# Использование макета СУБД

## Цель и содержание практики

# Цель проектно-технологической практики:

# - закрепление и углубление теоретической подготовки студентов покурсам «Информатика», «Основы программирования» и «Программирование на основе классов и шаблонов»,

# - использование библиотеки функций или классов для указанной в задании предметной области и для выполнения индивидуального задания.

## Содержание практики

* Разработать и записать в текстовые файлы таблицы с данными для указанной в задании предметной области. Каждая таблица хранится в отдельном файле. Данные хранятся в формате CSV с разделителем ‘|’. Для записи тестовых данных в файл использовать текстовый редактор MS Visual C++. Первая строка файла содержит заголовок таблицы, а остальные строки – данные.
* Изучить структуру DBTable, способную хранить данные произвольных таблиц.
* Использовать метод readTable() для чтения данных из файла в объект DBTable.

- Использовать метод printTable() для печати данных объекта класса DBTable

- Использовать метод writeTable() для записи данных объекта класса DBTable в файл в формате CSV;

- Разработать функцию main() для тестирования методов класса DBTable. Для вызова тестируемых функций из main() использовать меню.

- Разработать функции для выполнения индивидуального задания, использующие методы разработанных классов.

Для вызова функций из main() использовать меню.

## Общие указания

Перед началом программирования разработать таблицы с данными для указанной в задании предметной области. В качестве данных для тестового примера использовать таблицы БД, указанной в задании на практику.

***База данных библиотеки*** ***DBLibrary*** содержит 4х таблицы с данными: Abonements, Students, Books и Subjects.

Поля таблиц для БД Library:

*Таблица Students*: StudentID, Name, Group.

*Таблица Subjects*: SubjectID, Subject, где:

Subject – тематика книги (физика, математика, информатика)

*Таблица Books*: BookID, Title, SubjectID, Quantity, Number, где:

Quantity – количество поступивших в библиотеку экземпляров данной книги.

Number – количество выданных экземпляров данной книги.

*Таблица Abonements*: StudentID, BookID, OutDate, InDate, где:

OutDate, InDate – дата выдачи, дата возврата книги.

Заполнить файлы тестовыми данными. Ввести не менее трех книг по каждому из трех предметов, не менее 5 записей в таблицу Students и не менее 10 записей в таблицу Abonements.

При заполнении таблицы данными проверять уникальность значений идентификаторов и ссылочную целостность данных, то есть наличие в основных (родительских) таблицах Students, Books и Subjects значений идентификаторов StudentID, BookID и SubjectID, использованных в дочерних таблицах Abonements и Books.

***База данных DBCompany*** компании по оптовой торговле содержит 4 таблицы с данными: Orders, OrderDetails, Customers, Products.

Поля таблиц для БД ***DBCompany***:

*Customers*: CustomerID, CompanyName, ContactName

*OrderDetails*: ID, OrderID, ProductID, UnitPrice, Quantity

*Orders*: OrderID, CustomerID, OrderDate

*Products*: ProductID, UnitPrice, UnitInStock

***База данных DBBase*** продукции на складах содержит 3 таблицы с данными: Orders, Sklads, Products.

Поля таблиц для БД ***DBBase***:

*Sclads*: SkladID, SkladName

*Orders*: OrderID, SkladID, ProductID, Quantity, OrderDate

*Products*: SkladID, ProductID, Quantity

Базы данных хранятся на диске в текстовых файлах с разделителями (формат CSV).

Каждая таблица хранится в отдельном файле,

<имя файла>=<имя таблицы>.txt

Спецификация класса DBTable приведена ниже в Приложении 1.

## Варианты заданий

## Вариант 1

Оформление заказа на отпуск товаров со склада (добавить запись и сделать необходимые изменения в таблицах). Перед оформлением заказа сделать проверку наличия товара на складах (***База данных DBBase)***.

### Вариант 2

Выдача книги в библиотеке. На одних руках не должно быть двух книг по одному предмету. Перед выдачей сделать проверку наличия экземпляра книги в книгохранилище (***База данных DBLibrary)***.

### Вариант 3

Для книг, которых нет в библиотеке, вывести список студентов, взявших эти книги (***База данных DBLibrary)***.

### Вариант 4

Подсчитать доход компании от продажи товаров за указанный период времени (***База данных DBCompany)***.

Доход получается за счет разницы оптовой (указана в таблице Products) и отпускной (указана в таблице OrderDetails) цен товара (***База данных DBCompany)***.

### Вариант 5

Подсчитать расходы указанного покупателя за указанный период времени (***База данных DBCompany)***.

### Вариант 6

Вывести список книг по физике, находящихся на руках у заданного абонента (***База данных DBLibrary)***.

### Вариант 7

Определить покупателя, который потратил больше всех денег за указанный период времени (***База данных DBCompany)***.

### Вариант 8

Записать студента в библиотеку (обеспечить уникальность ID) и выдать ему книгу по указанному предмету (***База данных DBLibrary)***.

### Вариант 9

Определить студента, который дольше всех держит на руках экземпляр указанной книги (***База данных DBLibrary)***.

### Вариант 10

Определить предмет, по которому осталось меньше всего экземпляров книг в хранилище (***База данных DBLibrary)***.

## Вариант 11

Вывести список всех товаров, которые были отпущены со всех складов за указанный период времени (***База данных DBBase)***.

### Вариант 12

Определить склад, в котором осталось меньше всего товаров по количеству (***База данных DBLibrary)***.

*Приложение 1.* Файл dbmsLib.h

#pragma once

#ifndef \_dbmsLib\_

#define \_dbmsLib\_

#include <iostream>

#include <fstream>

#include <strstream>

#include <iomanip>

#include <string>

#include <vector>

#include <map>

using namespace std;

namespace dbmsLib

{

//====================класс DBDate======================

class DBDate {. . .};

//======================================================

//DBType-перечисление типов полей таблиц БД:

enum DBType{ NoType, Int32, Double, String, Date};

//typeNames - массив имен типов.

//Используется для преобразования имени типа (char\*) в объект DBType

static const char\* typeNames[]={

"NoType",

"Int32",

"Double",

"String",

"DBDate"

};

//Condition-перечисление условий, по которым могут сравниваються поля таблиц БД:

enum Condition{Undefined,Equal,NotEqual,Less,Greater,LessOrEqual,GreaterOrEqual};

const int LENGTH = 24;//длина имени таблицы и имени столбца.

//ColumnDesc - описание данных в столбце таблицы

//данные типа string имеют одинаковую длину (length) внутри одного столбца,

//но в разных столбцах их длина может отличаться

struct ColumnDesc {

char colName[LENGTH];//имя столбца

DBType colType;//тип данных в столбце таблицы

int length; //максимальное число символов, допустимое

//для представления данных в столбце

};

struct Strip{//полоса распечатки таблицы

int nField;//число полей

int\* fieldWidth;//ширина полей в полосе (массив)

};

typedef map<string, void\*> Row;

typedef map<string, ColumnDesc> Header;

//-----------------класс DBDate----------------------------

class DBTable{

public:

DBType GetType(char\* columnName);

const char\* TypeName(DBType type);

virtual ~DBTable(){};//виртуальный деструктор

virtual Header GetHeader()=0;

virtual vector<int> IndexOfRecord(void\* keyValue,string keyColumnName)=0;

//по значению ключа определяет

//порядковый номер (индекс) строки (записи) в data.

// keyValue-значение ключа (указатель на string, int или DBDate)

// keyColumnName-имя ключевого столбца (первичный или вторичный ключ)

//в таблице может быть несколько записей с одинаковым значением вторичного ключа.

virtual Row& GetRow(int index)=0;

virtual Row operator[](int index)=0;

virtual string valueToString(Row& row,string columnName)=0;

virtual int GetSize()=0;

virtual void SetHeader(Header& hdr)=0;

virtual void ReadDBTable(string fileName)=0;//fileName="..\\"+dbName+"\\"+tableName+".bin"|".txt"

virtual void WriteDBTable(string fileName)=0;

virtual void SetTableName(string tName)=0;

virtual void SetPrimaryKey(string key)=0;

virtual string GetTableName()=0;

virtual string GetPrimaryKey()=0;

virtual void SetFileName(string path)=0;

virtual string GetFileName()=0;

virtual void PrintTable(int screenWidth)=0;

virtual Row CreateRow()=0;

virtual void AddRow(Row row,int index)=0;

virtual DBTable\* SelfRows(string colName,Condition cond, void\* value)=0;

};

string GetTabNameFromPath(string path);

string ignoreBlanc(const string str);

void\* GetValue(string value, string columnName, Header hdr);

void\* SetValue(string value, string columnName,Header hdr);

bool comparator(/\*TableDataType\*/DBType type,void \*obj1,Condition condition,void \*obj);

int GetLength(ColumnDesc colDesc);

//===============class DBTableTxt=============================

class DBTableTxt:public DBTable{

public:

Header columnHeaders;

string tableName;

string primaryKey;

vector<Row> data;

string fileName;

public:

DBTableTxt(){}

DBTableTxt(string tabName);/\*: DBTable(tabName){}\*/

DBTableTxt(string tabName,Header hdr,string primKey);/\*:DBTable(tabName,hdr,primKey){}\*/

~DBTableTxt();//!!!деструктор перегружен,чтобы память,выделенная void\*,не утекала

vector<int> IndexOfRecord(void\* keyValue,string keyColumnName);

string valueToString(Row& row,string columnName);

void ReadDBTable(string tabName);//tabName=path+tableName

void ReadTableBin(string tabName);//чтение таблицы из бинарного файла

void PrintTable(int screenWidth);

void WriteDBTable(string tabName);//tabName=path+tableName

void WriteTableBin(string tabName);//запись таблицы в бинарный файл

int GetSize();

Row operator[](int ind);

void SetFileName(string path);

void SetTableName(string tName);

void SetPrimaryKey(string key);

string GetFileName();

string GetTableName();

string GetPrimaryKey(){return primaryKey;}

Header GetHeader();

void SetHeader(Header& hdr);

Row CreateRow();

Row& GetRow(int index);

void AddRow(Row row,int index);

DBTable\* SelfRows(string columnName,Condition cond, void\* value);

//Макет распечатки таблицы:

//screenWidth - ширина экрана (входной параметр)

//nStrips - число полос в распечатке (выходной параметр)

//strips[nStrips] - описание полос таблицы:

//число столбцов и ширина каждого столбца в полосе (выходной параметр)

void CreateTableMaket(Strip\* &strips,int &nStrips,int screenWidth);

};

//=====================Класс DBTableBin=====================================

class DBTableBin: public DBTable{. . .};

//==============Класс DBTableSet============================

struct Relation{. . .};

void PrintRow(Row row,DBTable\* tab);

//-------------------------класс DBTableSet---------------------

class DBTableSet

{. . .};

const int DELTA = 10;//запас в 10 строк для новых записей в таблицу.

//Если запас израсходован, то для его восстановления нужно увеличить

//на DELTA размер динамического массива char\* data[].

//const int LENGTH = 24;//длина имени таблицы и имени столбца.

string GetTabNameFromPath(string path);

string ignoreBlanc(const string str);

void\* GetValue(string value, string columnName,Header hdr);

void\* SetValue(string value, string columnName,Header hdr);

bool comparator(/\*TableDataType\*/DBType type,void \*obj1,Condition condition,void \*obj);

int GetLength(ColumnDesc colDesc);

}

#endif //конец \_dbmsLib\_

Шаблонный класс map<Type1,Type2> - это ассоциативный контейнер, построенный на основе бинарного дерева, в узлах которого хранятся пары значений типа pair<Type1,Type2>.

Type1 – тип ключа, Type2 – тип значения. Значение ключа должно быть уникальным в пределах контейнера. В качестве ключа могут использоваться только те типы, для которых определены операции сравнения.

Контейнер ***map*** можно рассматривать как массив, упорядоченный по значению ключа. Упорядочивание производится при записи данных в контейнер. Например, для добавления значения столбца value в строку заголовка columnName можно использовать оператор:

columnHeaders [columnName]=value;