

Readme.rmd

Sergio Pedro R Oliveira

2022-03-24

Objetivo

Estudo dirigido de MySQL.

Referência

Vídeo aulas “O curso completo de Banco de Dados e SQL, sem mistérios” - Udemey.

Módulo 2 - Teoria

Modelagem

Obs.: alguns softwares (ex.: brModelo) chamam a modelagem lógica de modelo conceitual.

1. Analise de requisitos

- Modelo das necessidades do Cliente, o que é do interesse do cliente e o que ele precisa no banco de dados.
- Processos a serem controlados pelo sistema.
- É uma fase de muita conversa e reunião com o cliente para investigar as regras do negocio.

2. processos de modelagem

- Fases 01 e 02 do projeto de banco de dados são feitos pelo administrador de dados:

i. Modelo conceitual

- Rascunho dos requisitos do projeto.
- Desenho conceitual.

ii. Modelo lógico

- Coloca os requisitos num programa de diagramas.
- Cria **entidades**, posteriormente serão tabelas.
- Cria **atributos**, posteriormente serão campos, colunas nas tabelas.
- **Atributos identificador**, posteriormente será **Chave Primaria Artificial**.
 - * Normalmente leva o nome “ID” + “o_nome_da_tabela”.
- Modelo **entidades-relacionamentos**, define os relacionamentos entre os agentes.

* Relacionamentos:

· Obrigatoriedade

A obrigatoriedade de preencher as duas tabelas/entidades.

Tipos:

0

Não existe obrigatoriedade, se entrar com dados em um, não é obrigado a entrar com dados no outro.

1

Existe obrigatoriedade, se entrar com dados em um, obrigatoriamente é necessario entrar com dados no outro.

- Cardinalidade
Maximo de preenchimentos:
Se obrigatoriedade 0, no minimo 0 e no maximo n dados.
Se obrigatoriedade 1, no minimo 1 e no maximo n dados.

* tipos de relacionamentos de entidade:

- (1,1)
É obrigatorio, pode entrar apenas com 1 dado.
- (0,1)
Não é obrigatorio, quando entrar, entrar com 1 dado.
- (1,n)
É obrigatorio, pode entrar com varios dados.
- (0,n)
Não é obrigatorio, pode entrar com varios dados.

* Como ler os relacionamentos entre entidades:

Exemplos:

- (1,1) -> (0,n)
Ignorar a primeira coordenanda de obrigatoriedade dos dois relacionamentos, e fica 1 para n, logo “um para muitos”.
- (0,n) -> (0,1)
Ignorar a primeira coordenanda de obrigatoriedade dos dois relacionamentos, e fica n para 1, logo “muitos para um”.

- Fase 03 do projeto de banco de dados é feita tanto pelo administrador de bancos de dados(DBA) quanto administrador de dados(AD):

iii. Modelo físico

- Criando banco de dados.

CREATE DATABASE *nome_do_banco_de_dados*;

- Conectando-se a um dos banco de dados do sistema.

USE *nome_do_banco_de_dados*;

- Criando tabela.

CREATE TABLE *nome_da_tabela*(
coluna1 tipo(tamanho) chave_ou_não restrições,
coluna2 tipo(tamanho) restrições,
 ...,
FOREIGN KEY(*nome_da_coluna_da_chave_estrangeira*)
REFERENCES *nome_da_tabela_da_chave_primaria*(*nome_da_coluna_da_chave_primaria*)
);

- Verificando os banco de dados no sistema.
SHOW DATABASES;
- Verificando as tabelas do banco de dados.
SHOW TABLES;
- *Descrevendo* como é a estrutura de uma tabela, verificando quais são as colunas.
DESC *nome_da_tabela;*
- Verificar em qual **DATABASE** esta conectado no momento.
STATUS;
- Deletando um banco de dados.
DROP DATABASE *nome_do_banco_de_dados;*
- Deletando uma tabela.
DROP TABLE *nome_da_tabela;*

Tipagem de campos

A tipagem correta diminui o tempo de resposta, otimiza os processos.

1. Tipo caracteres

- **CHAR**

- Usado quando o numero de caracteres não varia, separa na memoria um espaço determinado para ser preenchido.
- Sintaxe:
CHAR(*numero_maximo_de_caracteres*)

- **VARCHAR**

- Usado quando o numero de caracteres varia, dependendo da entrada adapta o espaço separado na memoria para caber os characters.
- Sintaxe:
VARCHAR(*numero_maximo_de_caracteres*)

2. Tipo **ENUM**

- Conjunto de dados enumerados, ou seja, um conjunto fixo de dados.
- Limita dados em uma coluna, lista de opções.
- tipo característico do **MySQL**.
- Sintaxe:
ENUM(*'primeira_opção'; 'segunda_opção'; ...*)

3. Tipo numerico

- **INT**

- Para numeros inteiros.
- Numero maximo de 11 digitos, para numeros maiores que isso usar **VARCHAR**.
- Sintaxe:
INT

- **FLOAT**

- Ponto flutuante, ou seja, numeros reais.
- Ao entrar com o valor (em **INSERT**, **UPDATE**, ...), usar “.” ao inves de “,” para separar as casas decimais.

- Para numeros com casas decimais.
FLOAT(*total*, *virgula*)

4. Para fotos e documentos

- **BLOB**

5. Tipo textos

- **TEXT**

Subtipos - regras e restrições

Restrições

- **PRIMARY KEY**

- Define que a coluna/campo é uma *Chave Primaria*.
- *Chave Primaria* é um campo que identifique todo registro como sendo único.

- **UNIQUE**

- Define aquela coluna/campo sem repetições.
- Tem valores unicos.

- **NOT NULL**

- A coluna/campo não aceita valor NULL, deve ser preenchida.

- **AUTO_INCREMENT**

- A coluna/campo se auto preenche com um valor inteiro não repetido, a cada registro.

Regras

- **FOREIGN KEY**

- *Chave Estrangeira* é a *Chave Primaria* de uma tabela, que vai ate a outra tabela, para fazer referencia entre registros.
- Regra de onde fica a *Chave Estrangeira* (**FK**):
 - * 1 x 1 (um pra um) a *Chave Estrangeira* fica na tabela mais fraca.
 - Se for 1 x 1, leva **UNIQUE**.
 - * 1 x n (um pra muitos) a *Chave Estrangeira* fica na tabela n.
 - * n x n (muitos pra muitos)...

- Sintaxe:

FOREIGN KEY(*nome_da_coluna_da_chave_estrangeira*)

- **REFERENCES**

- Aponta para onde a *Chave Estrangeira* faz referencia, qual *Chave Primaria*.

- Sintaxe:

REFERENCES *nome_da_tabela_da_chave_primaria*(*nome_da_coluna_da_chave_primaria*)

Obs.: A sintaxe para inserção de *Chave Estrangeira* em **MySQL** fica:

FOREIGN KEY(*nome_da_coluna_da_chave_estrangeira*)

REFERENCES *nome_da_tabela_da_chave_primaria*(*nome_da_coluna_da_chave_primaria*)

Sem virgula entre eles.

Módulo 3 - Comandos

Inserir registros na tabela - INSERT

- Existem diversas formas de inserir dados na tabela, entre eles temos:
 - Omitindo colunas/campos.
 - * Determina apenas a tabela, que puxa todos os campos para serem preenchidos, na ordem que aparece na tabela.
 - * Sintaxe:
INSERT INTO *nome_da_tabela*
VALUES (*valor_na_coluna_1*, *valor_na_coluna_2*,...);
 - Colocando as colunas.
 - * Especifica a ordem das entradas e os campos a serem preenchidos.
 - * Sintaxe:
INSERT INTO *nome_da_tabela*(*coluna_3*, *coluna_1*, *coluna_2*,...) **VALUES** (*valor_na_coluna_3*, *valor_na_coluna_1*,...);
 - INSERT COMPACTO, somente **MySQL**.
 - * Insere diversos registros de uma vez, na ordem que aparecem na tabela.
 - * Sintaxe:
INSERT INTO *nome_da_tabela*
VALUES (*valor_na_coluna_1_registro1*, *valor_na_coluna_2_registro1*,...),
(*valor_na_coluna_1_registro2*, *valor_na_coluna_2_registro2*,...),
...;
 - Inserindo dados num campo com **AUTO_INCREMENT**.
 - * Na coluna/campo em que tem **AUTO_INCREMENT**, insere-se o valor **NULL**, assim o **MySQL** entende que ele proprio deve auto incrementar aquele campo.

Consultando campos na tabela - SELECT

- O comando **SELECT** serve para projeção, seleção e junção.
- O comando **SELECT** seleciona os campos/colunas a serem mostrados.
- Projeta/constroi o que deve ser mostrado, não apenas os dados da tabela.
 - Exemplo de código:
SELECT 'SERGIO PEDRO' AS MEU_NOME;
 - Sintaxe:
SELECT 'algo a mostrar' AS alias_da_coluna;
- Seleciona o que deve ser mostrado da tabela.
 - Exemplo de código:
SELECT NOME, SEXO, EMAIL, ENDERECO FROM CLIENTE;
 - Sintaxe:
SELECT coluna_1, coluna_6, coluna_3, coluna_5 FROM tabela;
 - Seleciona todas as colunas da tabela:
SELECT * FROM tabela;
Obs.: '*', Diminui a eficiência da pesquisa na tabela.

Consultando registros na tabela - WHERE

- O comando **WHERE** serve para filtrar os registros/linhas da tabela, antes de mostrar.
 - Sintaxe:
SELECT *coluna_1*, *coluna_2* **FROM** *tabela*
WHERE *coluna_1* = *criterio*;
- O comando **WHERE** não precisa ter haver com a seleção **SELECT**.
 - Sintaxe:
SELECT *coluna_1*, *coluna_3* **FROM** *tabela*
WHERE *coluna_2* = *criterio*;
- Para trabalhar com *strings*, é útil usar o comando **LIKE** e os *caracteres coringas*.
 - Caracteres coringas:
 - * `'%'`
Qualquer coisa.
 - * `'_'`
Um único caracter.
 - Sintaxe:
SELECT *coluna_1*, *coluna_3* **FROM** *tabela*
WHERE *coluna_2* **LIKE** '*string_procurada*';
Obs.: Os caracteres coringas podem entrar em qualquer lugar da string para complementar o texto a procurar.
- Filtrando valores **NULL**.
 - Para filtrar valores **NULL**, basta utilizar o **IS NULL**, ao inves de '**= NULL**'.
 - * Sintaxe:
SELECT *coluna1*, *coluna2*, ... **FROM** *tabela*
WHERE *colunaX* **IS NULL**;
 - Para filtrar valores não **NULL**, basta utilizar **IS NOT NULL**, ao inves de uma expressão.
 - * Sintaxe:
SELECT *coluna1*, *coluna2*, ... **FROM** *tabela*
WHERE *colunaX* **IS NOT NULL**;

Módulo 5 - Operadores Lógicos, GROUP BY e ORDER BY

Operadores Lógicos e Performance de operadores lógicos

- Operadores lógicos:

- **OR**/OU

- * Apenas uma condição precisa ser verdadeira para dar verdadeiro.

- * Sintaxe:

- ```
SELECT * FROM tabela
WHERE (condição_1 OR condição_2);
```

- **AND**/E

- \* Todas as condições precisam ser verdadeiras para dar verdadeiro.

- \* Sintaxe:

- ```
SELECT * FROM tabela
WHERE (condição_1 AND condição_2);
```

- **NOT**/negação

- * Nega e inverte e inverte o valor de uma expressão.

- * Sintaxe:

- ```
SELECT * FROM tabela
WHERE (condição_1 AND NOT condição_2);
```

Obs.: Inverte o resultado da *condição\_2*.

- *Tabela verdade*

| ##   | A | NOT_A | B | NOT_B | A_OR_B | A_AND_B |
|------|---|-------|---|-------|--------|---------|
| ## 1 | V | F     | V | F     | V      | V       |
| ## 2 | V | F     | F | V     | V      | F       |
| ## 3 | F | V     | V | F     | V      | F       |
| ## 4 | F | V     | F | V     | F      | F       |

- Performance de operadores lógicos.

- Para melhorar a performance das consultas, com operadores lógicos, dois casos podem ser avaliados:

- \* No caso **OR**:

- Colocar a condição que oferece maior incidência de verdadeiro na frente.
    - Se a primeira condição é verdadeira, a segunda não é avaliada, melhorando assim a performance da consulta.

- \* No caso **AND**:

- Colocar a condição que oferece menor incidência de verdadeiro na frente.

- Se a primeira condição for falsa, a segunda nem é avaliada, pois o resultado é falso. Melhorando assim a performance da consulta.

## Agregador e funções de agregação - GROUP BY

- **COUNT(\*)**

- Conta o numero de registros.
- Sintaxe:  
**SELECT COUNT (\*) FROM tabela;**

- **GROUP BY**

- Agrupa dados em torno de determinado campo.
- Usar em conjunto com funções de agrupamento, como:
  - \* **COUNT (\*)**  
Conta todos os registros.
  - \* **COUNT (coluna\_x)**  
Conta os registros da coluna x.
  - \* **AVG (coluna\_x)**  
Calcula a media dos valores da coluna x.
  - \* **MAX (coluna\_x)**  
Encontra o valor maximo da coluna x.
  - \* **MIN (coluna\_x)**  
Encontra o valor minimo da coluna x.
  - \* **SUM (coluna\_x)**  
Calcula a soma dos valores na coluna x.
- Sintaxe:  
**SELECT coluna\_x, COUNT(\*) FROM tabela  
GROUP BY coluna\_x;**
- É possível agrupar mais de uma coluna de uma vez.
  - \* A ordem em que as colunas aparecem na instrução **GROUP BY**, determinam a ordem de prioridade no agrupamento.
  - \* Sintaxe:  
**SELECT coluna1, coluna2,.. FROM tabela  
GROUP BY coluna1, coluna2;**  
Obs.: Prioridade primeiro agrupar a *coluna1*, depois agrupar em função da *coluna1* a *coluna2*.

## Ordenando registros - ORDER BY

- ORDER BY

- Organiza os dados segundo uma ordem.
- Por default é ordem crescente, **ASC**.
- Para ordem decrescente só adicionar ao final **DESC**.
- Utilizado normalmente ao final de **WHERE** ou **GROUP BY**.
- Ao invés de colocar o nome da coluna, pode indicar a numeração da coluna na ordem em que aparece na instrução **SELECT**.
- Sintaxe:  
**SELECT** *coluna1, coluna2, ...* **FROM** *tabela*  
**GROUP BY** *coluna1*  
**ORDER BY** *coluna2*; (ou **ORDER BY** *2*;)
- Também é possível colocar em ordem, mais de uma coluna de uma vez.
  - \* Sintaxe:  
**SELECT** *coluna1, coluna2, ...* **FROM** *tabela*  
**GROUP BY** *coluna1*  
**ORDER BY** *coluna2 ASC, coluna1 DESC*; (ou **ORDER BY** *2 ASC, 1 DESC*;)



## Módulo 7 - Mais comandos UPDATE e DELETE

### Atualizando registros na tabela - UPDATE

- Atualizar todos os dados de uma coluna/campo de uma tabela, de uma vez.
  - Para atualizar todos os dados, de uma determinada coluna/campo, de uma tabela, para um dado determinado, basta usar **UPDATE** sem filtros.
  - Muito cuidado ao utilizar esse comando assim, pois pode gerar muitos problemas.
  - Sintaxe:  
**UPDATE** *tabela* **SET** *coluna\_a\_atualizar* = *valor\_atualizado*;
- Para atualizar um determinado registro.
  - Para atualizar um determinado dado de uma coluna/campo, utilizar o **UPDATE** em conjunto com a instrução **WHERE**.
  - Sintaxe:  
**UPDATE** *tabela* **SET** *coluna\_a\_atualizar* = *valor\_atualizado*  
**WHERE** *condição* = *valor*;

## Deletando registros - DELETE

- Deletar todos os registros de uma tabela.
    - Sintaxe:  
**DELETE FROM** *tabela*;
  - Deletar apenas determinados registros de uma tabela, usar **DELETE** em conjunto com filtro **WHERE**.
    - Sintaxe:  
**DELETE FROM** *tabela*  
**WHERE** *critério\_do\_que\_se\_quer\_deletar* = *valor*;
  - Dicas:
    - Antes de deletar qualquer registro, deve-se conferir através de uma consulta, se os dados que aparecem são os que querem ser deletados.  
**SELECT \* FROM** *tabela*  
**WHERE** *mesmo\_critério\_do\_delete* = *valor*;
    - Contar os registros antes, durante a consulta e depois do **DELETE**. Para ter certeza sobre o que foi deletado.  
**SELECT COUNT(\*) FROM** *tabela*  
**WHERE** *mesmo\_critério\_do\_delete* = *valor*;
- Obs.: Exemplo de consulta de quantos registros devem ser deletados.

## Transação - **START TRANSACTION**

- **START TRANSACTION;**

- As instruções dentro da transação, que serão avalidadas, ficam identadas dentro da transação.
- Sintaxe:  
**START TRANSACTION;**  
*instrução\_1;*  
*instrução\_2;*  
...

- **COMMIT;**

- aceita a transação (**START TRANSACTION;**). confirma as instruções da transação.
- Fica fora da identração da instrução **START TRANSACTION**.

- **ROLLBACK;**

- Nega a transação (**START TRANSACTION;**). Desfaz as instruções da transação.
- Instrução para voltar atrás em instruções.
- Desfaz instruções (como **UPDATE**, **DELETE**, ...), tudo que estiver dentro de **START TRANSACTION**.
- Fica fora da identração da instrução **START TRANSACTION**.

Obs.: Essas instruções (**START TRANSACTION**, **COMMIT** e **ROLLBACK**) levam “;” ao final delas, não esta errado como escrito a cima.

## Módulo 8 - Modelagem

### Primeira forma normal

- 3 Regras:
  1. Todo campo vetorizado se tornará outra tabela.
    - Campo vetorizado é todo campo que apresenta algo como um vetor dentro dele.
    - Varios dados do mesmo tipo (vetor).
    - Exemplo:  
*vetor* [VERDE, AMARELO, LARANJA,...]
  2. Todo campo multivalorado se tornará outra tabela.
    - Campo multivalorado é todo campo que apresenta algo como uma lista dentro dele.
    - Diversos dados de tipos diferentes (lista).
    - Exemplo:  
*list* (1, VERDE, CASA, ...)
  3. Toda tabela necessita de pelo menos um campo que identifique todo registro como sendo único (é o que chamamos de “**Chave Primaria**” ou “**Primary Key**”).
    - Tipos de **CHAVE PRIMARIA**:
      - \* NATURAL
        - Pertence ao registro intrinsecamente.
        - Muito útil, porem pouco confiavel. Depende de terceiros para existir, como o governo por exemplo.
        - Exemplo: CPF.
      - \* ARTIFICIAL
        - É criada pelo/para o banco de dados para identificar o registro.
        - Exemplo: ID.
        - Mais indicado de se trabalhar, pois oferece controle total por parte do administrador do banco de dados e não depende de terceiros para existir.

### Segunda forma normal

### Terceira forma normal

## Detalhes

- **Comentarios** no **MySQL**, diferente do **SQL** onde comentarios são '/\* \*/', no MySQL é '#'. Ou '-' para comentario de linha.
- O que são e o que fazem os administradores:
  - Administrador de dados(AD):

O Administrador de Dados (AD) tem o objetivo de gerenciar o Modelo de Dados Corporativo, contribuindo para assegurar a qualidade das informações, a integração dos sistemas, a retenção e a disseminação do conhecimento dos negócios.

Cabe a ele, guiado por certos princípios e através de atividades de planejamento, organização e controle dos dados corporativos, gerenciar os dados como recursos de uso comum da organização, promovendo-lhes os valores de autenticidade, autoridade, precisão, acessibilidade, seguridade e inteligibilidade.

Tem como função o planejamento central, a documentação e o gerenciamento dos dados a partir da perspectiva de seus significados e valores para a organização como um todo.
  - Administrador de banco de dados (DBA):

O DBA (database administrator), sigla em inglês para Administrador de Banco de Dados, é um profissional da área de tecnologia responsável pela criação, instalação, monitoramento, reparos e análise de estruturas de um banco de dados.

O banco de dados fica sob análise periódica do DBA, que trabalha para que não haja sobrecargas do sistema e que as informações inseridas tenham destino correto nos servidores. Outras funções também importantes são analisar o espaço em disco, buscar melhorias para os sistemas e realizar backups.
- Acesso ao **MySQL** pelo terminal é necessario usar o comando:  
mysql -u root -p
  - Depois colocar a senha.
- Ao final dos comandos do **SQL** e do **MySQL**, usar o ';' (delimitador), ele informa que o comando acabou e deve ser executado.

## **Andamento dos Estudos**

### **Assunto em andamento:**

Atualmente estou estudando Módulo 8.