СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ“
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА

КУРСОВ ПРОЕКТ
ПО Структури от данни и програмиране

Тема:

Система за управление на бази от данни

Студент:

Йордан Иванов Цолов група 2 ФН:45237

София, януари 2023 г.

- Формулировка на задачата

Реализиране на просто СУБД което поддържа работа с множество таблици, всяка от които се състои от произволен борй колони, като всяка колона може да е или тип число, дата, или символен низ. Цялата информация

на СУБД се съхранява на диска във формат .csv за всяка отделна таблица на която първият ред е информацията за таблицата следвана от цялата информация запазена в таблицата.

- # Описание на програмната реализация

Характеристики: В най-лошият случай за добавяне, премахване, търсене ще бъде O(n), тъй като ползваме тип хеш таблица, но вместо да имаме O(1) достъпване, правим линейно обхождане до момента в който срещнем ключа на данните, и след това обхождаме само децата на този елемент (реално представлява тип хеш таблица, и когато имат еднакви ключове два елемента, се свързват в списък). Реално със сегашната имплементация в базови случаи които ключовете са винаги различни операциите ще са доста бавни O(n) докато ако се беше направило с дърво щеше да е O(logn), тази имплементация обаче би могла да бъде по-бърза и по-добра в случаите в които имаме огромно количество данни които могат да бъдат разпределени по ключове (тоест има повтарящи се ключове). Така в случаи в които имаме много еднакви стойности на ключовете, примерно 100 000 реда, разпределени с 100 различни ключа, трябва да минем M = 100 в най-лошият случай, (нека предположим, че са равномерно разпредени за примера) K = 1000, тоест трябва да направим 1100 итерации, вместо 100 000. Което се приближава супер близко до това, ако бяхме ползвали дърво, но ключовете са еднакви. Тоест в такъв случаи O(M + K);

Реализацията на клас : **DataBase**

Атрибути: вектор от структури Base. Структура Base която съдържа пойнтер към TreeBase (ще обясним по-късно какво представлява този Клас), име на таблица, номер редове, размер на файла.

private Методи: :

std:: string getNameFromStringWhenCreating(string data) : минавайки през стринга който ни е зададен извличаме името което ще отговаря на таблицата която искаме да създадем.

void InsertInTable(string name): по подадено име и информация, първо проверяваме дали съществува файл с такова име, ако да, се оптиваме да заредим информацията от този файл, ако не: викаме InsertFunction() на тази таблица от вектора, чиито име отговаря на подаденото, и опитваме да добавим информацията там. Ако няма нито такова име в таблиците ни, нито има такъв файл, изкарва информация за това, че такава таблица не съществува!

void SelectFromTable(string name): по подадено име и информация, първо проверяваме дали съществува файл с такова име, ако да, се оптиваме да заредим информацията от този файл, ако не: викаме SelectFunction() на тази таблица от вектора, чиито име отговаря на подаденото, и опитваме да добавим информацията там. Ако няма нито

такова име в таблиците ни, нито има такъв файл, изкарва информация за това, че такава таблица не съществува!

void RemoveFromTable(string name): по подадено име и информация, първо проверяваме дали съществува файл с такова име, ако да, се оптиваме да заредим информацията от този файл, ако не: викаме DeleteFunction() на тази таблица от вектора, чиито име отговаря на подаденото, и опитваме да добавим информацията там. Ако няма нито такова име в таблиците ни, нито има такъв файл, изкарва информация за това, че такава таблица не съществува!

void RemoveFromTableAll(string name): По подадено име намира дали името съществува в базата от данни, ако да, запазва на това място в базата единствено името , размера и изтрива изцяло информацията която е съхранявана от класа TreeBase, извиквайки функцията DeleteFunctionAll(), тази функция също така записва всички текущи промени на базата на диска; ако не намери връща съобщение, че такава таблица не съществува.

void CreateTable(string info); по подадена информация вътре, която може да е или само име или име и (имена на колони и типове и къде е индексирано) създава нова таблица; Ако вече имаме таблица с това име, връща съобщение за грешка, че не може да се създаде таблица с вече същетствуващо име, ако се опитваме да дадем грешни аргументи за типове на колони, също връща грешка за невалидни аргументи.

void DropTable(string name); по подадено име, търси името на колоната, ако го намери я изтрива изцяло от базата ни данни, но този път изчиства и името, размер, всичко се занулява и се премахва. Ако не се намери име на такава таблица връща съобщение, че не е намерена търсената таблица.

void ListTables(); това е функция без аргументи, връща имената на всички таблици в базата.

void TableInfo(string name); по подадено име, търси в базата дали съществува такава таблица, ако да: то тогава показва информацията за тази таблица и колко редове има и какъв е размера на файла който е върху диска с това име(ако такъв съществува, ако няма просто връща за размер 0), ако не е намерено такова име, показва съобщение за това.

void RemoveFromTableAllSpecialCase(string name): по подадено име, ако е намерена такава таблица в базата изпълнява функцията deleteFunctionAllSpecialCase(); върху TreeBase обекта ни, единствената разлика с RemoveFromTableALl(), е това, че тук първо чистим, а после запаметяваме във файла, докато другото първо запаметяваме и после чистим информацията, причината е ,че този случай е единствено когато искаме да изтрием цялата информация от дадения обект и все още работим с него, а не се трие поради това, че се използва друг обект. Този случай се ползва когато продължаваме да работим с обекта и не сме приключили с него, другият се ползва когато просто искаме да почистим и запаметим информацията на диска, но вече не работим с нея, но бихме работили в бъдеще с нея. С две думи тук първо чистим после запазваме, а при другата първо запазваме после чистим.

public Методи:
void DetermineFunction(string data, ostream& out = cout); по информация и стрийм(дефолт е std::cout) , минавайки през информацията определя коя функционалност е нужна да се извика, дали ще е Select, Insert ... и т.н. След като определи коя е функционалността се опитва да изпълни нужната функция от горните, като при всяко извикване на функция, проверяваме дали все още работим със същата таблица, ако да, нищо не се променя, но ако не работим със същата, изтрива съдържанието на таблицата в базата изключайки името и размера, за да можем да я възвърнем от диска (единствено при коректно поискана команда). След което се опитва да изпълни нужната функционалност, връща съобщение при всеки опит за невалидно въведена информация.

DataBase(); -дефоултен конструктур, не прави нищо
DataBase(const DataBase& rhs); - копи конструктур, прави копие от rhs
DataBase& operator =(const DataBase& rhs); - оператор присвояване = първо на сегашния обект изчиства съдържанието, след което му се присвоява съдържанието на rhs
~DataBase(); - деструктур - изчиства цялата информация за базата

Реализацията на клас : **HashListBase**

Атрибути: string tableName, string TableInfo, int rowCount, int indexedAt; vector<Row> children; (това е вектор от редове), vector<string> types; (типове на колоните) .. структурата Row има атрибути (string key ; (ключ, индексирано по),, string data (целият ред) , Row* next; (ако се случи, че много редове да имат еднакъв ключ, те се нареждат в списък. Така получаваме вектор от елементи с различни ключове, а ако има еднакви ключове този даден елемент от вектора всъщност е свързан списък.)

private Методи:
void copy(const TreeBase& other); - правим сегашния обект да е копие на other
void clean(); - изтриваме напълно информацията за този обект

std::string getColumn(string data, int index) - по подаден ред и индекс връща, колоната която е на дадения ред, ползваме го при взимане на информация от ред който е пълен със стойности: Пример ако таблицата има Id:Int, Name:String, Date:Date, то тази функция бихме я ползвали за някой ред който е от тип (1, гошо, 10.10.2020);

std::string getColumnFromTable(int index) - по подаден индекс връща колона от таблицата, тоест връща ако таблицата е от тип ID:Int, Name:String, Date:Date, то връща името на колоната, тоест връща ID При 0 , Name при 1 , Date при 2;

int indexOfColumn(string data, string colName) - по подадено име и дата, връща на кой индекс се намира даденото име. Тоест ако дадем Name при горния пример би върнало 1, прескача Indexed и него не го брои за нова колона.

int asciiVal(string data) - минавайки буква по буква и събирайки резутлата, връща ascii стойността на стринг

void getAllStringInfo(string data, string& tblName, string& tblInfo, int &indxAt, vector<string> &tps) - минавайки през дадената ни информация от data, вземаме информация за таблицата, името на таблицата, къде е индексирана, и пълним типовете които са на колоните минавайки буква по буква, ако някоя от данните не е валидна throw std::invalid_argument(""); ако всичко е наред, но искаме да създадем таблица без информация, тогава я създава с дефолтни стойности.
Също така преобразува информация от тип : (ID:Int, Name:String) Index ON ID така ,че tableInfo = (ID:Int, Indexed; Name:String) ;

int compareData(string lhs, string rhs, string type) : тази функция сравнява два елемента в зависимост от това дали те са тип Int, Date, String, сравнявайки ги спрямо типа им. Ако lhs < rhs връща -1 ако lhs = rhs връща 0 иначе връща 1;

void addElement(string data): - по подадена информация , ред проверяваме дали информацията е валидна, ако не е throw std::invalid_argument(""); ако е валидна изцяло вземаме ключовата стойност на този ред, сравняваме го със всички ключови стойности които имаме до момента във вектора, ако не е намерен просто добавяме нов елемент във вектора, а ако е намерена, тогава добавяме този елемент като последен на списъка с такъв ключ.

void addElements(vector<string> rows); - извиква addElement(string data) за всеки елемент във вектора, служи за добавяне на повече от един елемент.

void removeElement(string data); това е функция която по подаден ред, взема му ключа за по-бързо търсене, и изтрива дадения ред от базата, ако е намерен.

bool checkIfValidArg(string data, string valueType, int type): буквално проверява дали всеки един тип е правилен като ако type = 0 проверява за имена на колони(тоест дали е част от имената на колоните които имаме в таблицата), 1 е за операции от тип >= < = и т.н, 2 е за стойности, ако е дата, ако е инт какви са характеристиките на тези типове, 3 ако е специална операция дали е AND, OR и т.н. Ако някоя от данните е невалидна връща съобщение за това. Ако е валидна информацията връща true;

bool ifConditionIsMet(vector<bool> & values, vector<string> operations, int & indxForOps, int& indxOfValues); - това е операция която по булев вектор който представлява 1 или 0, което представлява дали дадените операции са изпълнени при WHERE клаузите, пример за това би било ID >= 3 AND Name != Gosho ако стигнем до елемент на който ID >= 3 добавяме 1 , ако е < 3 добавяме 0 и т.н.
След което имаме вектор от операциите които са AND, OR , (,) (ако има NOT предварително сме обърнали знаците на нужните позиции. Пример: NOT (ID >= 3),
вместо да бъде добавено ако стойността на сегашния елемент е ID =5 , вместо да се добави 1 , се добавя 0. Вътре изцяло е функционалността в зависимост от това какъв символ е дадената операция. Реално логиката е да изпълнява операциите които са AND / OR или (,) върху булевият вектор, сравнявайки два по два елементите в зависимост от операцията, може да се наложи да се сравнят 1 и последен, и накрая връща последният елемент като резултат. Ако операцията е проста ... AND ... проверява само лев и десен елемент и присвоява нова стойност на десният елемент в булевият вектор, ако е по сложна от тип 1 AND (2 OR 3) , първо изчислява стойността

вътре в OR -> променя стойността на 3 , и след това проверява 1 AND 3 И променя стойността на 3.

Public Методи:

void orderByPrint(string columnName, vector<int> columnIndexes, bool allElem, vector<string> rowsToShow, bool typeofSort) : първо сортира в зависимост от typeOfSort може да сортира в нарастващ или в намаляващ ред, в зависимост от флага allElem , показва всички елементи или само нужните колони на които индексите са записани в columnIndexes, редовете които са за показване са във rowsToShow, в зависимост от WHERE клаузата преди викането на тази функция биха били или всички, или част от всички редове в базата.

string getTableName() - връща името на таблицата

string getTableInfo() - връща информацията за таблицата във формат който вече е преработен, пример : (ID:Int, Indexed; Name:String, Date:Date)

int getRowCount() - връща брой редове в базата;

void deleteFunction(string deleteRow): - по подадено deleteRow във формат ("Delete FROM TableName WHERE ID = 3 AND Name != Gosho" , извлича името на колоните (ID, Name) които са нужни, операциите (= , !=), стойностите (3, Gosho) , специалните операции (AND).. Ако информацията която е въведена е невалидна се показва съобщение в зависимост къде е била намерена грешката. Използваме int separator = 0; който нараства единствено ако е успешно било добавена дадената стойност независимо дали е в отдел колони, операции, стойности. И така реално редуваме в кой вектор да се вкара дадената ни дума, ако separator = 0 , в колони, ако е 1 в операции, ако е 2 в стойности, ако е 3 в специални операции, защото не е възможно да не са в този ред, ако не са в този ред, то тогава би намерило грешка и ще покаже, че е невалиден формат. След което в зависимост от WHERE clause-а дали съдържа ключа или не, следва пълненето на булевия вектор с 1 или 0, който използваме в ifConditionIsMet(), като сравняваме с всеки елемент използвайки compareData(), дали стойността отговаря на изискванията, ако да се добавя 1 ако, не 0. Всеки елемент който отговаря се добавя във вектор rowsToBeDeleted. За всеки елемент от този вектор се вика removeElement(); // изтрива данните за този елемент. След което запазваме промените във файла единствено ако, колоните са станали 0.

void deleteFunctionAll();- тази функция първо запазва информацията във файл , след което изчиства цялата информация в базата.

void deleteFunctionAllSpecialCase();- тази функция първо изчиства цялата информация в базата, след което запаметява във файла.

void print(vector<int> columnsToBeSHown, vector<string>rowsToBeSHown) - принтира всички колони които са нужни да бъдат показани от всички редове които са зададени на тази функция. Ако колоните са празни, това означава, че трябва всички елементи да се принтират.

void InsertFunction(string info) :след подадена информация Info , от нея взема цялата информация във ( ) и за всеки елемент пълним вектор rows, ако такава липсва, хвърля грешка, ако информацията е невалидна то addElements(rows[i]) хвърля грешка, ако информацията е валидна то тогава addElements() успешно добавя елементите в базата.

void loadFromFile() -- функцията зарежда информация от файл, ако информацията във файла е в некачествен формат, хвърля грешка, вземайки първият ред - това е tableInfo, текущото tableName+".csv" е името на файла който се опитваме да отворим, ако е успешно отварянето , опитваме да заредим информацията в базата, ако е невалиден файла хвърля грешка. Ако е валиден, пълним базата. Ако някъде по средата стане грешка, то трие автоматично всички предходно добавени елементи.

void selectFunction(string row)- Абсолютно аналогично на DeleteFunction() , абсолютно същата логика за WHERE клаузата, с разликата, че в началото имаме повече флагове (DISTINCT) , дали всички колони трябва да се хванат това става с * , или са изброени няколко колони.. Също така имаме флаг OrderBY който се ползва ако случайно има някакъв ред по който искаме да сортираме данните, и се извиква без значение дали има WHERE клауза, стига този флаг да го има, отново се проверява за коректност на данните, единствената разлика е ,че ако има флаг DISTINCT, когато добавяме в rowsToShow (което ще се ползва за колоните които искаме да принтираме) се добавят само такива които вече не са добавени. Следва логиката за пълненето на булевата функция която определя дали даден елемент отговаря на изискванията и дали ще бъде показан. В зависимост от различните флагове, се изпълняват различни функции за принтиране, начин на принтиране.

HashListBase() - дефолтен конструктур, създава таблица с име Sample + различно число, tableInfo= "(ID:Int, Indexed: Name :string; Date:Date) - > създава дефолтна таблица

HashListBase(std::string fullData) : конструктур по дата, създава таблица по дадена дата, реално нулира редовете и извиква getAllStringInfo(fulldata, tableName, tableInfo,indexedAt,types) за да запълни другите части.// ако е неуспешно създаването хвърля грешка от getAllStringInfo()..

HashListBase(const TreeBase& other) - копи конструктур извиква функция copy(other);
HashListBase& operator= (const TreeBase& rhs) = assignment operator - чисти, после копира;
~HashListBase() -деструктур , просто вика функцията clean() за да почисти цялата информация.

Други методи:

string returnOppositeOperation(string element) - в зависимост от кой тип операция е , дали е = връща !=, връща обратната операция, > връща <= и т.н.

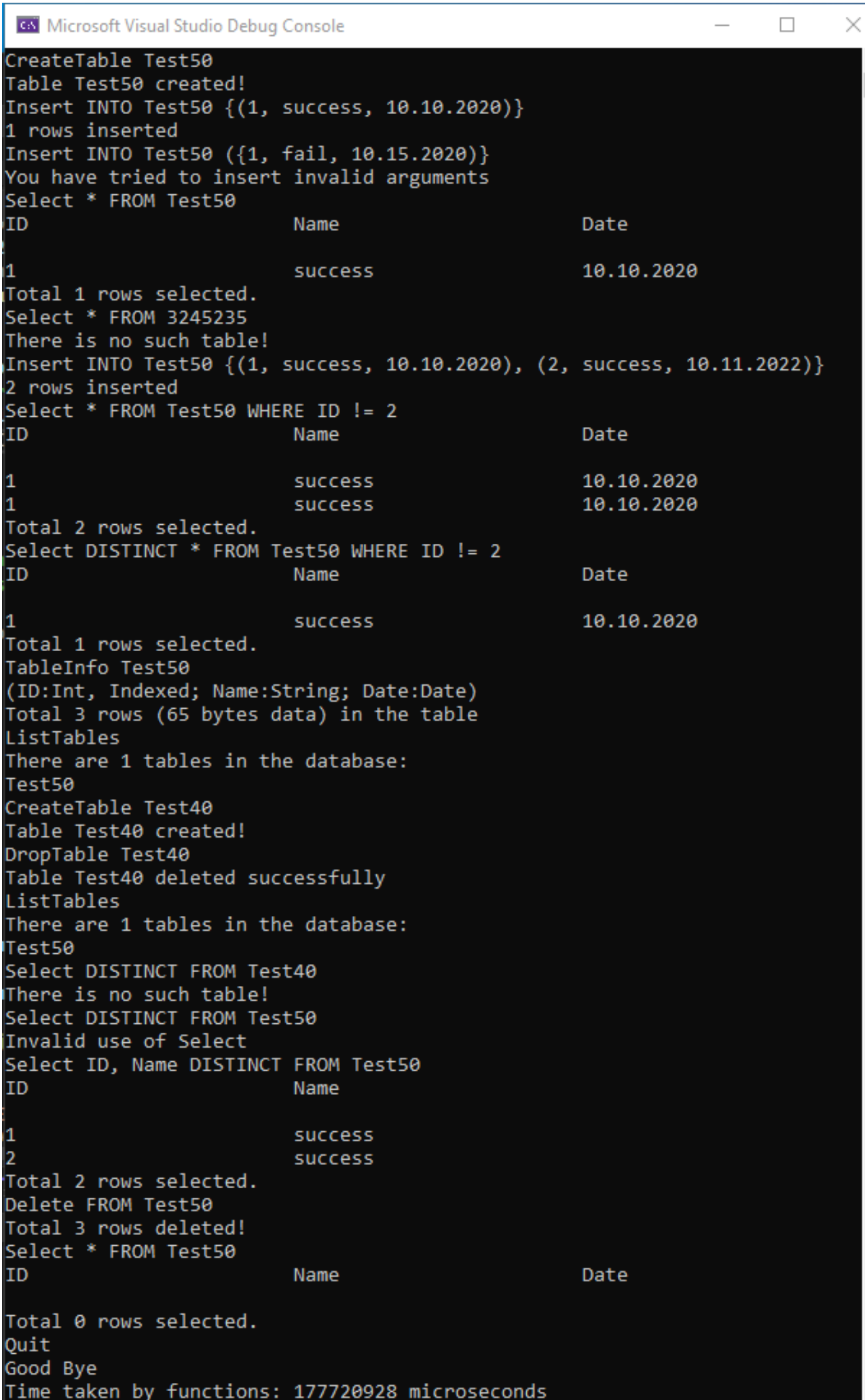
string returnOppoSiteSpecialOperation(string element) - в зависимост от операцията дали е AND връща OR, връща обратната на специалната операция, може да бъдат единствено AND , OR .

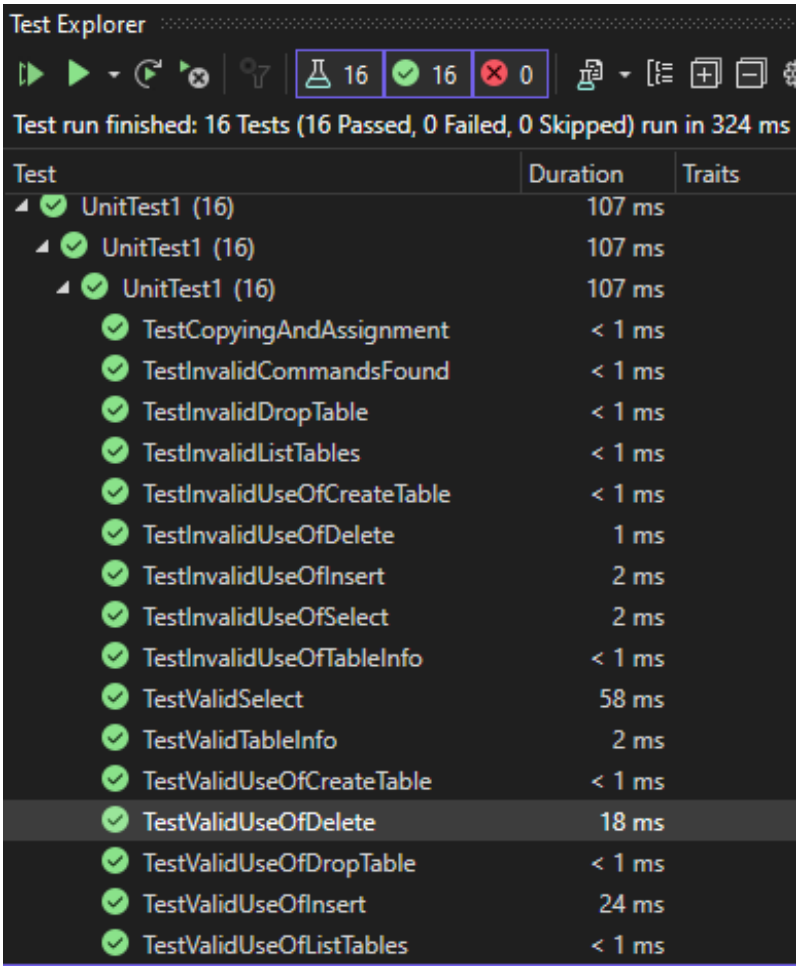
bool ifInt(string data) - връща true , ако дадения стринг е число, като проверим ascii кода на всяка буква. Ако не е число, връща false;

string getNameFromStringWhenCreating(string data) - връща втората пълна дума от data, ползваме го при създаване на таблица , CreateTable Tst1. Връща Tst1.

void readFromConsole(): това е функция която създава обект от тип DataBase tester и в зависимост от Input-а в конзолата ако е различен от Quit, извиква tester.DetermineFunction(line). Тоест в базата ни от данни да се прецени какво иска потребителя, изкарва съобщения за грешка, ако е невалидна датата, ако всичко е валидно, освен функционалността която се очаква показва и съобщение за това. Ако потребителя напише Quit, програмата приключва.

- Пример илюстриращ работата на програмната система


Microsoft Visual Studio Debug Console
CreateTable Test50
Table Test50 created!
Insert INTO Test50 {(1, success, 10.10.2020)}
1 rows inserted
Insert INTO Test50 ({1, fail, 10.15.2020)}
You have tried to insert invalid arguments
Select * FROM Test50
ID Name Date
1 success 10.10.2020
Total 1 rows selected.
Select * FROM 3245235
There is no such table!
Insert INTO Test50 {(1, success, 10.10.2020), (2, success, 10.11.2022)}
2 rows inserted
Select * FROM Test50 WHERE ID != 2
ID Name Date
1 success 10.10.2020
1 success 10.10.2020
Total 2 rows selected.
Select DISTINCT * FROM Test50 WHERE ID != 2
ID Name Date
1 success 10.10.2020
Total 1 rows selected.
TableInfo Test50
(ID:Int, Indexed; Name:String; Date:Date)
Total 3 rows (65 bytes data) in the table
ListTables
There are 1 tables in the database:
Test50
CreateTable Test40
Table Test40 created!
DropTable Test40
Table Test40 deleted successfully
ListTables
There are 1 tables in the database:
Test50
Select DISTINCT FROM Test40
There is no such table!
Select DISTINCT FROM Test50
Invalid use of Select
Select ID, Name DISTINCT FROM Test50
ID Name
1 success
2 success
Total 2 rows selected.
Delete FROM Test50
Total 3 rows deleted!
Select * FROM Test50
ID Name Date
Total 0 rows selected.
Quit
Good Bye
Time taken by functions: 177720928 microseconds



// UnitTest1 проектът са включените юнит тестове които тестват функционалността на всяка функция.