Readme.rmd

Sergio Pedro R Oliveira

2022-03-08

Objetivo

Estudo dirigido de SQL, utilizando SQLite.

Livro de referência

Introdução a linguagem SQL - abordagem pratica para iniciantes

Assuntos por capitulos 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, tambem 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 atraves de criterios **matematicos**.
  - Consultas atraves 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 auxilar 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**
      Filtra valores NULL.
    - **IS NOT NULL**
      Filtra valores não NULL.
    - **IS NULL OR**
      Adiciona NULL a filtragem, junto de outros criterios.
    - *coalesce*
      Transforma valores NULL em outra coisa.

Obs.: em situação normal, o valor NULL é ignorado pelos filtros matematicos, 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 tipicas 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 possivel 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 possivel dentro da *condição* fazer uso de operadores logicos:

  - **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.
   - Mantem todos os registros da tabela a esquerda.

- Diferente do **INNER JOIN**, não omite registros. Registros 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**

Outros tipos de operador JOIN Esses outros operadores não tem suporte no SQLite, porem tem nos outros banco de dados.

1. **RIGHT JOIN** - Mescla duas tabelas, uma há direita.
- Mantem todos os registros da tabela da direita.
- Diferente do **INNER JOIN**, não omite registros. Registros sem associação entre as tabelas recebe valor **NULL**.

2. **OUTER JOIN** - **OUTER JOIN** é um operador de associação externa completa.
- Inclui todos os registros das duas tabelas.
- Executa o **LEFT JOIN** e o **RIGHT JOIN** simultanemente.
- Busca registros orfãs nas duas direções.

Associando várias tabelas

1. Associação de diversas tabelas **INNER JOIN**
- Associa três ou mais tabelas atraves de colunas *CHAVES*, entre elas.
- Podem haver diversos tipos de relacionamentos entre as tabelas, dos mais complexos.
Ex.: tabela-filha com dois ou mais tabelas-pai; tabela-pai que é filha de outra tabela; etc.
- O importante é identificar os relacionamentos entre tabelas para poder mescla-las.
- Sintaxe:
SELECT
(colunas que deseja obter),
tabela.coluna_chave1,
tabela.coluna_chave2,
...
FROM *tabela1*
INNER JOIN *tabela2*
ON *tabela1.coluna_chave1* = *tabela2.coluna_chave1*
INNER JOIN *tabela3*
ON *tabela2.coluna_chave2* = *tabela3.coluna_chave2*

2. Agrupando **JOIN**s

- Apenas adicionar GROUP_BY ao final.
- Determinando quais devem ser as colunas a serem agrupadas.
- Por conseguencia é possivel usar as funções de agrupamento para conseguir novas informações.
- Sintaxe:

SELECT
coluna1,
coluna2,
...
FROM *tabela1*
INNER JOIN *tabela2*
ON *tabela1.coluna_chave1* = *tabela2.coluna_chave1*
INNER JOIN *tabela3*
ON *tabela2.coluna_chave2* = *tabela3.coluna_chave2*
GROUP BY *coluna1*, *coluna 2 (ou 1, 2)*

3. Associação de diversos **LEFT JOIN**s
   - É simples, basta ao inves de usar **INNER JOIN**, utilizar **LEFT JOIN**.
   - Utilizado para mostrar todos os registros da mescla de tabelas.
   - A sintaxe é basicamente a mesma da *associação de diversos* **INNER JOIN**.

Andamento dos Estudos

Estudando instrução **JOIN** - associando varias tabelas.

Assunto em andamento:

Em andamento:

Vazios:

Finalizando detalhes: