**《数据科学案例综合实训》实验报告**

实验序号：2 实验项目名称：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学　　号 | 2010716318 | 姓　　名 | 黄飞阳 | 专业、班 | 20大数据1班 | |
| 实验地点 | 精工1-214 | 指导教师 | 王琰 | 实验时间 | 2022/6/22 | |
| 1. 实验目的及要求   1、理解和掌握数据库中建表过程的基本原理和表结构信息存储的方法  2、Linux下，用C语言实现数据库建表的过程。  二、实验设备（环境）及要求  1、操作系统: Linux  2、参考的开源数据库管理系统：PostgreSQL数据库   1. 实验内容与步骤 2. 查看PostgreSQL源码中src目录下的makefile文件，找到它调用gcc编译具体c文件的命令位置。并说明如果要生成调试版的名为“postgres”的可执行文件，可以用哪些方法，并验证该方法的可行性。   在pgsrc下执行./configure --enable-debug -prefix=/home/hfy/pgsql  再make、make install      查看过程中执行的函数    栈帧，从下往上调用的。  删除断点并继续执行    2、假定数据库中，表名存储在数据字典表pg\_class（该表的id是1259），该表中的一行对应一张表的信息，每行由多个列值构成，分别存储该行所对应表的名称、该表对应的id（也是该表数据所在文件的名称）、该表的属性数量等。请设计一个结构体，用于将pg\_class表中一行数据存储在内存中。  3、假定数据库中，列的信息存储在数据字典表pg\_attribute（该表的id是1249），该表中的一行对应某一列信息，每行由多个列值构成，分别存储该行所对应的列属于哪张表（用表的id表示）、该列的名称、该列的类型、列的字节长度、该列是所在表的第几列、该列是否非空等。请设计一个结构体，用于将pg\_attribute表中一行数据存储在内存中  4、调用操作系统课程实验中所实现的insert函数，将pg\_class、pg\_attribute这两张表的表描述信息和列描述信息存储到上述的数据字典表中。（插入到文件）  5、设计一个建表函数create，功能是创建一张用户表（如student\_XXXX表，XXXX表示组长学号的后四位），  将用户表的描述信息插入pg\_class和pg\_attribute这两张字典表中。  该函数输入参数包括：第一个是表名，第二个是数组，数组中的每个元素对应该表中某列的描述。具体描述：生成表对应的id，只要与之前用过的id不重复即可。遍历pg\_class表，判断该表名是否已经存在，如存在则报错并退出。  调用insert函数，将用户表的表名、该表对应id和属性数量等信息插入pg\_class表。最后，循环调用insert函数，将对应的列信息插入pg\_attribute表。    如果表的文件id重复，那么是不允许创建的。    6、往新建的用户表中插入多行信息，要达到2个块  7、执行遍历程序，将该用户表的信息打印到控制台        8、执行遍历程序，将pg\_class表的信息打印到控制台      9、执行遍历程序，打印pg\_attribute表的信息打印到控制台    四、分析与讨论  c++和c语言不支持多类型数组，强行实现可以用void\*，但是这让代码难以维护且数据只能保存在内存当中，数据存到数据库中最好以string 的方式，还原时通过属性表（记录类型）来还原。 | | | | | | |
| 五、教师评语  签名：  日期： | | | | | | 成绩 |

附源程序清单：

main.cpp

#include <stdlib.h>

#include <time.h>

#include <cstdio>

#include <fstream>

#include <iostream>

#include <string>

#include <unordered\_set>

#include <vector>

#include <cstring>

#include <filesystem>

#include <typeinfo>

#include <random>

#define BLOCK\_SIZE 8192

using std::filesystem::directory\_iterator;

using *namespace* std;

vector <string> get\_folder\_filenames(string folder) {

    vector <string> filenames;

    for (*const* *auto* &file : directory\_iterator(folder)) {

        string s = file.path().string();

        for (*int* i = s.size() - 1; i >= 0; i--) {

            if (s[i] == '\\') {

                s = s.substr(i + 1, s.size() - 1);

                break;

            }

        }

        filenames.push\_back(s);

    }

    return filenames;

}

*struct* InfoObject {

*virtual* vector <string> to\_string\_vec() { return vector<string>(); }

*void* print() {

        for (string s : *this*->to\_string\_vec()) {

            cout << s << endl;

        }

    }

};

*/\*\**

*\* 对应一行表的信息*

*\* table\_name : 表名*

*\* table\_id : 表的 id*

*\* table\_attr\_cnt : 表的属性数量*

*\*/*

*struct* TableInfo : *public* InfoObject {

    string table\_name;

*int* table\_id;

*int* table\_attr\_cnt;

    TableInfo() {}

    TableInfo(

            string *table\_name*,

*int* *table\_id*,

*int* *table\_attr\_cnt*

    ) :

            table\_name(table\_name),

            table\_id(table\_id),

            table\_attr\_cnt(table\_attr\_cnt) {}

    vector <string> to\_string\_vec() {

        vector <string> ret;

        ret.push\_back(*this*->table\_name);

        ret.push\_back(std::to\_string(*this*->table\_id));

        ret.push\_back(std::to\_string(*this*->table\_attr\_cnt));

        return ret;

    }

*void* print() {

        printf("table\_name : %s\ntable\_id : %d\ntable\_attr\_cnt : %d\n", *this*->table\_name.c\_str(), *this*->table\_id,

*this*->table\_attr\_cnt);

    }

};

*/\*\**

*\* 对应一个属性（字段）的信息*

*\* table\_name : 所属表名*

*\* table\_id : 所属表的 id*

*\* column\_name : 字段名*

*\* column\_type : 字段类型*

*\* column\_bytes\_len : 字段长度*

*\* column\_index : 字段在第几列*

*\* is\_null : 字段是否为空*

*\*/*

*struct* AttributeInfo : *public* InfoObject {

    string table\_name;

*int* table\_id;

    string column\_name;

    string column\_type;

*int* column\_bytes\_len;

*int* column\_index;

*bool* is\_null;

    AttributeInfo() {}

    AttributeInfo(

            string *table\_name*,

*int* *table\_id*,

            string *column\_name*,

            string *column\_type*,

*int* *column\_bytes\_len*,

*int* *column\_index*,

*bool* *is\_null*

    ) :

            table\_name(table\_name),

            table\_id(table\_id),

            column\_name(column\_name),

            column\_type(column\_type),

            column\_bytes\_len(column\_bytes\_len),

            column\_index(column\_index),

            is\_null(is\_null) {}

    vector <string> to\_string\_vec() {

        vector <string> ret;

        ret.push\_back(*this*->table\_name);

        ret.push\_back(std::to\_string(*this*->table\_id));

        ret.push\_back(*this*->column\_name);

        ret.push\_back(*this*->column\_type);

        ret.push\_back(std::to\_string(*this*->column\_bytes\_len));

        ret.push\_back(std::to\_string(*this*->column\_index));

        ret.push\_back(std::to\_string(*this*->is\_null));

        return ret;

    }

};

*/\*\**

*\* @brief 块空间异常*

*\* 如果【要插入的数据 + 块内数据 > 块大小】，则抛出此异常*

*\**

*\*/*

*struct* BlockSpaceError : *public* exception {

*const* *char* *\**what() *const* throw() { return "Not enough block space"; }

};

*/\*\**

*\* @brief 表Id缺失异常*

*\* 如果将表的内容存放到文件中之前，没用设置表的Id，则抛出此异常*

*\*/*

*struct* TableIdMissingError : *public* exception {

*const* *char* *\**what() *const* throw() { return "Table id has not been set"; }

};

*/\*\**

*\* @brief 表Id重复异常*

*\* 如果将表保存到文件时，该表的 Id 已经存在，则抛出此异常*

*\*/*

*struct* TableIdRepeatError : *public* exception {

*int* repeat\_table\_id;

    TableIdRepeatError(*int* *repeat\_table\_id*) : repeat\_table\_id(repeat\_table\_id) {}

*const* *char* *\**what() *const* throw() {

        return ("Table id " + to\_string(*this*->repeat\_table\_id) + " already exists").c\_str();

    }

    string error() {

        return ("Table id " + to\_string(*this*->repeat\_table\_id) + " already exists").c\_str();

    }

};

*/\*\**

*\* @brief 块内保留空间，暂时没有作用*

*\**

*\*/*

*struct* Special {

*int* get\_size() { return 0; }

*void* print() { cout << "Special" << endl; }

};

*/\*\**

*\* @brief 块头，记录块的部分信息*

*\**

*\*/*

*struct* PageHeaderData {

*int* start\_special;

*int* free\_space\_start;

*int* free\_space\_end;

*int* table\_id;

    PageHeaderData() {

        start\_special = BLOCK\_SIZE;

        free\_space\_start = get\_size();

        free\_space\_end = 0;

        table\_id = 0;

    }

*int* get\_size() { return sizeof(PageHeaderData); }

*void* print() {

        printf(

                "start special : %d\nfree space start : %d\nfree space end : "

                "%d\ntable id : %d\n",

                start\_special, free\_space\_start, free\_space\_end, table\_id);

    }

};

*/\*\**

*\* @brief 块内数据项*

*\**

*\*/*

*struct* TupleData {

*int* n\_attr;

    vector <string> data\_vec;

*void* insert(vector <string> *data\_vec*, vector <string> *attr\_desc*) {

*this*->data\_vec = data\_vec;

*this*->n\_attr = attr\_desc.size();

    }

*void* print() {

        for (string s : *this*->data\_vec) {

            cout << s << "\t";

        }

        cout << endl;

    }

*int* get\_size() {

*int* size = sizeof(TupleData);

        for (string s : *this*->data\_vec) {

            size += s.size();

        }

        return size;

    }

};

*/\*\**

*\* @brief 块内数据项头部*

*\**

*\*/*

*struct* Item {

*int* tuple\_data\_start;

*int* len\_tuple\_data;

*int* get\_size() { return sizeof(Item); }

*void* insert(*int* *tuple\_data\_start*, *int* *len\_tuple\_data*) {

*this*->tuple\_data\_start = tuple\_data\_start;

*this*->len\_tuple\_data = len\_tuple\_data;

    }

*void* print() {

        cout << "tuple start : " << tuple\_data\_start

             << "\ttuple len : " << len\_tuple\_data << endl;

    }

};

*/\*\**

*\* @brief 计算空间大小*

*\**

*\*/*

*struct* SizeCalculator {

*static* *int* get\_string\_vec\_size(vector <string> *string\_vec*) {

*int* size = sizeof(string\_vec);

        for (*auto* x : string\_vec) {

            size += x.size();

        }

        return size;

    }

*static* *int* get\_item\_vec\_size(vector <Item> *item\_vec*) {

*int* size = sizeof(item\_vec);

        for (*auto* x : item\_vec) {

            size += x.get\_size();

        }

        return size;

    }

*static* *int* get\_tuple\_vec\_size(vector <TupleData> *tuple\_vec*) {

*int* size = sizeof(tuple\_vec);

        for (*auto* x : tuple\_vec) {

            size += x.get\_size();

        }

        return size;

    }

};

*/\*\**

*\* @brief 块*

*\**

*\*/*

*struct* Block {

    PageHeaderData phd;

    vector <Item> item\_vec;

    vector <TupleData> tuple\_data\_vec;

    Special special;

*void* set\_table\_id(*int* *table\_id*) { *this*->phd.table\_id = table\_id; }

*int* get\_size() {

*int* size = *this*->phd.get\_size() + *this*->special.get\_size();

        for (*int* i = 0; i < *this*->tuple\_data\_vec.size(); i++) {

            size += *this*->item\_vec[i].get\_size();

            size += *this*->tuple\_data\_vec[i].get\_size();

        }

        return size;

    }

*void* insert(vector <string> *data\_vec*, vector <string> *attr\_desc*) {

*int* data\_size = SizeCalculator::get\_string\_vec\_size(data\_vec);

        if (*this*->get\_size() + data\_size > BLOCK\_SIZE) {

            throw BlockSpaceError();

        }

        TupleData td;

        td.insert(data\_vec, attr\_desc);

*this*->tuple\_data\_vec.push\_back(td);

*int* tuple\_size =

                SizeCalculator::get\_tuple\_vec\_size(*this*->tuple\_data\_vec);

        Item it;

        it.insert(BLOCK\_SIZE - tuple\_size, data\_size);

*this*->item\_vec.push\_back(it);

*int* item\_size = SizeCalculator::get\_item\_vec\_size(*this*->item\_vec);

*this*->phd.free\_space\_end = BLOCK\_SIZE - special.get\_size() - data\_size;

*this*->phd.free\_space\_start = *this*->phd.get\_size() + item\_size;

    }

*void* print(*int* *block\_id*) {

        cout << "========================  Start Block (" << block\_id

             << ")  ========================\n"

             << endl;

        printf(" Block Size[ %d ] \n\n", *this*->get\_size());

        cout << "===========  PageHeaderData  ===========" << endl;

*this*->phd.print();

        cout << "\n\n===========  Item  ===========" << endl;

        for (Item it : *this*->item\_vec) {

            it.print();

        }

        cout << "\n\n===========  TupleData  ===========" << endl;

        for (TupleData td : *this*->tuple\_data\_vec) {

            td.print();

        }

        cout << "\n\n===========  Special  ===========" << endl;

*this*->special.print();

        cout << "\n\n========================  End Block (" << block\_id

             << ") ========================\n\n"

             << endl;

    }

};

*/\*\**

*\* 获取一个长度为 n 的随机名字*

*\* @param len*

*\* @return 随机名字*

*\*/*

string get\_random\_name(*int* *len*) {

    string name;

*char* first\_name = *char*(*int*(65 + rand() % 26)); *// A~Z  65~90*

    name.push\_back(first\_name);

    for (*int* i = 0; i < len; i++) {

*// a~z  97~122*

        name.push\_back(*int*(97 + rand() % 26));

    }

    return name;

}

*/\*\**

*\* 获取 n 个长度为 m 的随机字符串*

*\* @param m*

*\* @param n*

*\* @return 随机字符串*

*\*/*

vector <string> get\_random\_string(*int* m, *int* n) {

    vector <string> ret;

    for (*int* i = 0; i < n; i++) {

        ret.push\_back(get\_random\_name(m));

    }

    return ret;

}

*/\*\**

*\* 块管理器*

*\* 每个块管理器对应一张表*

*\*/*

*struct* BlockManager {

    vector <Block> block\_vec;

    BlockManager() {}

    BlockManager(*int* *table\_id*) {

        if (*this*->block\_vec.empty()) {

*this*->block\_vec.push\_back(Block());

        }

*this*->set\_table\_id(table\_id);

    }

*/\*\**

*\* 获取块管理器中的块数量*

*\* @return*

*\*/*

*int* get\_block\_num() { return *this*->block\_vec.size(); }

*/\*\**

*\* 获取表的 Id*

*\* @return*

*\*/*

*int* get\_table\_id() {

        if (*this*->block\_vec.empty()) {

        } else {

            return *this*->block\_vec[0].phd.table\_id;

        }

        return -1;

    }

*/\*\**

*\* 设置表的 Id*

*\* @param table\_id*

*\*/*

*void* set\_table\_id(*int* *table\_id*) {

        for (Block &block : *this*->block\_vec) {

            block.phd.table\_id = table\_id;

        }

    }

*/\*\**

*\* 插入数据到表中*

*\* @param data\_vec*

*\* @param attr\_desc*

*\*/*

*void* insert(vector <string> *data\_vec*, vector <string> *attr\_desc*) {

        if (*this*->block\_vec.empty()) {

*this*->block\_vec.push\_back(Block());

        }

*int* last\_block\_id = block\_vec.size() - 1;

        try {

            block\_vec[last\_block\_id].insert(data\_vec, attr\_desc);

        }

        catch (BlockSpaceError &bse) {

            cout << bse.what() << endl;

*this*->extend(data\_vec, attr\_desc);

        }

    }

*/\*\**

*\* @brief 当一个块的空间不足时，需要扩展*

*\**

*\* @param data\_vec 数据元组*

*\* @param attr\_desc 数据元组的属性描述*

*\*/*

*void* extend(vector <string> *data\_vec*, vector <string> *attr\_desc*) {

        Block new\_b;

        new\_b.insert(data\_vec, attr\_desc);

*this*->block\_vec.push\_back(new\_b);

    }

*/\*\**

*\* 打印表的内容*

*\*/*

*void* print() {

        for (*int* i = 0; i < *this*->block\_vec.size(); i++) {

            block\_vec[i].print(i);

        }

    }

*/\*\**

*\* @brief 将结构体信息写入文件*

*\**

*\*/*

*void* to\_file() {

        try {

            vector <string> folder\_filenames = get\_folder\_filenames("./db\_folder");

*int* table\_id = *this*->get\_table\_id();

            string file\_path = "./db\_folder/" + to\_string(table\_id);

            for (string filenames : folder\_filenames) {

                if (to\_string(table\_id) == filenames) {

                    throw TableIdRepeatError(*this*->get\_table\_id());

                }

            }

            FILE \*fp = fopen(file\_path.c\_str(), "wb");

            for (*int* i = 0; i < *this*->block\_vec.size(); i++) {

                Block \*block = (Block \*) calloc(1, BLOCK\_SIZE); *// malloc 报错*

                if (block == nullptr) {

                    cout << "申请空间失败" << endl;

                    return;

                }

*/\*Block cur = this->block\_vec[i];*

*block->phd = cur.phd;*

*block->item\_vec = cur.item\_vec;*

*block->tuple\_data\_vec = cur.tuple\_data\_vec;*

*block->special = cur.special;\*/*

                \*block = *this*->block\_vec[i];

                fwrite(block, BLOCK\_SIZE, 1, fp);

            }

            fclose(fp);

        } catch (TableIdRepeatError &e) {

            cout << e.error() << endl;

        } catch (TableIdMissingError &e) {

            cout << e.what() << endl;

        } catch (exception &e) {

            cout << e.what() << endl;

        }

    }

*/\*\**

*\* @brief 从文件中读取信息到结构体*

*\**

*\* @param bm 被写入的结构体*

*\*/*

*void* from\_file(BlockManager *&bm*) {

        try {

            string file\_path = "./db\_folder/" + to\_string(*this*->get\_table\_id());

            FILE \*fp = fopen(file\_path.c\_str(), "rb");

            if (fp == nullptr || feof(fp)) {

                cout << "file is null" << endl;

                return;

            }

            Block \*block = (Block \*) calloc(1, BLOCK\_SIZE);

            while (!feof(fp) && fread(block, BLOCK\_SIZE, 1, fp) == 1) {

                Block b(\*block);

                bm.block\_vec.push\_back(b);

                memset(block, 0, BLOCK\_SIZE);

            }

            fclose(fp);

        } catch (TableIdMissingError &e) {

            cout << e.what() << endl;

        } catch (exception &e) {

            cout << e.what() << endl;

        }

    }

};

*void* get\_info(

        vector <TableInfo> *&table\_info\_vec*,

        vector <AttributeInfo> *&attr\_info\_vec*

) {

*int* pg\_class\_id = 1001;

*int* pg\_attribute\_id = 1002;

*int* pg\_student\_id = 1003;

    vector <TableInfo> table\_info\_vec\_template = {

            TableInfo("pg\_class", pg\_class\_id, 3),

            TableInfo("pg\_attribute", pg\_attribute\_id, 6),

            TableInfo("pg\_student", pg\_student\_id, 3)

    };

    vector <AttributeInfo> attr\_info\_vec\_template = {

*// pg\_class 表的属性*

            AttributeInfo("pg\_class", pg\_class\_id, "table\_name", "string", 20, 1, false),

            AttributeInfo("pg\_class", pg\_class\_id, "table\_id", "int", 4, 2, false),

            AttributeInfo("pg\_class", pg\_class\_id, "table\_attr\_cnt", "string", 20, 3, false),

*// pg\_attribute 表的属性*

            AttributeInfo("pg\_attribute", pg\_attribute\_id, "table\_name", "string", 20, 1, false),

            AttributeInfo("pg\_attribute", pg\_attribute\_id, "table\_id", "int", 4, 2, false),

            AttributeInfo("pg\_attribute", pg\_attribute\_id, "column\_name", "string", 20, 3, false),

            AttributeInfo("pg\_attribute", pg\_attribute\_id, "column\_type", "string", 20, 4, false),

            AttributeInfo("pg\_attribute", pg\_attribute\_id, "column\_bytes\_len", "int", 4, 5, false),

            AttributeInfo("pg\_attribute", pg\_attribute\_id, "column\_index", "int", 4, 6, false),

            AttributeInfo("pg\_attribute", pg\_attribute\_id, "is\_null", "bool", 1, 7, false),

*// pg\_student 表的属性*

            AttributeInfo("pg\_student", pg\_student\_id, "sno", "int", 4, 1, false),

            AttributeInfo("pg\_student", pg\_student\_id, "sname", "string", 20, 2, false),

            AttributeInfo("pg\_student", pg\_student\_id, "sage", "int", 4, 3, false)

    };

    table\_info\_vec = table\_info\_vec\_template;

    attr\_info\_vec = attr\_info\_vec\_template;

}

*struct* StudentInfo : *public* InfoObject {

*int* sno;

    string sname;

*int* sage;

    StudentInfo() {}

    StudentInfo(*int* *sno*, string *sname*, *int* *sage*) : sno(sno), sname(sname), sage(sage) {}

    vector <string> to\_string\_vec() {

        return {to\_string(sno), sname, to\_string(sage)};

    }

};

*int* main() {

    system("chcp 65001");

*int* insert\_times = 200;

*int* pg\_class\_id = 1001;

*int* pg\_attribute\_id = 1002;

*int* pg\_student\_id = 1003;

    BlockManager pg\_class(pg\_class\_id);

    BlockManager pg\_attribute(pg\_attribute\_id);

    BlockManager pg\_student(pg\_student\_id);

    vector <TableInfo> table\_info\_vec;

    vector <AttributeInfo> attr\_info\_vec;

    get\_info(table\_info\_vec, attr\_info\_vec);

*// 创建 pg\_class 与 pg\_attribute 表*

    for (TableInfo &tableInfo : table\_info\_vec) {

        vector <string> pg\_class\_data\_vec = tableInfo.to\_string\_vec();

        pg\_class.insert(pg\_class\_data\_vec, vector<string>(pg\_class\_data\_vec.size(), ""));

    }

    for (AttributeInfo &attributeInfo : attr\_info\_vec) {

        vector <string> pg\_attr\_data\_vec = attributeInfo.to\_string\_vec();

        pg\_attribute.insert(pg\_attr\_data\_vec, vector<string>(pg\_attr\_data\_vec.size(), ""));

    }

*// 创建学生表*

    for (*int* i = 0; i < insert\_times; i++) {

        StudentInfo studentInfo(i + 1, get\_random\_name(4), *int*(rand() % 10 + 18));

        vector <string> pg\_stu\_data\_vec = studentInfo.to\_string\_vec();

        pg\_student.insert(pg\_stu\_data\_vec, vector<string>(pg\_stu\_data\_vec.size(), ""));

    }

*// 将表保存到文件中*

    {

        pg\_class.to\_file();

        pg\_attribute.to\_file();

        pg\_student.to\_file();

    }

*// 从文件中读取表，并打印*

    {

        BlockManager pg\_class\_buffer;

        pg\_class.from\_file(pg\_class\_buffer);

        pg\_class\_buffer.print();

        BlockManager pg\_attribute\_buffer;

        pg\_attribute.from\_file(pg\_attribute\_buffer);

        pg\_attribute\_buffer.print();

        BlockManager pg\_student\_buffer;

        pg\_student.from\_file(pg\_student\_buffer);

        pg\_student\_buffer.print();

    }

    return 0;

}