|  |  |  |
| --- | --- | --- |
| *Софийски университет* |  | *Факултет по математика* |
| „*Св. Климент Охридски“* | *и информатика* |
|  |  |  |

Бакалавърска програма   
„Информационни системи”

Дисциплина „Обектно-ориентирано програмиране”

Проект

“Бази от данни”

**Изготвил: Преподавател:**

Елис Данга Калин Николов

ф.н. 71998

**Съдържание**

Увод 2

Преглед на предметната област 3

Проектиране 4

Реализация 6

Заключение 8

**Увод**

Базите данни представляват колекция от логически свързани данни. Те са структурирани по определен начин в конкретна предметна област. Базите данни се използват за съхранение на административна информация и за по-специализирани данни, като инженерни данни или икономически модели.

Задачата е да се реализира програма, поддържаща операции с прости бази от данни. За целта е създаден главен файл (каталог), които съдържа списък от таблиците в базата данни, като за всяка таблица е зададено име и файл, в който тя е записана.

От друга страна всяка таблица съдържа колони, които могат да бъдат както от еднакъв, така и от различен тип. Приложението поддържа следните типове:

* **Цяло число** – поредица от цифри, без никакви други символи между тях. В началото на числото може да има знак '+' или '-'. (123, -123, +123)
* **Дробно число** – поредица от цифри, следвана от символ за точка и след нея друга поредица от цифри. В началото на числото може да има знак '+' или '-'. (123.456, -123.456, +123.456)
* **Символен низ (стринг)** – поредица от произволни символи оградени в кавички. Подобно на низовете в C++, ако е включен символа за кавичка в даден низ, трябва да се представи като \", а ако е включена наклонена черта, трябва да се представи като \\. ("Hello world!", "C:\\temp\\", "\"This is a quotation\"")
* Освен конкретна стойност, дадена клетка в даден ред на таблицата може да е “**празна**”. Такива клетки да се обозначават специално и се изписват като “NULL”.

В следващите няколко страници ще се опише подробно реализацията на проекта. Ще се прегледат основните концепции, дефиниции и методи. Ще се представят както архитектурата на програмата и диаграмите на поведение, така и извадка от кода и тестови сценарии. В обобщение ще се разгледа изпълнението на началните цели, бъдещото развитие и усъвършенстване.

## Преглед на предметната област

В проекта са дефинирани 2 основни класа – Catalog и Table, които са в основата на задачата, един FileSystem, отговарящ за основните функции в системата и Project6 в който се изпълняват основните операции. Всяка таблица е запазена в отделен файл, като приемаме структурата да е: на първия ред е изписано името на таблицата, на втория типа на колоните, а на останалите е самата таблица. По този начин обхождаме файловете и изпълняваме отделните операции.

[1] fs.read(file, buffer);

int first\_line\_length = (buffer.find("\n") != -1) ? buffer.find("\n") : buffer.size();

std::string table\_name = buffer.substr(0, first\_line\_length);

Тук е показано как четем даден файл, операция от класа FileSystem. С помощта на тернарен оператор, намираме дължината на първия ред от файла. Както се разбрахме по-нагоре, това е името на таблицата и откриваме с помощта на функцията substr().

Сложността на дадена задача се определя от вида на стъпките до реализирането й. В нашия случай те са три:

* Уточняват се начинът на съхранение на данни
* как ще се достъпват отделните елементи
* как ще се осъществят операциите

В процеса на разработка на системата, различните команди ги разглеждаме като отделни подзадачи или по-точно проблеми. Тук използваме функционалните изисквания. Всяка команда изпълнява някаква операция. Например чрез rename променяме името на таблицата, чрез describe се показва информация за типовете на колоните на дадена таблица, чрез export се записва таблица във файл и т.н. Освен основното действие, при някой операции, има и допълнителни условия. Например при rename, едно от допълнителните условия е ако името не е уникално, да се отпечатва грешка.

[2] if (command.compare("rename") == 0) {

if (args[0].size() && args[1].size()) {

if (c1.has\_table(args[1])) {

std::cout << "Error! The name isn't unique\n";

}

else {

c1.rename\_table(args[0], args[1]);

std::cout << "The name was changed\n";

}

}

else {

std::cout << "Please use format: \"rename <old name> <new name>\" \n";

}

}

## Проектиране

Както казахме по-нагоре, програмата е съставена от три основни класа. Като класа Table работи конкретно със съдържанието на таблицата. Има за член-данни най-важната информация за дадена таблица:

* Името на таблицата
* Името на файла, в който се пази таблицата

а като член-функции:

* конструктор с параметри(информацията за продукта)
* getters
* setters
* rename\_to\_file() – променя името на таблицата във файла
* print\_columns\_info() – извежда типа на колоните
* print\_rows() – извежда таблицата по редове

Класа Catalog работи с базата данни, в която се записва информацията за таблиците. Има за член-данни:

* името на файла в който е записана дадена таблица
* списък с таблиците, които са записани вътре

а, за член-функции:

* load\_from\_file() – въвежда таблица в каталога от файл
* fromString() – въвежда данни в каталога от стринг
* print() – извежда данните от каталога
* has\_table() – проверява дали имам таблица с такова име
* save\_to\_file() – запазва във файла
* get\_table() – „взима“ таблица
* rename\_table() – преименува таблица

Класа FileSystem има една член-данна

* Stream

и две член-функции

* read() – чете от файл
* write() – записва във файл

В главната функция Project6 са записани най-важните операции в програмата:

* import <file name> - Добавя в базата данни нова таблица от файл
* showtables - Показва списък с имената на всички заредени таблици
* describe <name> - Показва информация за типовете на колоните на дадена таблица
* print <name> - Показва всички редове от дадена таблица
* export <name> <file name> - Записва таблица във файл
* select <column-n> <value> <table name> - Извежда всички редове от таблицата, които съдържат стойността “value” в клетката с дадения пореден номер
* addcolumn <table name> <column name> <column type> - Добавя нова колона (с най-голям номер) в дадена таблица
* update <table name> <search column n> <search value> <target column n> <target value> - За всички редове в таблицата, чиято колона с пореден номер <search column n> съдържа стойността <search column value> се променят така, че колоната им с пореден номер <target column n> да получи стойност <target value>
* delete <table name> <search column n> <search value> - Изтрива всички редове в таблицата, чиято колона <search column n> съдържа стойността <search column value>
* insert <table name> <column 1> … <column n> - Вмъква нов ред в таблицата със съответните стойности
* innerjoin <table 1> <column n1> <table 2> <column n2> - Извършва операцията [Inner Join](https://en.wikipedia.org/wiki/Join_(SQL)#Inner_join) над две таблици спрямо колоните <column n1> в първата таблица и <column n2> във втората. Създава нова таблица и извежда идентификатора ѝ
* rename <old name> <new name> - Преименува таблица
* count <table name> <search column n> <search value> - Намира броя на редовете в таблицата, чиито колони съдържат дадената стойност
* aggregate <table name> <search column n> <search value> <target column n> <operation> - Извършва дадена операция върху стойностите от колоната <target column n> на всички редове, чиито колони с номер <search column n> съдържат стойността  <search value>

Нашата програма е създадена по команден ред, тоест в зависимост от командата която дадем се извършват различни операции. За да намерим каква команда е въвел потребителя и какви са нейните параметри, използваме следния подход :

[3] int delimiter\_pos = input.find(' ');

command = input.substr(0, delimiter\_pos);

input.erase(0, delimiter\_pos + 1);

if (delimiter\_pos != -1) {

while (!input.empty()) {

delimiter\_pos = (input.find(" ") != -1) ? input.find(" ") : input.size();

args[i] = input.substr(0, delimiter\_pos);

input.erase(0, delimiter\_pos + 1);

i++;

}

}

В променливата command запазваме името на командата, а аргументите в args (ако има такива). Project6.cpp се грижи за разпознаването на командите и за изпълнението на отделните операции. Например командата print:

[4] if (command.compare("print") == 0) {

if (args[0].size()) {

Table t = c1.get\_table(args[0]);

t.print\_rows(t.get\_file());

}

else {

std::cout << "Please use format: \"print <name>\" \n";

}

}

Тук имаме разпознаване на команда, проверка дали има отворен файл и проверка на синтаксиса на операцията, както и подходящи съобщения за грешка. За извършването на операцията са използвани функции от другите класове. В главната програма можем да видим връзката и значимостта между класовете.

## Реализация, тестване

В класа FileSystem имаме две член-функции :

- write() – записва input във файла, като първо изтрива всичко и след това го презаписва обновено.

[5] this->stream.open(file\_name, std::ios::in | std::ios::out | std::ios::trunc);

- read() – чете output ред по ред и връща съобщение дали е прочетено

В класа Table по-значими функции са:

* rename\_to\_file() – чете файла в буфера, след това намира дължината на първия ред(името на файла). Изтрива старото име и въвежда ново на негово място, след това се записва отново

[6] fs.read(this->file, tmp\_buffer);

int first\_line\_length = (tmp\_buffer.find("\n") != -1) ? tmp\_buffer.find("\n") : tmp\_buffer.size();

tmp\_buffer.erase(0, first\_line\_length);

tmp\_buffer.insert(0, new\_name);

fs.write(this->file, tmp\_buffer);

* print\_columns\_info() – логиката е същата като на [6].

fs.read(this->file, tmp\_buffer);

int first\_line\_length = (tmp\_buffer.find("\n")

!= -1) ? tmp\_buffer.find("\n") : tmp\_buffer.size();

std::string second\_line = tmp\_buffer.substr(first\_line\_length, tmp\_buffer.find("\n", first\_line\_length));

std::cout << second\_line;

В класа Catalog по-значими функции са:

* has\_table() – проверява дали даден функция съществува в каталога, като проверява дали се съдържа името й

for (std::list<Table>::iterator it = this- >list\_of\_table.begin(); it != this->list\_of\_table.end(); it++) {

if (it->get\_name().compare(table\_name) == 0) {

return true;

}

}

Тестване на програмата:

*Using: Catalog.txt*

*>showtables*

*table\_name\_1 t1.txt*

*table\_name\_2 t2.txt*

*>import*

*Please use format: "import file.txt"*

*>import t3.txt*

*The table has been imported!*

*>showtables*

*table\_name\_1 t1.txt*

*table\_name\_2 t2.txt*

*table\_name\_3 t3.txt*

*>describe table\_name\_3*

*double int*

*>print table\_name\_1*

*st1 1*

*st2 2*

*st3 3*

*st4 4*

*st5 5*

*Paging commands (previous, next or exit) : next*

*st6 6*

*st7 7*

*st8 8*

*st9 9*

*st10 10*

*Paging commands (previous, next or exit) : exit*

*>>rename table\_name\_2 second\_table*

*The name was changed*

*>showtables*

*table\_name\_1 t1.txt*

*second\_table t2.txt*

*table\_name\_3 t3.txt*

*>exit*

*Exiting the program*...

## Заключение

Програмата е реализирана и работи успешно. Началните цели са постигнати, някой от зададените операции и проверките към тях са изпълнени. Дописването на операциите и усъвършенстването им са част от бъдещото развитие на програмата.

*Линк към github -* [*https://github.com/elisdanga/DataBase*](https://github.com/elisdanga/DataBase)