SQL

DML - Data Manipulation Language (Linguagem de Manipulação de dados)

Linguagem de Manipulação de dados

- Uma linguagem de manipulação de dados (DML) permite aos usuários acessar ou manipular dados conforme organizados pelo modelo de dados apropriado.
- Compõe uma única linguagem de consulta banco de dados, como por exemplo, o SQL. Se assemelha a uma linguagem simples (do inglês) e torna fácil a interação do usuário com o sistema de banco de dados.

Linguagem de Manipulação de dados

- **DML** fornece comandos para
  que os usuários consigam manipular dados em um banco de dados. A manipulação envolve **inserir**, **recuperar**, **excluir** e **atualizar** dados em tabelas de banco de dados.

Linguagem de Manipulação de dados

- **INSERT:** adiciona novas informações no banco de dados
- **UPDATE:** altera informações armazenadas no banco de dados
- **DELETE:** deleta informações do banco de dados
- **SELECT:** recupera informações armazenadas no banco de dados

INSERT

- **INSERT** é uma declaração **SQL** que adiciona um ou mais **registros** em qualquer tabela simples de um **banco de dados relacional**.

INSERT

- Declarações INSERT têm a seguinte forma:
- **INSERT INTO tabela (coluna1, [coluna2, ...]) VALUES (valor1, [valor2, ...])**
- O número de colunas e valores devem ser o mesmo.
- Se uma coluna não for especificada, o valor padrão é usado. Os valores especificados (ou incluídos) pela declaração INSERT devem satisfazer todas as restrições aplicáveis (tais como chaves primárias, restrições CHECK e restrições NOT NULL).
- Se ocorrer um erro de sintaxe ou se algumas das restrições forem violadas, a nova linha não é adicionada à tabela e um erro é retornado.

INSERT exemplos

- INSERT padrão com colunas a serem adicionadas informados pelo usuário.

```
1  INSERT INTO EMPREGADOS (CODIGO, NOME, SALARIO, SECAO)
2  VALUES
3    (
4      1,
5      "HELBERT CARVALHO",
6      1.500,
7      1
8    );
```

- INSERT padrão sem informar colunas.

```
1  INSERT INTO EMPREGADOS
2  VALUES
3    (
4      1,
5      "HELBERT CARVALHO",
6      1500,
7      1
8    );
```

INSERT exemplos

- INSERT + SELECT comando utilizado para inserir informações no banco de dados a partir do retorno de uma consulta..

```
INSERT INTO EMPREGADOS (CODIGO, NOME, SALARIO, SECAO) SELECT
  CODIGO,
  NOME,
  SALARIO,
  SECAO
FROM
  EMPREGADOS_FILIAL
WHERE
  DEPARTAMENTO = 2;
```

- Comando utilizado para inserir registros em uma tabela a partir de um arquivo de texto.

```
BULK INSERT [ TABLE ]
FROM
  '"&[path filename]&"' WITH (
    fieldterminator = ' ',
    rowterminator = '\n'
  )
```

UPDATE

- **UPDATE** é uma declaração **SQL** que atualiza um ou mais **registros** em qualquer tabela simples de um **banco de dados relacional**.

UPDATE

- O comando para atualizar registros é **UPDATE**, que tem a seguinte sintaxe:
- **UPDATE nome_tabela SET CAMPO = "novo_valor“ WHERE CONDIÇÃO**

Onde:

- Nome_tabela: nome da tabela que será modificada
- Campo: campo que terá seu valor alterado
- Novo_valor: valor que substituirá o antigo dado cadastrado em campo
- Where: Se não for informado, a tabela intera será atualizada
- Condição: regra que impõe condição para execução do comando

UPDATE exemplos

- UPDATE padrão atualizando uma única informação da linha selecionada.

```
1  UPDATE DEPARTAMENTO
2  SET SALARIO = 1000
3  WHERE
4    CODIGODEP = 1;
```

- UPDATE padrão atualizando uma varias informações da linha selecionada.

```
1  UPDATE DEPARTAMENTO
2  SET NOME = "HELBERT CARVALHO",
3   SALARIO = 1000
4  WHERE
5    CODIGO = 1;
```

UPDATE exemplos

- UPDATE em todas as linhas utilizando SELECT como limitador.

```
1  UPDATE EMPREGADOS
2  SET SALARIO = salario * 1.1
3  WHERE
4    SALARIO = (
5      SELECT
6        MIN(salario)
7      FROM
8        EMPREGADOS
9    );
```

- UPDATE em registros usando como novo valor retorno de um SELECT.

```
1  UPDATE EMPREGADOS
2  SET SALARIO = (
3    SELECT
4      MAX(salario)
5    FROM
6      EMPREGADOS
7  )
8  WHERE
9    DEPARTAMENTO = 5;
```

DELETE

- O comando utilizado para apagar dados é o **DELETE**.

```
1  DELETE
2  FROM
3    nome_tabela
4  WHERE
5    condição
```

Onde:

- Nome_tabela: nome da tabela que será modificada
- Where: cláusula que impõe uma condição sobre a execução do comando

DELETE exemplos

- DELETE padrão.

```
1  DELETE FROM EMPREGADOS
2  WHERE CODIGO = 125
```

SELECT

- O **comando SELECT** permite recuperar os dados de um objeto do banco de dados, como uma tabela, view e, em alguns casos, uma stored procedure (alguns bancos de dados permitem a criação de procedimentos que retornam valor). A sintaxe mais básica do comando é:

```
1  SELECT
2    < lista_de_campos >
3  FROM
4    < nome_da_tabela ></ nome_da_tabela ></ lista_de_campos >
```

Exemplo SELECT

- O caractere * representa todos os campos. Apesar de prático, este caractere não é muito utilizado, pois, para o **SGBD** é mais rápido receber o comando com todos os campos explicitados. O uso do * obriga o servidor a consultar quais são os campos antes de efetuar a busca dos dados, criando mais um passo no processo.

```
1  SELECT
2    CODIGO,
3    NOME
4  FROM
5    CLIENTES SELECT
6      *
7    FROM
8      CLIENTES
```

COMANDO WHERE

- A cláusula Where permite ao comando SQL passar condições de filtragem:

```
1  SELECT CODIGO, NOME FROM CLIENTES
2  WHERE CODIGO = 10;
3
4  SELECT CODIGO, NOME FROM CLIENTES
5  WHERE UF = 'RJ';
6
7  SELECT CODIGO, NOME FROM CLIENTES
8  WHERE CODIGO >= 100 AND CODIGO <= 500;
9
10 SELECT CODIGO, NOME FROM CLIENTES
11 WHERE UF = 'MG' OR UF = 'SP';
12
13 SELECT CODIGO, NOME FROM CLIENTES
14 WHERE UF = 'RJ' OR (UF = 'SP' AND ATIVO = 'N');
15
16 SELECT CODIGO, NOME FROM CLIENTES
17 WHERE (ENDERECO IS NULL) OR (CIDADE IS NULL)
```

FILTRO DE TEXTO

- Para busca parcial de string, o SELECT fornece o operador LIKE. Veja o exemplo abaixo:

```
1  SELECT
2    CODIGO,
3    NOME
4  FROM
5    CLIENTES
6  WHERE
7    NOME LIKE 'MARIA % ';
```

- Neste comando, todos os clientes cujos nomes iniciam com Maria serão retornados. Se quisermos retornar os nomes que contenham 'MARIA' também no meio, podemos alterar para o exemplo a seguir:

```
SELECT
  CODIGO,
  NOME
FROM
  CLIENTES
WHERE
  NOME LIKE ' % MARIA % ';
```

FILTRO DE TEXTO

- O uso de máscara no início e no fim da string fornece maior poder de busca, mas causa considerável perda de performance. Este recurso deve ser utilizado com critério.
- ***Uma observação**: em alguns bancos de dados, a máscara de filtro não é representada por %. Consulte a referência do banco para verificar o caractere correto.*
- Por padrão, a SQL diferencia caixa baixa de caixa alta. Para eliminar essa diferença, utiliza a função UPPER. Veja abaixo:

```
1  SELECT
2    CODIGO,
3    NOME
4  FROM
5    CLIENTES
6  WHERE
7    UPPER(NOME) LIKE 'MARIA % SILVA % ';
```

ORDENAÇÃO

- A ordenação pode ser definida com o comando ORDER BY. Assim como no comando **WHERE**, o campo de ordenação não precisa estar listado como campo de visualização

```
1  SELECT
2    CODIGO,
3    NOME
4  FROM
5    CLIENTES
6  ORDER BY
7    NOME;
8
9  SELECT
10   CODIGO,
11   NOME
12 FROM
13   CLIENTES
14 ORDER BY
15   UF,
16   NOME;
```

ORDENAÇÃO

- A utilização da palavra DESC garante a ordenação invertida:

```
1  SELECT
2    CODIGO,
3    NOME
4  FROM
5    CLIENTES
6  ORDER BY
7    NOME DESC;
8
9  SELECT
10   CODIGO,
11   NOME
12 FROM
13   CLIENTES
14 ORDER BY
15   UF DESC;
```