PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS ESCOLA POLITÉCNICA

GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Fase 2 Implementar um interpretador

COMPIÇADORES

DANIEL XAVIER RODRIGUES

GOIÂNIA

2024

A linguagem escolhida foi o DBASE;

Escolhi os comandos básicos do DBASE que são o CREATE, INSERT, SELECT E DELETE.

Estrutura Geral do Código

Optei por definir constantes, estruturas e funções que gerenciam o banco de dados em memória, usando três comandos.

1. **Constantes:**
    - **MAX_CAMPOS**: Limita o número de campos por tabela a 40, o que é razoável para testes.
    - **MAX_REGISTROS:** Define um máximo de 1000 registros, ideal para manter a simplicidade e testar operações.
    - **MAX_TAMANHO_CAMPO:** Defini um tamanho máximo de 30 caracteres para nomes e valores de campos.
2. **Estruturas:**
    - **Campo:** Representa um campo da tabela, com o nome e o tipo (C para texto e N para número).
    - **Tabela:** Estrutura que armazena os campos e registros, com contadores de campos e registros.
    - **Token:** Representa tokens SQL (palavras-chave e símbolos) encontrados na análise da entrada do usuário.
3. **Enumeração (TipoToken):**
    - Usei o enumerador para definir tipos de tokens, incluindo comandos como TOKEN_CREATE, TOKEN_TABLE, TOKEN_VIRGULA, entre outros.

Funções

Token obter_token(const char entrada)**

Esta função analisa a string de entrada e identifica tokens SQL. A ideia foi simplificar a detecção de palavras-chave, caracteres especiais e literais, retornando o próximo token encontrado. Com isso fica mais fácil identificar o tipo de comando e processar.

- **Entrada**: Ponteiro para a string SQL.
- **Saída**: Um Token que representa o próximo token da entrada.

void inicializar_tabela()

Iniciei a estrutura da tabela com contadores de campos e registros zerados. Isso garante que a tabela esteja sempre vazia quando o programa começa.

void criar_tabela(const char entrada)**

Esta função lê tokens da entrada para criar uma tabela. Foi projetada para esperar um nome de tabela, seguido por uma lista de campos. Cada campo precisa de um nome e de um tipo (C ou N). A função incrementa o contador de campos ao processar.

- **Entrada**: Ponteiro para a string com o comando CREATE TABLE.
- **Saída**: Nenhuma (modifica a tabela diretamente).
- **Observação**: Adicionei verificações para garantir que a entrada tenha parênteses, nomes e tipos corretos. Qualquer erro de sintaxe é informado ao usuário.

void inserir_na_tabela(const char entrada)**

Usei esta função para inserir registros na tabela existente, esperando que os valores estejam entre parênteses e correspondam à quantidade de campos.

- **Entrada**: Ponteiro para o comando INSERT INTO.
- **Saída**: Nenhuma (modifica os registros diretamente).
- **Observação**: Adicionei uma verificação para garantir que a tabela exista e para verificar o número de campos. Também exibo mensagens de erro se houver alguma inconsistência.

void exibir_tabela()

Exibe todos os registros da tabela. Mostro os nomes dos campos na primeira linha e os valores dos registros abaixo, separados por tabulações. Se a tabela não tiver sido criada informo isso ao usuário.

- **Entrada**: Nenhuma.
- **Saída**: Impressão na tela.

void executar_comando(const char* entrada)

Aqui, criei um sistema de execução que interpreta comandos SQL digitados pelo usuário. Esta função identifica se o comando é CREATE TABLE, INSERT INTO ou SELECT * FROM, chamando a função correspondente.

- **Entrada**: String com o comando SQL.
- **Saída**: Nenhuma (executa o comando e, se necessário, imprime o resultado).
- **Observação**: Se o comando não for reconhecido, o programa informa o usuário.

excluir_ registros

- Implementa o comando DELETE FROM, que remove todos os registros da tabela.
- Exemplo de uso: DELETE FROM nome_tabela;

main()

A função principal configura o loop para leitura de comandos, encerrando o programa quando o usuário digita "sair". Usei fgets para ler as entradas do usuário, o que me permite lidar com strings completas.

Exemplo de Uso

CREATE TABLE:

```
C:\Users\danie\OneDrive\Área de Trabalho\Faculdade\10PERIODO\COMPILADORES\Interpretador>interpretador.exe
Digite um comando SQL ou 'sair' para encerrar:
> CREATE TABLE Alunos (Nome C, Idade N);
```

Comando digitado, quando for pressionado o enter será criada a tabela e dará a mensagem de “Tabela criada com sucesso!”, caso o comando esteja correto.

```
C:\Users\danie\OneDrive\Área de Trabalho\Faculdade\10PERIODO\COMPILADORES\Interpretador>interpretador.exe
Digite um comando SQL ou 'sair' para encerrar:
> CREATE TABLE Alunos (Nome C, Idade N);
Tabela criada com sucesso!
```

INSERT INTO

```
C:\Users\danie\OneDrive\Área de Trabalho\Faculdade\10PERIODO\COMPILADORES\Interpretador>interpretador.exe
Digite um comando SQL ou 'sair' para encerrar:
> CREATE TABLE Alunos (Nome C, Idade N);
Tabela criada com sucesso!
> INSERT INTO Alunos (Ana, 20);
Registro inserido com sucesso! (Registro 1)
>
```

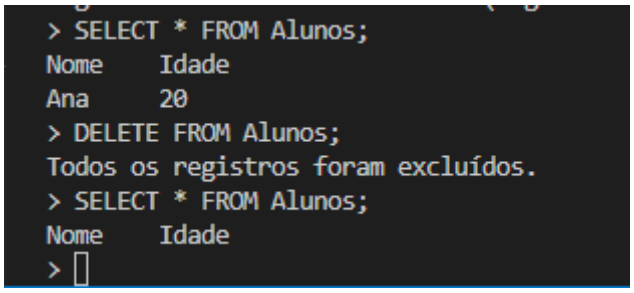
Comando digitado, e ao pressionar enter será inserido o registro dará a mensagem de “Registro inserido com sucesso!”, caso o comando esteja correto.

SELECT

```
C:\Users\danie\OneDrive\Área de Trabalho\Faculdade\10PERIODO\COMPILADORES\Interpretador>interpretador.exe
Digite um comando SQL ou 'sair' para encerrar:
> CREATE TABLE Alunos (Nome C, Idade N);
Tabela criada com sucesso!
> INSERT INTO Alunos (Ana, 20);
Registro inserido com sucesso! (Registro 1)
> SELECT * FROM Alunos;
Nome    Idade
Ana     20
>
```

Comando digitado, e ao pressionar enter será apresentado os registro que estiver na tabela, caso o comando esteja correto.

DELETE



Ao digitar o comando de DELETE, ele exclui todos os registros da tabela como mostra o print.