MySQL

Readme.rmd

Sergio Pedro R Oliveira

2022-03-30

Table of Contents

# 1 Objetivo

Estudo dirigido de MySQL.

# 2 Referência

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

# 3 Módulo 2 - Teoria

## 3.1 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.

1. processos de modelagem

* Fases 01 e 02 do projeto de banco de dados são feitos pelo administrador de dados:  
  1. Modelo conceitual
  + Rascunho dos requisitos do projeto.
  + Desenho conceitual.
  1. 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 obrigadorio, 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):  
  1. 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*;

## 3.2 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 caracters.
  + Sintaxe:  
    **VARCHAR**(*numero\_maximo\_de\_caracteres*)

1. Tipo **ENUM**

* Conjunto de dados enumerados, ou seja, um conjunto fixo de dados.
* Limita dados em uma coluna, lista de opções.
* tipo caracterisco do **MySQL**.
* Sintaxe:  
  **ENUM**(‘*primeira\_opção*’,‘*segunda\_opção*’,…)

1. 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*)

1. Para fotos e documentos

* **BLOB**

1. Tipo textos

* **TEXT**

## 3.3 Subtipos - regras e restrições

### 3.3.1 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.

### 3.3.2 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.

# 4 Módulo 3 - Comandos

## 4.1 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.

## 4.2 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 codigo:  
    **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 codigo:  
    **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 eficiencia da pesquisa na tabela.

## 4.3 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 util 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**;

# 5 Módulo 5 - Operadores Lógicos, **GROUP BY** e **ORDER BY**

## 5.1 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 incidencia 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 oference menor inicidencia 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.

## 5.2 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 possivel 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*.

## 5.3 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 inves 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 possivel 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**;)

# 6 Módulo 7 - Mais comandos **UPDATE** e **DELETE**

## 6.1 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*;

## 6.2 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 **WHERER**.  
  + Sintaxe:  
    **DELETE** **FROM** *tabela*  
    **WHERE** *criterio\_do\_que\_se\_quer\_deletar* **=** *valor*;
* Dicas:  
  + Antes de deletar qualquer registro, deve-se conferir atraves de uma consulta, se os dados que aparecem são os que querem ser deletados.  
    **SELECT** \* **FROM** *tabela*  
    **WHERE** *mesmo\_criterio\_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\_criterio\_do\_delete* **=** *valor*;  
    Obs.: Exemplo de consulta de quantos registros devem ser deletados.

## 6.3 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.

# 7 Módulo 8 - Modelagem

## 7.1 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,…]
  1. 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, …)
  1. 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.

## 7.2 Segunda forma normal

## 7.3 Terceira forma normal

# 8 Módulo 9 - PROJEÇÃO, SELEÇÃO E JUNÇÃO

Principais passos de uma consulta.

## 8.1 PROJEÇÃO

* O primeiro passo de uma consulta é montar o que quer ver na tela - **SELECT**.
* É tudo que você quer ver na tela.
* Sintaxe comentada:  
  **SELECT** *coluna\_1* (PROJEÇÃO)  
  **FROM** *tabela*; (ORIGEM)  
  ou  
  **SELECT** 2+2 **AS** *alias*; (PROJEÇÃO)  
  Obs.: o que esta entre parênteses é comentario.

## 8.2 SELEÇÃO

* O segundo passo de uma consulta é a seleção dos dados de uma consulta - **WHERE**.
* É filtrar.
* Trazer um subconjunto do conjunto total de registros de uma tabela.
* Sintaxe comentada:  
  **SELECT** *coluna\_1*, *coluna\_2*, *coluna\_3* (PROJEÇÃO)  
  **FROM** *tabela* (ORIGEM)  
  **WHERE** *critero* **=** *valor\_do\_criterio*; (SELEÇÃO)  
  Obs.: o que esta entre parênteses é comentario.

## 8.3 JUNÇÃO

### 8.3.1 Junção forma errada - gambiarra

* Usa seleção como uma forma de juntar tabelas.
* Como conseguencia:  
  + Uso de operadores lógicos para mais criterios de seleção - **WHERE**.
  + Ineficiencia na pesquisa, maior custo computacional.
* Sintaxe comentada:  
  **SELECT** *coluna1\_tab1*, *coluna2\_tab1*, *coluna1\_tab2* (PROJEÇÃO)  
  **FROM** *tabela1*, *tabela2* (ORIGENS)  
  **WHERE** *chave\_primaria\_tab1* **=** *chave\_estrangeira\_tab2*;(JUNÇÃO)  
  ou  
  **SELECT** *coluna1\_tab1*, *coluna2\_tab1*, *coluna1\_tab2* (PROJEÇÃO)  
  **FROM** *tabela1*, *tabela2* (ORIGENS)  
  **WHERE** *chave\_primaria\_tab1* **=** *chave\_estrangeira\_tab2* (JUNÇÃO)  
  **AND** *criterio* **=** *valor*;(SELEÇÃO com operador lógico)  
  Obs.: o que esta entre parênteses é comentario.

### 8.3.2 Junção forma certa - **JOIN**

* Junção **JOIN**, junta duas ou mais tabelas apartir das colunas de *chaves primarias* e *chaves estrangeiras*.
* Exclui os registros sem par (orfans) na outra tabela - **INNER**.
* Admite seleção - **WHERE** - sem maiores custos computacionais.
* Consulta com duas tabelas.  
  + Sintaxe comentada:  
    **SELECT** *coluna1\_tab1*, *coluna2\_tab1*, *coluna1\_tab2* (PROJEÇÃO)  
    **FROM** *tabela1* (ORIGEM)  
    **INNER** **JOIN** *tabela2* (JUNÇÃO)  
    **ON** *chave\_primaria\_tab1* **=** *chave\_estrangeira\_tab2*  
    **WHERE** *criterio* **=** *valor*;(SELEÇÃO)
* Consulta com mais de duas colunas.  
  + Indicar de onde vem cada coluna atraves de “*nome\_da\_tabela*.*nome\_da\_coluna*”.
  + Sintaxe comentada:  
    **SELECT**  
    *tabela1*.*coluna1\_tab1*,  
    *tabela1*.*coluna2\_tab1*,  
    *tabela2*.*coluna1\_tab2*,  
    *tabela3*.*coluna1\_tab3* (PROJEÇÃO)  
    **FROM** *tabela1* (ORIGEM)  
    **INNER** **JOIN** *tabela2* (JUNÇÃO)  
    **ON** *tabela1*.*chave\_primaria\_tab1* **=** *tabela2*.*chave\_estrangeira\_tab2*  
    **INNER** **JOIN** *tabela3* (JUNÇÃO)  
    **ON** *tabela1*.*chave\_primaria\_tab1* **=** *tabela3*.*chave\_estrangeira\_tab3*  
    **WHERE** *criterio* **=** *valor*;(SELEÇÃO)  
    Obs.: o que esta entre parênteses é comentario.
* Ponteiramento (alias para tabelas)  
  + Melhora a performance da consulta.
  + Sintaxe comentada:  
    **SELECT**  
    *A*.*coluna1\_tab1*,  
    *A*.*coluna2\_tab1*,  
    *B*.*coluna1\_tab2*,  
    *C*.*coluna1\_tab3*  
    **FROM** *tabela1* *A* (PONTEIRAMENTO DA TABELA 1)  
    **INNER** **JOIN** *tabela2* *B* (PONTEIRAMENTO DA TABELA 2)  
    **ON** *A*.*chave\_primaria\_tab1* **=** *B*.*chave\_estrangeira\_tab2*  
    **INNER** **JOIN** *tabela3* *C* (PONTEIRAMENTO DA TABELA 3)  
    **ON** *A*.*chave\_primaria\_tab1* **=** *C*.*chave\_estrangeira\_tab3*  
    **WHERE** *criterio* **=** *valor*;

# 9 Categoria de comandos

## 9.1 **DML** - *Data Manipulation Language* (Linguagem de Manipulação de Dados)

É um conjunto de instruções usada nas consultas e modificações dos dados armazenados nas tabelas do banco de dados.

* **INSERT**  
  + Adiciona registros numa 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*,…),  
    …;
* **UPDATE**  
  + Altera os dados de um ou mais registros em uma tabela.
  + Sintaxe:  
    **UPDATE** *tabela* **SET** *coluna\_a\_atualizar* **=** *valor\_atualizado*  
    **WHERE** *condição* **=** *valor*;
* **DELETE**  
  + Remove um ou mais registros de uma tabela.
  + Sintaxe:  
    **DELETE** **FROM** *tabela*  
    **WHERE** *criterio\_do\_que\_se\_quer\_deletar* **=** *valor*;

## 9.2 **DDL** - *Data Definition Language* (Linguagem de definição de dados)

É um conjunto de instruções usado para criar e modificar as estruturas dos objetos armazenados no banco de dados.

* **CREATE**  
  Utilizada para construir um novo banco de dados, tabela, índice ou consulta armazenada.  
  + **DATABESE**  
    - Criação de banco de dados.
    - Sintaxe:  
      **CREATE** **DATABASE** *nome\_banco\_de\_dados*;
  + **TABLE**  
    - Criação de tabela.
    - Sintaxe:  
      **CREATE** **TABLE** *nome\_tabela* (  
      *coluna1* *tipo* *regra* *retrições*,  
      *coluna2* *tipo* *regra* *retrições*,  
      …  
      );
* **DROP**  
  Remove um banco de dados, tabela, índice ou visão existente.  
  + **DATABESE**  
    - Remove banco de dados.
    - Sintaxe:  
      **DROP** **DATABASE** *nome\_do\_banco\_de\_dados*;
  + **TABLE**  
    - Remove tabela.
    - Sintaxe:  
      **DROP** **TABLE** *nome\_da\_tabela*;
* **ALTER**  
  + Modifica um objeto existente do banco de dados.
  + É possível incluir, eliminar e alterar colunas.  
    - **CHANGE**  
      * Altera o nome e o tipo da coluna/campo.
      * Para alterar apenas o tipo, é necessario repetir o nome da coluna/campo.
      * Sintaxe:  
        **ALTER TABLE** *nome\_tabela*  
        **CHANGE** *nome\_coluna* *(novo)nome\_coluna* *modificação\_tipo*;
    - **MODIFY**  
      * Altera o tipo e regras de uma coluna/campo.
      * Sintaxe:  
        **ALTER TABLE** *nome\_tabela*  
        **MODIFY** *nome\_coluna* *modificação\_tipo*;
    - **ADD C0LUMN**  
      * Adicionando uma nova coluna.
      * Sintaxe:  
        **ALTER TABLE** [nome\_database.]*nome\_tabela*  
        **ADD COLUMN** *nome\_coluna* *tipo*;
      * Para alterar a posição de entrada da coluna na tabela, usar **FIRST** (para aparecer na primeira posição da tabela) ou **AFTER** (depois de tal coluna).
      * Sintaxe:  
        **ALTER TABLE** [nome\_database.]*nome\_tabela*  
        **ADD COLUMN** *nome\_coluna* *tipo*  
        **FIRST**;  
        ou  
        **ALTER TABLE** [nome\_database.]*nome\_tabela*  
        **ADD COLUMN** *nome\_coluna* *tipo*  
        **AFTER** *coluna\_de\_referencia*;
    - **DROP COLUMN**  
      * Deleta uma determinada coluna de uma tabela.
      * Sintaxe:  
        **ALTER TABLE** [nome\_database.]*nome\_tabela*  
        **DROP COLUMN** *nome\_coluna*;
* **TRUNCATE**  
  + Esvazia imediatamente todo o conteúdo de uma tabela ou objeto que contenha dados.
  + É muito mais rápido que um comando DELETE, pois, ao contrário deste, não armazena os dados sendo removidos no log de transações. Por esse motivo, em vários SGBDs é um comando não-transacional e irrecuperável, não sendo possível desfazê-lo com **ROLLBACK**.
  + Sintaxe:  
    **TRUNCATE** **TABLE** *nome\_tabela*;
* **COMMENT**
* **RENAME**

## 9.3 **DCL** - *Data Control Language* (Linguagem de Controle de Dados)

São usados para controle de acesso e gerenciamento de permissões para usuários em no banco de dados. Com eles, pode facilmente permitir ou negar algumas ações para usuários nas tabelas ou registros (segurança de nível de linha).

* **GRANT**  
  + Permitir que usuários especificados realizem tarefas especificadas.
  + Sintaxe:
* **REVOKE**  
  + Cancela permissões previamente concedidas ou negadas.
  + Sintaxe:
* Privilégios que podem ser CONCEDIDOS à ou REVOCADOS de um usuário:  
  + **CONNECT**
  + **SELECT**
  + **INSERT**
  + **UPDATE**
  + **DELETE**
  + **EXECUTE**
  + **USAGE**

## 9.4 **TCL** - *Tool Command Language* (Linguagem de Comandos de Ferramentas)

São usados para gerenciar as mudanças feitas por instruções DML. Ele permite que as declarações a serem agrupadas em transações lógicas.

* **BACKROLL**
* **COMMIT**

# 10 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.

# 11 Andamento dos Estudos

## 11.1 Assunto em andamento:

Atualmente estou estudando Módulo 9.