Curso: Projetar Banco de Dados Relacionais (72h)

Prof. André Nascimento

andre.donascimento@ba.senac.br

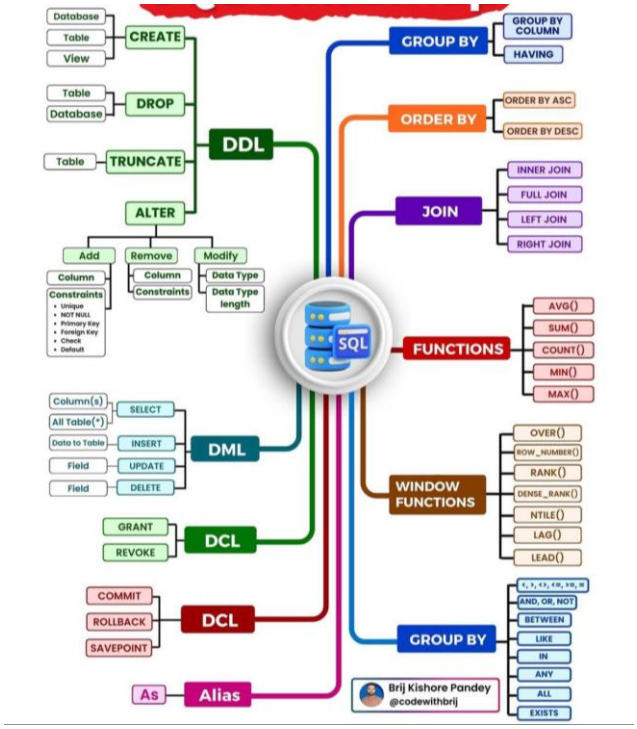
https://linktr.ee/andreanivaldo

1

SQL

- Modelo físico com MySQL

2



3

Download e instalação do MySQL

- Download: https://dev.mysql.com/downloads/installer/
- Tutorial de instalação: https://www.youtube.com/watch?v=IEUgVwjXF0o
- Documentação oficial: https://dev.mysql.com/doc/

4

Introdução a SQL

- Todo Sistema Gerenciador de Banco de Dados (SGBD) deve oferecer aos seus usuários e administradores meios de criar e manipular dados armazenados em seus banco de dados
- A linguagem SQL, sigla em inglês para Structured Query Language ou Linguagem de Consulta Estruturada é a linguagem padrão adotada por diferentes SGBDs para criar e manipular banco de dados relacionais
- A linguagem SQL possui uma sintaxe única, porém alguns SGBDs trazem variações na sintaxe. Porém a lógica estrutural da linguagem é sempre a mesma
- O SQL tem o objetivo de criar de forma estruturada e declarativa as tabelas de bancos de dados

5

DDL – Linguagem de Definição de Dados

- Os comandos DDL permite ao utilizador criar novas tabelas, alterar a estrutura de uma tabela ou excluir uma tabela;
- Os comandos DDL são:
  - **CREATE TABLE** (Criar Tabela)
  - **DROP TABLE** (Excluir Tabela)
  - **ALTER TABLE** (Alterar Tabela)

6

DML – Linguagem de Manipulação de Dados

- Os comando DML permite ao utilizador inserir registros em uma tabela, atualizar registros inseridos em uma tabela, deletar registros em uma tabela e selecionar registros em uma tabela;

- Os comandos DML são:

  - **INSERT** (Inserir)
  - **UPDATE** (Atualizar)
  - **DELETE** (Deletar)
  - **SELECT** (Selecionar)

7

SQL: comentários

- Para adicionar comentários em SQL, pode-se utilizar os seguintes símbolos:

  - **Comentários de linha única**: Iniciar o comentário com dois hífens (--). Para maior clareza, é recomendado deixar um espaço após os hífens

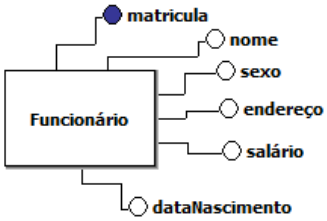
  - **Comentários de várias linhas**: Iniciar o comentário com /* e terminá-lo com */

8

Registros: DDL e DML

- ...

COMANDOS DDL
Create, Alter e Drop



COMANDOS DML
Insert, Update, Delete e Select

Matricula	Nome	Sexo	Endereço	Salário	dataNasc
1	Jackson Henrique	Masculino	Rua X...	5.000,00	30/06/1987
2	Hanna Karoline	Feminino	Rua Y...	6.000,00	23/10/1990
3	Jaqueline Leão	Feminino	Av. B	8.000,00	21/06/1993

9

DDL: comandos básicos

- Criação de Banco de Dados:
- Sintaxe:
  - `CREATE DATABASE nome_do_banco_de_dados;`
- Exemplo:
  - `CREATE DATABASE locadora;`

10

DDL: CREATE TABLE

- Sintaxe:

```
CREATE TABLE nome_da_tabela (
        nomeatributo1 tipodedado tipoatributo,
        nomeatributo2 tipodedado,
        nomeatributo3 tipodedado
);
```

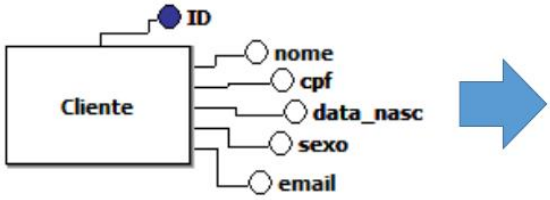
11

DDL: CREATE TABLE

- Tipo de **Dados**:
  - VARCHAR (100) -> Textos
  - INTEGER -> Números Inteiros
  - FLOAT -> Números Fracionados
  - DOUBLE -> Números Fracionados
  - DATE -> Datas
  - TIME -> Horas
  - DATETIME -> Data e Horas
- Tipos de **Atributos**:
  - PRIMARY KEY -> Chave Primária
  - FOREIGN KEY -> Chave Estrangeira
  - NOT NULL -> Não Nulo
  - UNIQUE -> Único
  - AUTO_INCREMENT -> Incremento Numérico Automático;

12

DDL: CREATE TABLE




```
CREATE TABLE Cliente (
        id_cli_pk INTEGER PRIMARY KEY,
        nome_cli VARCHAR (100),
        cpf_cli VARCHAR(15),
        data_cli DATE,
        sexo_cli VARCHAR (10),
        email_cli VARCHAR (100)
);
```

13

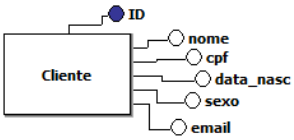
Regras de boas práticas

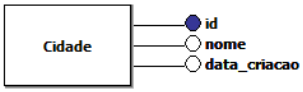
- Todo atributo deve possuir um **sufixo** ou **prefixo** que identifique a sua tabela de origem
- Os sufixo ou prefixo deve possuir de **3 a 5 letras**
- O sufixo ou prefixo devem ser separado por **underline** do nome do atributo
- Todo nome de atributo começa com a letra **minúscula**
- Todo nome de tabela começa com a letra **MAIÚSCULA**
- **Não utilize acentuação** nos nomes dos atributos e tabelas

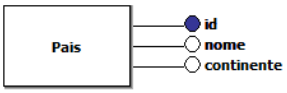
14

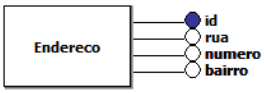
Prática SQL 01

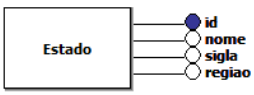
- Crie as tabelas seguir no seu MySQL. Salve o script com o nome **pratica01** e crie um banco de dados chamado **pratica_clientes**











- Organize seus scripts para consultas posteriores!

15

Chaves estrageiras no SQL

- As chaves estrangeiras são atributos do tipo INTEGER e devem possuir um COMANDO que o identifica como FOREIGN KEY

- Para criar uma Chave Estrangeira devemos aplicar alguns comandos, veja as regras para isso:

- Regra nº 1: Cria-se um atributo simples do tipo INTEGER que SERÁ a Chave Estrangeira;
- Regra nº 2: O nome do atributo será o MESMO da Chave Primária a que ele se refere (Origem);
- Regra nº 3: No final do atributo deve-se adicionar a fim de identificá-lo como Chave Estrangeira através do nome do atributo; _fk
- Regra nº 4: Na próxima linha cria-se o comando FOREIGN KEY que transformará o atributo id_cli_fk em uma Chave Estrangeira de fato;

16

Chaves estrageiras no SQL

- EXEMPLO:
- Nome da Chave Primária de Origem:
  - `id_cli INTEGER`
- Nome do Chave Estrangeira:
  - `id_cli_fk INTEGER`

➢ **EXEMPLO:**

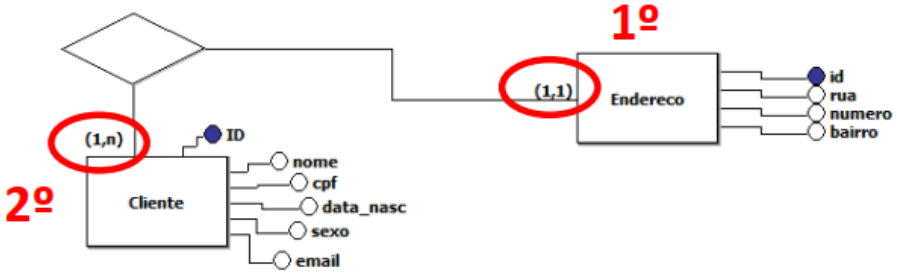
✓ **FOREIGN KEY (id_cli_fk) | REFERENCES | Cliente (id_cli);**

Diz para o SGBD qual atributo será a chave estrangeira	Faz a referencia entre as tabelas	Diz para o SGBD de qual TABELA e ATRIBUTO será a referencia

17

Chaves estrageiras no SQL

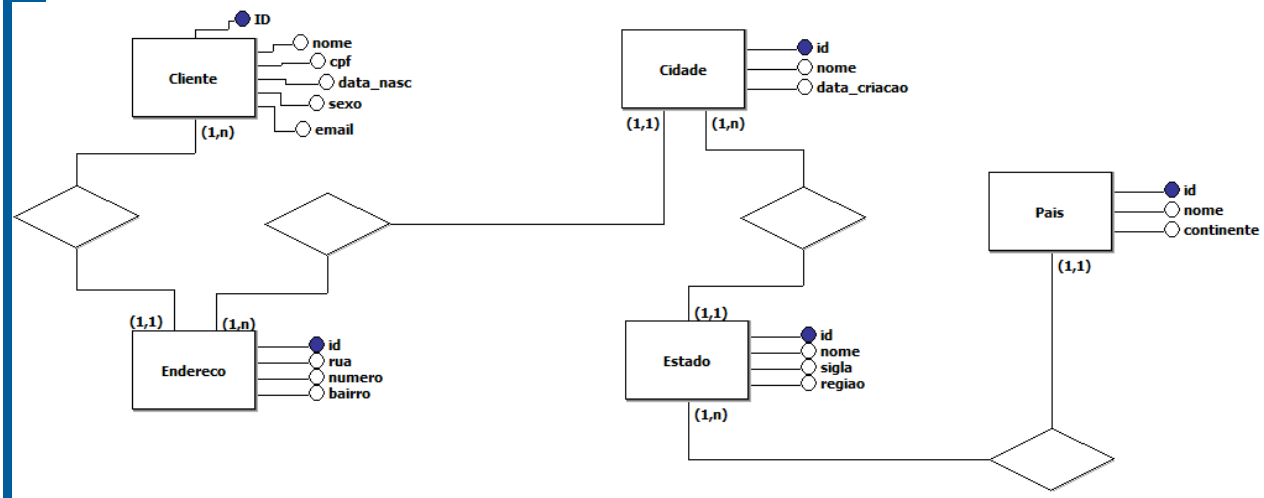
- Na criação das Chaves Estrangeiras só podemos fazer referência a uma tabela que JÁ EXISTE



18

Prática SQL 01, parte 2

- Criar as entidades com suas devidas chaves estrangeiras



19

DDL: ALTER TABLE

- Usando o comando ALTER é possível realizar as alterações na estrutura de uma tabela, tais como:
  - Adicionar Atributos
  - Excluir Atributos
  - Alterar o tipo e o nome de um atributo já existente
- Para cada tipo de alteração, existe um operador específico a ser utilizado no comando

20

DDL: ALTER TABLE

- Sintaxe do Comando:
  - `ALTER TABLE nome_tabela`
  - `OPERADOR nome_atributo (...);`
- Para cada tipo de alteração, existe um operador específico a ser utilizado no comando
- Tipos de Operadores:
  - Adicionar: ADD
  - Excluir: DROP
  - Alterar: ALTER

21

DDL: ALTER TABLE

- O operador ADD é utilizado no comando ALTER TABLE para adicionar um novo atributo em uma tabela já existente
- Sintaxe:
  - `ALTER TABLE name_table ADD column_name data_type;`
- Exemplos:
  - `ALTER TABLE Cliente ADD data_nascimento_cli DATE;`
  - `ALTER TABLE Cliente ADD renda_cli FLOAT;`
  - `ALTER TABLE Cliente ADD indicacao_cli VARCHAR(80);`

22

DDL: ALTER TABLE

- O operador CHANGE é utilizado no comando ALTER TABLE para alterar um atributo em uma tabela já existente
- Com ele podemos mudar apenas o nome, o tipo ou o nome e o tipo ao mesmo tempo
- Sintaxe:
- ```
  ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type
  ```

23

DDL: ALTER TABLE

- Exemplo para alterar o NOME do atributo:
  - ```
    ALTER TABLE table_name
    ```
  - ```
    RENAME COLUMN column_name TO new_column_name;
    ```
- Exemplo para alterar o TIPO do atributo:
  - ```
    ALTER TABLE table_name
    ```
  - ```
    MODIFY column_name data_type;
    ```

24
    ```

ADD FK após o CREATE TABLE

- Podemos criar as tabelas sem as chaves estrangeiras / foreign keys, e fazer uma atualização na tabela posteriormente para adicionar as FKs

- Para isso, é necessário criar o atributo para a FK e posteriormente transformar ele em FK:

```
ALTER TABLE table_name
ADD column_name_fk INTEGER;
```

```
ALTER TABLE table_name
ADD FOREIGN KEY(column_name_fk)
REFERENCES origin_table_name(column_name_pk);
```

25

DDL: operador DROP

- O operador DROP é utilizado no comando ALTER TABLE para excluir um atributo em uma tabela já existente;

- Sintaxe:

```
ALTER TABLE table_name DROP column_name;
```

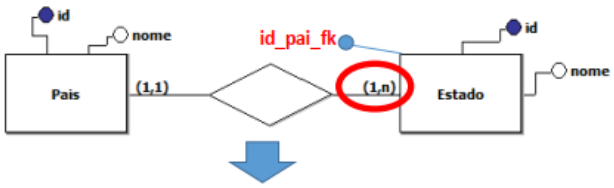
- Exemplos:

```
ALTER TABLE Cliente DROP nome_cli;
ALTER TABLE Cliente DROP email_cli;
```

26

DDL: operador DROP

- Não é possível excluir uma tabela que possua registros dependentes de registros de outra tabela!
- Exemplo:



Pais	
id_pai	nome_pai
1	Brasil
2	Chile

Estado		
id_est	nome_est	id_pai_fk
1	São Paulo	1
2	Rio de Janeiro	1

Pergunta: É possível **excluir** a tabela PAÍS?

Resposta: **NÃO**, porque o registro com id nº 1 está vinculado com um ou mais registros na tabela Estado. Ou seja, para que os registros nº 1 e 2 da tabela Estado exista eles dependem de um registro (FK) de Pais.

Pergunta: É possível **excluir** a tabela Estado?

Resposta: **SIM**, porque nenhum registro da tabela Estado possui o id vinculado a um ou mais registros da tabela Pais;

27

Prática SQL 01, parte 03

- No banco de dados **locadora**, faça:

1. Adicione o atributo **RG** na tabela Cliente;
2. Adicione o atributo **TELEFONE** na tabela Cliente;
3. Adicione o atributo **REFERENCIA** na tabela Endereco;
4. Exclua o atributo **CONTINENTE** na tabela Pais;
5. Altere o nome do atributo **SEXO** para **GENERO**;
6. Altere o tipo de dados do atributo **DATA_NASC** de **DATE** para **DATETIME** em Cliente;

28

Prática SQL 02, parte 01

- Crie o modelo físico, com os devidos comandos SQL/DDL, dos seguintes modelos conceituais:
  - **Hospital**,
  - Campeonato de **Futebol** e
  - Agência de **Viagens**
- Criar o Modelo Físico a partir do Modelo Conceitual. Serão avaliados os seguintes quesitos:
  - Relacionamentos
  - Cardinalidades e possíveis desmembramentos de entidades
  - Normalização de Dados (deve estar na 3FN)
  - Utilização correta (inclusive boas práticas) na criação do script em SQL
- Enviar o script SQL e modelo relacional atualizado via Tarefa Teams

29

DML: comando INSERT

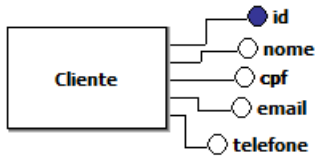
- O comando INSERT é utilizado para inserir um registro em uma tabela, que equivale a uma linha de uma tabela, fazendo novamente uma analogia a uma tabela do Excel
- Existem duas formas de fazer um INSERT, fazendo referencia ou não aos atributos. Vamos analisar as duas formas:
  - **COM Referência ao Atributo**: É possível escolher QUAIS atributos serão preenchidos no momento da inserção dos dados
  - **SEM Referência ao Atributo**: Não é possível escolher QUAIS atributos serão preenchidos, assim, TODOS os atributos devem receber um valor

30

DML: comando INSERT

- Sintaxe – COM Referencia:

```
INSERT INTO Cliente (id_cli_pk, nome_cli)
VALUES (1, 'José da Silva');
INSERT INTO Cliente (id_cli_pk, nome_cli, telefone_cli)
VALUES (2, 'Ana Maria', '699999 8888');
INSERT INTO Cliente (id_cli_pk, nome_cli, email_cli)
VALUES (3, 'Gustavo Silva', 'gustavo@gmail.com');
```



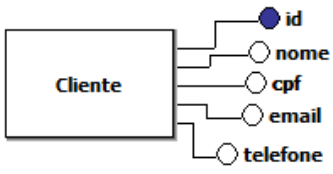
Cliente				
id_cli	nome_cli	cpf_cli	email_cli	telefone_cli
1	José da Silva	null	null	null
2	Ana Maria	null	null	69 9999 8888
3	Gustavo Silva	null	gustavo@gmail.com	Null

31

DML: comando INSERT

- Sintaxe – COM Referencia:

```
INSERT INTO Cliente VALUES (1, 'José da Silva', null, null, null);
INSERT INTO Cliente VALUES (2, 'Ana Maria', null, null, '69 9999 8888');
INSERT INTO Cliente VALUES (3, 'Gustavo Silva', null, 'gustavo@gmail.com', null);
INSERT INTO Cliente VALUES (4, 'Marcos Pereira', '123.456.789-15',
'pereira@gmail.com', '69 98888 7777');
```



Cliente				
id_cli	nome_cli	cpf_cli	email_cli	telefone_cli
1	José da Silva	null	null	null
2	Ana Maria	null	null	69 9999 8888
3	Gustavo Silva	null	gustavo@gmail.com	Null
4	Marcos Pereira	123.456.789-15	pereira@gmail.com	69 98888 7777

32

DML: regras do INSERT

- Registros do tipo VARCHAR, DATE e TIME são inseridos entre **aspas simples**
  - Exemplo: ‘Jackson Henrique’, ‘1987-06-30’, ‘08:00:00’;
- Chaves primárias não se repetem! Se repetir o SGBD retornará um erro!
- SE você utilizar **auto_increment** na criação do atributo da Chave Primária, não será preciso digitar o número da correspondente a Chave Primária, somente o valor NULL deve ser digitado.
  - Exemplo: INSERT INTO Cliente VALUES (null, ‘José da Silva’, null, null, null);
- Atributos com not null são obrigatórios e sem not null são opcionais;

33

DML: regras do INSERT

- Registros do tipo INTEGER, FLOAT ou DOUBLE são inseridos sem aspas. Exemplo: 500, 500.65 ou 5123.45
- Registros do tipo FLOAT ou DOUBLE são inseridos com ponto no lugar da vírgula na separação entre a casa decimal do valor inteiro.
  - Exemplo:
    - Mundo Real: R$ 1.550,50
    - Banco de Dados: 1550.50
- Só mencione uma Chave Estrangeira se a Chave Primária a que ela se referencia REALMENTE EXISTIR na TABELA de ORIGEM

34

DML: auto_increment

```
CREATE TABLE Animals (
     id_ani_pk MEDIUMINT NOT NULL AUTO_INCREMENT,
     name_ani CHAR(30) NOT NULL,
     PRIMARY KEY (id_cli_pk)
);

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich’);

INSERT INTO animals (id_ani_pk, name_ani) VALUES(0, 'groundhog’);
INSERT INTO animals (id_ani_pk, name_ani) VALUES(NULL, 'squirrel');
```

35

Prática SQL 01, parte 04

- Dando continuidade ao banco de dados **locadora**, faça:
    - Insira 02 registros na tabela **Pais** mencionando os atributos
    - Insira 02 registros na tabela **Pais** sem mencionar os atributos
    - Insira 02 registros na tabela **Estado** mencionando os atributos
    - Insira 02 registros na tabela **Estado** sem mencionar os atributos
    - Insira 02 registros na tabela **Cidade** mencionando os atributos
    - Insira 02 registros na tabela **Cidade** sem mencionar os atributos
    - Insira 02 registros na tabela **Endereco** mencionando os atributos
    - Insira 02 registros na tabela **Endereco** sem mencionar os atributos
    - Insira 02 registros na tabela **Cliente** mencionando os atributos
    - Insira 02 registros na tabela **Cliente** sem mencionar os atributos

36

DML: comando UPDATE

- O comando UPDATE é utilizado para alterar um ou vários registros em uma tabela, lembrando que um registro equivale a uma linha de uma tabela, fazendo novamente uma analogia a uma tabela do Excel

- Sintaxe:

```
UPDATE nome_da_tabela
SET atributo = valor
WHERE condição;
```

- Mais antes utilizar o UPDATE é preciso compreender o conceito de CONDIÇÃO, pois é ela que dirá qual ou quais registros serão alterados

37

DML: comando UPDATE

- Uma condição é uma expressão lógica formada por valores e operadores de comparação que deverá retornar um valor, podendo este valor ser Verdadeiro (V) ou de Falso (F);

- Caso o resultado da comparação destes valores seja verdadeiro, dizemos que essa condição é verdadeira e um comando poderá ser executado;

- Caso o resultado seja negativo, o comando não será executado;

- Os valores de uma condição, que serão comparados, podem ser representados por: ✓Números Inteiros ✓Números com Casa Decimal ✓Datas ✓Horas ✓Textos

38

DML: comando UPDATE

- Os comparadores, ou seja sinais de comparação utilizados no SQL, a princípio serão:

  = IGUAL
  > MAIOR
  < MENOR
  >= MAIOR OU IGUAL
  <= MENOR OU IGUAL
  <> DIFERENTE

39

DML: comando UPDATE

- Cada condição é composta por um valor, um sinal de comparação e um segundo valor, tudo entre parênteses:

- Sintaxe da Condição:

  - (valor1 operador valor2)

- Exemplos de Condição:

  - (18 > 20) = Falso
  - (‘André’ = ‘And’) = Falso
  - (‘1987-06-30’ < ‘2020-09-06’) = Verdadeiro
  - (45 = 45) = Verdadeiro

40

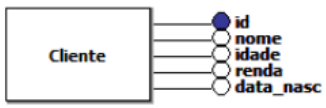
DML: comando UPDATE

- Exemplo de Condição com Atributo:
  - (media > 60) = ?
  - SE o valor do atributo media for 80 então a condição é **Verdadeira**
  - SE o valor do atributo media for 50 a condição é **Falsa**
- LEMBRE-SE: os valores de uma condição podem ser valores reais ou representado por um atributo

41

DML: comando UPDATE

- No comando UPDATE a condição é essencial para determinar QUAL ou QUAIS registros serão alterados. Vamos ao exemplo:
- Considere o seguinte banco de dados:



Cliente				
id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

42

DML: comando UPDATE

- Objetivo 1: Altere o nome do cliente nº 3 para 'Gustavo H. Silva'
- Comando 1: **`UPDATE Cliente SET nome_cli = 'Gustavo H. Silva' WHERE (id_cli = 3);`**
- Objetivo 2: Aumente em 1000 reais a renda dos clientes com mais de 30 anos
- Comando 2: **`UPDATE Cliente SET renda_cli = renda_cli + 1000 WHERE (idade_cli > 30);`**

Cliente – ANTES UPDATE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	31	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – APÓS UPDATE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	2500.50	1987-01-30
2	Ana Maria	31	3500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	6000.00	1987-06-30

43

DML: comando UPDATE sem condição ⚠

- Em algumas raras situações podemos utilizar o Update sem uma condições
- MAS LEMBRE-SE: Sem condição, todos os registros da tabela serão afetados
- Objetivo: Aumente a renda dos clientes em 10%
- Comando: **`UPDATE Cliente SET renda_cli = renda_cli * 1.1`**

Cliente – ANTES UPDATE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	31	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – APÓS UPDATE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1650.55	1987-01-30
2	Ana Maria	31	2750.00	1990-02-20
3	Gustavo H. Silva	20	5500.00	2000-01-31
4	Marcos Pereira	27	1122.00	1993-06-21
5	Thiago Souza	33	5500.00	1987-06-30

44

Prática SQL 01, parte 05

- Continue as práticas no banco de dados **locadora**

1. Teste os updates realizados nos slides anteriores
2. Crie um update que mude o nome do cliente nº 2 para ‘Lucas Matos’
3. Crie um update que aumentar a renda em 20% dos clientes nascidos antes do ano 2000
4. Crie um update para diminuir a renda em 27% dos clientes com renda maior ou igual a 5000 reais
5. Crie um update para alterar a data de nascimento do cliente nº 1

45

DML: UPDATE com múltiplas condições

- É possível utilizarmos mais de uma condição no comando Update para personalizar melhor quais registros serão alterados
- Para isso é preciso separar as condições com operadores lógicos, são eles o AND (E) e o OR (OU)
- O uso dos operadores lógicos define quais condições devem ser verdadeiras para que o comando seja realizado
  - **AND**: Todas as condições devem ser Verdadeiras
  - **OR**: Pelo menos uma das condições precisa ser verdadeira

46

DML: UPDATE com múltiplas condições

- **Sintaxe com AND:** V V
  - ***UPDATE*** *nome_da_tabela* ***SET*** *atributo = valor* ***WHERE*** *(condição)* ***AND*** *(condição);*

Update Executado

- ***Sintaxe com OR:*** V F
  - ***UPDATE*** *nome_da_tabela* ***SET*** *atributo = valor* ***WHERE*** *(condição)* ***OR*** *(condição);*

Update Executado

- **Sintaxe com AND:** F V
  - ***UPDATE*** *nome_da_tabela* ***SET*** *atributo = valor* ***WHERE*** *(condição)* ***AND*** *(condição);*

Update NÃO Executado

- ***Sintaxe com OR:*** F F
  - ***UPDATE*** *nome_da_tabela* ***SET*** *atributo = valor* ***WHERE*** *(condição)* ***OR*** *(condição);*

Update NÃO Executado

47

DML: UPDATE com múltiplas condições

- Objetivo: Considerando a tabela Cliente novamente é preciso aumentar em 20% a renda dos cliente com idade entre 20 e 30 anos.
- Comando:
  - `UPDATE Cliente SET renda_cli = renda_cli * 1.2`
  - `WHERE (idade_cli >= 20) AND (idade_cli <= 30);`

Cliente – ANTES UPDATE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – APÓS UPDATE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	3000.00	1990-02-20
3	Gustavo H. Silva	20	6000.00	2000-01-31
4	Marcos Pereira	27	1224.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

48

DML: UPDATE com múltiplas condições

- Esse se substituíssemos o AND pelo OR? O que aconteceria?
- Comando:
  - `UPDATE Cliente SET renda_cli = renda_cli * 1.2`
  - `WHERE (idade_cli >= 20) OR (idade_cli <= 30);`

Cliente – ANTES UPDATE				
id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – APÓS UPDATE				
id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1800.60	1987-01-30
2	Ana Maria	30	3000.00	1990-02-20
3	Gustavo H. Silva	20	6000.00	2000-01-31
4	Marcos Pereira	27	1224.00	1993-06-21
5	Thiago Souza	33	6000.00	1987-06-30

49

DML: UPDATE com múltiplas condições

- Objetivo: Aumentar em 20% a renda dos cliente com idade entre 20 e 30 anos e com renda menor do que 6 mil reais;
- Comando:
  - `UPDATE Cliente SET renda_cli = renda_cli * 1.2`
  - `WHERE (idade_cli >= 20) AND (idade_cli <= 30) AND (renda_cli < 6000);`

Cliente – ANTES UPDATE				
id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	6000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – APÓS UPDATE				
id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	3000.00	1990-02-20
3	Gustavo H. Silva	20	6000.00	2000-01-31
4	Marcos Pereira	27	1224.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

50

DML: UPDATE com múltiplas condições

- Objetivo: Aumentar em 80% a renda dos clientes com idade menor que 30 anos ou maior do que 50 anos, e nascidos após o ano 2000;

- Comando:

  - `UPDATE Cliente SET renda_cli = renda_cli * 1.8`
  - `WHERE ((idade_cli < 30) OR (idade_cli > 50)) AND (data_nasc_cli >= '2000-01-01');`

Cliente – ANTES UPDATE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – APÓS UPDATE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	9000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

51

Prática SQL 01, parte 06

- Use o banco de dados **locadora** criado nas práticas anteriores e faça:

1. Teste os comandos explicados anteriormente;
2. Altere para 50 a idade dos clientes com ID entre 2 e 5 ou nascidos entre 1970 e 1980;
3. Diminua 500 reais na renda dos clientes nascidos entre 1980 e 2000 que tenha renda superior a 1000 reais e idade menor que 50;
4. Aumente 1000 reais na renda dos clientes com idade entre 10 e 20 anos ou 40 e 60 anos.

52

DML: comando DELETE

- O comando DELETE é utilizado para apagar um ou vários registros em uma tabela, lembrando que um registro equivale a uma linha de uma tabela, fazendo novamente uma analogia a uma tabela do Excel
- Sintaxe:
  - **`DELETE FROM nome_da_tabela`**
  - **`WHERE (condição);`**
- No DELETE não é possível excluir registros que possuam chaves primárias vinculadas a chaves estrangeiras em outras tabelas

53

DML: comando DELETE ⚠

- CUIDADO: comandos DELETE sem uma condição (WHERE) excluem todos os registros de uma tabela. **Nunca são utilizados na prática**.
- Exemplo:
  - `DELETE FROM Cliente;`
- Observações:
  - Não existe o comando DELETE de somente um atributo específico
  - O comando DELETE apaga toda a linha do registro selecionado na condição

54

DML: comando DELETE

- No comando DELETE a condição é essencial para determinar QUAL ou QUAIS registros serão apagados. Vamos ao exemplo:
- Objetivo: Exclua o cliente com ID nº 1
- Comando: **`DELETE FROM Cliente WHERE (id_cli = 1);`**

Cliente – ANTES DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – DEPOIS DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

55

DML: comando DELETE

- No comando DELETE a condição é essencial para determinar QUAL ou QUAIS registros serão apagados. Vamos ao exemplo:
- Objetivo: Exclua os clientes com renda menor do que 1500 reais
- Comando: **`DELETE FROM Cliente WHERE (renda_cli < 1501);`**

Cliente – ANTES DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – DEPOIS DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

56

Prática SQL 01, parte 07

- Teste os comandos vistos anteriormente no banco de dados **praticas_clientes**
    - Você mesmo irá testar situações em seu banco de dados

57

Operadores de comparação especiais

- Além dos Operadores de Comparação aprendidos até aqui (MAIOR, MENOR, IGUAL e DIFERENTE) podemos utilizar alguns Operadores Especiais na cláusula WHERE para personalizar melhor as condições
- Os operadores especiais que iremos aprender são:
    - **BETWEEN**
    - **LIKE**

58

Operadores BETWEEN

- O operador BETWEEN pode ser usado para verificar se o valor de um atributo está em um intervalo de valores;
- Facilita a construção de condições que buscam delimitar intervalos de valores. Pode ser usado nos comandos UPDATE, DELETE e SELECT;
- Sintaxe da Condição com BETWEEN:
  - **(atributo BETWEEN valor1 AND valor2);**
- Exemplo:
  - **(renda_cli BETWEEN 500 AND 1000);**

59

Operadores BETWEEN

- Objetivo: Exclua os clientes com idade entre 30 e 40 anos.
- Comando:

```
DELETE FROM Cliente
WHERE (idade_cli BETWEEN 30 AND 40);
```

Cliente – ANTES DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – DEPOIS DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21

60

Operadores BETWEEN

- Objetivo: Exclua os clientes com idade entre 30 e 40 anos e que possuam renda entre 500 e 2000 reais.

- Comando:

  - DELETE FROM Cliente
  - WHERE (idade_cli **BETWEEN** 30 AND 40)
  - AND (renda_cli **BETWEEN** 500 AND 2000);

Cliente – ANTES DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – DEPOIS DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

61

Operadores LIKE

- O operador LIKE pode ser utilizado para comparar cadeias de caracteres usando padrões de comparação para um ou mais caracteres

- Utiliza o símbolo percentual % como um coringa que substitui um ou mais caracteres desconhecidos e o coringa underline _ que substitui um único caractere desconhecido

- O LIKE pode ser usado em valores entre aspas simples, ou seja, atributos do tipo VARCHAR, TIME, DATE e DATETIME

62

Operadores LIKE

- Sintaxe da Condição com LIKE:
  - `(atributo LIKE valor)`
- Exemplos:
  - `(nome_cli LIKE 'André%')`
  - `(nome_cli LIKE '%Nascimento%')`
  - `(nome_cli LIKE '%Silva')`
  - `(data_nasc_cli LIKE '1987%')`
  - `(hora_nasc_cli LIKE '21%')`

63

Operadores LIKE

- Objetivo: Exclua os clientes com sobrenome Silva.
- Comando:

```
DELETE FROM Cliente
WHERE (nome_cli LIKE '%Silva%');
```

Cliente – ANTES DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – DEPOIS DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
2	Ana Maria	30	2500.00	1990-02-20
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

64

Operadores LIKE

- Objetivo: Exclua os clientes que nasceram em 1990.
- Comando:
  - `DELETE FROM Cliente`
  - `WHERE (data_nasc_cli LIKE '1990%');`

Cliente – ANTES DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – DEPOIS DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

65

Operadores LIKE

- Objetivo: Exclua o cliente com o nome Jose da Silva. Porém garanta que os nomes acentuados também sejam excluídos
- Comando:
  - `DELETE FROM Cliente`
  - `WHERE (nome_cli LIKE 'Jos_ da Silva');`

Cliente – ANTES DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

Cliente – DEPOIS DELETE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

66

Prática SQL 01, parte 08

- Teste todos os comandos anteriores;
- Usando **BETWEEN** e **LIKE** faça os comandos a seguir:
  1. Exclua os clientes com renda entre 500 e 2000 e nascidos em 1987
  2. Exclua os clientes com sobrenome Pereira ou Souza e que possuam idade entre 20 e 30 anos
  3. Exclua os clientes com idade entre 10 e 60 anos e com renda entre 1000 e 10000 e nascidos em 2000
- Enviar o Script SQL no Teams

67

DML: comandos SELECT

- Uma das principais funções de um banco de dados é possibilitar a consulta dos registros armazenados nas tabelas
- Para consultar um ou muitos registros utilizamos o comando SELECT
- O comando SELECT não modifica nenhum registro, apenas mostra para o usuário os registros armazenados na tabela de acordo com a condição
- O comando SELECT pode ser usando sem condição, neste caso, todos os registros da tabela serão selecionados;

68

DML: comandos SELECT

- Sintaxe:
  - `SELECT atributo1, atributo2, ...`
  - `FROM nome_da_tabela`
  - `WHERE (condição);`
- Também é possível omitir os nomes dos atributos usando o símbolo asterisco *. Com o * no lugar atributos todos os atributos serão selecionados;
- Sintaxe com *:
  - `SELECT * FROM nome_da_tabela`
  - `WHERE (condição);`

69

DML: comandos SELECT

- Objetivo: Selecione o id, o nome e a renda dos clientes com renda superior a 1600 reais
- Comando:
  - `SELECT id_cli, nome_cli, idade_cli`
  - `FROM Cliente`
  - `WHERE (renda_cli > 1600);`

TABELA CLIENTE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

RESULTADO DA CONSULTA

id_cli	nome_cli	renda_cli
2	Ana Maria	2500.00
3	Gustavo H. Silva	5000.00
5	Thiago Souza	5000.00

70

DML: comandos SELECT

- Objetivo: Selecione os clientes com renda superior a 1600 reais;
- Comando:
  - `SELECT * FROM Cliente`
  - `WHERE (renda_cli > 1600);`

TABELA CLIENTE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

RESULTADO DA CONSULTA

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
5	Thiago Souza	33	5000.00	1987-06-30

71

DML: comandos SELECT com ORDER BY

- Podemos utilizar a palavra-chave ORDER BY para ordenar os registros selecionadas em uma consulta a partir de um atributo
- O ORDER BY é usado sempre no final da comando SELECT
- Caso queira ordena os valores de forma decrescente basta adicionar a palavra DESC após o nome do atributo ordenador
- Sintaxe:
  - `SELECT atributo1, atributo2, ...`
  - `FROM nome_tabela WHERE (condição)`
  - `ORDER BY atributo;`

72

DML: comandos SELECT com ORDER BY

- Objetivo: Selecione os clientes com renda maior do que 1000 e ordenados pelo nome;
- Comando:
  - `SELECT * FROM Cliente`
  - `WHERE (renda_cli > 1000)`
  - `ORDER BY nome_cli;`

TABELA CLIENTE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

RESULTADO DA CONSULTA

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
2	Ana Maria	30	2500.00	20/02/1990
3	Gustavo H. Silva	20	5000.00	31/01/2000
1	José da Silva	33	1500.50	30/01/1987
4	Marcos Pereira	27	1020.00	21/06/1993
5	Thiago Souza	33	5000.00	30/06/1987

73

Operadores especiais: IS NULL

- O operador IS NULL é usado em uma condição para selecionar valores que sejam NULL (nulos) ou NOT NULL (não nulos);
- Importante destacar que no MySQL os valores NULL não podem ser comparados com o operador = (igual), somente com o IS;
- Sintaxe:
  - `(atributo IS NULL) ou (atributo IS NOT NULL)`
- Exemplo:
  - `(nome_cli IS NULL)`

74

Operadores especiais: IS NULL

- Objetivo: Selecione o id, nome e renda dos clientes que não possuem renda, ou seja, que estão nulas.
- Comando:
  - `SELECT id_cli, nome_cli, idade_cli FROM Cliente`
  - `WHERE (renda_cli IS NULL);`

TABELA CLIENTE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	null	1987-06-30

RESULTADO DA CONSULTA

id_cli	nome_cli	renda_cli
5	Thiago Souza	null

75

Funções especiais

- As Funções Especiais no MySQL retornam um ou vários registros de acordo com o atributo de entrada da função
- Toda função tem um objetivo específico. A forma como esse objetivo é cumprido é de responsabilidade do SGBD
- Toda função tem uma entrada. Essa entrada é o atributo da tabela que está sendo selecionada
- As funções são utilizadas entre o SELECT e o FROM, substituindo um atributo da tabela foco da seleção

76

Funções especiais

- Sintaxe do SELECT com Função;
  - `SELECT Nome_Função(atributo)`
  - `FROM nome_tabela`
  - `WHERE (condição);`
- As principais funções no MySQL são:
  - COUNT(atributo) – Retorna a quantidade total de registros não nulos de um atributo
  - SUM(atributo) – Função que retorna a soma dos valores de um atributo
  - AVG(atributo) – Função que retorna a média dos valores de um atributo
  - MIN(atributo) – Função que retorna o menor valor de um atributo
  - MAX(atributo) – Função que retorna o maior valor de um atributo

77

Funções especiais: COUNT

- Objetivo: selecione e mostre quantos clientes estão armazenados no banco de dados.
- Comando:
  - `SELECT COUNT(id_cli_pk)`
  - `FROM Cliente;`

TABELA CLIENTE

id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

RESULTADO DA CONSULTA

COUNT(id_cli)
5

78

Funções especiais: SUM

- Objetivo: selecione e mostre a soma das rendas de todos os clientes armazenados no banco de dados.
- Comando:
  - `SELECT SUM(renda_cli)`
  - `FROM Cliente;`

TABELA CLIENTE				
id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

RESULTADO DA CONSULTA

SUM(renda_cli)
15020.50

79

Funções especiais: AVG

- Objetivo: Selecione e mostre a idade média dos clientes armazenados no banco de dados.
- Comando:
  - `SELECT AVG(idade_cli)`
  - `FROM Cliente;`

TABELA CLIENTE				
id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

RESULTADO DA CONSULTA

AVG(idade_cli)
28.6

80

Funções especiais: MAX

- Objetivo: Selecione e mostre a maior renda entre os clientes armazenados no banco de dados.
- Comando:
  - `SELECT MAX(renda_cli)`
  - `FROM Cliente;`

TABELA CLIENTE				
id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

RESULTADO DA CONSULTA

MAX(renda_cli)
5000.00
5000.00

81

Funções especiais: MIN

- Objetivo: selecione e mostre a menor renda entre os clientes armazenados no banco de dados.
- Comando:
  - `SELECT MIN(renda_cli)`
  - `FROM Cliente;`

TABELA CLIENTE				
id_cli	nome_cli	idade_cli	renda_cli	data_nasc_cli
1	José da Silva	33	1500.50	1987-01-30
2	Ana Maria	30	2500.00	1990-02-20
3	Gustavo H. Silva	20	5000.00	2000-01-31
4	Marcos Pereira	27	1020.00	1993-06-21
5	Thiago Souza	33	5000.00	1987-06-30

RESULTADO DA CONSULTA

MIN(renda_cli)
1020.00

82

Funções especiais: REGRAS

1. Funções não podem ser usadas em conjunto com atributos no mesmo SELECT.
   - Exemplo: `SELECT nome_cli, MAX(renda_cli) FROM Cliente;`
2. Funções não podem ser usadas dentro de condições.
   - Exemplo: `SELECT * FROM Cliente WHERE (renda_cli > MAX(renda_cli))`
3. As Funções AVG, COUNT e SUM retornam apenas um valor
4. Funções só podem ser utilizadas com valores numéricos, exceto COUNT

83

Prática SQL 01, parte 09

- No banco de dados **locadora**, script um script para o que se pede e envie no Tarefas no Teams:
  1. Realize, no mínimo, 10 inserts em cada entidade (os inserts devem conter dados para que as consultas abaixo apresentem, no mínimo, uma linha de retorno)
  2. Realize uma consulta para exibir todos os clientes que não moram no Brasil
  3. Realize uma consulta para exibir todas as cidades da Bahia
  4. Realize uma consulta para exibir todos os clientes que nasceram após 2020
  5. Realize uma consulta para exibir todos os clientes do sexo masculino, que nasceram antes de 1975

84

Prática para o Projeto Integrador

- A turma será dividida em três equipes
- Cada equipe ficará responsável por um projeto do PI (baseado nos protótipos do Figma)
- Entregar, via e-mail:
  - Modelo Conceitual, desenvolvido no BR Modelo
  - Dicionário de Dados, elaborado em planilha
  - Script SQL do Modelo Físico, contendo:
    - 6 registros em cada tabela
    - 2 updates e 2 deletes, inclusive usando operadores especiais
    - 4 selects com joins, envolvendo uma situação que deverá ser comentada
    - 2 triggers, 2 stored procedures e 2 fuctions (analise a melhor situação para essas aplicações)
    - O arquivo deve estar organizado para ser executado em sequência, sem erros de execução
- Entrega e apresentação em 29/01/2025

85

Regras básicas do SELECT

- Uma das principais atividades de um Desenvolvedor de Software é gerar consultas personalizadas sobre os registros disponíveis no banco de dados
- A razão de utilizar sistemas comerciais vai além do controle de processos. Um sistema deve gerar informação e conhecimento para os usuários
- Esse conhecimento é gerado através de consultas (selects) na base de dados
- Basicamente existem dois tipos de consultas:
  - **Consulta Regular**: por comparação direta de chaves
  - **Consulta por Junção**: com utilização de JOINS

86

Regras básicas do SELECT

- O usuário do sistema não sabe e não precisa saber o que é Chave Primária, Estrangeira, Atributo, Tipo de dados, entre outros
- As informações devem ser mostradas na mesma linguagem do usuário, para que ele compreenda o conhecimento gerado pelo sistema
- Uma consulta jamais deve:
  - Mostrar número de Chave Estrangeira
  - Mostrar nome de Atributo

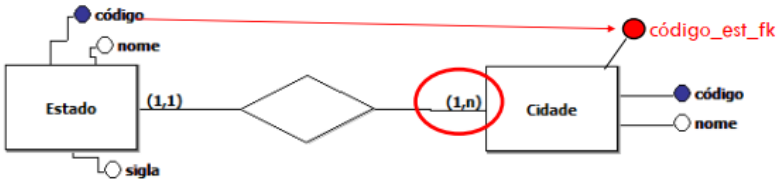
87

SELECT com múltiplas tabelas

- Duas tabelas estão relacionadas quando possuem um relacionamento N para 1. Assim a tabela com o N recebe a chave estrangeira (FK) da tabela com o 1
- No SELECT por comparação direta devemos comparar o atributos comum entre as 02 tabelas relacionadas
- Ou seja, comparamos a chave primária da tabela com o 1 com chave estrangeira que está na tabela com o N

88

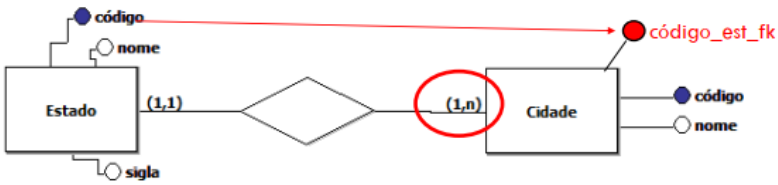
SELECT com múltiplas tabelas



- Podemos afirmar que:
  - Um (1) Estado está relacionado a muitas (N) Cidades;
  - Uma (1) Cidade está relacionada a Um (1) Estado;
- Logo:
  - cidade está com o N, então recebe a Chave Estrangeira de Estado que está com o 1;

89

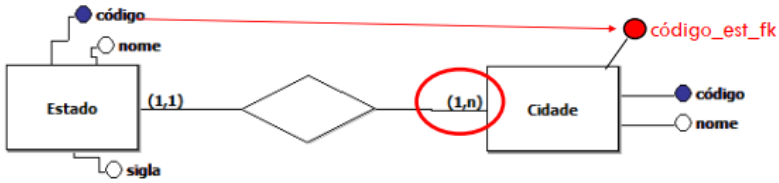
SELECT com múltiplas tabelas



- Sintaxe:
  - `SELECT tabela1.atributo1, tabela2.atributo2`
  - `FROM tabela1, tabela2`
  - `WHERE (tabela1.atributoPK = tabela2.atributoFK);`

90

SELECT com múltiplas tabelas



- **Tarefa**: Selecione todos os registros de Cidade, porém, substitua a Chave Estrangeira de Estado pelo Nome do Estado.

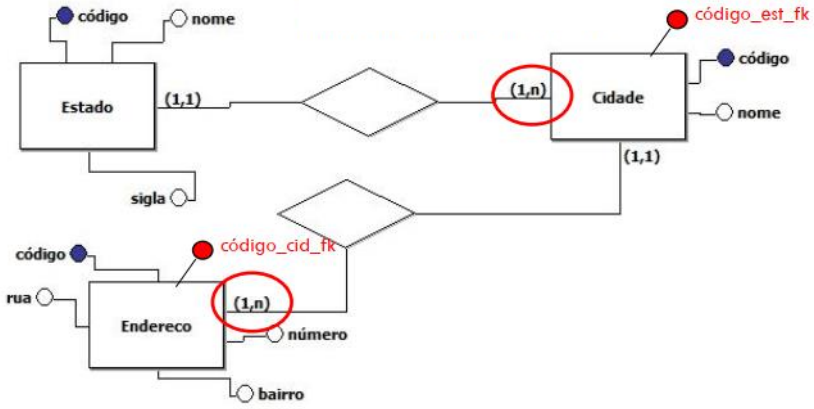
- **Solução**:

```
SELECT CIDADE.nome_cid AS 'Nome Cidade', ESTADO.nome_est AS 'Nome Estado'
FROM CIDADE, ESTADO
WHERE (ESTADO.cod_est_pk = CIDADE.cod_est_fk);
```

91

SELECT com três tabelas

- **Tarefa**: Selecione o nome do Estado, nome da Cidade, rua, número e bairro da Cidade.



92

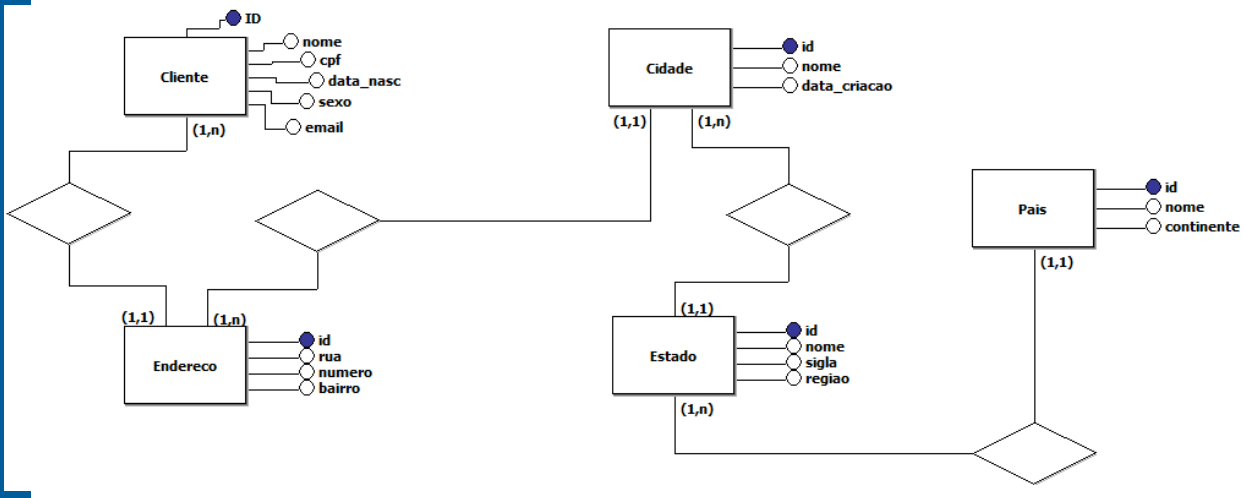
SELECT com três tabelas

- **Solução:**

```
SELECT
CIDADE.nome_cid AS 'Nome Cidade',
ESTADO.nome_est AS 'Nome Estado',
ENDERECO.rua_end AS 'Rua',
ENDERECO.numero_end AS 'Nº',
ENDERECO.bairro_end AS ‘Bairro’
FROM
CIDADE, ESTADO, ENDERECO
WHERE
(ESTADO.cod_est_pk = CIDADE.cod_est_fk) AND
(CIDADE.cod_cid_pk = ENDERECO.cod_cid_fk);
```

93

Prática SQL 01, parte 10



94

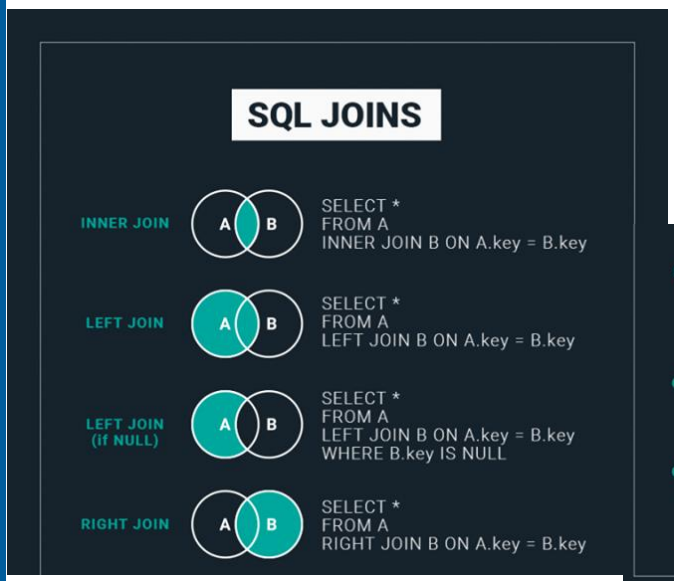
Prática SQL 01, parte 10

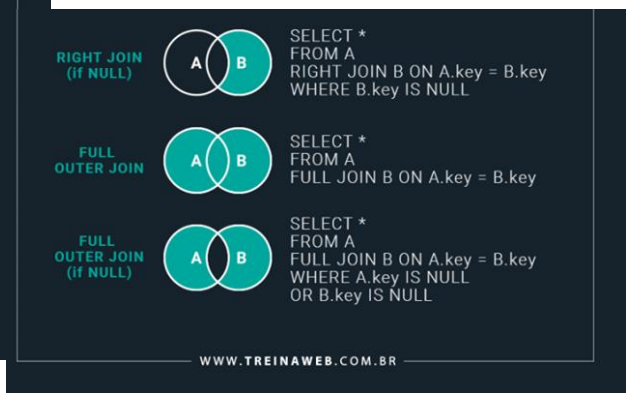
- Na base de dados **locadora**:

1. Faça uma consulta que retorne somente o nome e estado de cada Cliente.
2. Faça uma consulta que retorne todas as Cidades com seus devidos Estados.
3. Faça uma consulta que retorne todos os Estados com seus devidos Países.
4. Faça uma consulta que retorne somente o nome dos Clientes com seus devidos nome dos Países.
5. Faça uma consulta que retorne somente o nome dos **clientes**, informando qual sua **cidade**, **estado** e **país**.

95

SELECT com JOINS





96

SELECT com JOINS

- Joins (junções) são um recurso presente nos bancos de dados relacionais, através da qual é possível juntar o conteúdo de duas tabelas através de um critério;
- É um conceito que muitas vezes quem está iniciando no mundo dos bancos de dados relacionais tem dificuldade de entender;
- Os Joins mais utilizadas são: INNER, LEFT, RIGHT, CROSS
- Sintaxe:

```
SELECT Tabela1.atributo1, Tabela2.atributo1
FROM Tabela1 TIPO JOIN Tabela2
ON (tabela1.atributoPK = tabela2.atributoFK);
```

97

SELECT com JOINS, com mais de 2 tabelas

- Sintaxe:

```
SELECT
    Tabela1.coluna_tabela1,
    Tabela2.coluna_tabela2,
    Tabela3.coluna_tabela3
FROM
    Tabela1
INNER JOIN
    Tabela2 ON Tabela1.chave_primaria = Tabela2.chave_estrangeira
INNER JOIN
    Tabela3 ON Tabela2.chave_primaria = Tabela3.chave_estrangeira;
```

98

SELECT com JOINS – CROSS JOIN

- Usaremos o script **BD_Herois**, disponível na pasta de arquivos do Teams
- Quando queremos juntar duas ou mais tabelas por cruzamento. Ou seja, para cada linha da tabela ORIGEM queremos todos os HEROIS ou vice-versa

```
SELECT
ORIGEM.nome_ori AS 'Origem’,
HEROI.nome_hero AS 'Nome Heroi'
FROM origem CROSS JOIN heroi;
```

99

SELECT com JOINS – INNER JOIN

- Usado quando queremos juntar duas ou mais tabelas por coincidência e tem o mesmo efeito do SELECT por comparação direta de chaves
- No INNER JOIN são selecionados apenas os registros que possuem relação, ou seja, que possuem a FK preenchida. Já os registros com a FK Nula são descartados
- No caso de HEROI e ORIGEM os atributos internos coincidentes são cod_ori_pk na tabela ORIGEM e cod_ori_fk na tabela HEROI

100

SELECT com JOINS – INNER JOIN

- **Objetivo**: mostrar os valores vinculados através da chave estrangeira (cod_ori_fk). Os registros não vinculados SÃO DESCARTADOS

- **Solução**:

```
SELECT
Origem.nome_ori AS 'Origem’,
Heroi.nome_hero AS 'Nome Herói'
FROM Origem INNER JOIN Heroi
ON (Origem.cod_ori_pk = Heroi.cod_ori_fk);
```

101

SELECT com JOINS – LEFT JOIN

- Observando o resultado do SELECT com o INNER que os heróis Huck e Big Hero não apareceram porque não possuíam relacionamento

- Percebe-se também a origem Internet também não apareceu, porque não possui nenhum herói relacionado a ela

- Se desejarmos listar todos os Heróis com suas respectivas Origens, incluindo os heróis sem origem, a exemplo de Huck e Big Hero, devemos o LEFT no lugar INNER, assim, todos os registros à esquerda do JOIN serão mostrados, independente se possuem ou não relação

102

SELECT com JOINS – LEFT JOIN

- **Exemplo**: buscar tudo que esta na tabela da esquerda da comparação (join) e vincula com a tabela direita (join) inclusive os registros sem relação;

- **Solução**:

```
SELECT
Origem.nome_ori as 'Origem',
Heroi.nome_hero as 'Nome Heroi'
FROM Heroi LEFT JOIN Origem
ON (Heroi.cod_ori_fk = Origem.cod_ori_pk);
```

103

SELECT com JOINS – RIGHT JOIN

- Observando o resultado da consulta com o LEFT JOIN notamos que a origem Internet não apareceu, pois ela não possui relacionamentos e a tabela Origem estava a direita da comparação (JOIN)

- Se desejarmos listar todas as ORIGENS e seus respectivos HEROIS, incluindo as ORIGENS sem HEROIS, poderíamos usar o RIGTH JOIN;

104

SELECT com JOINS – RIGHT JOIN

- **Objetivo**: busca tudo que está na tabela da direita da comparação (antes do join) e vincula com a tabela esquerda (depois do join) inclusive os registros null da tabela a esquerda;

- **Solução**:

```
SELECT
Origem.nome_ori as 'Origem’,
Heroi.nome_hero as 'Nome Heroi'
FROM Heroi RIGHT JOIN Origem
ON (Heroi.cod_ori_fk = Origem.cod_ori_pk);
```

105

Prática SQL – BD_HEROIS

- Use o **BD_Herois**:

1. Liste todos os nomes de origens com as devidas armas dos heróis
2. Liste todos os nomes dos heróis que tenham a origem ‘Marvel’
3. Liste os nomes dos heróis, que tenham como arma o ‘dinheiro’, com suas respectivas origens
4. Crie um SELECT utilizando JOIN a partir de uma necessidade visualizada por você e detalhe, nos comentários, seu objetivo

106

VIEWS

- Visões em SQL são consultas armazenadas em uma estrutura de fácil acesso baseadas num comando SELECT
- Essa consulta armazenada funciona como uma tabela virtual, com comportamento similar a uma tabela real, entretanto, sem armazenar dados ou modificar as tabelas de origem da consulta
- Os dados que são exibidos nas visões são gerados dinamicamente toda vez que a visão é consultada
- É importante ressaltar que o SGBD armazena apenas o nome da visão e o comando SELECT associado
- Os dados visualizados nas visões são gerados dinamicamente toda vez que é solicitada uma consulta sobre a VIEW. Isso implica que uma visão está sempre atualizada, ou seja, ao se modificar dados nas tabelas referenciadas na descrição da visão, uma consulta a visão reflete automaticamente essas alterações

107

VIEWS

- Sintaxe:

```
CREATE VIEW nome_da_visão AS
SELECT atributo1, atributo2
FROM tabela
WHERE condição;
```

108

VIEWS no BD_HEROIS

- Exemplo:

```
CREATE VIEW Herois_e_Origem AS
SELECT
ORIGEM.nome_ori AS 'Origem’,
HEROI.nome_hero AS 'Nome do Herói’
FROM
ORIGEM, HEROI
WHERE
(HEROI.cod_ori_fk = ORIGEM.cod_ori_pk);
```

109

Vantagens das VIEWS

- **Reuso**: as VIEWS são objetos de caráter permanente. Pensando pelo lado produtivo isso é excelente, já que elas podem ser lidas por vários usuários simultaneamente

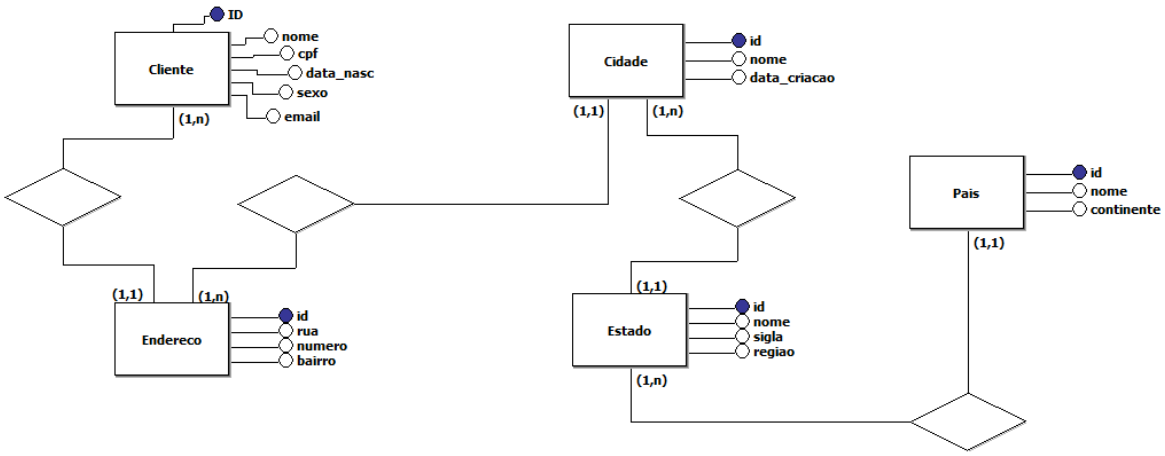
- **Segurança**: as VIEWS permitem que ocultemos determinadas colunas de uma tabela. Para isso, basta criarmos uma VIEW com as colunas que acharmos necessário que sejam exibidas e as disponibilizarmos para o usuário

- **Simplificação do código**: as VIEWS nos permitem criar um código de programação muito mais limpo, na medida em que podem conter um SELECT complexo. Assim, criar views para os programadores a fim de poupá-los do trabalho de criar SELECT’s é uma forma de aumentar a produtividade da equipe de desenvolvimento

110

Prática SQL 01 (locadora), parte 11

- Criar, no mínimo, duas VIEWS no BD **locadora**, envolvendo mais de uma tabela



111

Stored Procedures

- É um conjunto de instruções SQL armazenadas no servidor de banco de dados que pode ser **executado sob demanda**. Ela permite encapsular lógica complexa de banco de dados, como operações repetitivas ou cálculos complexos, facilitando a reutilização de código e melhorando o desempenho.

- As Stored Procedures são particularmente úteis para:

  - **Reduzir a quantidade de código** enviado pela aplicação ao servidor

  - **Centralizar a lógica** de negócios no banco de dados

  - **Melhorar a segurança** ao limitar o acesso direto às tabelas

112

Stored Procedures

- Sintaxe:

```
CREATE PROCEDURE NomeDaProcedure
    @Parametro1 INT,
    @Parametro2 VARCHAR(50)
AS
BEGIN
    -- Instruções SQL
    SELECT * FROM Tabela
    WHERE Coluna1 = @Parametro1
    AND Coluna2 = @Parametro2;
END
```

113

Stored Procedures

- **Cenário:** No contexto de uma administradora de condomínios, queremos criar uma procedure que liste todas as unidades de um condomínio específico com os moradores associados

```
DELIMITER //
CREATE PROCEDURE ListarUnidadesPorCondominio(IN CondominioID INT)
BEGIN
    SELECT
        u.Numero AS Numero_Unidade,
        u.Tipo AS Tipo_Unidade,
        m.Nome AS Nome_Morador,
        m.CPF AS CPF_Morador
    FROM
        Unidade u
    LEFT JOIN
        Morador m ON u.ID_Unidade = m.ID_Unidade
    WHERE
        u.ID_Condominio = CondominioID;
END //
DELIMITER ;
```

- ✓ **Parâmetro de Entrada (IN):** CondominioID é usado para filtrar as unidades de um condomínio específico.
- ✓ **LEFT JOIN:** Inclui todas as unidades mesmo que não tenham moradores associados.
- ✓ **WHERE:** Garante que apenas as unidades do condomínio especificado sejam listadas.

114

Stored Procedures

- Para chamar a procedure:

```
CALL NomeDaProcedure @Parametro1 = 1, @Parametro2 = 'Exemplo';
```

- Parâmetros da Procedure:
  - **IN**: Valor fornecido ao chamar a procedure (somente entrada).
  - **OUT**: Valor retornado pela procedure.
  - **INOUT**: Permite entrada e modificação do valor do parâmetro.

115

Stored Procedures

- Vantagens
  - **Reutilização**: O código SQL pode ser reutilizado várias vezes sem reescrevê-lo.
  - **Performance**: Reduz o tráfego entre aplicação e banco de dados.
  - **Segurança**: Permite abstrair a lógica de acesso às tabelas.
- Desvantagens
  - **Dependência do banco**: Lógica centralizada no banco pode dificultar a portabilidade entre sistemas.
  - **Complexidade**: Procedimentos extensos podem ser difíceis de manter.

116

Function

- É um bloco de código reutilizável que realiza uma tarefa específica e retorna um valor único.
- As funções permitem encapsular lógica que pode ser chamada em consultas SQL, simplificando a reutilização de código e a manutenção do banco de dados.
- Elas são diferentes de procedures, pois são usadas exclusivamente para retornar valores e não alteram diretamente os dados no banco.

117

Function

- Características:
  - Retorna apenas um valor (obrigatório).
  - Pode aceitar zero ou mais parâmetros de entrada.
  - É usada em consultas SQL, como em cláusulas SELECT, WHERE, ou GROUP BY.
  - Não pode executar comandos que modifiquem o banco de dados, como INSERT, UPDATE, ou DELETE.
  - São usadas em consultas como uma função embutida (ex.: SUM, COUNT).

118

Function

- Sintaxe:

```
CREATE FUNCTION nome_da_funcao (parametro1 tipo, parametro2 tipo, ...)
RETURNS tipo_de_retorno
DETERMINISTIC/NOT DETERMINISTIC
BEGIN
    DECLARE variavel tipo;
    -- lógica da função
    RETURN valor;
END;
```

- **RETURNS** tipo_de_retorno: Define o tipo de dado que a função irá retornar (ex.: INT, VARCHAR).
- **DETERMINISTIC**: Declara que a função sempre retorna o mesmo resultado para os mesmos parâmetros (ou NOT DETERMINISTIC se o resultado variar).
- **BEGIN** ... **END**: Delimita o bloco de código da função.
- **RETURN**: Define o valor de saída da função.

119

Function

- **Cenário**: Criar uma função que calcule o custo total de todos os serviços prestados a um condomínio, dado o ID do condomínio.

```
DELIMITER //
CREATE FUNCTION CalculaCustoTotalServicos (p_ID_Condominio INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE total DECIMAL(10, 2);
    SELECT SUM(s.Custo)
    INTO total
    FROM Prestacao_de_Servico ps
    INNER JOIN Servico s ON ps.ID_Servico = s.ID_Servico
    WHERE ps.ID_Condominio = p_ID_Condominio;
    RETURN COALESCE(total, 0); -- Garante que o retorno seja 0 se não
                               -- houver serviços para o condomínio
END; //
DELIMITER ;
```

120

Function

- Para utilizar a function:

```
SELECT
    Nome AS ‘Nome Condomínio’,
    CalculaCustoTotalServicos(ID_Condominio) AS ‘Custo Total’
FROM
    Condominio;
```

121

Function

- Vantagens
  - **Reutilização**: A lógica encapsulada pode ser reutilizada em várias consultas.
  - **Manutenção**: Facilita a atualização de lógica complexa em um único local.
  - **Legibilidade**: Simplifica as consultas SQL, deixando-as mais legíveis.

122

Triggers

- Uma trigger (**gatilho**) é um objeto de banco de dados que é automaticamente executado ou "disparado" em resposta a um evento específico, como um **INSERT**, **UPDATE** ou **DELETE**, em uma tabela.

- Triggers são usadas para manter a integridade do banco de dados, realizar cálculos automáticos, auditar mudanças ou implementar lógica adicional.

- Principais características:

  - Associada a uma tabela específica.
  - Disparada antes (BEFORE) ou depois (AFTER) do evento.
  - Pode ser criada para os eventos INSERT, UPDATE ou DELETE.
  - Não retorna valores diretamente.

123

Triggers

- Sintaxe:

```
CREATE TRIGGER nome_trigger
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE }
ON nome_tabela
FOR EACH ROW
BEGIN
    -- Aqui você escreve as instruções que serão executadas
END;
```

124

Triggers

- Elementos da sintaxe:
  - **CREATE TRIGGER nome_trigger:** Define o nome da trigger. Deve ser único dentro do banco de dados.
  - **BEFORE ou AFTER**: Especifica quando a trigger será disparada:
    - **BEFORE**: Antes da operação (útil para validar ou modificar dados).
    - **AFTER**: Após a operação (geralmente usado para registros ou ações baseadas na mudança).
  - **INSERT, UPDATE, DELETE:** Define o evento que dispara a trigger. A trigger será ativada quando uma dessas operações for realizada na tabela.
  - **ON nome_tabela:** Indica a tabela à qual a trigger está associada.
  - **FOR EACH ROW:** A trigger será executada para cada linha afetada pela operação.
  - **BEGIN ... END:** Define o bloco de código que será executado quando a trigger for acionada. Nesse bloco, você pode incluir várias instruções SQL.

125

Triggers

- Exemplo de utilização:
  - Tentativa de Inserção com Valor Negativo:

```
INSERT INTO Financeiro (Tipo, Descricao, Valor, Data, ID_Condominio)
VALUES ('Despesa', 'Manutenção', -100.00, '2025-01-25', 1);
```

  - Erro retornado:

```
ERROR 1644 (45000): O valor da transação não pode ser negativo.
```

126

Triggers

- Cenário: No sistema da **administradora de condomínios**, queremos que toda vez que uma despesa ou receita seja inserida ou alterada na tabela Financeiro, o saldo total do condomínio correspondente seja automaticamente atualizado na tabela Condomínio.

- Objetivo da Trigger:

  - Se for uma **receita**, o saldo do condomínio aumenta com o valor da transação.
  - Se for uma **despesa**, o saldo do condomínio diminui com o valor da transação.

127

Triggers

```
DELIMITER //
CREATE TRIGGER atualizar_saldo_condominio
AFTER INSERT ON Financeiro
FOR EACH ROW
BEGIN
    -- Atualizar o saldo do condomínio com base no tipo de transação
    IF NEW.Tipo = 'Receita' THEN
        UPDATE Condominio
        SET Saldo = Saldo + NEW.Valor
        WHERE ID_Condominio = NEW.ID_Condominio;
    ELSEIF NEW.Tipo = 'Despesa' THEN
        UPDATE Condominio
        SET Saldo = Saldo - NEW.Valor
        WHERE ID_Condominio = NEW.ID_Condominio;
    END IF;
END //
DELIMITER ;
```

128

Prática SQL 02, parte 02

- Crie:
  - **Hospital** (modelo conceitual, dicionário de dados e modelo físico/script SQL),
  - Campeonato de **Futebol** (modelo conceitual e modelo físico/script SQL) e
  - Agência de **Viagens** (modelo conceitual)
- Um script para cada modelo solicitado, a ser executado de uma única vez (sequencial), com:
  - **INSERT** (mínimo 4 por tabela, com suas devidas chaves estrangeiras)
  - **UPDATE** (mínimo 2 por tabela)
  - **DELETE** (mínimo 1 por tabela)
  - Uso dos operadores **BETWEEN** e **LIKE** (usar no DELETE e UPDATE)
  - 3 **SELECTs** para cada banco de dados, utilizando cláusulas WHERE e operadores especiais; todos os SELECTs devem ser comentados, definindo o objetivo de cada um (os SELECTS precisam retornar, no mínimo, um registro)
  - 1 **TRIGGER**, 1 **STORED PROCEDURE** e 1 **FUNCTION** para cada modelo
- Enviar os arquivos (arquivo BR Modelo, planilha e script SQL) atualizado via trabalhos/tarefas no Teams até 30/01/2025

129

Prática para o Projeto Integrador

- A turma será dividida em três equipes
- Cada equipe ficará responsável por um projeto do PI (baseado nos protótipos do Figma)
- Entregar, via e-mail:
  - Modelo Conceitual, desenvolvido no BR Modelo
  - Dicionário de Dados, elaborado em planilha
  - Script SQL do Modelo Físico, contendo:
    - 6 registros em cada tabela
    - 2 updates e 2 deletes, inclusive usando operadores especiais
    - 4 selects com joins, envolvendo uma situação que deverá ser comentada
    - 1 triggers, 1 stored procedures e 2 fuctions (analise a melhor situação para essas aplicações)
    - O arquivo deve estar organizado para ser executado em sequência, sem erros de execução
- Entrega e apresentação em 31/01/2025

130

...

- ...

131

Referências

- Slide baseado no material do **Prof. Jackson Henrique da Silva Bezerra** (IFRO – Campus Ji-Paraná)
- Documentação oficial: https://dev.mysql.com/doc/

132