**ROMÂNIA**

**MINISTERUL APĂRĂRII NAȚIONALE**

**ACADEMIA TEHNICĂ MILITARĂ “FERDINAND I”**

**FACULTATEA DE SISTEME INFORMATICE ȘI SECURITATE CIBERNETICĂ**

**Specializarea: Calculatoare Și Sisteme Informatice Pentru Apărare Și Securitate Națională**



**IMPLEMENTAREA UNUI SISTEM DE GESTIUNE A UNEI BAZE DE DATE**

REALIZAT DE:

**Sd. Sg. DARIA PASCU**

**Sd. Sg. DIANA-MIHAELA NEGUT**

**BUCUREȘTI**

**2024**

### Introducere

Acest proiect are ca scop implementarea unui sistem de gestiune al unei baze de date dezvoltat in limbajul C. Sistemul include un parser de comenzi SQL, un interpretor de SQL și suport pentru procesarea multithreading, ceea ce permite mai multor clienti să interacționeze simultan cu baza de date.

#### Parser SQL

Parserul SQL reprezintă o componentă centrală a proiectului, responsabilă de intrepretarea comenzilor SQL primite de la fiecare client conectat. Acesta este de asemenea responsabil și de transformarea acestora în operații gestionabile de către sistemul bazei de date.

Parserul implementat permite și recunoaște comenzi SQL precum: **SELECT** , **INSERT** , **UPDATE** , **DELETE** și **CREATE TABLE**.

Funcția ***initSQLParser*** asociază fiecare metodă din structura SQLParser cu implementările corespunzătoare. Acest proces configurează parserul astfel încât să poată răspunde comenzilor SQL transmise.

Este necesară existența unei funcții pentru a identifica tipul comenzii SQL, care verifică dacă aceasta corespunde operațiilor suportate de către parser, urmând să redirecționeze serverul către funcția necesară pentru o procesare detaliată.

Parserul permite comenzi de tipul ***SELECT*** impreună cu clauza ***WHERE***. Clauza where permite condiții de tipul coloană, operator (=,<,>,>= , <=, != ) și valoarea.

De asemenea, este necesară o funcție de validare a datelor care vor fi inserate, identificând și tabelul în care trebuie să fie stocate valorile.

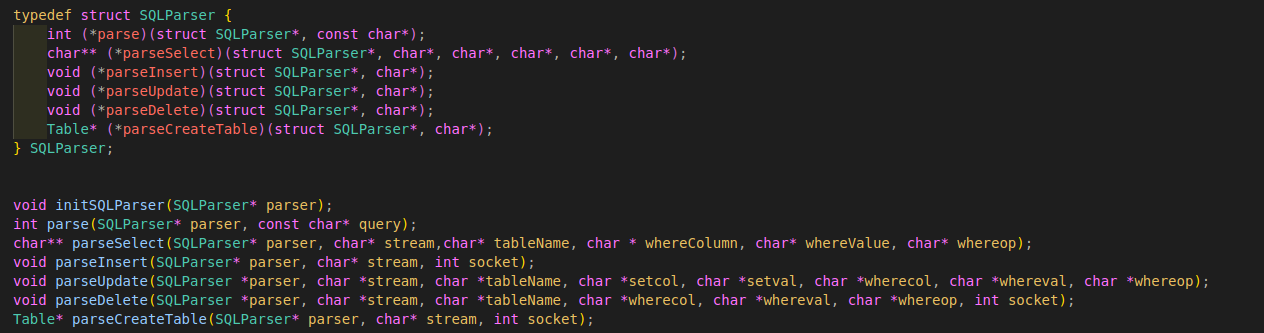


Fig. Structură SQL Parser

### Interpretor SQL

Interpretorul SQL reprezintă componenta sistemului care gestionează execuția comenzilor SQL procesate de parser. Acesta le transformă în operații asupra tabelelor din sistemul bazei de date, gestionând funcționalitățile necesare pentru a răspunde solicitărilor clienților. Interpretorul utilizează structurile și funcțiile definite în header-ul ***Gestionare Tabele***, oferind suport pentru operații precum inserarea, afișarea, modificarea și salvarea datelor.

Interpretorul lucrează direct cu obiecte de tip ***Table***, definite în gestionareTabele.h, și implementează funcționalități specifice pentru a opera asupra acestor structuri.

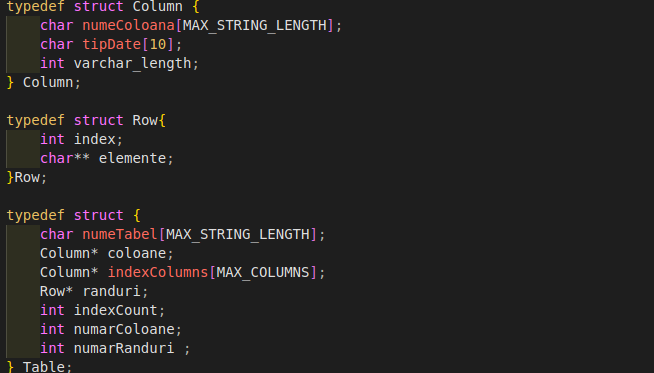


Fig. Structură Tabel

Funcționalitățile interpretorului:

1. Interpretorul utilizează funcția ***loadTable*** pentru tabelul din fișierul în care acesta este stocat. Aceasta analizează structura și datele tabelului stocate pe disc și creează o instanță de tip ***Table*** în memorie.



Fig. Semnătură funcție loadTable

1. Funcția ***creeazaTabel*** permite inițializarea unui nou tabel pe baza unui set de coloane definite de utilizator.



Fig. Semnătură funcție creeazaTabel

1. Interpretorul oferă suport pentru afișarea tabelului într-un format prietenos, folosind funcția ***afiseazaTabel*** (afisare\_nice). Aceasta afișează atât numele coloanelor selectate, cât și datele din rândurile selectate dacă există o clauza “where”.



Fig. Semnătură funcție afisare\_nice

### Gestionarea conexiunilor mai multor client

Thread pool-ul implementat în acest proiect este o soluție eficientă pentru gestionarea concurenței în cadrul aplicației. Un thread pool este o colecție de thread-uri prealocate, care așteaptă în mod continuu să execute sarcini specifice. Această abordare elimină consumul de timp pentru crearea și distrugerea thread-urilor la fiecare cerere.

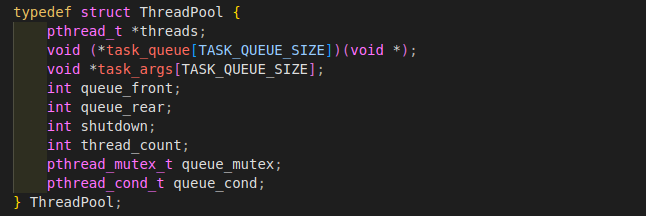


Fig. Structură ThreadPool

Implementarea constă într-o structură ***ThreadPool*** care conține un număr fix de thread-uri, o coadă circulară pentru sarcini, precum și mecanisme de sincronizare pentru gestionarea accesului concurent. Când este creat, thread pool-ul inițializează un număr de thread-uri specificat și le asociază o funcție ***threadWorker***, care preia și execută sarcinile din coadă. Fiecare thread așteaptă în stare inactivă până când o sarcină este disponibilă, asigurând astfel utilizarea optimă a resurselor.

Funcția **addTaskToPool** adaugă o sarcină în coada de așteptare a thread pool-ului. Dacă coada este plină, sarcina este respinsă, iar utilizatorul este notificat printr-un mesaj. Thread-urile accesează sarcinile din coadă într-un mod sincronizat, folosind un mutex pentru a preveni race conditions. În plus, o variabilă de condiție este utilizată pentru a semnala thread-urilor disponibile că există sarcini în coadă.

Funcția ***destroyThreadPool*** asigură închiderea corectă a thread pool-ului. Aceasta marchează pool-ul ca fiind închis, semnalează toate thread-urile să se oprească și așteaptă finalizarea acestora. După încheierea execuției, toate resursele utilizate de pool sunt eliberate.

### Optimizarea consumului de resurse

###### Maparea directă a memoriei

Maparea fișierelor direct în memoria procesului se realizează prin funcții precum ***mmap*** în Linux. Această tehnică permite manipularea fișierelor ca și cum ar fi o zonă de memorie obișnuită, eliminând necesitatea de a apela în mod repetat funcții de citire și scriere, care introduc un overhead pentru fișierele de dimensiuni mari.

Fișierele aferente tabelelor vor fi mapate în memorie pentru a facilita accesul rapid la rânduri și coloane. Orice modificare a datelor va fi efectuată direct asupra zonei mapate, fără a implica operații suplimentare de scriere. Astfel, operațiile de citire și scriere devin semnificativ mai rapide, deoarece sunt eliminate treceriile din user-space in kernel-space, iar modificările sunt reflectate imediat în fișierul mapat și sincronizate automat la închiderea zonei de memorie.

###### Mecanism Caching

Folosind un mecanism de ***caching*** putem îmbunătăți performanța proiectului, reducând o parte din problemele de suprasolicitare a serverului. Prin stocarea temporară a datelor pe partea clientului, caching-ul reduce numărul cererilor trimise către server. Aceasta permite serverului să gestioneze un număr mai mare de clienți și să proceseze cererile care chiar necesită intervenția serverului.

Dacă un client execută mai multe tranzacții care folosesc aceleași date, acestea pot fi preluate direct din cache fără a accesa serverul.

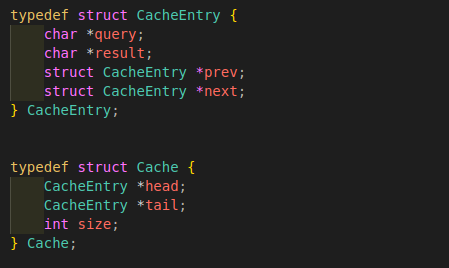


Fig. Structură Cache

### Mecanism de backup

Thread-ul de backup implementează un mecanism automat de copiere a fișierelor din directorul curent, utilizând lista fișierelor specificată în ***fișierul master***. Această funcție asigură protecția datelor prin crearea de copii de siguranță ale fișierelor la fiecare 5 minute, fără a necesita intervenția utilizatorului. La fiecare execuție, funcția verifică dacă directorul de destinație, backup, există. Dacă nu este găsit, acesta este creat automat cu permisiuni adecvate. Apoi, funcția citește fișierele din fișierul master, care conține numele fișierelor ce trebuie copiate, câte unul pe linie.

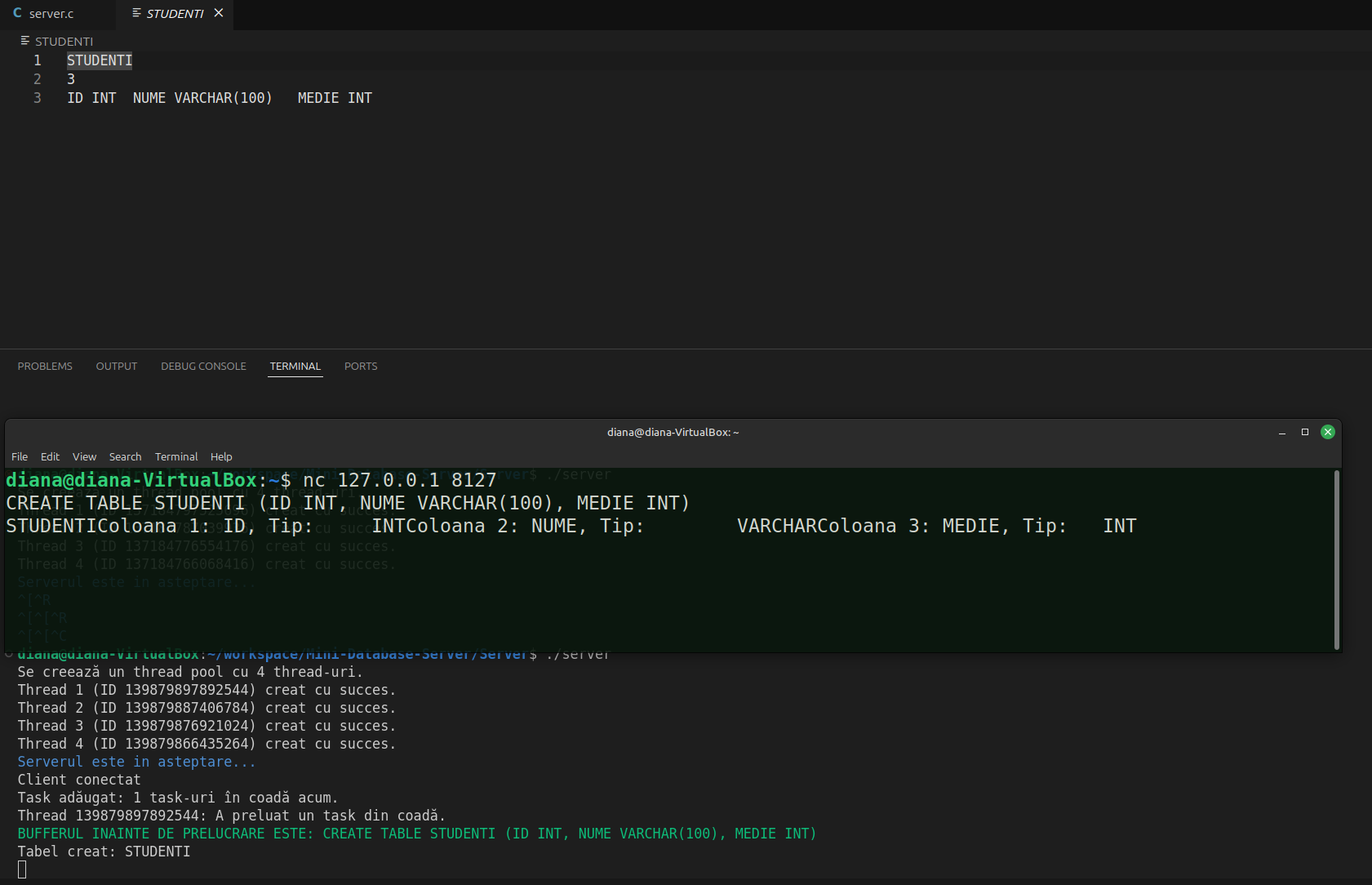
Pentru fiecare fișier listat, funcția construiește o cale completă către directorul de backup și apelează o funcție de copiere care asigură transferul conținutului fișierului sursă în fișierul de destinație. Acest proces repetitiv garantează că orice modificare sau fișier nou adăugat în lista master este inclus în backup-ul următor. Thread-ul funcționează în fundal și este conceput să ruleze continuu, reducând riscul pierderii datelor. Fiecare ciclu de execuție include o perioadă de așteptare de 5 minute (sleep(300)), pentru a menține o frecvență regulată a copiilor de siguranță, fără a suprasolicita resursele sistemului.



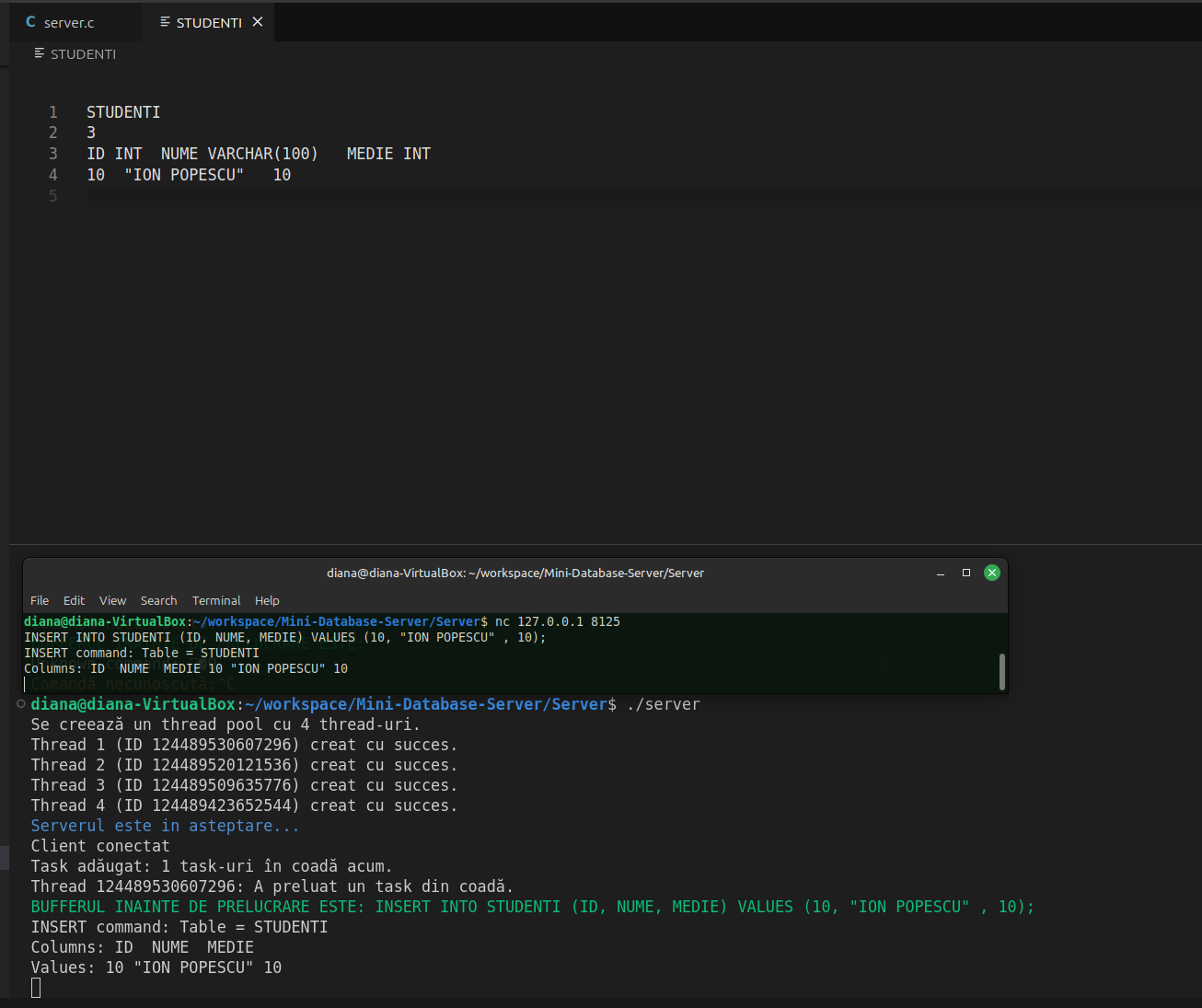
Fig. Semnătură funcție backupThread

### Exemple funcționale

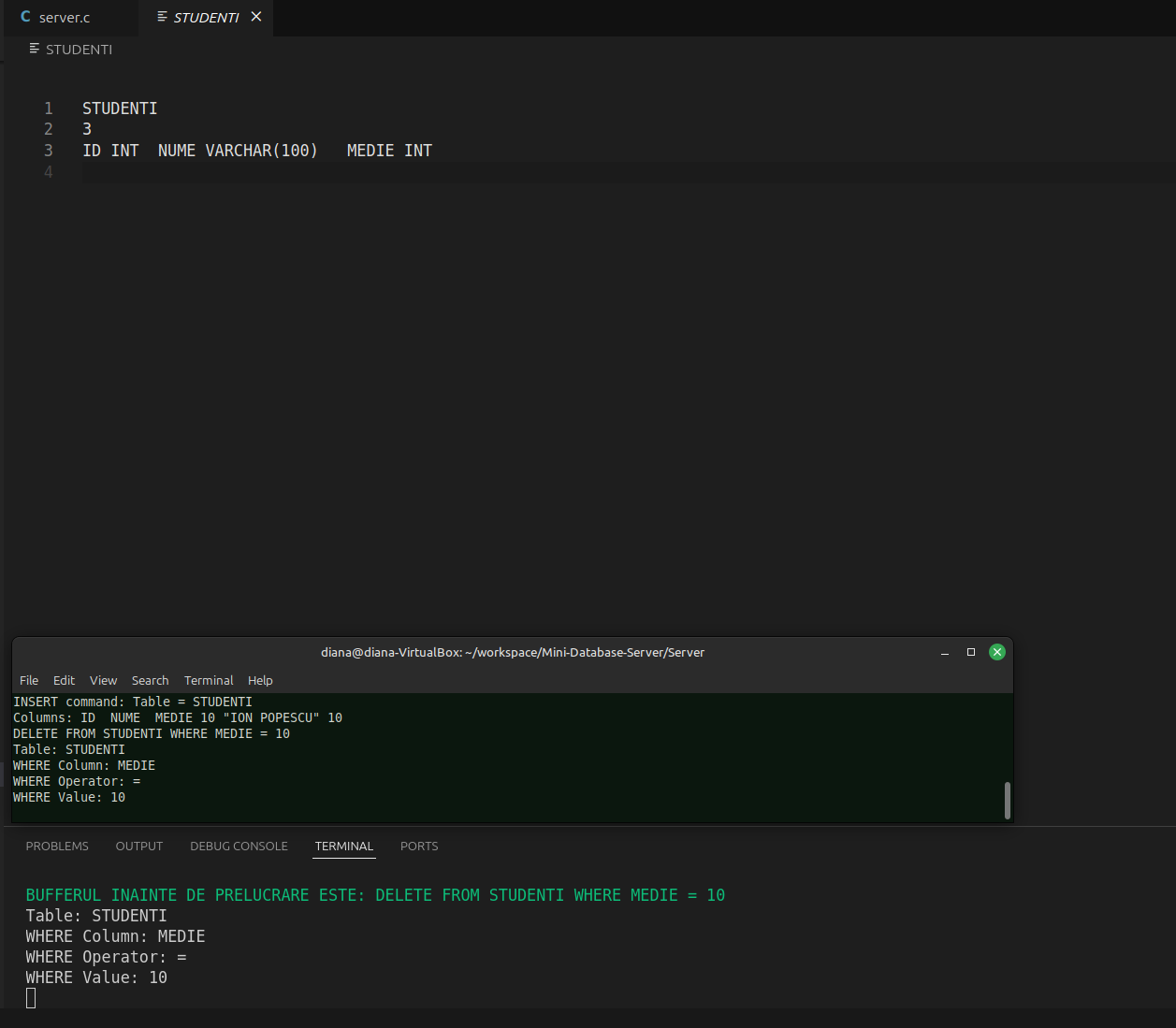
* Creare tabel



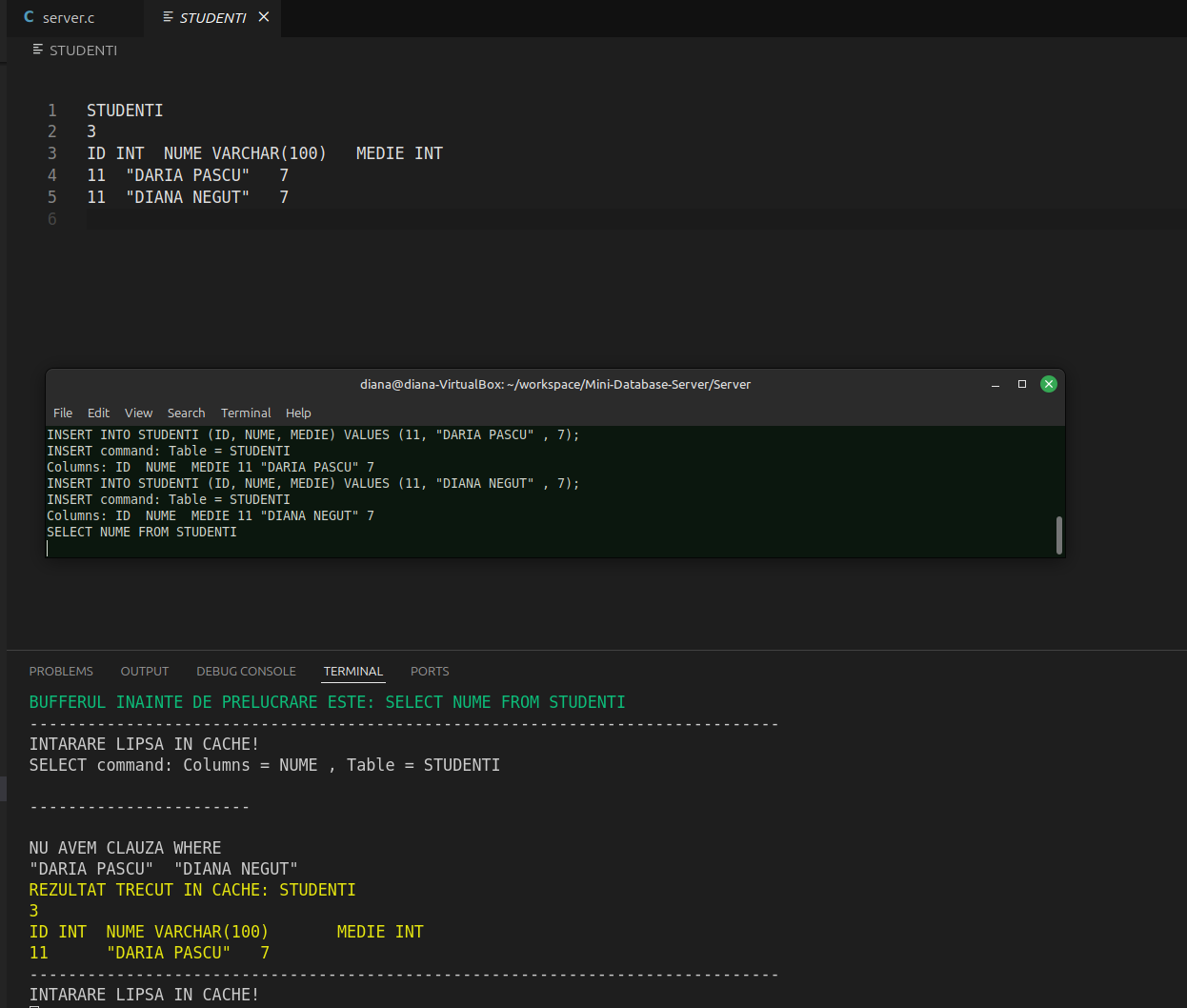
* Inserare element



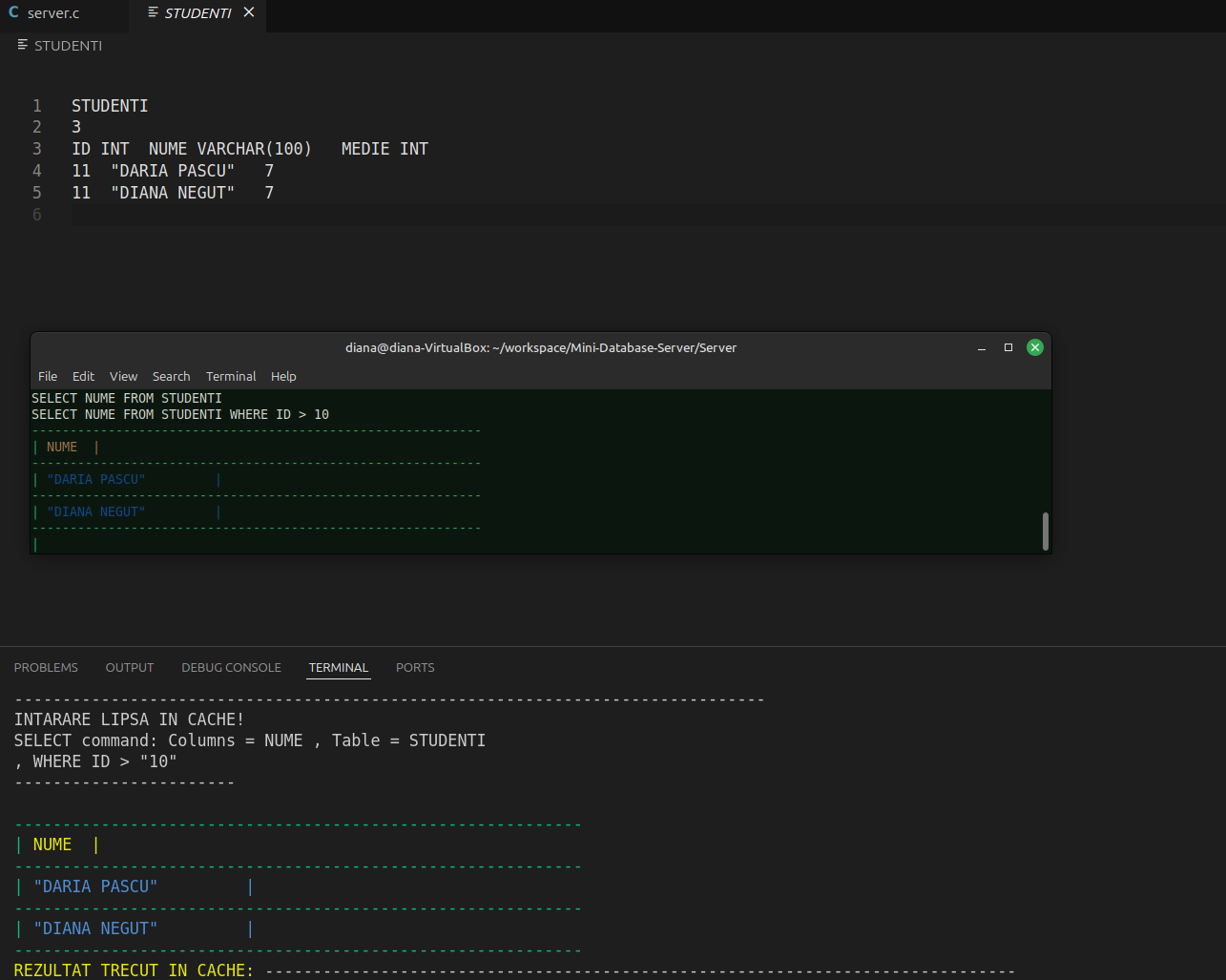
* Ștergere element tabel



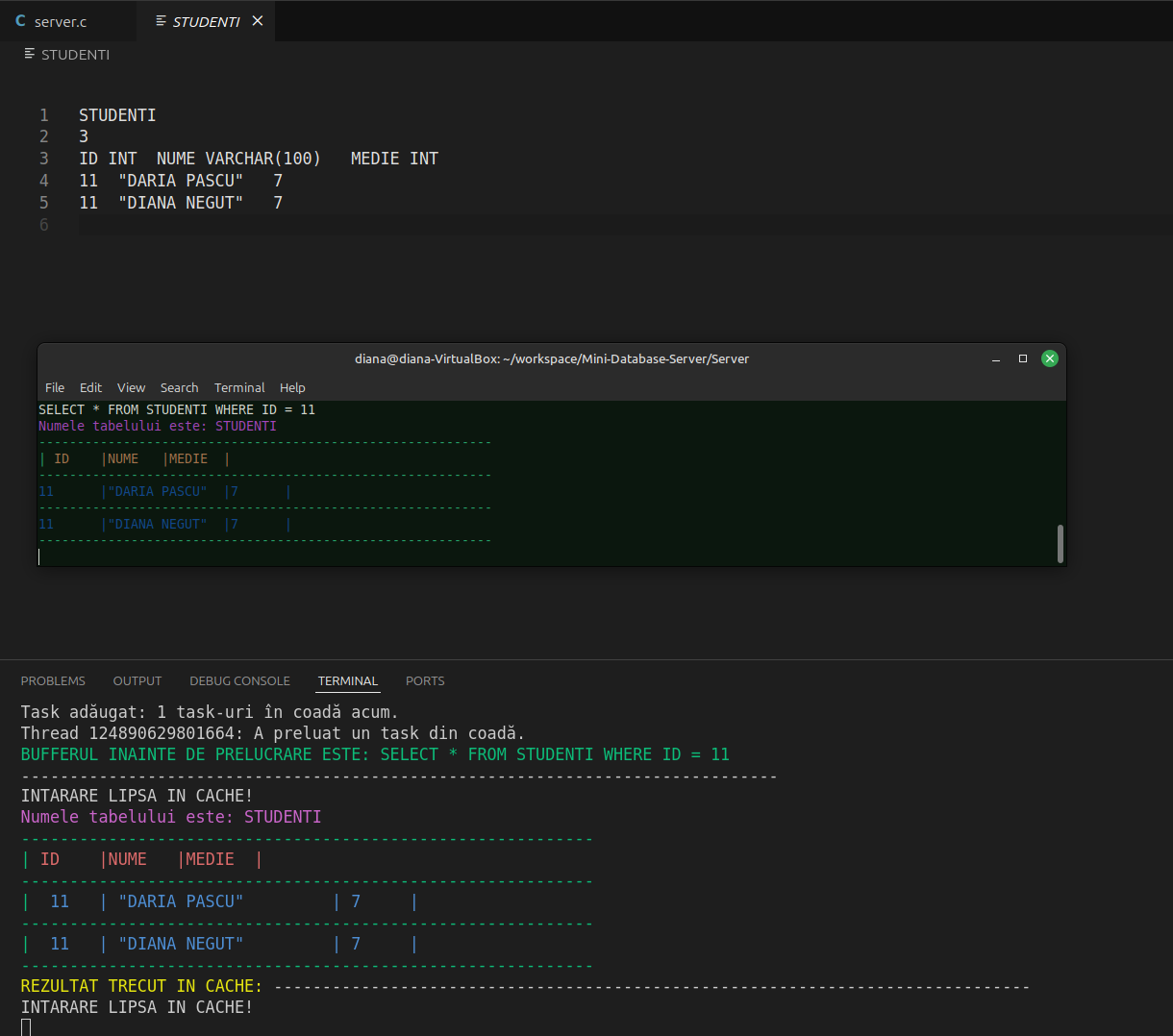
* Select fără clauză where, se poate observa și folosirea mecanismului de caching



* Select clauză where



* Suport pentru comanda de tipul Select \* cu clauza where



* Suport Select \*

