

# Readme.rmd

Sergio Pedro R Oliveira

2022-03-06

## **Objetivo**

Estudo dirigido de SQL, utilizando SQLite.

## **Livro de referência**

Introdução a linguagem SQL - abordagem pratica para iniciantes

# Assuntos por capítulos e resumos

## Capítulo 4

### SELECT:

- Extrai dados de uma tabela e exibe os resultados.
- Uso do (\*) para especificar todas as colunas.
- Uso do **AS** para criar nova coluna, também serve para mudar nome de coluna, na consulta.
- Uso da função *round()* para arredondamentos.
- Uso da função *coalesce()* para alterar o valor NULL de determinada coluna para outro valor estabelecido. Usado em conjunto com o **AS** para trocar o nome da coluna, na consulta.

Obs.: na expressão o uso do ponto para representar o numero decimal.

### Operadores matematicos:

##	Operador	Descrição
## 1	+	soma
## 2	-	subtração
## 3	*	multiplicação
## 4	/	divisão
## 5	%	resto da divisão

### Concatenação de textos:

- Mescla dois ou mais dados.
- O operador de concatenação é especificado por um **pipe duplo** (||).
- Após a mesclagem de dados o retorno é no dado tipo texto.

Obs.: no MySQL a função que faz concatenação é **CONCAT()**.

## Capítulo 5

### WHERE:

- **Filtro** de dados(registros) para consulta.
  - Consultas através de criterios **matematicos**.
  - Consultas através de criterios em formato **texto**.
- Uso da função *length* em conjunto com **WHERE**, função para determinar o numero de caracteres.
- Uso do **BETWEEN** para filtragem inclusiva de dados, buscar dados entre valores.
- Uso da expressão **LIKE**, para utilização de caracteres curingas na utilização de filtros.
- Uso de operadores logicos para auxiliar na filtragem de dados:
  - *OR*  
Uso de mais de um criterio para filtragem.
  - *AND*  
Criterios bem definidos

tabela verdade:

##	p	NOT_p	q	NOT_q	p_AND_q	p_OR_q
## 1	V	F	V	F	V	V
## 2	V	F	F	V	F	V
## 3	F	V	V	F	F	V
## 4	F	V	F	V	F	F

- Uso de **listas**:
  - *IN*  
fornece uma lista validade valores como criterio de filtragem.
  - *NOT IN*  
Todos os dados, exceto os fornecidos pela lista.

##	Operador	
## 1	AND	
## 2	OR	
## 3	BETWEEN	
## 4	IN	
## 5	NOT	
## 6	IS NULL	
## 7	IS NOT NULL	
##		Descricao_op_logc
## 1		Verifica se todas as expressões booleanas são verdadeiras

```

## 2          Verifica se alguma expressão booleana é verdadeira
## 3 Verifica se um valor se encaixa inclusivamente dentro de um intervalo
## 4          Verifica se um valor existe dentro de uma lista de valores
## 5          Nega e inverte o valor em uma expressão booleana
## 6          Verifica se um valor é nulo
## 7          Verifica se um valor não é nulo
##          Exemplo
## 1          x AND y
## 2          x OR y
## 3 a BETWEEN x AND y
## 4          a IN (x,y,w,z)
## 5 a NOT IN (x,y,w,z)
## 6          a IS NULL
## 7          a IS NOT NULL

```

- uso de *booleanos* no filtro, em conjunto com NOT para transformar um true em false (1 -> 0).

– **true** = 1.

– **false** = 0.

obs.: SQLite só aceita 1 e 0. MySQL aceita true e false.

- Tratamento de NULL, valor nulo.
  - funções para trabalhar com NULL:
    - \* **IS NULL**  
Filtrar valores NULL.
    - \* **IS NOT NULL**  
Filtrar valores não NULL.
    - \* **IS NULL OR**  
Adiciona NULL a filtragem, junto de outros critérios.
    - \* *coalesce*  
Transforma valores NULL em outra coisa.

Obs.: em situação normal, o valor NULL é ignorado pelos filtros matemáticos, se não especificado.

## Capítulo 6

### GROUP BY e ORDER BY

Agragação de dados, também conhecido como totalização, resumo ou agrupamento.

#### GROUP BY

- Agrupamento de registros.
- É comum ser usado em conjunto com **WHERE** para selecionar dados.
- Normalmente é usado com conjunto com funções típicas de sumarização (resumo), como:

```
##          Funcao
## 1          avg(X)
## 2        count(X)
## 3        count(*)
## 4          max(X)
## 5          min(X)
## 6          sum(X)
## 7 group_concat(X)
##
##                               Descricao_func_tipica_groupby
## 1 Calcula a media de todos os valores da coluna X (Omite valores nulos)
## 2                               Contao o numero de valore não nulos da coluna X
## 3                               Conta o numero registros
## 4          Encontra o valor maximo da coluna X (Omite valores nulos)
## 5          Encontra o valor minimo da coluna X (Omite valores nulos)
## 6          Calcula a soma dos valores da coluna X (Omite valores nulos)
## 7                               Concatena os valores não nulos da coluna X.**
```

Obs.: Você também pode fornecer um segundo argumento que especifica um separador, como a virgula.

- Existem duas formas possiveis de escrever os argumentos de GROUP BY\*\*:

1. Escrevendo o nome das colunas especificadas em **SELECT**.
2. Dando o numero da ordem das colunas que aparecem especificadas em **SELECT**.  
Essa segunda forma não funciona no Oracle e no SQL Server.

#### ORDER BY

- Ordenando registros.
- Por padrão a instrução ORDER BY organiza por ordem crescente os registros.
- Operadores **ORDER BY**:
  1. **ASC**  
Organiza os registros. em ordem crescente

## 2. DESC

Organiza os registros em ordem decrescente.

## HAVING

- Filtra registros de acordo com um valor agregado.
- Substitui o **WHERE** para filtrar valores agregados por **GROUP BY**.
- Sintaxe no Oracle é ligeiramente diferente, é preciso especificar a função de agregação ao usar o **HAVING**.  
ex.: **HAVING SUM(precipitation) > 30**

## DISTINCT

- Instrução para obter registros distintos, sem duplicatas, sem valores repetidos.

## Capítulo 7

### CASE

- Esse comando nos permite substituir o valor de uma coluna por outro valor, de acordo com uma ou mais condições.
- Equivalente ao **IF**, **ELIF**, **ELSE** de outras linguagens.
- Sintaxe do **CASE**:  
**CASE**  
**WHEN** (*condição*) **THEN** (*valor1*)  
**ELSE** (*valor2*)  
**END AS** (*nome da nova coluna*)

### Truque CASE ZERO/NULL

- Onde é possível colocar a instrução CASE dentro de uma função de agregação, substituindo assim o uso do **WHERE**.
- Aplicando assim mais de um filtro distinto na mesma pesquisa.
- Sintaxe:  
**SUM(CASE WHEN** (*condição*) **THEN** (*valor1*) **ELSE** (*valor2*) **END**) **AS** (*nome da nova coluna*)
- É possível dentro da *condição* fazer uso de operadores lógicos:
  - **OR**
  - **AND**
  - **NOT**

## Capítulo 8

### JOIN

#### Banco de dados relacional

- Duas ou mais tabelas se relacionam (relacionais) determinado campo de uma tabela aponta para o campo de outra tabela.
- Colunas *Chave* são as colunas que interligam as tabelas, contem valores unicos que guardam identificações que não vão se repetir, identificadores de determinado objeto.
- Dizemos que uma tabela é pai da outra quando a segunda tabela depende de informações da primeira tabela. a primeira tabela é pai e a segunda tabela é filha.
- Tipos de relacionamento entre tabela-pai e tabela-filha:
  - *Um para muitos.* (a mais comum)  
**Um** registro da tabela-pai pode estar associado a **diversos** registros da tabela-filha.
  - *Um para um.*  
**Um** registro da tabela-pai pode estar associado a **um** registro da tabela-filha.
  - *Muitos para muitos.*  
**Diversos** registros da tabela-pai podem estar associados a **diversos** registros da tabela-filha.

#### INNER JOIN

- Une duas tabelas, relacionadas, para efetuar consultas mais eficientes.
- A mescla é feita apartir de algum campo comum, para que os registros se alinhem, colunas *chave*.
- Sintaxe:  
**SELECT** (colunas consultadas das duas tabelas), *tabela-pai.coluna\_chave*  
**FROM** *tabela-pai* **INNER JOIN** *tabela-filha*  
**ON** *tabela-pai.coluna\_chave* = *tabela-filha.coluna\_chave*;
- Obs.:
  - No **SELECT** é preciso selecionar a *coluna\_chave*, tanto faz se for da tabela-pai ou filha.
  - É dentro do **FROM** que é executado o **JOIN INNER**.
  - Quanto a exibição dos resultados, só é exibido registros que existam nas duas tabelas.
  - Caso queiramos incluir consultas que mostrem todos os registros, mesmo os que só existam em uma tabela, podemos usar **LEFT JOIN**.

#### LEFT JOIN

##### 1. LEFT JOIN



- Mescla duas tabelas, uma há esquerda.
- Diferente do **INNER JOIN**, não omite registros. Registros sem sem associação entre as tabelas recebe valor **NULL**.
- Sintaxe:  
**SELECT** (colunas consultadas das duas tabelas), *tabela-pai.coluna\_chave*  
**FROM** *tabela-pai(A ESQUERDA)* **LEFT JOIN** *tabela-filha(A DIREITA)*  
**ON** *tabela-pai.coluna\_chave = tabela-filha.coluna\_chave*;

## 2. **LEFT JOIN + WHERE NULL**

- Pode ser usado em conjunto com filtro **WHERE** procurando valores **NULL** para achar registros sem relação entre tabelas.  
ex.: pedidos sem cliente ou clientes sem pedidos.
- Sintaxe:  
**SELECT** (colunas consultadas das duas tabelas), *tabela-pai.coluna\_chave*  
**FROM** *tabela-pai(A ESQUERDA)* **LEFT JOIN** *tabela-filha(A DIREITA)*  
**ON** *tabela-pai.coluna\_chave = tabela-filha.coluna\_chave*;  
**WHERE** (*coluna\_procurada* ou *coluna\_chave*) = **NULL**

## Andamento dos Estudos

Estudando instrução **JOIN** - operadores **JOIN**.

**Assunto em andamento:**

**Em andamento:**

**Vazios:**

**Finalizando detalhes:**