲区状态如图 4-4 所示。

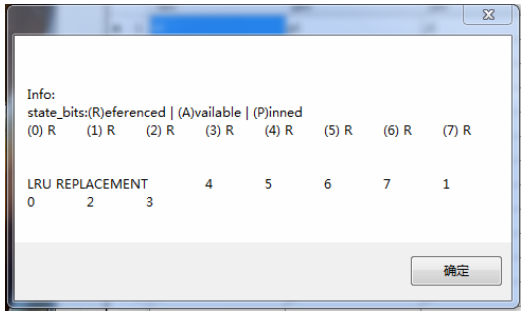

图 4-4 测试显示的缓冲区状态图

经过测试，缓冲管理器能够在单线程的条件下很好地完成读写操作，并且能够在多线程的情况下，保持系统的正确运行。

第五章　存取方法模块

5.1　设计原理

存取方法管理实际上主要是对索引进行管理，因而，存取方法管理模块实际上就是索引管理模块。系统提供了堆文件和 B 树索引两种存取方法[12]。

5.1.1　堆文件

堆文件的类图如图 5-1 和图 5-2 所示。

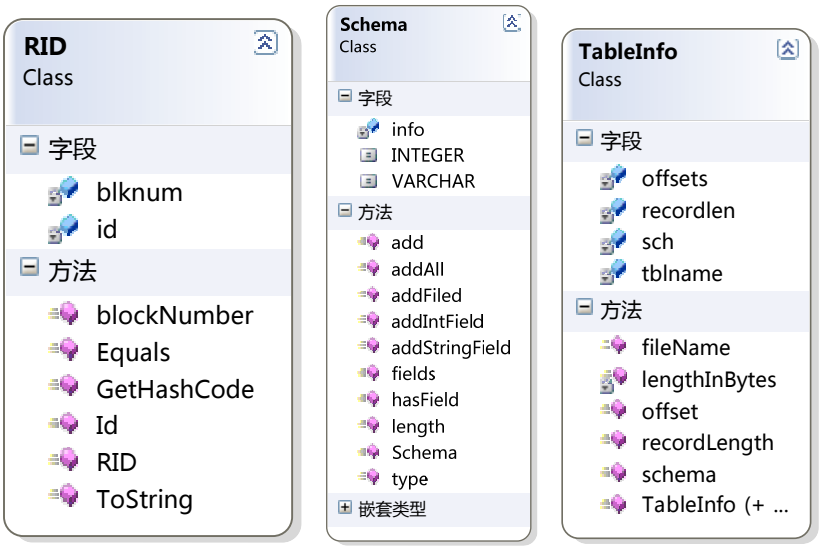

图 5-1　堆文件的类图(1)

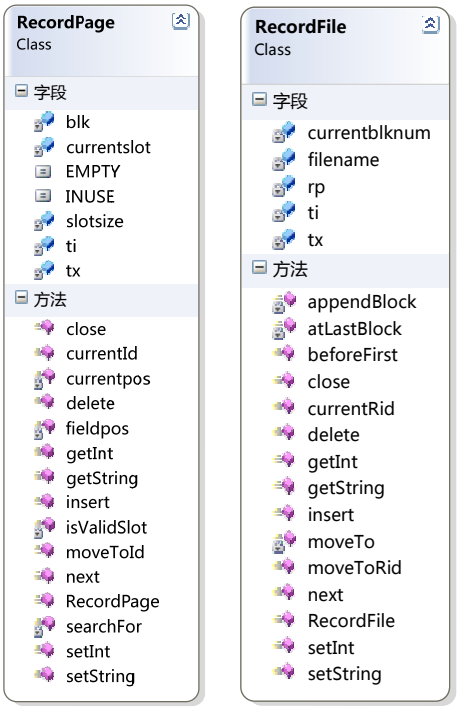

图 5-2　堆文件的类图(2)

5.1.2 B 树

B 树，即二叉搜索树，它具有以下几个特点：

1. 所有非叶子结点至多拥有两个儿子（Left 和 Right）；
2. 所有结点存储一个关键字；
3. 非叶子结点的左指针指向小于其关键字的子树，右指针指向大于其关键字的子树；

B 树的搜索[11]，从根结点开始，如果查询的关键字与结点的关键字相等，那么就命中；否则，如果查询关键字比结点关键字小，就进入左儿子；如果比结点关键字大，就进入右儿子；如果左儿子或右儿子的指针为空，则报告找不到相应的关键字；如果 B 树的所有非叶子结点的左右子树的结点数目均保持差不多（平衡），那么 B 树的搜索性能逼近二分查找；但它比连续内存空间的二分查找的优点是，改变 B 树结构（插入与删除结点）不需要移动大段的内存数据，甚至通常是常数开销。

B 树的数据结构如图 5-3 所示。

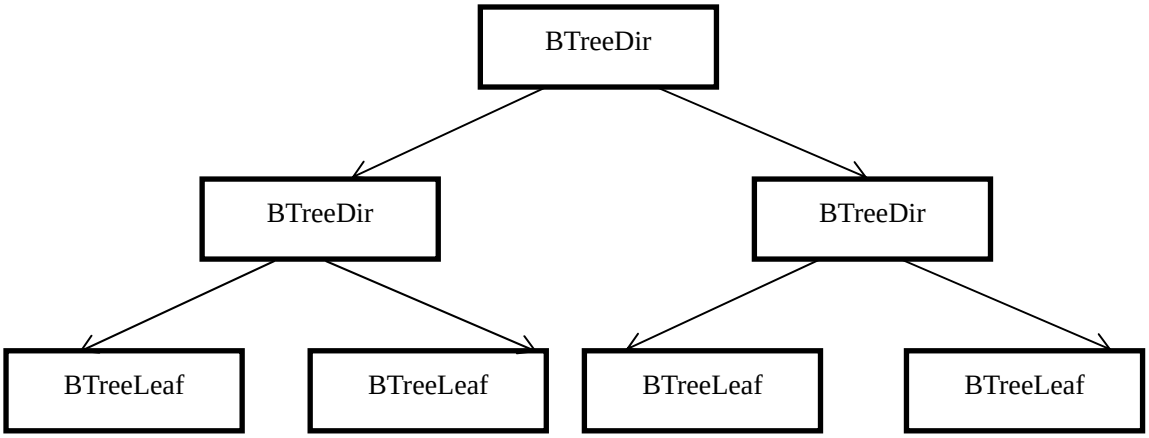

图 5-3 B 树的数据结构

B 树的类图如图 5-4 所示。

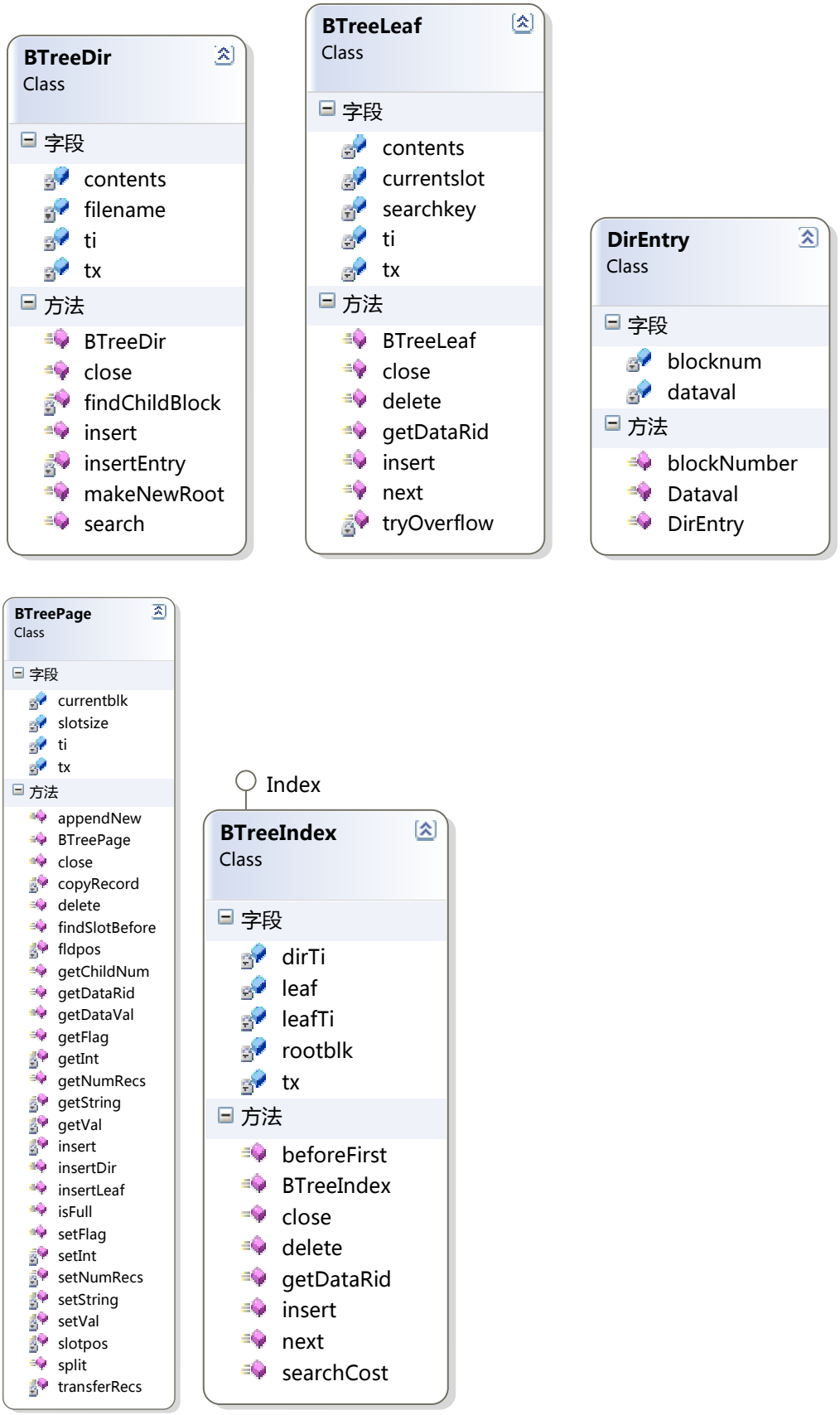
BTreeDir
Class
字段
contents
filename
ti
tx
方法
BTreeDir
close
findChildBlock
insert
insertEntry
makeNewRoot
search
BTreeLeaf
Class
字段
contents
currentslot
searchkey
ti
tx
方法
BTreeLeaf
close
delete
getDataRid
insert
next
tryOverflow
DirEntry
Class
字段
blocknum
dataval
方法
blockNumber
Dataval
DirEntry
BTreePage
Class
字段
currentblk
slotsize
ti
tx
方法
appendNew
BTreePage
close
copyRecord
delete
findSlotBefore
fldpos
getChildNum
getDataRid
getDataVal
getFlag
getInt
getNumRecs
getString
getVal
insert
insertDir
insertLeaf
isFull
setFlag
setInt
setNumRecs
setString
setVal
slotpos
split
transferRecs
Index
BTreeIndex
Class
字段
dirTi
leaf
leafTi
rootblk
tx
方法
beforeFirst
BTreeIndex
close
delete
getDataRid
insert
next
searchCost

图 5-4 B 树索引的类图

5.2 具体实现

5.2.1 Schema

概述：定义了一个表的模式。

成员变量：

```
private Dictionary<string, FieldInfo> info = new Dictionary<string, FieldInfo>();
```

FieldInfo，包含两个公有变量，type 表示字段的数据类型，length 表示字段的长度。info<string, FieldInfo>，将字段名和字段的信息对应起来。

成员方法：

public void addFiled(string fldname, int type, int length)

描述：为表的模式增加一个字段，字段名、类型、长度已经给出。

入口参数：fldname，字段名；type，字段类型；length，字段长度。

实现：用参数 type 和 length 两个变量构建一个 FieldInfo 类，再跟 fldname 组合加入到字典 info 中。

public void addIntField(string fldname)

描述：为表的模式增加一个整型字段，字段名给出。

入口参数：fldname，字段名。

实现：调用 addFiled 函数，type 参数用 INTEGER，length 参数用 0。

public void addStringField(string fldname, int length)

描述：为表的模式增加一个字符串字段，字段名和长度给出。

入口参数：fldname，字段名；length，字段长度。

实现：调用 addFiled 函数，type 参数用 VARCHAR。

public void add(string fldname, Schema sch)

描述：将另一模式中的字段加入到当前模式。

入口参数：fldname，字段名；sch，模式类型。

实现：取出 fldname 在 sch 模式中的类型和长度，再作为参数调用 addFiled 函数来实现字段的添加。

public void addAll(Schema sch)

描述：将 sch 模式中的所有字段添加到当前模式中。

入口参数：sch，给定的模式。

实现：扫描 sch 模式中的每个字段，调用 addFiled 方法将其逐个加入到当前模式中。

public ICollection<string> fields()

描述：返回字段名的集合。

入口参数：无。

实现：将字段名的集合返回。

public bool hasField(string fldname)

描述：判断当前模式是否含有给定的字段 fldname。

入口参数：fldname，需要检验的字段名。

实现：运用 info 的 ContainsKey 方法来检测模式中是否含有字符按 fldname。

public int type(string fldname)

描述：查看给定字段的数据类型。

入口参数：fldname，给定的字段名。

实现：返回 info 字典中 fldname 对应的 type 属性

public int length(string fldname)

描述：查看给定字段的数据长度。

入口参数：fldname，给定的字段名。

实现：返回 info 字典中 fldname 对应的 length 属性。

5.2.2 TableInfo

概述：描述表的一些基本信息。

成员变量：

```
private Schema sch;
private Dictionary<string, int> offsets;
private int recordlen;
private string tblname;
```

sch，Schema 类型，表的模式，offsets，字典类型，将字段名和偏移量对应起来存储。Tblname，字符串类型，表名，recordlen，整型类型，表示一条记录的长度。

成员方法：

public TableInfo(string tblname, Schema schema)

描述：构造函数，将传入的参数分别赋值给变量，对其余的变量进行初始化。

入口参数：tblname，给定的表名；schema，表的模式。

实现：将 tblname 和 schema 赋值给类的 tblname 和 sch，根据 sch 中的字段名获取各个字段的长度，再依次对 offsets 进行初始化。

public string fileName()

描述：返回表文件的名字。

入口参数：无。

实现：表名加上“.tbl”组成文件名，并且返回该字符串。

public Schema schema()

描述：返回表的模式。

入口参数：无。

实现：返回类的 sch 变量值。

public int offset(string fldname)

描述：查看给定字段的便宜量。

入口参数：fldname，给定的字段名。

实现：返回 offsets 字典中 fldname 对应的整型数值。

public int recordLength()

描述：查看这个表中的一个记录的长度。

入口参数：无。

实现：返回类的 recordlen 变量值。

private int lengthInBytes(string fldname)

描述：计算给定字段所占的字节数。

入口参数：fldname，给定的字段名。

实现：先取出 fldname 字段的类型，在调用 Page 类中指定函数来计算该字段所占的字节数，并返回结果。

5.2.3 RID

概述：记录的记录号，找到记录的唯一标识。

成员变量：

```
private int blknum;
private int id;
```

blknum，整型类型，表示记录所在的页号；id，整型类型，表示记录在所在页的记录号。

成员方法：

public RID(int blknum, int id)

描述：构造函数，根据给出的页号和记录号构造一个 RID 类。

入口参数：blknum，给定的页号；id，给定的记录号。

实现：将其入口参数分别赋值给 blknum 和 id。

public int blockNumber()

描述：返回记录的页号。

入口参数：无。

实现：返回类的 blknum 变量值。

public int Id()

描述：返回记录的记录号。

入口参数：无。

实现：返回类的 id 变量值。

public override bool Equals(object obj)

描述：判断当前的 RID 类与给定的类是否相等。

入口参数：obj，作为比较的类。

实现：将两个类的 blknum 和 id 两个变量的值作比较，两者都相等的认为两个类相等，返回 true，否则认为两个类不相等，返回 false。

5.2.4 RecordPage

概述：表示记录页的类，规定了记录页面的格式。

成员变量：

```
private Block blk;
private TableInfo ti;
private Transaction tx;
private int slotsize;
private int currentslot = -1;
```

blk，Block 类对象，当前页的页号，currentslot，整型类型，当前槽号，ti，TableInfo 类对象，存储表的基本信息，slotsize，整型类型，表示槽的大小。

成员方法：

public RecordPage(Block blk, TableInfo ti, Transaction tx)

描述：构造函数，根据传入的参数构造一个记录页。

入口参数：blk，给定的页标识；ti，给定的表信息；tx，操作的事务。

实现：将传入的参数分别赋值，并且 slotsize 设置为记录的长度加上一个整型数的长度，并且将 blk 页面钉在缓冲区中。

public void close()

描述：关闭当前记录页。

入口参数：无。

实现：将当前页 unpin，并且将 blk 变量置空。

public bool next()

描述：移动到下一个记录。

入口参数：无。

实现：调用 searchFor 函数来查找下一条 INUSE 标志的记录。

public int getInt(string fldname)

描述：将指定的字段按照整型取出。

入口参数：fldname，指定的字段名。

实现：先调用 fieldpos 将定位到 fldname 字段的位置，在运用 tx 的 getInt 方法来取出数据并返回。

public string getString(string fldname)

描述：将指定的字段按照字符串类型取出。

入口参数：fldname，指定字段名。

实现：先调用 fieldpos 将定位到 fldname 字段的位置，在运用 tx 的 getString 方法来取出数据并返回。

public void setInt(string fldname, int val)

描述：将 fldname 字段的数据设置为 val。

入口参数：fldname，指定的字段名；val，要设置的值。

实现：先调用 fieldpos 将定位到 fldname 字段的位置，在运用 tx 的 setInt 方法来将指定位置的数据进行设置。

public void setString(string fldname, string val)

描述：将 fldname 字段的数据设置为 val。

入口参数：fldname，指定的字段名；val，要设置的值。

实现：先调用 fieldpos 将定位到 fldname 字段的位置，在运用 tx 的 setString 方法来将指定位置的数据进行设置。

public void delete()

描述：删除当前记录。

入口参数：无。

实现：先调用 currentpos 函数来定位到当前记录，在运用 tx 的 setInt 函数将记录设置为空值。

public bool insert()

描述：插入一条空记录。

入口参数：无。

实现：运用 searchFor 函数来查找 EMPTY 的记录，运用 tx 的 setInt 函数设置为 INUSE，返回 searchFor 的查找结果。

public void moveToId(int id)

描述：将当前记录移至给定的记录号 id。

入口参数：id，给定的记录号。

实现：将变量 currentslot 变量赋值为 id。

public int currentId()

描述：返回当前记录的 id。

入口参数：无。

实现：返回类的 currentslot 变量值。

private int currentpos()

描述：返回当前记录的偏移位置。

入口参数：无。

实现：返回 currentslot 与 slotsize 的乘积。

private int fieldpos(string fldname)

描述：返回当前记录给定字段的偏移位置。

入口参数：fldname，给定的字段名。

实现：当前记录的偏移量可以调用 currentpos 来得到，在根据 ti 中的 fldname 得到记录内部字段的偏移量，两者之和再加上一个整型数的长度，所得到的值即是需要返回的值。

private bool isValidSlot()

描述：判断当前槽是否合法。

入口参数：无。

实现：查看当前记录的偏移量加上一个槽的长度是否超过一页的大小，如果超过了，则返回 false，如果没有，返回 true。

private bool searchFor(int flag)

描述：查找符合给定标志位的下一条记录。

入口参数：flag，标志位，分为 INUSE 和 EMPTY 两种。

实现：依次扫描槽，查看标志位，如果满足 flag 的要求即返回，如果不满足，继续扫描。

5.2.5 RecordFile

概述：记录文件的虚拟类，负责管理记录文件。

成员变量：

```
private TableInfo ti;
private Transaction tx;
private string filename;
private RecordPage rp;
private int currentblknum;
```

currentblknum，当前的页号，filename，当前的文件名，rp，RecordPage 类对象，ti，TableInfo 对象，表的信息，tx，对文件操作的事务。

成员方法：

public RecordFile(TableInfo ti, Transaction tx)

描述：构造函数，根据参数来构造一个记录文件类。

入口参数：ti，给出的表信息；tx，执行操作的事务。

实现：将参数分别赋值，filename 赋值为 ti 中的 fileName 方法的返回值，如果文件是空的，那么先创建一个新页，将当前记录位置移动到 0 位。

public void close()

描述：关闭当前的记录文件。

入口参数：无。

实现：调用 rp 的 close 来实现。

public void beforeFirst()

描述：将当前记录指针指向第一条记录。

入口参数：无。

实现：调用 moveTo 来将指针移动到第 0 位。

public bool next()

描述：向后扫描一条记录。

入口参数：无。

实现：调用 rp 的 next 函数实现，如果成功，返回 true，如果当前页是最后一页，那么返回 false。

public int getInt(string fldname)

描述：将指定字段的数据按照整型类型取出。

入口参数：fldname，指定字段名。

实现：调用 rp 的 getInt 方法来实现。

public string getString(string fldname)

描述：将指定字段的数据按照字符串类型取出。

入口参数：fldname，指定字段名。

实现：调用 rp 的 getString 方法来实现。

public void setInt(string fldname, int val)

描述：将指定字段的数据按照整型类型设置为 val。

入口参数：fldname，指定字段名；val，将要设置的值。

实现：调用 rp 的 setInt 方法来实现。

public void setString(string fldname, string val)

描述：将指定字段的数据按照字符串类型设置为 val。

入口参数：fldname，指定字段名；val，将要设置的值。

实现：调用 rp 的 setString 方法来实现。

public void delete()

描述：删除当前记录。

入口参数：无。

实现：调用 rp 的 delete 方法实现。

public void insert()

描述：插入一条空记录。

入口参数：无。

实现：先调用 rp 的 insert 方法实现，如果执行失败，则查看是否是最后一页，如果是，在其后面追加一页，将当前记录指针向后移动一位。

public void moveToRid(RID rid)

描述：将当前记录指针指向给定的记录上。

入口参数：rid，给定的记录。

实现：先调用 moveTo 来将当前页移动到 rid 的页上，再调用 rp 的 moveToId 移动到指定的记录。

public RID currentRid()

描述：返回当前记录标识。

入口参数：无。

实现：调用 rp 的 currentId 方法得到记录的 id，再结合当前页号构建出一个 RID 类，并将其返回。

private void moveTo(int b)

描述：将当前页面移动至指定页。

入口参数：b，指定的页号。

实现：将 currentblknum 赋值为 b。

private bool atLastBlock()

描述：判断当前页是否为最后一页。

入口参数：无。

实现：判断 currentblknum 是否等于页数减 1，如果等于，返回 true，如果不等于，返回 false。

private void appendBlock()

描述：在最后一页后面追加一新页。

入口参数：无。

实现：调用 rp 的 append 方法来实现。

5.2.6 BTreePage

B 树的页对象，包含的私有变量：currentblk，Block 类对象，当前页的页号，ti，表的信息，slotsize，单个槽的大小，tx，对页操作的事务。

一系列的 get 和 set 函数都是将具体的属性信息返回。CopyRecord(int from , int to)是将 from 的记录复制到 to 的记录位置。findSlotBefore(Constant searchkey)是根据 searchkey，找到值为 searchkey 前一个槽。

5.2.7 BTreeDir

概述：B 树的目录节点对象，对于目录节点来说，记录分为 dataval 和 block 两个字段，记录着索引值和索引页号。

成员变量：

```
private TableInfo ti;
private Transaction tx;
private string filename;
private BTreePage contents;
```

ti，表的信息，tx，对表操作的事务，filename，文件名也是表名，contents，BTreePage 类对象，存储页的内容。

成员方法：

public BTreeDir(Block blk, TableInfo ti, Transaction tx)

描述：构造函数，根据提供的参数构造一个目录节点类。

入口参数：blk，指定的页标识；ti，表的信息；tx，执行操作的事务。

实现：分别将参数进行赋值，将 ti 中的文件名赋值给 filename 即可。

public void close()

描述：关闭这个节点。

入口参数：无。

实现：调用 contents 的 close 来实现。

public int search(Constant searchkey)

描述：查找含有 searchkey 的叶子节点。

入口参数：searchkey，查找关键字。

实现：调用 findChildBlock 来查找孩子节点，如果有，那么继续调用该函数进行查找，直到找到为止，返回该页的页号。

public void makeNewRoot(DirEntry e)

描述：为 B 树新建一个根节点，新的根节点的其中一个孩子是旧的根节点，另一个孩子给出。

入口参数：e，给出的另一个孩子节点。

实现：调用 contents 的 split 方法来将已有树中的所有节点向后移动完毕，取原来根节点的值和新建的页号组合建立一个新的根节点，在想树中插入原来的根节点和已给出的另一个孩子节点。

public DirEntry insert(DirEntry e)

描述：插入一个新的目录项。

入口参数：e，给定的新的目录项。

实现：如果在 level0 的位置，那么直接进行插入，如果不在，则需要逐层查找目录项直到找到适合插入 e 的位置，将 e 插入，返回检测值。

5.2.8 BTreeLeaf

概述：B 树的叶子节点对象，由它可以直接找到合适的记录，对于叶子节点来说，记录分为 dataval、block 和 id 三个字段，记录着索引值和索引页号和具体的记录 id。

成员变量：

```
private TableInfo ti;
private Transaction tx;
private Constant searchkey;
private BTreePage contents;
private int currentslot;
```

ti，表的信息，tx，对表操作的事务，filename，文件名也是表名，contents，BTreePage 类对象，存储页的内容。

成员方法：

public BTreeLeaf(Block blk, TableInfo ti, Constant searchkey, Transaction tx)

描述：构造函数，根据给出的参数构造出一个 B 树的叶子节点。

入口参数：blk，页号；ti，表的信息；searchkey，查找关键字；tx，执行操作的事务。

实现：将各个参数逐一赋值，contents为新建的一个B数页类，currentslot调用findSlotBefore函数来定位到searchkey前一个记录。

public void close()

描述：关闭当前的数据页。

入口参数：无。

实现：调用contents的close方法来实现。

public bool next()

描述：将当前槽的指针指向下一槽。

入口参数：无。

实现：将currentslot指针加1，判断currentslot是否超出记录数的范围，超出范围返回false，否则返回true。

public RID getDataRid()

描述：返回当前记录的 RID。

入口参数：无。

实现：调用contents的getDateRid来实现。

public void delete(RID datarid)

描述：删除指定的记录。

入口参数：dararid，指定要删除的 RID。

实现：依次扫描记录，找到RID与指定记录的RID相等的记录，调用contents的delete函数来将其删除。

public DirEntry insert(RID datarid)

描述：将指定的记录插入到叶子节点当中。

入口参数：datarid，指定要插入的记录 RID。

实现：将当前指针加 1 后插入指定的 RID，如果此时页面不满，则返回空，如果页面满了，那么就需要移动之后的叶子节点，保证树的平衡性。

5.3 功能测试

测试代码：

```
INSERT INTO S(SSNO,SNAME,STATUS,SCITY) VALUES('S1','Andrew',20,'Beijing');
INSERT INTO S(SSNO,SNAME,STATUS,SCITY) VALUES('S2','Bob',10,'Beijing');
INSERT INTO S(SSNO,SNAME,STATUS,SCITY) VALUES('S3','Charles',30,'Beijing');
INSERT INTO S(SSNO,SNAME,STATUS,SCITY) VALUES('S4','John',20,'Tianjin');
INSERT INTO S(SSNO,SNAME,STATUS,SCITY) VALUES('S5','Michael',30,'Shanghai');
```

执行之后，表中的数据如图 5-5 所示。

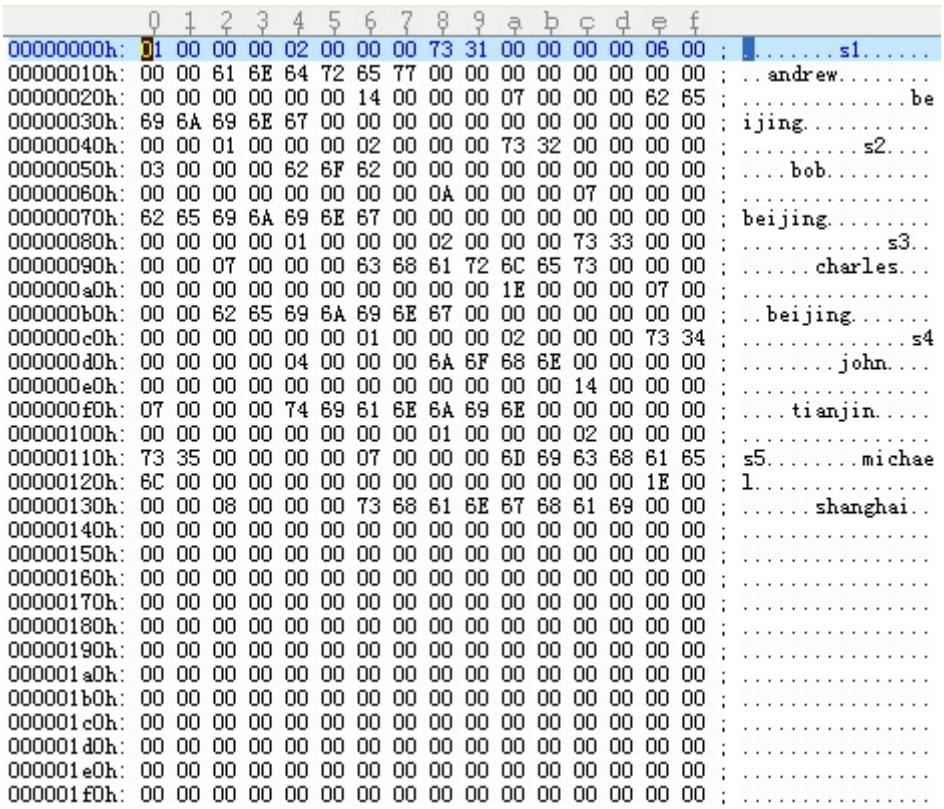

图 5-5 表的数据存储

测试结果：数据读取无错误，保证了数据库的正确性，未对性能进行测试评估。

第六章　总结与展望

TBase 数据库系统是一个功能完全的教学用数据库，具有基本数据库的存储管理，事务管理和查询优化三部分。本文实现的是它的存储管理子系统，虽然并非是事务处理的直接逻辑单元，但是从底层实现上，提供了对事务处理的支持，并且保证了良好的存储管理性能。论文在构建存储管理子系统的过程中，做了以下工作：

第一，采用了三种缓冲管理算法，可根据具体数据的分布来切换替换算法。该系统在实现缓冲管理模块的过程中，考虑到了在瞬间大量临时页面扫描过程中，页面失效过快而导致缓冲池频繁替换，引起系统性能下降的特殊情况，。

第二，实现了支持并发访问的 B 树索引结构。系统可以随时保持树的平衡，可以将数据的存取速度提高，保证数据的存取速度对于系统的性能提高有很大的重要性。

第三，面向对象的程序设计。程序的主体部分使用 C#语言编写，在整个程序中所有的功能模块都用类进行封装，并且在运行时各个模块以对象的形式存在。通过面向对象的程序设计，提高了程序源码的可读性和重用性。

第四，大规模的项目编程。与同学合作实现了一个基本的教学数据库，实现部分的源代码达到千余行。

本文实现的存储管理子系统目前还只是数据库的存储管理子系统的一个简单的原型，还有很多的地方需要改善，比如：系统仅仅实现了一种 B 树索引结构，对于目前被广泛使用的哈希表还没有支持，存取方法的实现过于单一等等，这些都需要在以后的工作中进一步改进和完善。

参考文献

[1] M.M.Astrahan， M.W.Blasgen，D.D.Chamberlin et al．System R: Relational Approach to Database Management[J]．ACM Transactions on Database Systems，1976，1(2)：97－137．

[2] Oracle Technology Network．Oracle Berkeley DB 11g[EB/OL]．http://www.oracle.com/technology/products/berkeley-db/index.html．2010．

[3] 罗摩克里希纳(Raghu Ramakrishnan)，格尔基(Johannes Gehrke)．数据库管理系统原理与设计（第三版）[M]．北京：清华大学出版社，2004．207－295．

[4] Hong-Tai Chou，David J．DeWitt．An Evaluation of Buffer Management Strategies for Relational Database Systems[C]．Proc．11th International Conference on Very Large Data Bases．Stockholm，1985：127－141．

[5] Douglas Comer．The Ubiquitous B-Tree[J]．ACM Computing Surveys，1979，11(2)：121－137．

[6] 加西亚(Hector Garcia－Molina)，沃尔曼(Jeffrey D．Ullman)，威德姆(Jennifer D．Widom)．数据库系统实现[M]．北京：机械工业出版社，2001．1－238．

[7] 李泽，陈彬，唐俊翟等．C#函数适用手册[M]．北京：冶金工业出版社，2005．107－122．

[8] 斯梅切尔．C#和.NET 2.0 实战[M]．北京：人民邮电出版社，2008．94－107．

[9] 内格尔 (Nagel，C) 等．C#高级编程(第 6 版)[M]．北京：清华大学出版社，2008．1－358．

[10] D．DeWitt，H．T．Chou，R．Katz and A．Klug． Design and implementation of the Wisconsin Storage System. Software Practice and Experience，1985，15(10)：943-962．

[11] Ibrahim Jaluta，Seppo Sippu and Eljas Soisalon-Soininen．Concurrency control and recovery for balanced B trees[J]．The VLDB Journal，2005，14(2)：257 － 277．

[12] Mohan，C．An Efficient Method for Performing Record Deletions and Updates Using Index Scans[A]，Proc．28th International Conference on Very Large Databases[C]．Hong Kong，2002：940－949．

[13] C．Mohan and D．Haderle．Algorithms for Flexible Space Management in Transaction Systems Supporting Fine-Granularity Locking[A]．In Proceedings of the International Conference on Extending Database Technology[C]．Cambridge：1994．131－144．

[14] Rakesh Agrawal，Anastasia Ailamaki et al．The Claremont Report on Database Research [C]．ACM SIGMOD Record．Berkeley，2008，37(3)：8－19．

[15] Mark L．McAuliffe，Michael J．Carey and Marvin H．Solomon. Towards Effective and Efficient Free Space Management[J]．ACM SIGMOD Record，1996，25(2):：389-400．

[16] D．Batory，J．Barnett，J．Garza，K．Smith，K．Tsukuda，B．Twichell and T．Wise．GENESIS：An extensible database management system．In S．Zdonik and D．Maier，editors，Readings

in Object-Oriented Databases．Morgan Kaufmann，1990．

致　　谢

本文的写作过程中得到了很多老师、同学的帮助和指导。

感谢张坤龙老师，在张老师的一步步指导下，我完成了毕业设计的程序实现和论文的写作。跟随张老师的两年里，他在做学问和做人方面给予我莫大的启迪和指引。得益于张老师的长期培养，我有幸走进数据库管理系统的科研领域，体验到科学的魅力和做研究的快乐。

感谢杨亚军和杨晓科同学，他们与我共同完成了整个 TBase 教学数据库。我们一起讨论系统实现中遇到的问题，他们经常能在关键时刻给予我启发。

感谢实验室的谭龙飞师兄，在我的论文写作阶段，他们给予了我很多论文写作的建议，让我少走了不少弯路。

感谢我的挚友韩博、马国宁、何明、黄典和刘岫同学，在接近半年的毕业设计期间，是他们的关心和帮助，让我走出了情绪的低谷，顺利完成了毕业设计的各项任务。

再次对所有帮助过我的人表示衷心的感谢！