SQLite e C

ECM404

SQLite

- Permite utilizar os comandos em SQL e acessar bancos de dados sem a necessidade de um servidor (serverless).
- Os dados ficam armazenados em arquivos de texto.
- Permite maior facilidade na distribuição de aplicações.

SQLite – Integração com C

- Criação uma conexão com o banco.

```c
#include "sqlite3.h"

int main(){
    sqlite3 *db = NULL;
    int conexao;

    printf("Criando conexao com o banco:\n");
    conexao = sqlite3_open("pokemonCenter.db", &db);

    if (conexao != SQLITE_OK){
        printf("Erro ao conectar ao banco\n");
        exit(-1);
    }
    printf("Sucesso na conexao!\n");

    sqlite3_close(db);
    return 0;
}
```

SQLite – Integração com C

- **sqlite3 *db:** cria um ponteiro para uma estrutura de dados manipular o banco;
- **sqlite3_open(“arquivo.db” , &db):** abre o arquivo, atribuindo seu endereço à estrutura **db.** Retorna a constante inteira **SQLITE_OK** ou outro valor referente ao erro que ocorreu;
- **sqlite3_close(db):** fecha a conexão com o banco.

SQLite – Integração com C

- As pesquisas realizadas no banco são realizadas utilizando estruturas de pesquisas **statements**. Essas estruturas são implementadas como ponteiros do tipo **sqlite3_stmt ***.
- **Statements** são ponteiros para estruturas que precisam ser preparadas (como se um código estivesse sendo compilado para se tornar um programa). Depois de preparado, deve-se executar a instrução da string SQL.
- As estruturas podem ser reutilizadas no código.

SQLite – Integração com C

- Criação de um **statement**.

```c
#include "sqlite3.h"

int main(){
    sqlite3 *db = NULL;
    sqlite3_stmt *stmt = NULL;
    int conexao, i;

    printf("Criando conexao com o banco:\n");
    conexao = sqlite3_open("pokemonCenter.db", &db);

    if (conexao != SQLITE_OK){
        printf("Erro ao conectar ao banco\n");
        exit(-1);
    }
    printf("Sucesso na conexao!\n");
```

SQLite – Integração com C

- Prepara a consulta ao banco.

```
    // prepara a consulta ao banco
    sqlite3_prepare(db, "SELECT * FROM vendas", -1, &stmt, NULL);
```

- Escreve o cabeçalho com o nome das colunas.

```
    // escreve o cabeçalho com o nome das colunas
    for (i = 0; i < sqlite3_column_count(stmt); i++)
        printf("%s\t", sqlite3_column_name(stmt, i));
    printf("\n"); // termina a exibição do cabeçalho
```

SQLite – Integração com C

- Escreve os demais dados da tabela.

```
    // para cada linha da tabela resultado
    // while (sqlite3_step(stmt) == SQLITE_ROW)  OU

    while (sqlite3_step(stmt) != SQLITE_DONE){
        // para cada coluna
        for (i = 0; i < sqlite3_column_count(stmt); i++)
            printf("%s\t", sqlite3_column_text(stmt, i));
        printf("\n"); // termina a exibição de uma linha
    }

    sqlite3_finalize(stmt);
    sqlite3_close(db);
    return 0;
}
```

SQLite – Integração com C

- **sqlite3_stmt *stmt:** cria um ponteiro para uma estrutura de dados realizar as operações no banco;
- **sqlite3_prepare(db, STRING_SQL, -1, &stmt, NULL):** compila, mas não executa, a string SQL que irá manipular os registros no banco de dados **db** e retornar o ponteiro com o resultado das operações em **stmt**;
- Os parâmetros **-1** e **NULL** não alterados, pois representam o tamanho máximo da string em bytes e o endereço da posição final da string respectivamente.

SQLite – Integração com C

- **sqlite3_step(stmt):** Define o valor do ponteiro da estrutura **stmt** para uma linha da tabela resultante. Além disso, retorna **SQLITE_ROW**, quando o ponteiro apontar uma linha da tabela ou **SQLITE_DONE** quando não houverem mais dados;
- Deve ser utilizado para se obter cada uma das linhas da tabela resultante;
- Pode ser utilizado como um **procedimento**, critério de uma **estrutura condicional** ou de **repetição**.
- Para facilitar a composição da string SQL utilizaremos a função **sprintf**.

Inserir ou Alterar Valores

- Inserir novos valores.

```
sprintf(sql, "INSERT INTO vendas
     (idVenda, cod, quant) VALUES (%i, %i, %i)",
                    idVenda, codigo, quantVendida);

sqlite3_prepare(db, sql, -1, &stmt, NULL);
sqlite3_step(stmt);
```

- Alterando valores.

```
sprintf(sql, "UPDATE estoque
      SET quant=quant-%i WHERE cod=%i", quantVendida, codigo);

sqlite3_prepare(db, sql, -1, &stmt, NULL);
sqlite3_step(stmt);
```

Acessando Retorno

- Quando uma query retornar somente um valor em uma linha.

```
sprintf(sql, "SELECT SUM(quant*preco) FROM
         tabela WHERE idVenda = %i", idVenda);

sqlite3_prepare(db, sql, -1, &stmt, NULL);

if (sqlite3_step(stmt) == SQLITE_ROW){
  // acessa a coluna 0 da linha recebida
}

else {
  // se necessário
}
```

Acessando Retorno

- Quando uma query retornar uma tabela.

```
sprintf(sql, "SELECT * FROM estoque
                    WHERE quant < %i", quant);

sqlite3_prepare(db, sql, -1, &stmt, NULL);

// para cada linha
while (sqlite3_step(stmt) != SQLITE_DONE){

 // para cada coluna
 for (i=0; i<sqlite3_column_count(stmt); i++){
  // acessa a coluna i da linha de stmt
 }
}
```

Acessando Retorno

- **sqlite3_column_count(stmt):** Conta a quantidade de colunas da tabela resultado;
- **sqlite3_column_name(stmt,i):** Retorna a string com o nome da coluna **i** do registro atual. Os índices começam em **0**!
- Os dados de cada coluna podem ser obtidos como:
  - **sqlite3_column_text(stmt,i):** Retorna o valor em formato de texto da coluna **i** do registro atual;
  - **sqlite3_column_int(stmt,i):** Retorna o valor em formato de número inteiro da coluna **i** do registro atual;
  - **sqlite3_column_double(stmt,i):** Retorna o valor em formato de número real da coluna **i** do registro atual;