

**Mini-db详细设计文档**

BIT最强小分队



组长：蔡新帆 1120142030

组员：陈凯余 1120142031

叶南飞 1120142055

胡成杰 1120142036

赵堃宇 1120142056

日期 2016年9月1日星期四

200

目录

[引言 1](#_Toc460845483)

[编写目的 1](#_Toc460845484)

[背景 1](#_Toc460845485)

[术语 2](#_Toc460845486)

[参考资料 2](#_Toc460845487)

[变量及函数命名规范 3](#_Toc460845488)

[对命名规范的详细解释如下表： 3](#_Toc460845489)

[软件基本结构 6](#_Toc460845490)

[系统模块划分与设计 7](#_Toc460845491)

[词法分析模块： 7](#_Toc460845492)

[核心处理模块： 7](#_Toc460845493)

[文件处理模块： 7](#_Toc460845494)

[接口设计 8](#_Toc460845495)

[用户接口： 8](#_Toc460845496)

[外部接口: 8](#_Toc460845497)

[内部接口： 8](#_Toc460845498)

[数据结构 8](#_Toc460845499)

[算法设计 8](#_Toc460845500)

[1.关键字设计 8](#_Toc460845501)

[2.B+树索引算法 9](#_Toc460845502)

[技术难点及其解决方案 10](#_Toc460845503)

[项目源代码 11](#_Toc460845504)

[测试用例 15](#_Toc460845505)

# 引言

## 编写目的

为了深入学习和了解数据库以及小学期课程需要，我们小组决定自己编写一个简单的mini-database。

## 背景

1970年，IBM的研究员，有“关系数据库之父”之称的[埃德加·弗兰克·科德](http://baike.baidu.com/view/6819004.htm)（Edgar Frank Codd或E. F. Codd）博士在刊物《Communication of the ACM》上发表了题为“A Relational Model of Data for Large Shared Data banks（大型共享数据库的[关系模型](http://baike.baidu.com/view/176484.htm)）”的论文，文中首次提出了数据库的关系模型的概念，奠定了关系模型的理论基础。20世纪70年代末，关系方法的理论研究和软件系统的研制均取得了很大成果，IBM公司的San Jose实验室在IBM370系列机上研制的关系数据库实验系统System R历时6年获得成功。1981年IBM公司又宣布了具有System R全部特征的新的数据库产品SQL/DS问世。由于关系模型简单明了、具有坚实的数学理论基础，所以一经推出就受到了学术界和产业界的高度重视和广泛响应，并很快成为数据库市场的主流。20世纪80年代以来，计算机厂商推出的[数据库管理系统](http://baike.baidu.com/view/68446.htm)几乎都支持关系模型，数据库领域当前的研究工作大都以关系模型为基础。

本项目的主要命令都参考mysql完成。MySQL是一个关系型数据库管理系统，由瑞典MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 最流行的关系型数据库管理系统，在 WEB 应用方面MySQL是最好的 RDBMS (Relational Database Management System，关系数据库管理系统) 应用软件之一。MySQL是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。MySQL所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

## 术语

|  |  |  |
| --- | --- | --- |
| 序号 | 术语/缩略语 | 说明 |
| **1** | **B+tree** | **B+树** |
| **2** | **SGA** | **System Global Area 系统全局区** |
| **3** | **shared pool** | **共享池** |
| **4** | **data buffer area** | **数据缓冲区** |
| **5** | **redo log buffer** | **日志缓冲区** |
| **6** | **library cache** | **库缓存区** |
| **7** | **data directory cache** | **数据字典缓存区** |
| **8** | **SMON** | **System Monitor** |
| **9** | **DBWR** | **Data Base Writer** |
| **10** | **LGWR** | **Log Writer** |
| **11** | **CKPT** | **Check point** |

## 参考资料

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 文档名 | 作者 | 版本 |
| 1 | 数据库系统实现 | Hector Garcia-Molina  Jeffrey D.Ullman  Jennifer Widom | 第二版 |
| 2 | 数据库系统概念 | Abrahanm Silberschatz Henry F.Korth S.Sudarshan | 第六版 |
| 3 | 数据库系统基础教程 | Jeffrey D.ullman Jennifer | 第三版 |
| 4 | 数据库系统全书 | Hector.Garcia-Molina  Jeffrey D.ullman  Jennifer Widom | 第一版 |

# 变量及函数命名规范

变量名称定义：变量单词首字母大写，前面再加一个前缀（小写）；

类型定义：类型单词全大写，前面加前缀（大写）；

函数名定义：函数名（大驼峰式命名法）；

常数命名：用大写命名，中间用\_符号隔开；

文件命名：根据驱动模块命名(字母全部小写);

驱动应用程序的驱动名加”app”。

## 对命名规范的详细解释如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| **变量类型** | **前缀** | **示例** | **备注** |
| **全局变量** | g\_ | *g\_usFsmCurrentState* |  |
| **静态变量** | s\_ | *s\_ulCounter* |  |
| **数组Array** | a | *g\_astrStateTbl[FSM\_STATE\_MAX]* | **全局变量的一个结构体数组** |
| **Char** | c | *cName* |  |
| **Int** | i | *iNumber* |  |
| **unsigned char** | uc | *ucTemp* |  |
| **BOOL(int)** | b | *bLogic* |  |
| **unsigned long(32)** | ul | *ulRetVal* |  |
| **unsigned short(16)** | us | *usMinNo* |  |
| **signed long(32)** | l | *lVoltageOffset* |  |
| **signed short(16)** | s | *sSumVal* |  |
| **float** | f | *fGain* |  |
| **double** | d | *dGain* |  |
| **结构体** | str | *g\_astrTriggerTbl[FSM\_MAX];* | **全局变量+数组+结构体** |
| **指针** | p | *\* pRegState* |  |
| **结构体指针** | pstr | *\* pstrState* |  |
| **枚举** | e | *eType* |  |
| **位定义** | b(数字） | INT16U b1EnableFlag :1; | **数字表示bit数量** |
| INT16U b1DisableFlag :1; |
| INT16U b3SetTimeUnit :3; |
| **联合体** | un | unWordType |  |
|  |  |  |  |
| **类型定义** | 前缀 | 示例 | **备注** |
| **枚举** | ENUM\_ | typedef enum \_ENUM\_TYPE{  eType1=0,  eType2,  eType3 }ENUM\_TYPE; |  |
| **联合体** | UN\_ | typedef union \_UN\_WORD{  unsigned long ulWord;  struct{  unsigned short usWordL;  unsigned short usWordH;  }Word; }UN\_WORD; |  |
| **结构体** | STR\_ | typedef struct \_STR\_DATA{  unsigned short usData1;  unsigned long ulData2;  unsigned char ucData3;  ENUM\_TYPE eType;  UN\_WORD unData4;  union {  unsigned short usWord;  struct{  unsigned short b1Ena:1;  unsigned short b1Flag:1;  unsigned short b14Flag:14;  }Bit;  }unData5; }STR\_DATA; |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **函数命名规则** |  |  |  |
| **函数名（驼峰式）** |  | HasFault(void) |  |
|  |  | RegisterState(FSM\_TYPE \* pRegState) |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **常数命名** |  |  |  |
| **用大写命名，中间用\_符号隔开** |  | #define ALARM\_INFO\_MAX\_NUM 200 |  |
|  | #define ALARM\_GROUP\_MAX\_NUM 12 |  |
|  | #define ALARM\_BIT\_MAX\_NUM 16 |  |
|  | #define ALARM\_MAX\_NUM 300 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **文件命名** | 根据驱动模块命名(字母全部小写) | 如：spi.c、spi.h |  |
| 驱动应用程序的驱动加app | 如：spiapp.c\spiapp.h |

|  |  |  |
| --- | --- | --- |
| **编程细则** | | |
| **使用条件语句时，常量放左边，函数或变量放右边** | 如：if (compare==1){  ++i;  } |  |

# 软件基本结构

软件由实例和数据库组成：

 数据库由数据文件（包含表头文件、纪录文件、索引等数据）、控制文件（包括每个表的操作信息）、日志文件（数据操作sql语句）、参数文件（分配内存区域并定位控制文件的位置）、口令文件、日志归档文件（归档模式下）（服务器崩溃、硬盘损坏情况下，通过日志恢复时用）

\* 实例由 内存结构（memory strutct） 和 后台进程（background processor）组成。

SGA： System Global Area 系统全局区，主要是给实例使用，包括共享池 （shared pool） 、数据缓冲区（ data buffer area） ,日志缓冲区（ redo log buffer） .

**共享池(shared pool):**

包括library cache 、data directory cache 组成，其中 library cache 主要保存最近的sql 检查、编译、执行计划， 下次有同样语句过来的时候，可以重用这些，避免重复的检查编译执行计划。 data directory cache 主要保存数据库数据表的字段定义、索引数据等， shared pool 的大小直接影响到数据库的性能。

data buffer area : 主要保存用户对数据的修改，查询操作。该内存区域的大小直接影响数据库的性能。

redo log buffer area： 主要保存最近用户对数据库的操作记录，该大小对数据库性能没有多大影响

\*SMON（System Monitor） 监控SGA的健康情况，收集SGA碎片内存，监控实例健康情况

\*DBWR（Data Base Writer） 维护data buffer area 和物理表数据的一致性

\*LGWR （Log Writer）维护redo log buffer area 内存数据和日志文件的一致性

\*CKPT（Check point） 设置检查点，在oracle 实例出现问题的时候，可以恢复到实例失败前的情况。



# 系统模块划分与设计

## 词法分析模块：

主要是处理用户的输入，将用户的输入转换成程序可执行的语句。它拟提供以下功能：程序使用简要说明，对用户输入进行词法分析并提取参数及关键字，对于错误输入报错并提示正确的输入格式，将用户输入的信息备份在日志中。

|  |  |  |
| --- | --- | --- |
| LexicalAnalysis句法分析模块 | | 传入参数 |
| Analysis() | 从输入流中取出主要命令，调用各命令对参数的分析 |  |
| CommandHelp() | 当输入出现错误时对用户提供帮助 | int调用的帮助项目序号 |
| Use() | 提供USE命令的句法分析 |  |
| Back() | 提供BACK命令的句法分析 |  |
| CreateDataBase() | 提供CREATE DATABASE命令的句法分析 |  |
| CreateTable() | 提供CREATE TABLE命令的句法分析 |  |
| Inster() | 提供INSTER命令的句法分析 |  |
| Update() | 提供UPDATE命令的句法分析 |  |
| Delete() | 提供DELETE命令的句法分析 |  |
| Select() | 提供SELECT命令的句法分析 |  |
| Add() | 提供ADD命令的句法分析 |  |
| Save() | 提供SAVE命令的句法分析 |  |

## 核心处理模块：

对已从磁盘加载进内存中的数据进行处理，并利用索引实现增删改查等功能，将结果显示在控制台上.

## 文件处理模块：

将内存与磁盘数据同步(日志、表单读取、新建库或表单)

|  |  |  |
| --- | --- | --- |
| FilesProcessing句法分析模块 | | 传入参数 |
| ChangePath () | 根据输入参数提取出文件路径 |  |
| CreateFolder () | 创建文件夹（即库） | string传入文件路径 |
| DeleteFolde() | 删除文件夹 | string传入文件路径 |
| Rename() | 更改名称 | string原名称，string现名称 |
| WriteTxt() | 将记录写入txt文件中 | string表名，vector记录内容 |
| ReadTxt() | 从txt中读出记录 | string表名，vector记录内容 |

# 接口设计

## 用户接口：

用户在控制台输入与SQL命令类似的语句来操作软件，具体的语句格式见需求分析文档。

本项目中Analysis()是从控制台获取用户输入并且进行分析的主要函数，他从输入中提取到关键命令后将调用函数对参数进行进一步提取。而后CreateDataBase()，CreateTable()等函数将在提取完参数后调用核心处理模块。进而完成用户接口的功能

## 外部接口:

本软件主要通过文件处理模块与磁盘就行交互。

外部接口：

CreateFolder ()，DeleteFolde()，Rename()等文件及文件夹的创建删除等功能。

WriteTxt()和ReadTxt()函数主要负责完成内存和磁盘的同步

## 内部接口：

将上述三个模块封装为三个类，模块之间用类的成员函数进行调用，尽量在设计类时减小它们间的耦合度，实现内部接口功能。

# 数据结构

1．表头为结构体数组，结构体包含字段名、数据类型等

2．数据记录结构为一系列数组，每一列数据对应一个数组，将角标相同的各数组元素逻辑上视为一条记录中的成员。

3．索引结构为B+树，每个节点指向一个记录的角标。

# 算法设计

## 1.关键字设计

**CREATE：**在文件夹中新建一个表头和一个数据文件，表头内容由词法分析模块提供的参数确认

**INSTER：**将插入的数据保存在每一个对应数组的最后一个，也就是指数据的存盘是追加的形式。

**UPDATE：**循环加判断，找到所有符合条件的值进行替换，先在内存中处理再进行数据的存盘

**DELETE**：用循环，找到删除以后直接break，先在内存中处理再进行数据的存盘

**SELECT**： 循环

（1）直接按格式将内容输出

（2）在取完整条数据内容后加条件输出

（3）在循环时取指定内容进行判断，符合条件的输出

（4）先按要求排序（如果数据结构支持则不需要这一步），然后按照INSTER算法进行输出

其中的排序和查找由B+树本身的性质就可以高效实现。

## 2.B+树索引算法

1.B+树叶子节点定义

struct LeafNode

{

vector<Key> keys;

vector<Value> values;

PagePointer next\_page;

};

2.B+树内部节点定义

struct InteriorNode

{

vector<Key> keys;

vector<PagePointer> pointers;

};

3.search操作

/\*\*

\* finds the leaf node that \_should\_ contain the entry with the specified key

\*/

LeafNode search(Node root, Key key)

/\*\*

\* Inserts a key/val pair into the tree.

\* Returns the root of the new tree which \_may\_ be different

\* from the old root node.

\*/

InteriorNode insert\_into\_tree(InteriorNode root, Key newk4.ey, Value val)

4.insert操作

1） 寻找insert的正确的目标leaf node

2） 向目标leaf node中尝试insert操作

InteriorNode insert\_into\_tree(InteriorNode root, Key newkey, Value val)

{

LeafNode leaf = search(root, newkey);

return insert\_into\_node(leaf, newkey, val);

}

其中，insert\_into\_node中，要做如下的一些事：

/\*\*

\* Tries to inserts the (newkey/val) pair into

\* the node.

\*

\* If `target` is an interior node, then `val` must be a page pointer.

\*/

InteriorNode insert\_into\_node(Node target, newkey, val)

{

if( ... CASE 1 ... )

{

/\* handle CASE 1 \*/

}

else if( ... CASE 2 ... )

{

/\* handle CASE 2 \*/

}

else if( ... CASE 3 ... )

{

/\* handle CASE 2 \*/

}

}

其中三个不同的case包括：

A) 目标leaf node有足够的空间保存key

B) 目标leaf node已满，但是它的parent node（父节点）有足够的空间保存key

C) 目标leaf node和它的parent node已满。

4.delete操作

删除算法是insert算法的逆过程

# 技术难点及其解决方案

1. 词法分析模块对于用户多变的输入，如何正确地判段输入的合法性，再提取出关键字和参数是相当麻烦的，我们暂拟定用正则表达式来实现主体部分。
2. 对记录的数据结构方面，我们开始打算使用变长的结构体，然后发现并不能在程序运行后来确定结构体大小，所以我们就使用一系列数组来实现，只是要将角标相同的元素逻辑上视为一个记录结构体的成员就行。
3. 对于索引方面，首先B+树这种结构该如何存盘就是个非常棘手的问题，如果我们不能找到有效的方法的话，我们将采用二叉查找树来代替B+树。
4. 文本文件的读写可能会出现意外情况，而这方面我们了解不多。

# 项目源代码

files\_processing .h

#pragma once

//文件处理类的定义

#include<fstream>

#include<iostream>

#include<string>

#include<vector>

#include<direct.h>

using namespace std;

class FilesProcessing

{

public:

static void ChangePath(string path\_name = "");

static void CreateFolder(string dir\_name);

static void DeleteFolder(string dir\_name);

static void Rename(string pre\_name, string new\_name);

static void WriteTxt(string name, vector<string> &record);

static void ReadTxt(string name, vector<string> &record);

};

lexical\_analysis.h

#pragma once

//词法分析类的定义

#include<iostream>

#include<string>

#include<regex>

using namespace std;

class LexicalAnalysis

{

public:

LexicalAnalysis(string str);

~LexicalAnalysis();

void Analysis(); //判断是否合法，并归类命令

void CommandHelp(int n); //显示输入非法并提供命令格式帮助

void Use();

void Back();

void CreateDataBase();

void CreateTable();

void Inster();

void Update();

void Delete();

void Select();

void Add();

void Save();

private:

string mStr; //要处理的字符串

string mKeyWord; //关键字

vector<string> mParameter; //参数表

bool mCorrect; //输入是否合法

};

files\_processing .cpp

#include"files\_processing .h"

void FilesProcessing::ChangePath(string path\_name)

{

if (path\_name.empty()) {

char path[\_MAX\_DIR];

\_getcwd(path, \_MAX\_DIR);

string path0 = path;

int tmp = path0.find\_last\_of("/\\");

path\_name = path0.substr(0, tmp);

}

\_chdir(path\_name.c\_str());

}

void FilesProcessing::CreateFolder(string dir\_name)

{

\_mkdir(dir\_name.c\_str());

}

void FilesProcessing::DeleteFolder(string dir\_name)

{

\_rmdir(dir\_name.c\_str());

}

void FilesProcessing::Rename(string pre\_name, string new\_name)

{

rename(pre\_name.c\_str(), new\_name.c\_str());

}

void FilesProcessing::WriteTxt(string name, vector<string> &record)

{

ofstream ptxt(name, ios::out);

if (!ptxt.is\_open())

cout << "写入文件出错！！" << endl;

int len = record.size();

for (int i = 0; i < len; i++)

ptxt << record[i] << endl;

ptxt.close();

}

void FilesProcessing::ReadTxt(string name, vector<string> &record)

{

ifstream gtxt(name, ios::in);

if (!gtxt.is\_open())

cout << "读取文件出错！！" << endl;

string tmp;

while (getline(gtxt, tmp))

record.push\_back(tmp);

gtxt.close();

}

lexical\_analysis.cpp

#include"files\_processing .h"

void FilesProcessing::ChangePath(string path\_name)

{

if (path\_name.empty()) {

char path[\_MAX\_DIR];

\_getcwd(path, \_MAX\_DIR);

string path0 = path;

int tmp = path0.find\_last\_of("/\\");

path\_name = path0.substr(0, tmp);

}

\_chdir(path\_name.c\_str());

}

void FilesProcessing::CreateFolder(string dir\_name)

{

\_mkdir(dir\_name.c\_str());

}

void FilesProcessing::DeleteFolder(string dir\_name)

{

\_rmdir(dir\_name.c\_str());

}

void FilesProcessing::Rename(string pre\_name, string new\_name)

{

rename(pre\_name.c\_str(), new\_name.c\_str());

}

void FilesProcessing::WriteTxt(string name, vector<string> &record)

{

ofstream ptxt(name, ios::out);

if (!ptxt.is\_open())

cout << "写入文件出错！！" << endl;

int len = record.size();

for (int i = 0; i < len; i++)

ptxt << record[i] << endl;

ptxt.close();

}

void FilesProcessing::ReadTxt(string name, vector<string> &record)

{

ifstream gtxt(name, ios::in);

if (!gtxt.is\_open())

cout << "读取文件出错！！" << endl;

string tmp;

while (getline(gtxt, tmp))

record.push\_back(tmp);

gtxt.close();

}

# 测试用例

CREATE 学生 （学号 INT,姓名 CHAR,性别 CHAR,生日 DATA,年龄 INT）;

USE 学生:

INSERT

VALUES ("0001", "张三", "男", #1996-1-1#, "20");

INSERT

VALUES ("0001", "张三", "男", #1996-1-1#, "20");（设置两组完全相同的数据）

INSERT

VALUES (0001, "张三", "男", #1996-1-1#, "20");（没加引号，报错）

INSERT

VALUES ("0002", "张四", "女", #1996-1-1#, "20");

INSERT

VALUES ("0003", "张五", "男", #1985-1-1#, "21");

INSERT

VALUES ("0004", "张六", "男", #1995-1-1#, "21");

INSERT

VALUES ("0005", "张七", "女", #1995-1-1#, "21");

INSERT

VALUES ("0006", "张八", "男", #1995-1-1#, "21");

INSERT (学号,姓名,性别)

VALUES ("0007", "李四", "女");

UPDATE SET 年龄= 31 WHERE 姓名="张五";

SELECT \*;（输出所有记录，输出格式没给没办法写预测输出）

SELECT 姓名,性别;（输出姓名，性别两列）

SELECT 姓名,性别；（分号不同）

SELECT 姓名,性别 WHERE 年龄<20；

ADD 成绩 INT;

ADD 成绩 INT;（添加相同表头）

ADD 成绩 IN;（没有的数据类型，报错）

ADD 1 CHAR;

BACK;

CREAT 老师（工号 INT,姓名 CHAR）;

USE 老师；

SELECT \*;（输出空表）

BACK；

USE 学生;

CREATE;（层级不对，应该报错）

INSERT (雇员号,姓名,性别)

VALUES ("0002", "王宏", "男");（表头没有雇员号，应该报错）

UPDATE SET 出生日期= #1960-1-11# WHERE 姓名="张磊";（找不到符合条件的人，应该报错）

DELETE WHERE 学号 = "0002";

SELECT \*;(应该输出中没有2号学生)

SAVE;

(关闭程序再打开)

USE 学生；

SELECT \*;（输出应该和关闭前不变）