POO - Specificații proiect 2020
www.acs.ase.ro/cpp

Baza mea de date SQLite

De ce să fac proiectul?

Acest lucru este echivalent cu a te întreba de ce să înveți?

Cum mă va ajuta?

Această misiune nu este „una ușoară” și va necesita implicarea ta pentru următoarele săptămâni. Necesită angajament și implicare săptămânală. Nu poți face proiectul în ultimele 2-3 zile înainte de termen.

Dacă reușești să-l faci, chiar dacă nu în totalitate, vei avea o experiență practică care te va ajuta să vezi cum se combină diferite clase pentru a dezvolta un produs software „real” (SQLite este real și folosit foarte mult)

Soluția finală va fi „oribilă” din perspectiva arhitecturii/a design-ului, dar nu iti fa griji. Vei câștiga ceva ce nu poți obține altfel, se numește experiență.

Care este scopul final?

Implementarea unei versiuni limitate a unui sistem de gestionare a bazelor de date, cum ar fi SQLite https://www.sqlite.org/index.html

Faza 1 - Cerințele procesorului de comandă

În următoarea descriere, | înseamnă logică SAU și [] înseamnă că parametrul nu este obligatoriu.

Implementați diferite clase C ++ care vă vor permite să interpretați diferite comenzi SQL primite de la consolă sau dintr-un fișier:

1. Soluția va fi utilizată pentru a gestiona o singură bază de date - doar una, cu mai multe tabele
2. comenzile acceptate pentru gestionarea structurii bazei de date sunt CREATE TABLE, DROP TABLE, DISPLAY TABLE

a. CREATE TABLE nume_tabel [IF NOT EXISTS] ((nume_coloană_1, tip, dimensiune, valoare_implicită), (nume_coloană_2, tip, dimensiune, valoare_implicită), ...) - comanda ar trebui să primească cel puțin 1 coloană

b. DROP TABLE table_name
c. DISPLAY TABLE table_name
d. CREATE INDEX [IF NOT EXISTS] index_name ON table_name (column_name) - opțional pentru faza 1; definește un index pentru coloana column_name
e. DROP INDEX index_name - opțional pentru faza 1
f. tipurile acceptate sunt text, integer, float

3. Comenzile CRUD acceptate pentru gestionarea datelor sunt INSERT, SELECT, UPDATE , DELETE
4. Comanda acceptată pentru comenzile CRUD este (utilizați https://www.sqlite.org/lang.html pentru a vă face o idee)

a. INSERT INTO nume_tabela VALUES(...); valorile sunt separate prin, și au numărul și ordinea exacta ca definiția tabelului
b. DELETE FROM nume_tabela WHERE nume_coloană = valoare (DELETE permite doar o coloană în clauza WHERE)
c. SELECT (cel_putin_o_coloana, ...) | ALL FROM nume_tabela [WHERE nume_coloană = valoare] - clauza WHERE este opțională
d. UPDATE nume_tabela SET nume_coloană = valoare WHERE nume_coloană = valoare (coloana SET poate fi diferită de cea WHERE; UPDATE merge doar pentru o coloana)

În acest moment ar trebui să dezvoltați funcții care vor primi un șir de caractere (char * sau un string C ++) și identificați comenzile. Dacă comanda nu este bine formatată, trebuie să returnați un cod/mesaj de eroare.

Acestea sunt cerințele funcționale. Cerințele specifice POO sunt publicate pe www.acs.ase.ro/cpp în faza 1 a proiectului.

Recomandări

- Începeți cu o funcție/clasa generică care identifică tipul de comandă (SELECT, UPDATE etc.)
- Implementați clase specifice care vor valida un format și parametri specifici tipului de comandă
- Utilizați vectori pentru a stoca datele comenzilor
- Definiți clasele pentru entitățile care sunt descrise de un substantiv

Rezultate:

- Unul sau mai multe fișiere .h care conține toate definițiile și implementarea claselor
- Unul sau mai multe fișiere sursă C ++ (utilizați extensia .cpp) care conține main ()
- O singură aplicație consolă C++ care va permite utilizatorilor să scrie comenzi din lista anterioară. Ieșirea pentru fiecare comandă este

- Mesaj de eroare dacă comanda nu este ok
- Detalii despre parametrii comenzii dacă comanda este ok

Soluții acceptate: sunt acceptate numai soluțiile complete și fără erori de compilare.

Faza 2 - Stocarea datelor în fișiere

Legătura cu faza anterioară:

- Toate comenzile implementate în modului anterior vor genera date ce vor fi salvate în fișiere și/sau vor fi afișate la consolă;
- Interfața aplicației este tot de tipul consolă și utilizatorul introduce comenzile una după alta;
- Folosind datele din fișiere se vor introduce validări care să nu permită utilizatorului să creeze 2 tabele cu aceeași denumire sau să execute comenzi INSERT, DELETE, UPDATE pe tabele care nu există sau nu au structura corectă;
- Comenzile de tip INSERT, DELETE și UPDATE vor produce efecte în fișierele de date asociate tabelelor
- Comenzile de tip CREATE TABLE și DROP TABLE vor produce efecte în structura de fișiere
- Entitățile care gestionează fișierele și operațiile pe fișiere se implementează prin clase (codul sursa prin care sunt accesate fișierele trebuie sa fie inclus în metode care să aparțină unor clase noi sau existente)

Tipuri de fișiere utilizate în această fază

- Fișiere de tip text de comenzi transmise ca argumente pentru funcția main(). Sunt procesate la pornirea aplicației
- Fișiere de tip text/binar de configurare încărcate la pornirea aplicației. Fișierele conțin descrierea tabelelor existente. Fișierele au denumiri predefinite (alese de programatori) și locația lor este cunoscută. Această informație nu este solicitată utilizatorilor.
- Fișiere de tip binar ce conțin datele din tabele. Fiecare tabelă are asociat un fișier binar. Crearea unei tabele conduce la crearea fișierului asociat. Ștergerea tabelei conduce la ștergerea fișierului.
- Fișiere CSV ce conțin date ce pot fi importate în tabele (o alternativă la o secvență de comenzi INSERT).

Cerințe specifice

- Se implementează un modul (una sau mai multe clase) prin care aplicația primește unul sau mai multe fișiere de intrare, de tip text, prin argumentele funcției main

Exemplu: dacă aplicația se numește proiectPOO.exe atunci ea este lansată în execuție cu comanda

proiectPOO.exe comenzi.txt [comenzi2.txt];

Scenariu posibil: Aplicația primește un fișier de comenzi prin care sunt definite tabelele și indexul pentru acestea prin comenzi de tipul CREATE. Un al 2-lea fișier ar conține comenzi de tipul INSERT.

- Pot fi folosite doar fișiere text ca argumente pentru funcția main(). Maxim 5
- Aplicația poate fi pornită fără a primi argumente prin funcția main (se lansează în execuție cu proiectPOO.exe).
- Descrierea tabelelor (nume coloane, tipul lor, etc) existente în sistem se va stoca în fișiere text de configurare asociate acestor structuri (numele fișierului poate corespunde numelui tabelei). Fișierul sau fișierele de configurare sunt cunoscute de programator și sunt accesate automat de aplicație la start.

Recomandare: folosind fișiere binare sau text aplicația încarcă descrierea tabelelor din sistem. La pornire aplicația știe ce tabele există pe baza acestor fișiere de configurare.

- Toate datele inserate de utilizatori în tabele prin comenzi INSERT se vor stoca în fișiere binare
- Fiecare tabelă din sistem va avea asociat un fișier binar ce are numele tabelei și o extensie la alegere
- Se implementează un modul (una sau mai multe clase) prin care datele aplicației sunt salvate în fișiere binare. Datele aplicației se considera a fi acele date obținute din fișierele text primite ca argumente pentru main sau date încărcate de utilizatori în timpul sesiunii de lucru prin comenzi de tip INSERT sau IMPORT

Recomandare: pentru a gestiona tipuri diferite de date este recomandată stocarea datelor sub formă de șiruri de bytes. La citirea lor se va folosi descrierea tabelei pentru a le converti la tipurile primitive asociate (string, numeric întreg, etc)

- Se implementează modulul prin care execuția comenzilor de tip SELECT sau DISPLAY TABLE generează automat rapoarte pentru datele afișate (de ex. lista date dintr-o anumită tabelă, etc) la consola dar și în fișiere text (numele fișierelor text sunt generate automat).

De exemplu: pentru o comanda SELECT se va genera fișierul text SELECT_1.txt ce conține rezultatul acelei comenzi. Pentru următoarea comanda SELECT se generează SELECT_2.txt. Rezultatele afișate de comenzi pe ecran se vor salva și în aceste fișiere.

- Aplicația permite încărcarea de date din fișiere CSV (comma separated values);

simbolul ales pentru a separa valorile (, ; # | sau altul) este ales de programator și este folosit pentru toate fișierele CSV ale aplicației. Importul fișierelor CSV se va face prin adăugarea comenzii

IMPORT nume_tabela nume_fisier.CSV

- Dacă fișierul CSV nu conține separatorul așteptat sau nu are structura aferentă tabelei indicate atunci este afișat un mesaj de eroare care indică separatorul și structura tabelei respective
- Pentru comanda IMPORT cele 2 argumente sunt obligatorii
- Faza 2 se considera realizata dacă sunt implementate cel puțin 75% dintre cerințe

Cerințe tehnice:

- Minim 3 clase noi ce conțin relații de tipul has-a cu alte clase
- Adăugarea în clasele existente sau în clasele noi a minim 2 vectori de obiecte. Minim unul dintre vectori este dinamic.
- Entitățile care gestionează fișierele și operațiile pe fișiere se implementează prin clase (codul sursa prin care sunt accesate fișierele trebuie sa fie inclus în metode care să aparțină unor clase noi sau existente)

Bonus (cerință opțională):

- Implementarea modului care construiește index pentru tabela indicată (vezi comanda CREATE INDEX)
- Indexul trebuie să permită căutarea rapidă a datelor ca răspuns la comenzile SELECT cu clauza WHERE unde condiția este indexul
- Prin utilizarea indexului, se va defini un fișier binar asociat acestuia în care este stocată valoarea coloanei index și poziția înregistrării aferente (offset-ul) în fișierul binar asociat tabelei respective
- Pentru comenzile SELECT cu clauza WHERE pe coloana index căutarea se va face în index și apoi sunt extrase înregistrările aferente (fără index, căutarea înregistrărilor se va face secvențial în fișierul binar)

Faza 3 - Finalizare proiect

- refactoring prin utilizarea relațiilor de tip is-a între clase
- utilizarea de colecții de tip STL și adăugarea de noi funcții
- Realizarea unui video de 5 minute în care echipa să prezinte un demo live al aplicației

Cerințe prezentare

- În funcție de solicitările coordonatorului de la seminar - Realizarea unui video de 5 minute (maxim 5 minute) în care echipa să prezinte un demo live al aplicației care să se concentreze doar pe funcțiile aplicației (NU este prezentat cod sursă). În înregistrare este prezentată doar utilizarea aplicației prin exemplificarea introducerii de diferite comenzi și a rezultatelor obținute (pe ecran sau în fișierele cu date)

Cerințe specifice

- Toate cerințele solicitate în fazele anterioare trebuie să fie funcționale
- Aplicația trebuie sa ruleze "la cheie" fără a genera erori la execuție
- Opțional - implementarea unui modul care sa gestioneze o forma simpla de meniu (folosind taste numerice) prin care utilizatorul are posibilitatea sa selecteze o serie de rapoarte predefinite (ex. Lista de tabele din sistem sau înregistrările din anumite tabele predefinite). Meniul se poate activa automat la pornirea aplicației (după încărcarea fișierelor primite din linia de comanda sau) sau se activează printr-o comanda specifica (ex MENU). Meniul trebuie sa conțină o opțiune care sa permita utilizatorului sa revină în modul de introducere a comenzilor.

Cerințe tehnice

- Implementarea/Modificarea unei clase care sa exemplifice conceptul compunerii claselor prin relații de tip has-a (dacă nu a fost deja implementată în fazele anterioare)
- Extinderea claselor definite în fazele anterioare prin adaugarea de atribute noi (necesare în aceasta faza) se face doar prin derivare și NU prin modificarea claselor existente - minim 2 extinderi
- Se implementează minim o clasa abstracta (cu sau fără atribute) ce trebuie sa conțină minim 2 metode virtuale pure
- Adăugarea într-o clasa existentă sau într-o clasă nouă a minim 2 metode virtuale (altele decat cele virtuale pure) care să fie supradefinite în clasele derivate
- Prin intermediul clasei abstracte sau a unei clase de bază (nou definită sau existentă) se defineste minim o ierarhie care sa descrie o familie de clase (ex. pentru tipuri de cheltuieli, etc); un exemplu de familie de clase definite pentru a descrie forme geometrice
- Se implementează minim un vector de pointeri la clasa de baza (a ierarhiei) prin care se gestionează elemente concrete dintr-o familie de clase (o familie de clase este dată de un set de clase aflate în relații de derivare/mostenire ce au la bază un părinte comun - un exemplu de familie de clase definite pentru a descrie forme geometrice)
- Sa se implementeze din libraria STL cel puțin un vector, un set, o lista și un map pentru a gestiona datele aplicației (colecțiile se folosesc fie în clase existente sau în unele noi adăugate în această fază). Puteți modifica o clasă existentă înlocuind vectorii clasici (statici sau dinamici) cu colecții STL.