SQL

DML - Data Manipulation Language (Linguagem de Manipulação de dados) parte 2

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:

```
SELECT
  < lista_de_campos >
FROM
  < 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;
```

JUNÇÃO DE TABELAS

- O SELECT permite juntar duas ou mais tabelas no mesmo resultado. Isso pode ser feito de várias formas.

JUNÇÃO DE TABELAS

- Nesta linha as tabelas relacionadas CLIENTES e PEDIDOS são unificadas através do campo chave, em uma operação de igualdade. Repare que os nomes dos campos passam a ser prefixados pelo nome das tabelas, resolvendo duplicidades.

```
1  SELECT
2     CLIENTES.CODIGO,
3     CLIENTES.NOME,
4     PEDIDOS. DATA
5  FROM
6     CLIENTES,
7     PEDIDOS
8  WHERE
9     CLIENTES.CODIGO = PEDIDOS.CODCLIENTE
```

JUNÇÃO DE TABELAS

- O uso de aliases no código SQL torna a manutenção mais simples.

```
1  SELECT A.CODIGO, A.NOME, B.DATA, B.VALOR
2  FROM CLIENTES A, PEDIDOS B
3  WHERE A.CODIGO = B.CODCLIENTE
```

JUNÇÃO DE TABELAS

- No comando abaixo temos várias tabelas unificadas em uma mesma cláusula.

```
1  SELECT A.CODIGO, A.NOME, B.DATA, B.VALOR, C.QTD, D.DESCRIC
2  FROM CLIENTES A, PEDIDOS B, ITENS C, PRODUTOS D
3   WHERE A.CODIGO = B.CODCLIENTE
4   AND B.CODIGO = C.CODPEDIDO
5   AND C.CODPRODUTO = D.CODIGO
```

Comando JOIN

- Uma cláusula JOIN em SQL, correspondente a uma operação de junção em álgebra relacional, combina colunas de uma ou mais tabelas em um banco de dados relacional. Ela cria um conjunto que pode ser salvo como uma tabela ou usado da forma como está.

- Um JOIN é um meio de combinar colunas de uma (auto-junção) ou mais tabelas, usando valores comuns a cada uma delas. O SQL padrão ANSI especifica cinco tipos de JOIN: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN e CROSS JOIN.

Comando JOIN

- Como um caso especial, uma tabela (tabela base, visão ou tabela juntada) pode se juntar a si mesma em uma auto-união (self-join).

- Em um banco de dados relacional, os dados são distribuídos em várias tabelas lógicas. Para obter um conjunto completo e significativo de dados, é necessário consultar dados dessas tabelas usando junções (JOINs).

- Como já foi mencionado acima, existem diferentes tipos de JOINs no SQL. Vamos definir e exemplificar cada um deles.

INNER JOIN

A cláusula INNER JOIN compara cada linha da tabela **A** com as linhas da tabela **B** para encontrar todos os pares de linhas que satisfazem a condição de junção. Se a condição de junção for avaliado como **TRUE**, os valores da coluna das linhas correspondentes das tabelas **A** e **B** serão combinados em uma nova linha e incluídos no conjunto de resultados.

INNER JOIN

```
1   SELECT <select_list>
2   FROM Tabela A
3   INNER JOIN Tabela B
4   ON A.Key = B.Key
```

LEFT JOIN

Retorna todos os registros da tabela esquerda e os registros correspondentes da tabela direita.

Para cada linha da tabela A, a consulta a compara com todas as linhas da tabela B. Se um par de linhas fizer com que a condição de junção seja avaliado como TRUE, os valores da coluna dessas linhas serão combinados para formar uma nova linha que será incluída no conjunto de resultados.

Se uma linha da tabela “esquerda” A não tiver nenhuma linha correspondente da tabela “direita” B, a consulta irá combinar os valores da coluna da linha da tabela “esquerda” A com NULL para cada valor da coluna da tabela da “direita” B que não satisfaça a condição de junto (FALSE).

Em resumo, a cláusula LEFT JOIN retorna todas as linhas da tabela “esquerda” A e as linhas correspondentes ou valores NULL da tabela “esquerda” A.

LEFT JOIN

```
1  SELECT <select_list>
2  FROM Tabela A
3  LEFT JOIN Tabela B
4  ON A.Key = B.Key
```

RIGHT JOIN

Retorna todos os registros da tabela direita e os registros correspondentes da tabela esquerda.

A RIGHT JOIN combina dados de duas ou mais tabelas. A RIGHT JOIN começa a selecionar dados da tabela “direita” B e a corresponder às linhas da tabela “esquerda” A.

A RIGHT JOIN retorna um conjunto de resultados que inclui todas as linhas da tabela “direita” B, com ou sem linhas correspondentes na tabela “esquerda” A. Se uma linha na tabela direita B não tiver nenhuma linha correspondente da tabela “esquerda” A, a coluna da tabela “esquerda” A no conjunto de resultados será nula igualmente ao que acontece no LEFT JOIN.

RIGHT JOIN

```
1  SELECT <select_list>
2  FROM Tabela A
3  RIGHT JOIN Tabela B
4  ON A.Key = B.Key
```

CROSS JOIN

A cláusula CROSS JOIN retorna todas as linhas das tabelas por cruzamento, ou seja, para cada linha da tabela esquerda queremos todos os linhas da tabelas direita ou vice-versa. Ele também é chamado de produto cartesiano entre duas tabelas. Porém, para isso é preciso que ambas tenham o campo em comum, para que a ligação exista entre as duas tabelas.

Para entender melhor, pense que temos um banco de dado, onde temos uma tabela FUNCIONÁRIO e uma tabela CARGO, assim poderíamos ter vários cargos para um único FUNCIONÁRIO, e usando o CROSS JOIN podemos trazer todos os CARGOS de todos os FUNCIONÁRIOS.

CROSS JOIN

```
1   SELECT <select_list>
2   FROM Tabela A
3   CROSS JOIN Tabela B
```

SUB SELECT

Uma **Subquery** (também conhecida como **SUBCONSULTA** ou **SUBSELECT**) é uma instrução do tipo **SELECT** dentro de outra instrução **SQL**, que efetua consultas que, de outra forma, seriam extremamente complicadas ou impossíveis de serem feitas.

SUB SELECT

```sql
SELECT
    *
FROM
    tabela1 AS T
WHERE
    coluna1 IN
    (
        SELECT
            coluna2
        FROM
            tabela2 AS T2
      WHERE
            T.id = T2.id
    )
```

SUB SELECT

Como utilizar?

Existem algumas formas de utilizar **subqueries**. Neste artigo abordaremos os seguintes meios:

•**Subquery** como uma nova coluna da consulta (SELECT AS FIELD);

•**Subquery** como filtro de uma consulta (utilizando IN, EXISTS ou operadores de comparação);

•**Subquery** como fonte de dados de uma consulta principal (SELECT FROM SELECT).

Subquery como uma nova coluna da consulta

Uma das formas possíveis de realizar uma subquery é fazendo com que o resultado de outra consulta seja uma coluna dentro da sua consulta principal.

Como exemplo buscaremos o título de todos os projetos cadastrados e adicionaremos uma coluna com a quantidade de comentários existentes em cada projeto, realizando assim uma consulta principal na tabela projetos e uma subconsulta na tabela comentarios, que gerará uma nova coluna.

Subquery como uma nova coluna da consulta

titulo	Quantidade_Comentarios
Aplicação C#	2
Aplicação Ionic	3
Aplicação Python	2

```
SELECT
    P.titulo,
    (SELECT
        COUNT(C.id_projeto)
      FROM
        comentarios C
      WHERE
        C.id_projeto = P.id ) AS Quantidade_Comentarios
FROM
    projetos P
GROUP BY
    P.id
```

Subquery como filtro de uma nova consulta

Outro exemplo da utilização de subqueries é fazendo filtros no resultado de outras consultas. Para esse modelo podemos utilizar as cláusulas **IN**, **EXISTS** ou operadores de comparação, como **=**, **>=**, **<=**, dentre outros.

Para exemplificar buscaremos todos os projetos que possuam algum comentário, ou seja, uma consulta principal na tabela projetos, e filtrar o resultado com base no resultado da subconsulta na tabela comentários.

Subquery como filtro de uma nova consulta

id	titulo	data
1	Aplicação C#	2018-04-01
2	Aplicação Ionic	2018-05-07
3	Aplicação Python	2018-08-05

```
SELECT
    P.id,
    P.titulo,
    P.data
FROM
    projetos P
WHERE
    P.id IN
    (
        SELECT
            C.id_projeto
        FROM
            comentarios C
        WHERE
            P.id = C.id_projeto
    );
```