Linguagem SQL

Módulo 1

Conhecendo a linguagem SQL

Por: Igor Coelho

NASAJON SISTEMAS

HISTÓRIA DO SQL

INTRODUÇÃO:

SQL (Structure Query Language) – é a linguagem comumente utilizada para criar, modificar, recuperar e manipular dados de sistemas gerenciadores de bancos de dados relacionais (SGBD).

Ela foi criada no ano de 1970 pela IBM inicialmente para a recuperação e manipulação de dados do sistema de banco de dados **System R** e em primeiro momento foi chamada de **SEQUEL**.

Posteriormente foi abreviada para **SQL** pois a palavra SEQUEL já era marca registrada de uma companhia aérea inglesa da época.

Após algum tempo, a IBM em paralelo a outra companhia, a Relational Software (hoje **Oracle**), visualizou o potencial comercial dos conceitos do SQL e lançaram no verão de 1979 o primeiro banco de dados relacional totalmente baseado no SQL, o **Oracle 2**.

Pelo grande sucesso que teve, o SQL foi adotado como padrão **ANSI** em 1986 e **ISO** em 1987.

ESTRUTURA DA LINGUAGEM SQL

Apesar de definida pelos padrões ANSI e ISO, existem muitas variações da linguagem, como o PL/SQL da Oracle, SQL PL da IBM e T-SQL da Microsoft.

Todas seguem as linhas base definidas pelo padrão ANSI, mas implementam características próprias, como tipos de dados e funções específicas.

O SQL foi planejado para atender a um propósito específico: consultar e manipular dados contidos em bancos de dados relacionais. Por isso, o SQL “puro” não é uma linguagem de programação clássica. Mas por outro lado, não deixa de ser uma linguagem de programação, por ter uma sintaxe específica e o objetivo de ser programável.

A linguagem SQL é dividida em subconjuntos com objetivos distintos. Focaremos o treinamento em dois destes subconjuntos:

- **DDL – Data Definition Language**
- **DML – Data Manipulation Language**

DDL E DML

DEFINIÇÕES DA ESTRUTURA

DDL – Data Definition Language
É o subconjunto de comandos que serve para a definição do banco de dados com as funções de manipulação de estrutura das tabelas. Seus principais comandos são:

- **CREATE** (cria banco de dados, tabelas, índices, etc...);
- **ALTER** (altera a estrutura de uma tabela);
- **DROP** (apaga uma tabela ou um índice existente no banco de dados).

DML – Data Manipulation Language
É o subconjunto de comandos responsável por promover a manutenção dos dados, como inserção, alteração, exclusão e recuperação dos dados. Os principais comandos são:

- **INSERT** (usado para inserir dados em uma tabela do banco de dados);
- **UPDATE** (usado para modificar valores em um campo ou conjunto de linhas);
- **DELETE** (remove linhas existentes no banco de dados);
- **SELECT** (seleciona e exibe dados em uma ou mais tabelas).

NASAJON SISTEMAS

CRIANDO UM BANCO DE DADOS E SUAS TABELAS

Vamos iniciar pela criação de um banco de dados com o nome **curso_sql**.

Abriremos o editor de scripts do SGDB que será utilizado e escreveremos o comando abaixo:

CREATE DATABASE curso_sql;

Antes de prosseguir, vamos entender o comando acima:

CREATE - informa para o SGDB que algo será criado.
DATABASE - define o objeto que será criado no SGDB.
curso_sql - nome dado ao banco de dados que será criado (não deve conter espaços)

Após executar o comando acima podemos reparar que o banco de dados **curso_sql foi** criado**.**

CRIANDO UM BANCO DE DADOS E SUAS TABELAS

Após a criação do banco de dados, vamos criar as tabelas que iremos trabalhar. Para isto veja os comandos abaixo:

```
CREATE TABLE CLIENTES
(
codcliente INT not null,
nome VARCHAR(70) not null,
documento VARCHAR(14),
endereco VARCHAR(70),
bairro VARCHAR(30),
cidade VARCHAR(30),
uf VARCHAR(2),
cep VARCHAR(9),
PRIMARY KEY (codcliente)
);
```

```
CREATE TABLE PEDIDOS
(
codpedido INT not null,
codcliente INT not null,
datapedido DATE not null,
tipopedido INT,
valor NUMERIC(9,2),
PRIMARY KEY (codpedido),
FOREIGN KEY (codcliente) REFERENCES CLIENTES
);
```

CRIANDO UM BANCO DE DADOS E SUAS TABELAS

Nos comandos do slide anterior, temos alguns termos que precisaremos entender:

INT VARCHAR(X) DATE NUMERIC(X, Z)	São definições de tipos de dados para cada campo a ser criado. Ex.: cidade VARCHAR(30) - onde significa que será criado o campo cidade que aceitará preenchimento alfanumérico de até 30 dígitos.
NOT NULL	Informa que o preenchimento do campo será obrigatório.
PRIMARY KEY	Cria um identificador único para cada linha da tabela.
FOREIGN KEY	Indica o campo que relaciona uma tabela com a outra.

ALTERANDO TABELAS EXISTENTES

Podemos realizar alterações em tabelas. Vamos exemplificar a Adição e Remoção de campos.

Adicionando um campo na tabela:

ALTER TABLE PEDIDOS **ADD COLUMN** situacao VARCHAR(20) Default 'Aberto';

OBS.: o comando **default** define um valor padrão para a coluna mesmo que ela não receba nenhuma entrada de dados.

Removendo um campo da tabela:

ALTER TABLE PEDIDOS **DROP COLUMN** tipopedido;

OBS: Podemos apagar também uma tabela inteira do banco de dados. Para isso, basta executar o comando **DROP TABLE + nome_da_tabela**. Devemos ter cuidado ao executar este comando, pois ele excluirá toda a tabela juntamente com os seus dados.

D
M
L

INSERINDO DADOS NAS TABELAS EXISTENTES

Agora que já temos a estrutura do nosso banco de dados montada, vamos aprender como inserir dados. Acompanhe os scripts abaixo:

INSERT INTO CLIENTES
(codcliente, nome, documento, endereco, bairro, cidade, uf, cep)
VALUES
(1, 'João Silva', '11436468236', 'Av. Rio Branco 45', 'Centro', 'Rio de Janeiro', 'RJ', '20003-009');

INSERT INTO PEDIDOS
(codpedido, codcliente, datapedido, valor)
VALUES
(10001, 1, '2018-11-30', 275.50);

OBS: Valores em string ou data devem conter aspas antes e depois do valor, e os valores numéricos devem ser passados diretamente sem as aspas. Para os valores do tipo data, devem ser informados na seguinte ordem: Ano-Mês-Dia. Por fim, os valores numéricos reais devem ser informados com ponto ao invés de vírgula para separar os decimais.

Clique **aqui** e efetue o download do script de inclusão de dados!

NASAJON SISTEMAS

ALTERANDO DADOS NAS TABELAS

Para realizar a alteração/atualização de dados já gravados nas tabelas devemos utilizar o comando **UPDATE**. Acompanhe os scripts abaixo:

UPDATE CLIENTES **SET** UF = 'SP';

UPDATE PEDIDOS **SET** VALOR = 500;

OBS: Nos exemplos acima, o comando UPDATE atualizará a UF de todos os clientes para o SP, e o valor de todos os pedidos para 500.

Na maioria das vezes, isto não é o esperado, ou seja, queremos atualizar valores dentro de um determinado conjunto específico de dados. Para isto devemos utilizar a cláusula WHERE que veremos no próximo slide.

Pergunta: *Como atualizar dois ou mais valores no mesmo comando em SQL?*

UTILIZANDO A CLÁUSULA WHERE

A cláusula **WHERE** é utilizada para limitar um conjunto de dados, seja em uma pesquisa, alteração ou exclusão de dados. Ela deve ser inserida logo após o comando SQL da operação desejada.

Aproveitando os exemplos de atualização do slide anterior, vamos reescrevê-los de forma que a alteração aconteça somente para um determinado conjunto. Acompanhe:

UPDATE CLIENTES **SET** UF = 'SP' **WHERE** codcliente = 1;

UPDATE PEDIDOS **SET** VALOR = 500 **WHERE** codpedido = 10001;

Veja que agora as alterações somente serão feitas nos registros que atenderem a limitação da cláusula WHERE, ou seja, a UF alterada será somente a do cliente com o código 1 e somente o pedido de código 10001 terá seu valor alterado para 500.

Pergunta: *Como colocar mais de uma condição na cláusula WHERE?*

EXCLUINDO DADOS DAS TABELAS

Para realizar a exclusão de uma ou mais linhas da tabela, utilizamos o comando **DELETE**.

Acompanhe os exemplos abaixo:

DELETE FROM CLIENTES **WHERE** codcliente = 99;

DELETE FROM PEDIDOS **WHERE** valor = 275.50;

Observe que a exclusão acontecerá somente se os registros satisfizerem a condição da cláusula WHERE. Se for necessário excluir ‘todos’ os registros da tabela, basta não utilizar a cláusula WHERE, mas devemos ter muito cuidado neste caso.

OBS.: Caso a linha a ser excluída possua um dado vinculado a outra tabela, o SGBD irá exibir um alerta e não permitirá a exclusão.

RECUPERANDO DADOS - O INCRÍVEL "SELECT"

Como já foi dito anteriormente, o SQL foi planejado com o objetivo de permitir a consulta de dados contidos em tabelas de banco de dados relacionais.

Por isso o comando **SELECT** é o mais importante e elaborado da linguagem SQL.

A sintaxe básica é:

SELECT * **FROM** CLIENTES;

Quando utilizado o *, todos os campos da tabela serão exibidos. Se desejarmos exibir somente alguns campos a sintaxe será:

SELECT campo1, campo2, campo3 **FROM** CLIENTES;

RECUPERANDO DADOS – ORDENANDO O RESULTADO

No slide anterior, a consulta retornará uma listagem com todos os registros da tabela, mas, geralmente precisamos exibir os dados de uma forma que seja de fácil entendimento.

Para isto, podemos utilizar o comando **ORDER BY**, que irá exibir os dados ordenando alfabeticamente conforme o campo apontado.

Veja os exemplos abaixo:

SELECT * **FROM** CLIENTES **ORDER BY** nome;

SELECT * **FROM** PEDIDOS **ORDER BY** datapedido;

OBS.: Se precisamos apresentar os dados com a ordenação inversa, basta incluir a palavra DESC após o campo da ordenação:

SELECT * **FROM** PEDIDOS **ORDER BY** datapedido **DESC**;

Pergunta: *Posso ordenar uma consulta por mais de um campo da tabela?*

RECUPERANDO DADOS - DEFININDO CONDIÇÕES

Normalmente quando realizamos uma consulta, temos a necessidade de aplicar algumas condições. Sendo assim, para retornar dados das tabelas com o **SELECT**, podemos aplicar condições utilizando em conjunto a cláusula **WHERE**.

Acompanhe alguns exemplos:

SELECT * **FROM** CLIENTES **WHERE** uf = 'SP';

SELECT * **FROM** PEDIDOS **WHERE** codcliente = 5;

Utilizando os conceitos do slide anterior, podemos gerar uma consulta, aplicando uma condição e ordenando o resultado. Vejamos o exemplo:

SELECT * **FROM** CLIENTES **WHERE** uf = 'SP' **ORDER BY** nome;

A consulta acima retornará os clientes da uf SP ordenados alfabeticamente pelo nome.

RECUPERANDO DADOS - OPERADORES DE COMPARAÇÃO

Os operadores de comparação são muito utilizados quando definimos condições em uma consulta.

Os mais comuns são:

=	Igual
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual

<> ou !=	Diferente ou não igual
LIKE ou ILIKE	Como ou Parecido
BETWEEN	Entre
IN	Na lista
NOT IN	Não está na lista

RECUPERANDO DADOS - OPERADORES AND E OR

Operador AND (E)
Somente retornará os dados quando todas as condições forem satisfeitas.

Exemplo:

SELECT * **FROM** clientes **WHERE** uf = 'RJ' **AND** cidade = 'Rio de Janeiro';

Operador OR (OU)
Retornará os dados quando qualquer uma das condições for satisfeita.

Exemplo:

SELECT * **FROM** clientes **WHERE** uf = 'SP' **OR** cidade = 'Barueri';

RECUPERANDO DADOS - DISTINCT

A cláusula **DISTINCT** deve ser utilizada quando desejamos eliminar valores iguais no retorno de uma consulta (por padrão compara todas as colunas retornadas). Ela deve ser colocada logo após o SELECT, antes da declaração dos campos a serem retornados.

Exemplo:

SELECT DISTINCT cidade **FROM** clientes **ORDER BY** cidade;

O retorno da query acima irá trazer todas as cidades em que existem clientes cadastrados, em ordem alfabética, e sem repetições. Caso o DISTINCT não seja utilizado, poderá ocorrer de existirem vários clientes com a mesma cidade.

Podemos ainda utilizar o DISTINCT para eliminar valores repetidos considerando, somente uma, ou algumas colunas a serem retornadas. Com isso a sintaxe muda um pouco:

SELECT DISTINCT ON (UF) cidade **FROM** clientes **ORDER BY** uf, cidade;

RECUPERANDO DADOS - OPERADORES DE AGREGAÇÃO

Uma função de agregação processa um conjunto de valores contidos em uma única coluna de uma tabela e retorna um único valor como resultado.

Estudaremos os principais:

SELECT COUNT(*) **FROM** pedidos; – retorna a **quantidade** de pedidos

SELECT SUM (valor) **FROM** pedidos; – retorna a **soma** dos valores

SELECT AVG (valor) **FROM** pedidos; – retorna a **média** dos valores

SELECT MAX (valor) **FROM** pedidos; – retorna o **maior** valor

SELECT MIN (valor) **FROM** pedidos; – retorna o **menor** valor

RECUPERANDO DADOS – GROUP BY

O comando **GROUP BY** permite que a apresentação dos dados retornados da consulta seja agrupada de acordo com uma determinada condição. É sempre utilizado em conjunto com o comando **SELECT** e um operador de agregação.

SELECT SUM(valor), codcliente **FROM** pedidos **GROUP BY** codcliente;
- Retornará a soma dos pedidos de cada cliente.

SELECT COUNT(codcliente), codcliente **FROM** pedidos **GROUP BY** codcliente;
- Retornará a quantidade de pedidos de cada cliente.

SELECT AVG(valor), codcliente **FROM** pedidos **GROUP BY** codcliente;
- Retornará a média de valor dos pedidos para cada cliente.

SELECT MAX(valor), codcliente **FROM** pedidos **GROUP BY** codcliente;
- Retornará o maior valor de pedido para cada cliente.

SELECT MIN(valor), codcliente **FROM** pedidos **GROUP BY** codcliente;
- Retornará o menor valor de pedido para cada cliente.

RECUPERANDO DADOS - HAVING

Usamos a cláusula **HAVING** em conjunto com **GROUP BY** para filtrar os resultado que serão submetidos a agregação dentro de um **SELECT**.

Veja o exemplo:

SELECT COUNT(codcliente), codcliente **FROM** pedidos
GROUP BY codcliente **HAVING COUNT**(codcliente) > 3;

- Retornará o cliente e a sua quantidade de pedidos, quando a quantidade for maior que 3.

SELECT AVG(valor), codcliente **FROM** pedidos
GROUP BY codcliente **HAVING AVG**(valor) <= 300;

- Retornará o cliente e sua média de valor dos pedidos, quando a média for menor ou igual a 300.

RECUPERANDO DADOS - INTERVALOS DEFINIDOS (1)

Operador LIKE (Parecido ou Como)

Utilizamos o operador **LIKE** quando precisamos aplicar uma condição buscando dentro de um texto.

SELECT * FROM clientes **WHERE** bairro **LIKE** '%o**';**
O símbolo % é utilizado para definir o intervalo em que a busca irá ocorrer dentro o texto. No exemplo, serão exibidos os clientes cujo bairro terminar com a letra **O.**

Operador BETWEEN (Entre)

Utilizado para definir intervalos de número ou datas.

SELECT * **FROM** pedidos **WHERE** datapedido **BETWEEN** '2018-12-01' **AND** '2018-12-31';
O exemplo acima exibirá todos os pedidos cuja data esteja em Dezembro de 2018.

RECUPERANDO DADOS - INTERVALOS DEFINIDOS (2)

Operador IN (Na lista)

Utilizado em comparações com uma lista de valores pré definida.

SELECT * **FROM** clientes **WHERE** uf **IN** ('RJ', 'SP');
Serão exibidos os clientes cuja uf seja RJ ou SP.

Operador NOT IN (Não está na lista)

Utilizado em comparações para **negar** uma lista de valores pré definida.

SELECT * **FROM** pedidos **WHERE** codcliente **NOT IN** (2, 3, 4, 5);
Serão exibidos os pedidos cujo codcliente não seja 2, 3, 4 ou 5.

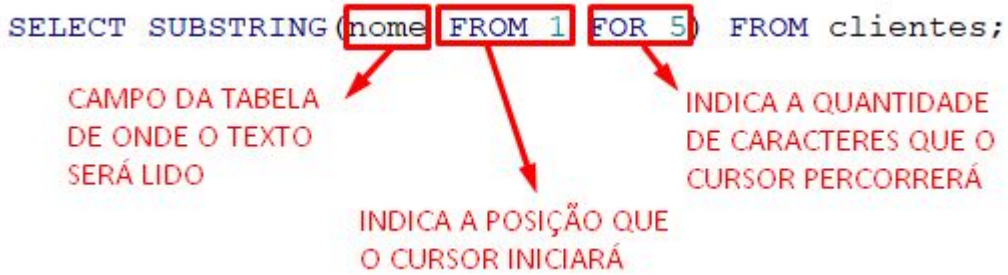
RECUPERANDO DADOS - SUBSTRINGS

O comando **SUBSTRING** permite retornar um campo de texto considerando uma determinada posição, ou seja, podemos apresentar parte do texto contido em um campo.

SELECT SUBSTRING(nome **FROM** 1 **FOR** 5) **FROM** clientes;

No exemplo acima, serão retornados os nomes dos clientes exibindo o texto da primeira para a quinta posição.

Explicação:



FIXANDO O CONHECIMENTO

LISTA DE EXERCÍCIOS - Parte 1

1 - FAÇA UMA QUERY QUE RETORNE TODOS OS CLIENTES DO ESTADO DO RIO DE JANEIRO.

2 - FAÇA UMA QUERY QUE RETORNE OS CLIENTES COM EXCEÇÃO DOS RESIDENTES DO ESTADO DO ESPÍRITO SANTO.

3 - FAÇA UMA QUERY QUE RETORNE TODOS OS PEDIDOS COM VALORES ENTRE 100 E 200.

4 - INSIRA UM CLIENTE DO ESTADO DE MINAS GERAIS NO BANCO DE DADOS.

5 - FAÇA UMA QUERY QUE RETORNE DE FORMA ORDENADA PELA DATA, TODOS OS PEDIDOS CUJO VALOR SEJA MENOR OU IGUAL A 100.

6 - FAÇA UMA QUERY QUE ATUALIZE EM 10% O VALOR DE TODOS OS PEDIDOS QUE POSSUEM O VALOR MAIOR QUE 200.

NASAJON SISTEMAS

FIXANDO O CONHECIMENTO

LISTA DE EXERCÍCIOS - Parte 2

7 - FAÇA UMA QUERY QUE RETORNE O VALOR MÁXIMO, VALOR MÍNIMO, VALOR MÉDIO E SOMA DOS VALORES DOS PEDIDOS DE CADA CLIENTE.

8 - INSIRA 2 PEDIDOS COM DATAS E VALORES DIFERENTES PARA O CLIENTE DO ESTADO DE MG CRIADO NO EXERCÍCIO 4.

9 - FAÇA UMA QUERY QUE RETORNE A UF E A QUANTIDADE DE CLIENTES CADASTRADOS NESTA UF.

10 - FAÇA UMA QUERY QUE RETORNE A QUANTIDADE DE PEDIDOS DE CADA CLIENTE, E ORDENE O RETORNO PELO CLIENTE.

11 - FAÇA UMA QUERY QUE EXIBA O NOME DOS CLIENTES DO ESTADO DO RIO DE JANEIRO E AS 3 PRIMEIRAS LETRAS DA CIDADE.

12 - FAÇA UMA QUERY QUE EXIBA O NOME E ENDEREÇO DOS CLIENTES QUE POSSUAM A PALAVRA 'Rua' NO ENDEREÇO.

NASAJON SISTEMAS

FIXANDO O CONHECIMENTO

LISTA DE EXERCÍCIOS - CORREÇÃO - Parte 1

1 - FAÇA UMA QUERY QUE RETORNE TODOS OS CLIENTES DO ESTADO DO RIO DE JANEIRO.
SELECT * FROM CLIENTES WHERE UF = 'RJ';

2 - FAÇA UMA QUERY QUE RETORNE OS CLIENTES COM EXCEÇÃO DOS RESIDENTES DO ESTADO DO ESPÍRITO SANTO.
SELECT * FROM CLIENTES WHERE UF <> 'ES';

3 - FAÇA UMA QUERY QUE RETORNE TODOS OS PEDIDOS COM VALORES ENTRE 100 E 200.
SELECT * FROM PEDIDOS WHERE VALOR BETWEEN 100 AND 200;

4 - INSIRA UM CLIENTE DO ESTADO DE MINAS GERAIS NO BANCO DE DADOS.
INSERT INTO CLIENTES (codcliente, nome, documento, endereco, bairro, cidade, uf, cep)
VALUES (6, 'Maria Silva', '88888888888', 'Largo Santa Rita, 51', 'Centro', 'Cataguases', 'MG', '20003-009');

NASAJON SISTEMAS

FIXANDO O CONHECIMENTO

LISTA DE EXERCÍCIOS - CORREÇÃO - Parte 2

5 - FAÇA UMA QUERY QUE RETORNE DE FORMA ORDENADA PELA DATA, TODOS OS PEDIDOS CUJO VALOR SEJA MENOR OU IGUAL A 100.

SELECT * FROM PEDIDOS WHERE VALOR <= 100 ORDER BY DATAPEDIDO;

6 - FAÇA UMA QUERY QUE ATUALIZE EM 10% O VALOR DE TODOS OS PEDIDOS QUE POSSUEM O VALOR MAIOR QUE 200.

UPDATE PEDIDOS SET VALOR = VALOR*1.1 WHERE VALOR > 200;

7 - FAÇA UMA QUERY QUE RETORNE O VALOR MÁXIMO, VALOR MÍNIMO, VALOR MÉDIO E SOMA DOS VALORES DOS PEDIDOS DE CADA CLIENTE.

SELECT MAX(VALOR), MIN(VALOR), AVG(VALOR), SUM(VALOR), CODCLIENTE FROM PEDIDOS GROUP BY CODCLIENTE

8 - INSIRA 2 PEDIDOS COM DATAS E VALORES DIFERENTES PARA O CLIENTE DO ESTADO DE MG CRIADO NO EXERCÍCIO 4.

INSERT INTO PEDIDOS (codpedido, codcliente, datapedido, valor)
VALUES (10017, 6, '2018-11-30', 210);
INSERT INTO PEDIDOS (codpedido, codcliente, datapedido, valor)
VALUES (10018, 6, '2018-12-02', 85.25);

NASAJON SISTEMAS

FIXANDO O CONHECIMENTO

LISTA DE EXERCÍCIOS - CORREÇÃO - Parte 3

9 - FAÇA UMA QUERY QUE RETORNE A UF E A QUANTIDADE DE CLIENTES CADASTRADOS NESTA UF.
SELECT COUNT(*), UF FROM CLIENTES GROUP BY UF;

10 - FAÇA UMA QUERY QUE RETORNE A QUANTIDADE DE PEDIDOS DE CADA CLIENTE, E ORDENE O RETORNO PELO CLIENTE.
SELECT COUNT(*), CODCLIENTE FROM PEDIDOS GROUP BY CODCLIENTE ORDER BY CODCLIENTE;

11 - FAÇA UMA QUERY QUE EXIBA O NOME DOS CLIENTES DO ESTADO DO RIO DE JANEIRO E AS 3 PRIMEIRAS LETRAS DA CIDADE.
SELECT NOME, SUBSTRING(CIDADE FROM 1 FOR 3) FROM CLIENTES WHERE UF = 'RJ';

12 - FAÇA UMA QUERY QUE EXIBA O NOME E ENDEREÇO DOS CLIENTES QUE POSSUEM A PALAVRA 'Rua' NO ENDEREÇO.
SELECT NOME, ENDERECO FROM CLIENTES WHERE ENDERECO LIKE 'Rua%';

NASAJON SISTEMAS

OBRIGADO!

FIM!

NASAJON SISTEMAS