SQLite

Sergio Pedro R Oliveira

14 junho 2023

# Objetivo

Estudo dirigido de SQL, utilizando SQLite.

# Livro de referência

Introdução a linguagem SQL - abordagem pratica para iniciantes. (Nield e Prates, 2016)

# Capítulo 4 - **SELECT**

## **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 número decimal.

## **Operadores matemáticos**

Operadores matemáticos.

| Operador | Descrição |
| --- | --- |
| + | Soma |
| - | Subtração |
| \* | Multiplicação |
| / | Divisão |
| % | 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 número 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:

Tabela verdade.

| P | NOT P | Q | NOT Q | P AND Q | P OR Q |
| --- | --- | --- | --- | --- | --- |
| V | F | V | F | V | V |
| V | F | F | V | F | V |
| F | V | V | F | F | V |
| 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.

Operadores lógicos.

| Operadores Lógicos | Descrição | Exemplo |
| --- | --- | --- |
| AND | Verifica se todas as expressões booleanas são verdadeiras | x AND y |
| OR | Verifica se alguma expressão booleana é verdadeira | x OR y |
| BETWEEN | Verifica se um valor se encaixa inclusivamente dentro de um intervalo | a BETWEEN x AND y |
| IN | Verifica se um valor existe dentro de uma lista de valores | a IN (x,y,w,z) |
| NOT | Nega e inverte o valor em uma expressão booleana | a NOT IN (x,y,w,z) |
| IS NULL | Verifica se um valor é nulo | a IS NULL |
| IS NOT NULL | Verifica se um valor não é nulo | 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 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 tipicas de sumarização (resumo), como:

Funções tipicas do GROUP BY.

| Função | Descrição |
| --- | --- |
| avg(X) | Calcula a media de todos os valores da coluna X (Omite valores nulos) |
| count(X) | Conta o número de valore não nulos da coluna X |
| count(\*) | Conta o número registros |
| max(X) | Encontra o valor máximo da coluna X (Omite valores nulos) |
| min(X) | Encontra o valor mínimo da coluna X (Omite valores nulos) |
| sum(X) | Calcula a soma dos valores da coluna X (Omite valores nulos) |
| group\_concat(X) | 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 número 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**

## **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 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.  
   - 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 JOINs  
   - 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 JOINs  
   - 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.

# CAPÍTULO 9 - DESIGN DE BANCO DE DADOS

## Planejando um banco de dados

* O design de banco de dados serve para **criar** novas tabelas, assim como **inserir**, **atualizar** e **excluir** registros.
* Uma dica para o design é fazer o diagrama RE (relaciomento de entidade), no qual exibe as tabelas e como elas estão relacionadas.
* Principais perguntas que devem ser feitas para planejar um banco de dados:  
  1. Perguntas relativas ao design:
  + Quais são os requisitos no negocio?
  + Que tabelas são necessarias para atender a esses requisitos?
  + Que colunas cada tabela conterá?
  + Como as tabelas serão *normalizadas*?  
    A *normalização* é a separação dos diferentes tipos de dados em suas proprias tabelas em vez de serem inseridos na mesma tabela.
  + Quais serão seus relacionamentos pai/filho?
  1. Perguntas relacionadas aos dados:
  + Quantos dados serão fornecidos nessas tabelas?
  + Quem ou o que fornecerá os dados para as tabelas?
  + De onde virão os dados?
  + Precisamos de processos que preencham automaticamente as tabelas?
  1. Perguntas relacionadas a segurança:
  + Quem deve ter acesso a esse banco de dados?
  + Quem deve ter acesso a que tabelas? Acesso somente de leitura? Acesso de gravação?
  + Esse banco de dados é critico para as operações empresariais?
  + Que planos de *backup* temos para o caso de desastre/falha?
  + As alterações feitas nas tabelas devem ser registradas?
  + Se o banco de dados for usado por sites ou aplicativos *web*, isso é seguro?
* SQLite tem poucos recursos de segurança, porem os bancos de dados centralizados lidam com essas áreas.

## Chave Primaria e Chave Externa

1. Chave Primaria:

* A chave primaria em uma tabela é um campo especial (ou uma combinação de campos) que fornecem uma identidade exclusiva para cada registro.
* Chave primaria serve para definir relacionamento e costuma formar base de associação.
* Chave primaria aumenta a eficienca nas consultas do software de banco de dados.
* Não são permitidas duplicatas da chave primaria, ou seja, não pode ter dois registros iguais. Se isso acontecer ocorrerá um **ERRO**.

1. Chave Externa:

* Chave externa não é o mesmo que chave primaria, a chave primaria existe na tabela-pai, a chave externa existe na tabela-filha.
* A chave externa de uma tabela-filha aponta para a chave primaria de uma tabela-pai.
* A chave externa não exige exclusividade, relacinamento “*um para muitos*”.

1. Chave Primaria vs Chave Externa:

* A chave externa e a chave primaria não precisam compartilhar o mesmo nome.

## Esquema

Dicas para montar e analisar esquematicos:  
- O diagrama exibe as tabelas, as colunas e os relacinamentos.  
- Todas as chaves primarias e chaves externas são conectadas por setas. Saindo da chave primaria e apontando para a chave externa.  
- As setas demonstram com as tabelas-pai fornecem dados para as tabelas-filha.  
- Analisar duas, ou três, tabelas por vez, para evitar se perder.  
- Para notar se esta bem *normalizado* o banco de dados, verificar se as chaves primarias/externas estão sendo usadas de maneira eficientes.

## Criando um novo banco de dados

* Extensão de banco de dados “.db”.

## Criando **TABLE**

Criação de tabelas.

* Ao criar tabelas é preciso criar as colunas e definir o tipo, as restrições e regras elas devem seguir.
* Sintaxe:

CREATE TABLE nome\_da\_tabela(  
nome\_da\_coluna1 tipo regra restrição,  
nome\_da\_coluna2 tipo regra restrição,  
...  
);

* tipos:  
  + **INTEGER**  
    Valores inteiros.
  + **REAL**  
    Ponto flutuante.
  + **VARCHAR**  
    Texto com ate 100 caracteres.
  + **BOOLEAN**  
    Aceita valores booleanos, 1 é verdadeiro e 0 é falso.
  + **TIME**  
    Tempo.
* regras:  
  + PRIMARY KEY  
    - *Chave primaria*. Determina coluna(s) identificadoras da tabela.
    - Também usado para forjar relações entre tabelas (identificador de tabela-pai).
  + FOREIGN KEY  
    - *Chave externa*. Determina as relações entre tabelas-pai e filha.
    - Sintaxe:  
      REFERENCES nome\_da\_tabela-pai (coluna\_chave\_da\_tabela-pai)
  + NOT NULL
    - Não aceitar valor NULL.
  + DEFAULT
    - Determina um valor default para o registro, muito util para tipo **BOOLEANO**.
    - Sintaxe:  
      DEFAULT (0)
* restrições:  
  + AUTOINCREMENT  
    Adiciona valores automaticamente no registro.

## Criando **VIEWS**

* Quando salvamos uma consulta em um banco de dados, ela se chama *view*.
* Podemos consultar uma *view* como se ela fosse uma tabela, ou seja, chamar a *view*, apelicar:  
  + SELECT
  + WHERE
  + CASE
  + …
* Sintaxe:
* CREATE VIEW nome\_da\_view AS  
  SELECT  
  nome\_da\_tabela.coluna1  
  nome\_da\_tabela.coluna2  
  ...  
  FROM tabela1  
  INNER JOIN tabela2  
  ON tabela1.coluna\_chave1 = tabela2.coluna\_chave1  
  ...;

# Capitulo 10 - GERENCIANDO DADOS

As principais ações do gerenciamento de dados são inserir, excluir e atualizar registros.

## **INSERT**

* INSERT  
  + O comando serve para inserir registros no banco de dados.
  + Campos não preenchidos no registro, recebem valor NULL, ou valores pré-determinados.
  + Se um campo não for preenchido e tiver a restrição NOT NULL, o INSERT falhara, pois não tem valor pré-definido e não pode ser desconsiderado aquele registro.
  + Sintaxe:
* INSERT INTO nome\_da\_tabela (coluna1\_do\_registro, coluna2\_do\_registro)  
  VALUES ('dado\_1','dado\_2');
* Multiplos INSERT simultâneos  
  + É possivel inserir diversos registros de uma só vez.
  + Processo muito útil para inserções automatizada atraves de linguagens de programação, como:  
    - Python
    - R
    - Java
    - …
  + Sintaxe:
* INSERT INTO nome\_da\_tabela (coluna1\_do\_registro, coluna2\_do\_registro)  
  VALUES  
  ('dado\_1','dado\_2'),  
  ('dado\_3','dado\_4'),  
  ...,  
  ('dado\_n','dado\_n+1');
* Chaves externas  
  + Se for inserido um registro em que a *chave externa* esteja errada, registro orfão, o registro não será aceito.

## **DELETE**

* DELETE  
  + Deleta todos os registros de uma determinada tabela.
  + Sintaxe:  
    DELETE FROM tabela;
* DELETE WHERE  
  + Pode ser usado em conjunto com a instrução WHERE para deletar apenas determinados registros.
  + Sintaxe:
* DELETE FROM tabela  
  WHERE (instrução);

Obs.: No **MySQL** a melhor forma de *deletar* todos os registros de uma tabela é pela instrução TRUNCATE TABLE.  
Ex. Sintaxe:  
TRUNCATE TABLE nome\_tabela;

## **UPDATE**

* UPDATE  
  + Modifica registros existentes.
  + Pode moficicar diversos registros de uma vez por meio de uma função.
  + Sintaxe:  
    UPDATE tabela SET coluna = função(coluna);
* UPDATE varias colunas diferentes  
  + Pode modificar diversos registros de colunas diferentes de uma só vez.
  + Sintaxe:
* UPDATE tabela SET  
  coluna1 = função(coluna1),  
  coluna2 = função(coluna2);
* UPDATE WHERE  
  + Pode ser usado em conjunto com a instrução WHERE para modificar apenas determinados registros.
  + Sintaxe:
* UPDATE tabela SET coluna = valor  
  WHERE tabela IN (lista\_dos\_registros);

## **DROP TABLE**

* Deleta determinada tabela especificada.
* Sintaxe:  
  DROP TABLE nome\_da\_tabela;

# CAPITULO 11 - TÓPICOS ESPECIAIS

## Funções de data e horario

1. DATE

* Função DATE, serve para manipular datas no SQL.
* O formato para trabalhar com data é **‘AAAA-MM-DD’** (*ano* **traço** *mês* **traço** *dia*, **entre aspas**).
* A função DATE aceita como outros argumentos, somar ou subtrair anos, mês e dias.
* Sintaxe:  
  DATE('aaaa-mm-dd','+1 day')
* Outro argumento que a função DATE aceita é o uso do ‘*now*’, para pegar a data no sistema.  
  + Sintaxe:  
    DATE('now')

1. TIME

* Função TIME, serve para manipular horarios no SQL.
* O formato para trabalhar com data é **‘HH:MM:SS’** (*horas* **dois pontos** *minutos* **dois pontos** *segundos*, **entre aspas**).
* A função TIME aceita como outros argumentos, somar ou subtrair horas, minutos, segundos.
* Sintaxe:  
  TIME('hh:mm:ss','+1 minute')
* Outro argumento que a função TIME aceita é o uso do ‘*now*’, para pegar o horario no sistema.  
  + Sintaxe:  
    TIME('now')

1. DATETIME

* A função DATETIME, serve para manipulação de data e horario ao mesmo tempo.
* O formato para trabalhar com DATETIME é **‘AAAA-MM-DD HH:MM:SS’** (*ano* **traço** *mês* **traço** *dia*, **espaço**, *horas* **dois pontos** *minutos* **dois pontos** *segundos*, **entre aspas**)
* A função aceita como argumentos soma e subtração de data e horario.
* Sintaxe:  
  DATETIME ('aaaa-mm-dd hh:mm:ss','+1 day', '-3 hour')

## Transações

* É uma instrução que só executa as instruções dentro dela, no caso (INSERT, UPDATE, DELETE), apenas se todas as instruções sejam concluidas com sucesso.
* Caso alguma instrução dentro dela dê ERRO, tudo é desfeito.
* Muito util para fazer operações de transação financeira entre contas.  
  + Exemplo de transação financeira, transferencia de dinheiro entre contas:  
    - Subtrair dinheiro de uma conta.
    - Somar dinheiro em outra conta.
* Sintaxe:

BEGIN TRANSACTION  
UPDATE tabela SET coluna1\_a\_modificar = expressão1  
WHERE tabela IN (lista\_dos\_registros\_a\_modificar)  
UPDATE tabela SET coluna2\_a\_modificar = expressão2  
WHERE tabela IN (lista\_dos\_registros\_a\_modificar)  
END TRANSACTION

# ANDAMENTO DOS ESTUDOS

Concluído.

# REFERÊNCIA

NIELD, T.; PRATES, R. [**Introdução à Linguagem SQL: Abordagem prática para iniciantes**](https://books.google.com.br/books?id=5HQdDAAAQBAJ). [s.l.] Novatec Editora, 2016.