



**ser**  
educacional

gente criando o futuro

# Banco de Dados Unidade – III e IV

Professor Adilson da Silva

# Objetivos



- Apresentar os conceitos da linguagem SQL;

- Criar objetos

- Aprender como são feitas as consultas básicas em Linguagem SQL

- Alterar Objetos

- Saber como restringir e ordenar dados
- Estudar as funções de agregação
- Tecnologias Emergentes



# SQL



**ser**  
educacional  
gente criando o futuro

# SQL

---



- SQL é considerada a razão principal para o sucesso dos bancos de dados relacionais comerciais
- Tornou-se a linguagem padrão para bases relacionais
- Funciona entre diferentes produtos
- Embedded SQL: Java, C/C++, Cobol...
- Fácil uso para o usuário



# SQL

---



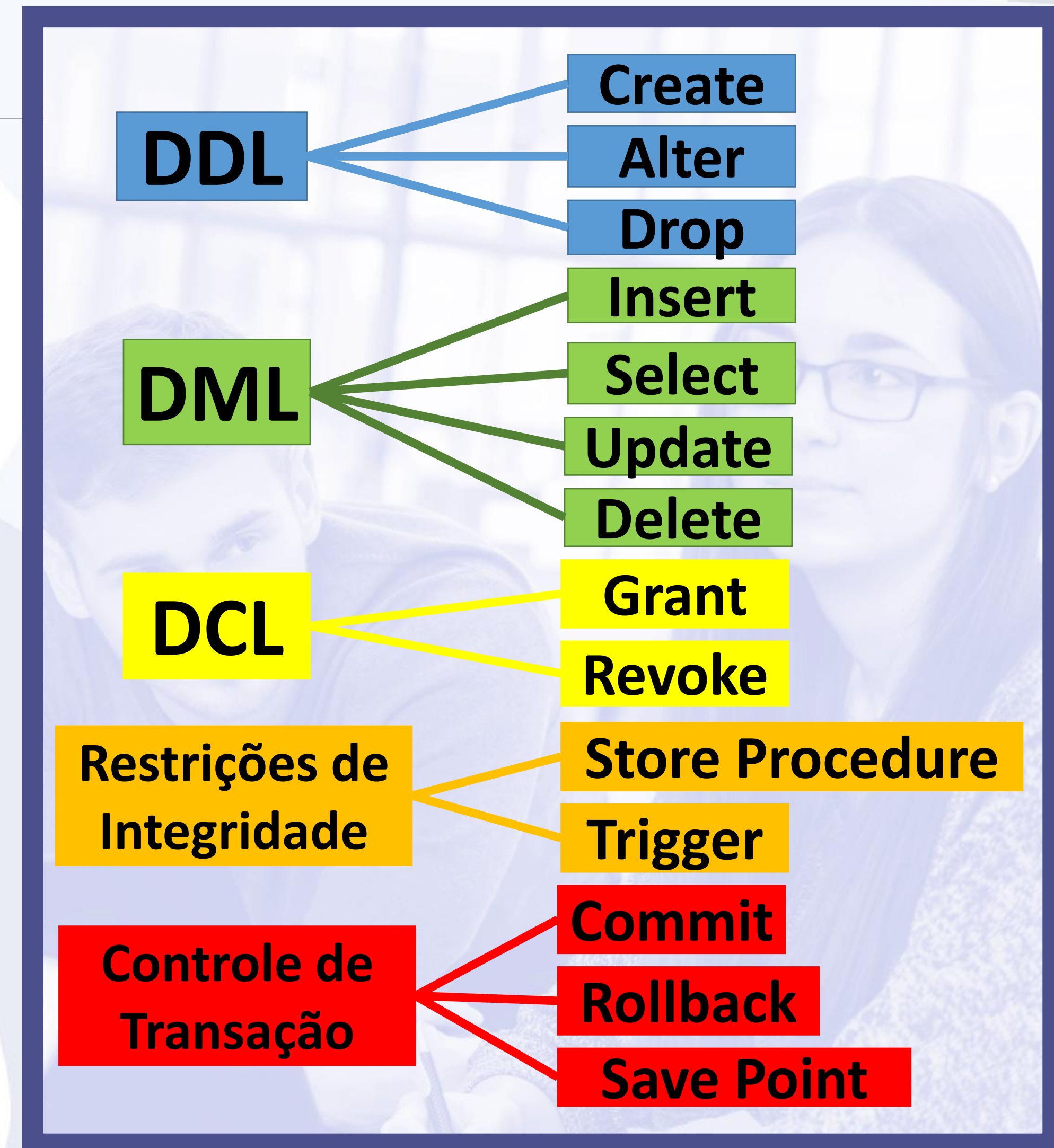
- Permite especificar:
  - O esquema de cada relação
  - O domínio dos valores associados a cada atributo
  - Restrições de integridade
  - O conjunto de índices
  - Visões
  - Permissão de acesso às relações





# Linguagem SQL

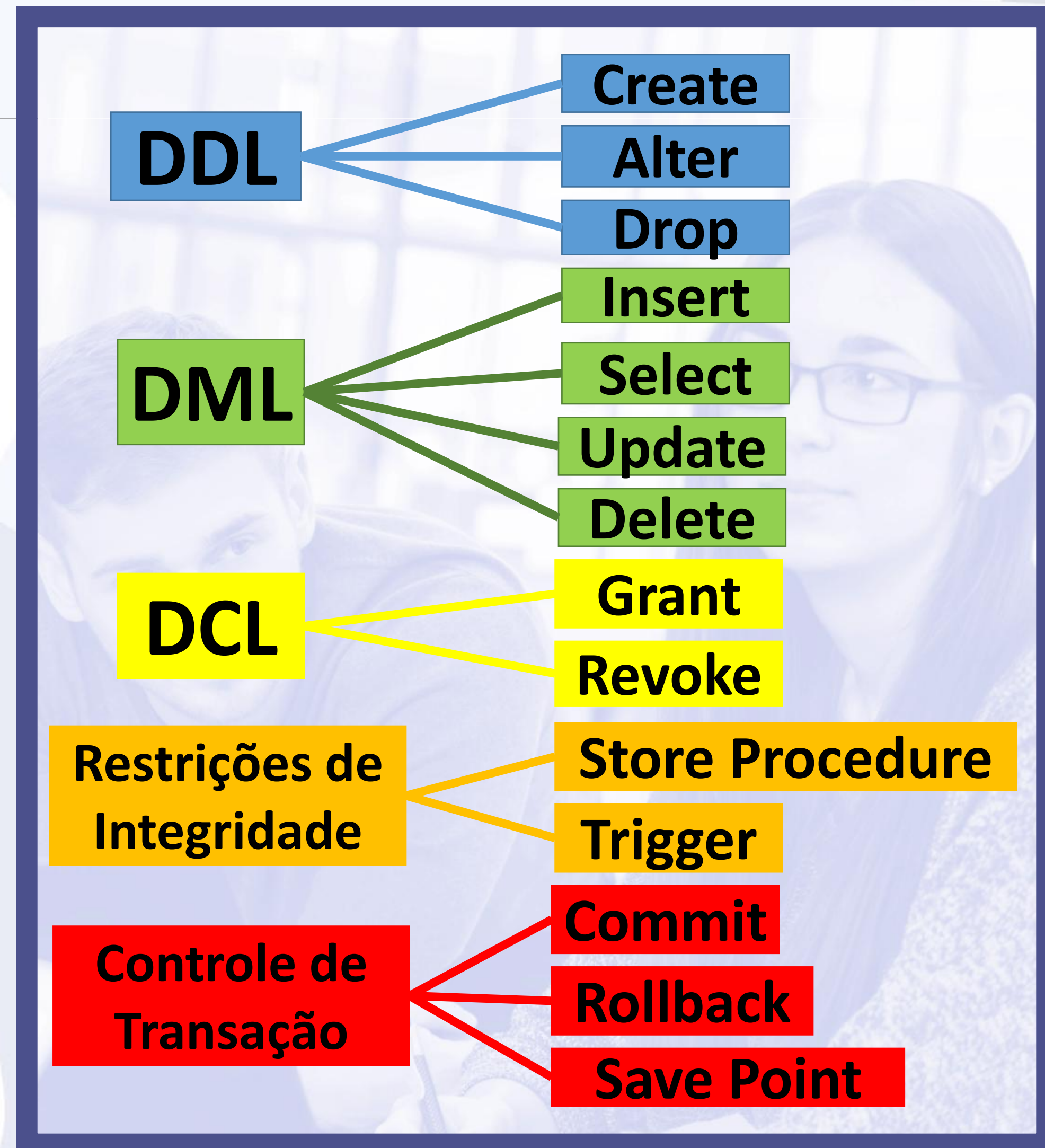
- A linguagem SQL tem cinco vertentes
- **DDL** - Data Definition Language – Linguagem de definição de dados
  - Criar (CREATE) objetos;
  - Remover (DROP) objetos;
  - Alterar (ALTER) objetos;
  - Descrever restrições de integridade(CONSTRAINTS-PK e FK)
- **DML** - Data Manipulation Language – Linguagem de Manipulação de Dados
  - Atualização da Base de Dados (Instruções INSERT, SELECT, DELETE e UPDATE)
  - Realiza a inserção, consulta, alteração e a exclusão de dados nas tabelas.



# Linguagem SQL



- **DCL** – Data Control Language - Linguagem de Controle de Dados - controla os aspectos de autorização de dados e licenças de usuários para controlar quem tem acesso para ver ou manipular dados dentro do banco de dados.
  - GRANT - autoriza ao usuário executar ou setar operações.
  - REVOKE - remove ou restringe a capacidade de um usuário de executar operações.



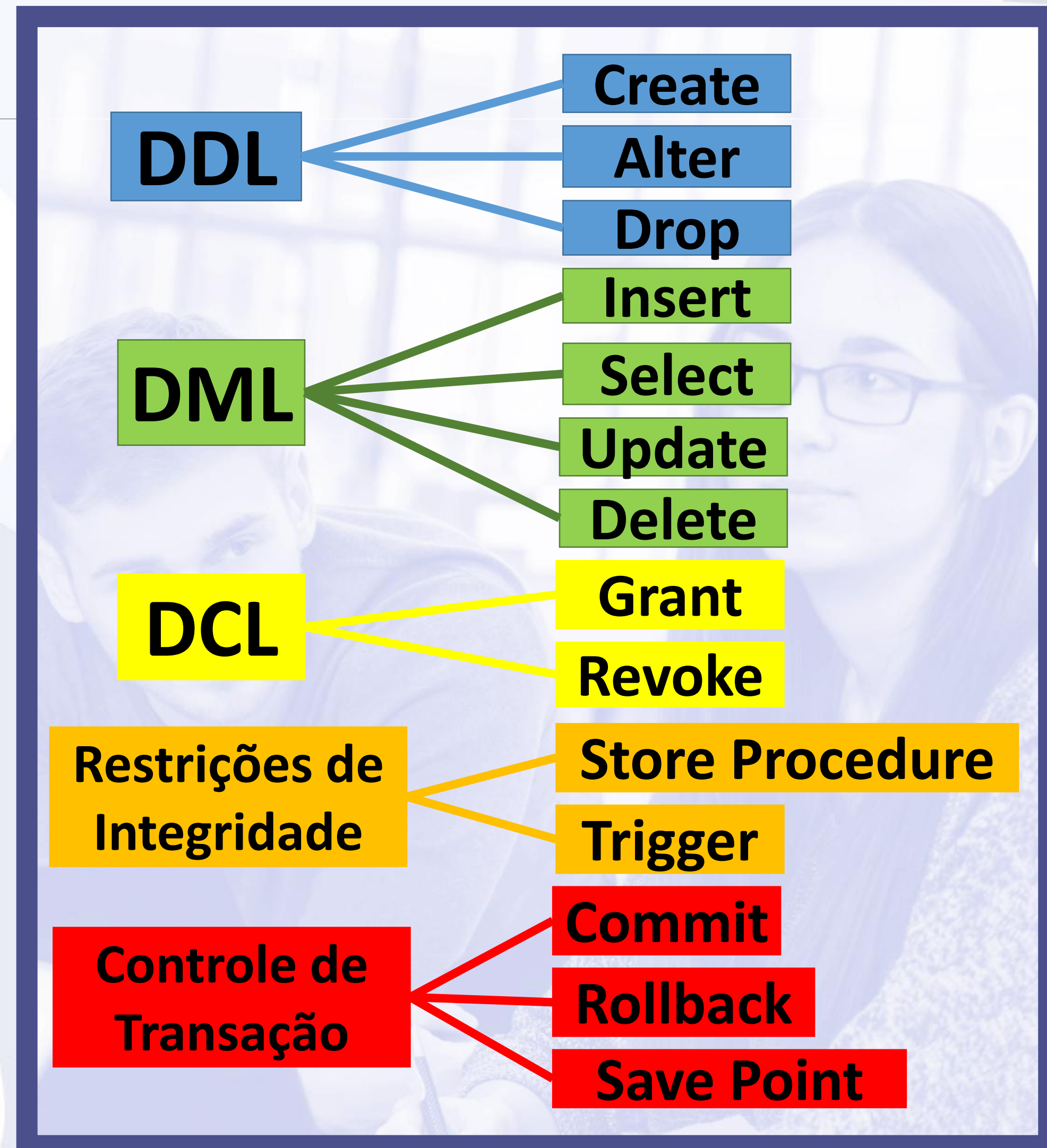


# Linguagem SQL



- **Restrições de Integridade**

- É possível realizar ações que contribuam para a integridade dos dados, como:
- STORED PROCEDURES – Um conjunto de comandos em SQL que podem ser executados de uma só vez.
- TRIGGERS – Gatilho que é ativado quando um evento especial acontece numa tabela, algumas vezes devendo afetar outras tabelas.

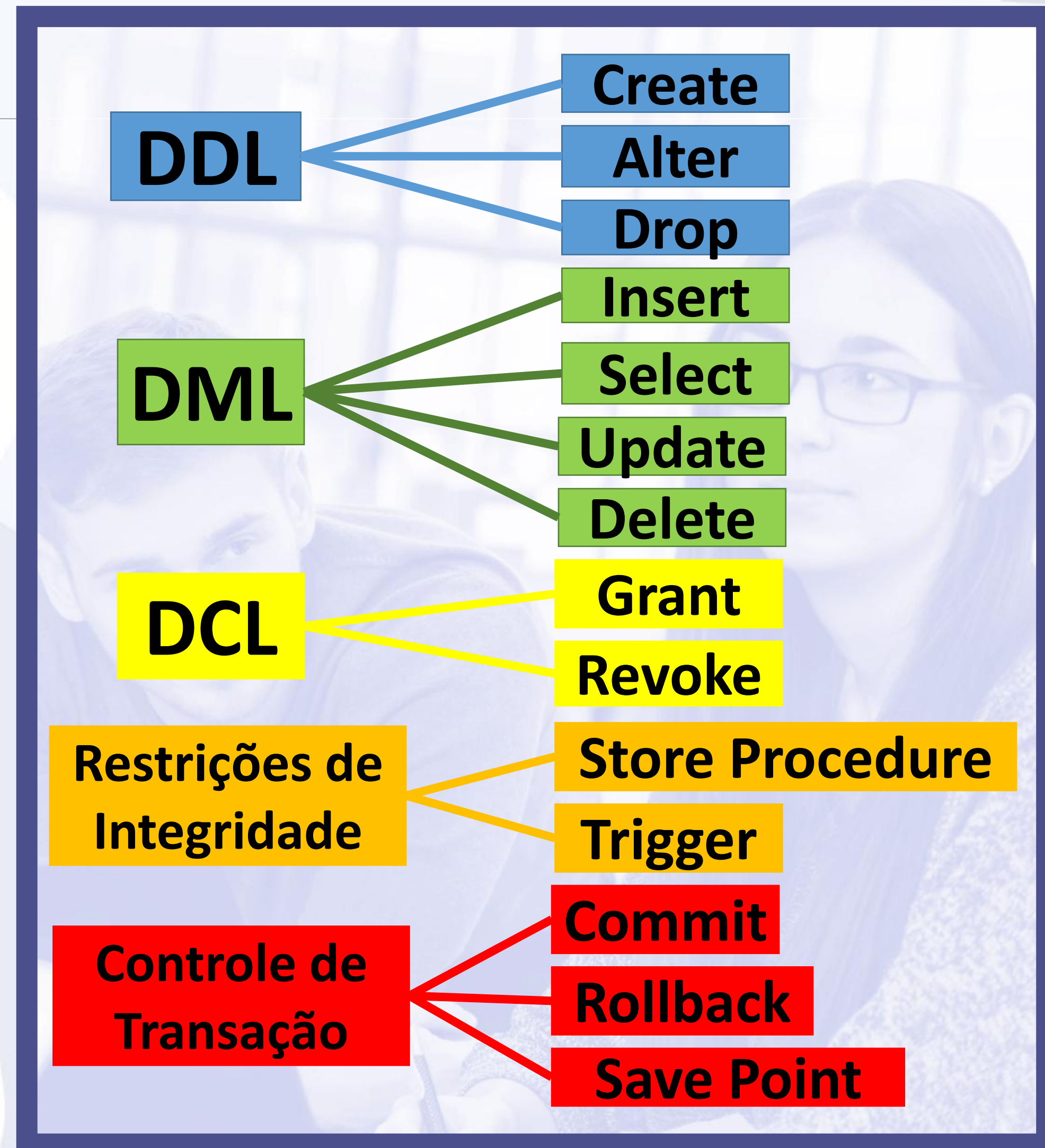




# Linguagem SQL



- **Controle de Transação**
- Permite garantir a efetividade de ações realizadas no banco de dados, como:
  - COMMIT – Efetiva alterações pendentes em um banco de dados.
  - ROLLBACK – Desfaz uma alteração antes de ser efetivada no banco de dados.
  - SAVEPOINT – Permite uma subdivisão lógica de uma transação longa.



# Comandos básicos



**ser**  
educacional

gente criando o futuro





# Comandos SQL

## SHOW DATABASES;

- O SGBD organiza o banco de dados em diversas bases de dados e, em cada uma, podemos criar várias tabelas com seus atributos. Logo, ao nos referenciarmos ao nosso banco de dados, usaremos o termo “base de dados”, que são sinônimos, mas apenas para não haver confusão sobre qual base está sendo referida.

### SCHEMAS



**northwind**



Tables could not be fetched



categorias



clientes



detalhes do pedido



fornecedores



funcionarios



pedidos



produtos



transportadoras



Views could not be fetched



Stored Procedures could not be fetched



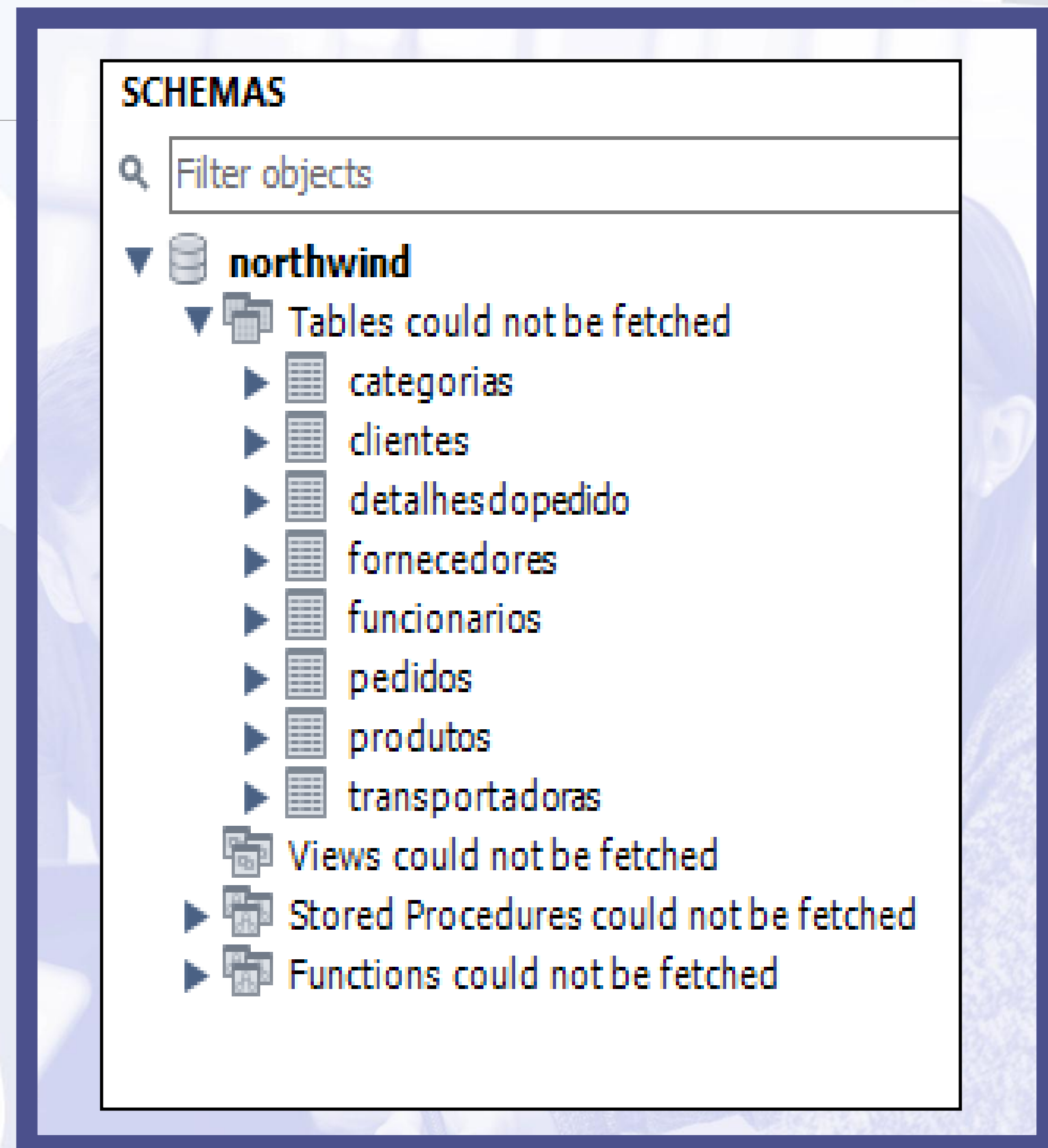
Functions could not be fetched



# Comandos SQL

## SHOW DATABASES;

- Cada banco de dados organizado pelo SGBD é, portanto, composto por sua base de dados, com suas tabelas e atributos totalmente independentes e acessados separadamente de outras bases de dados no mesmo SGBD.



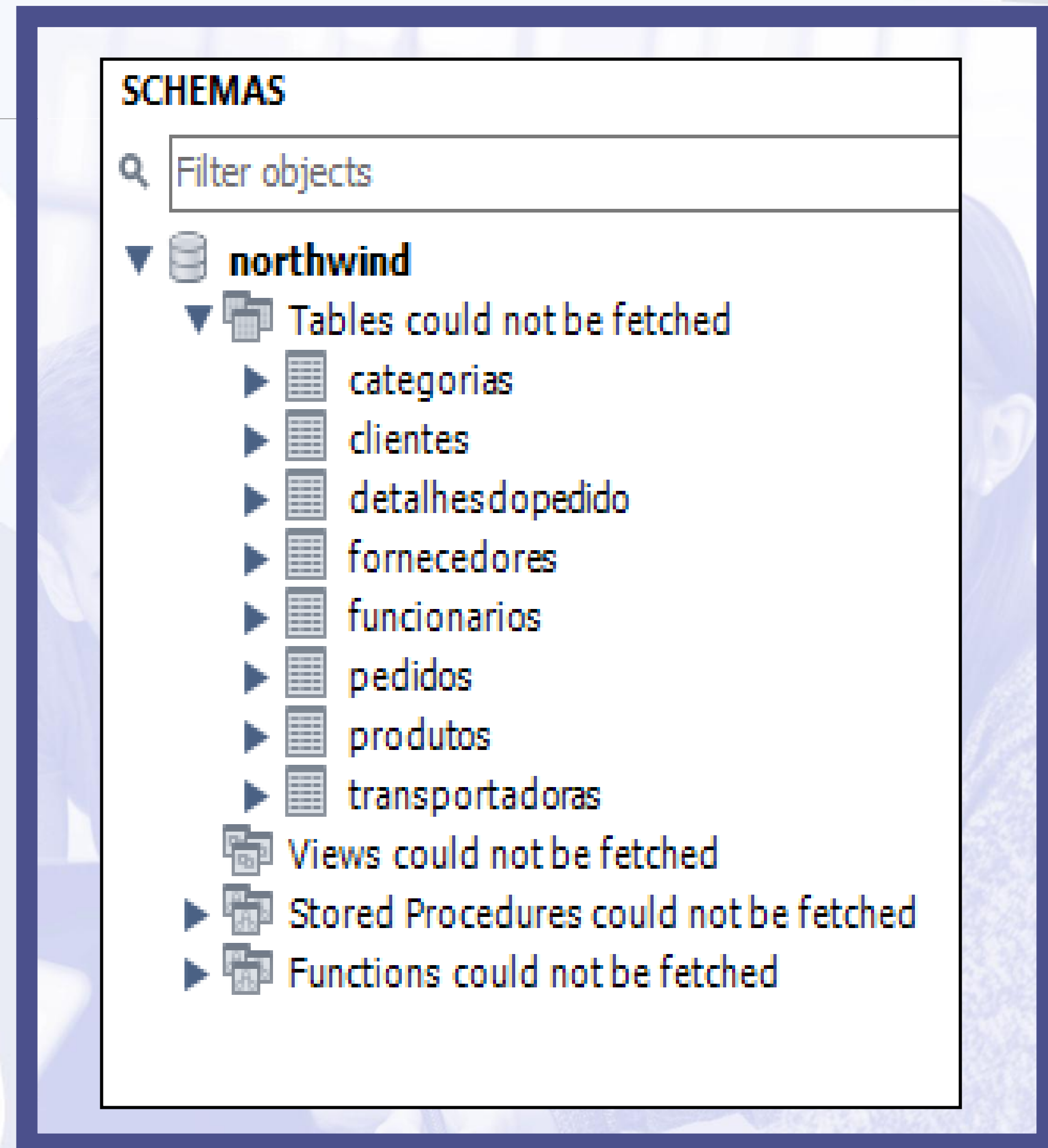




# Comandos SQL

## CREATE DATABASES;

- Para criar uma nova base de dados chamada Controle\_de\_Vendas, mas, como o MySQL normalmente converterá o nome das bases de dados para minúsculo, vamos então referenciá-la sempre em minúsculo: **controle\_de\_vendas**.
- O comando para criação de bases de dados é **CREATE DATABASE**, seguido do nome da base de dados. Logo, o comando deverá ser digitado da seguinte forma
- **CREATE DATABASE** controle\_de\_vendas;





# Comandos SQL

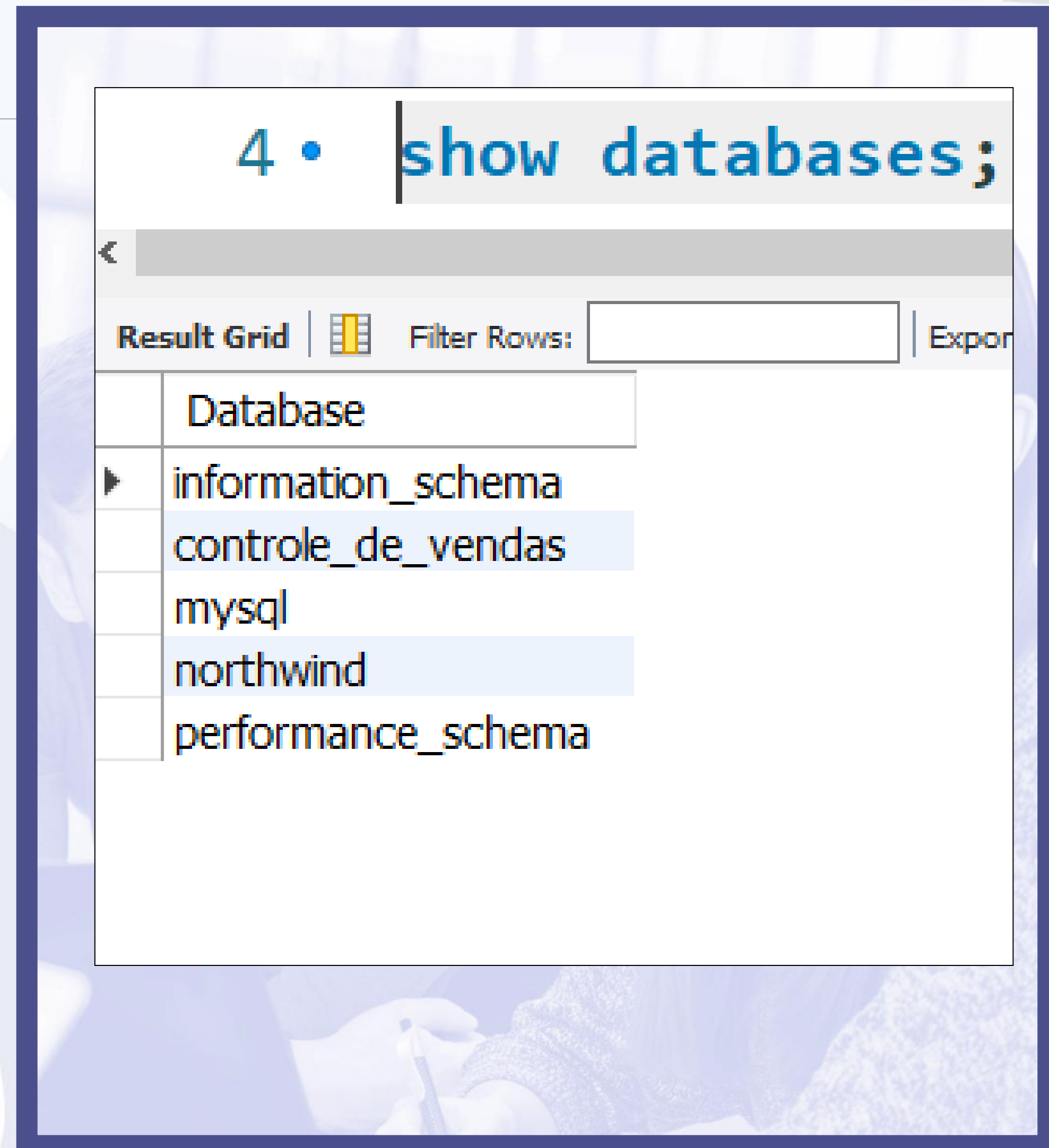
## Comando SHOW

**SHOW DATABASES;** – listará as bases criadas.

**SHOW TABLES;** – listará as tabelas criadas.

**SHOW CREATE TABLE** nome da tabela; – listará a estrutura da tabela indicada.

**SHOW CREATE DATABASE** nome da base de dados; – listará dados sobre a base de dados indicada.





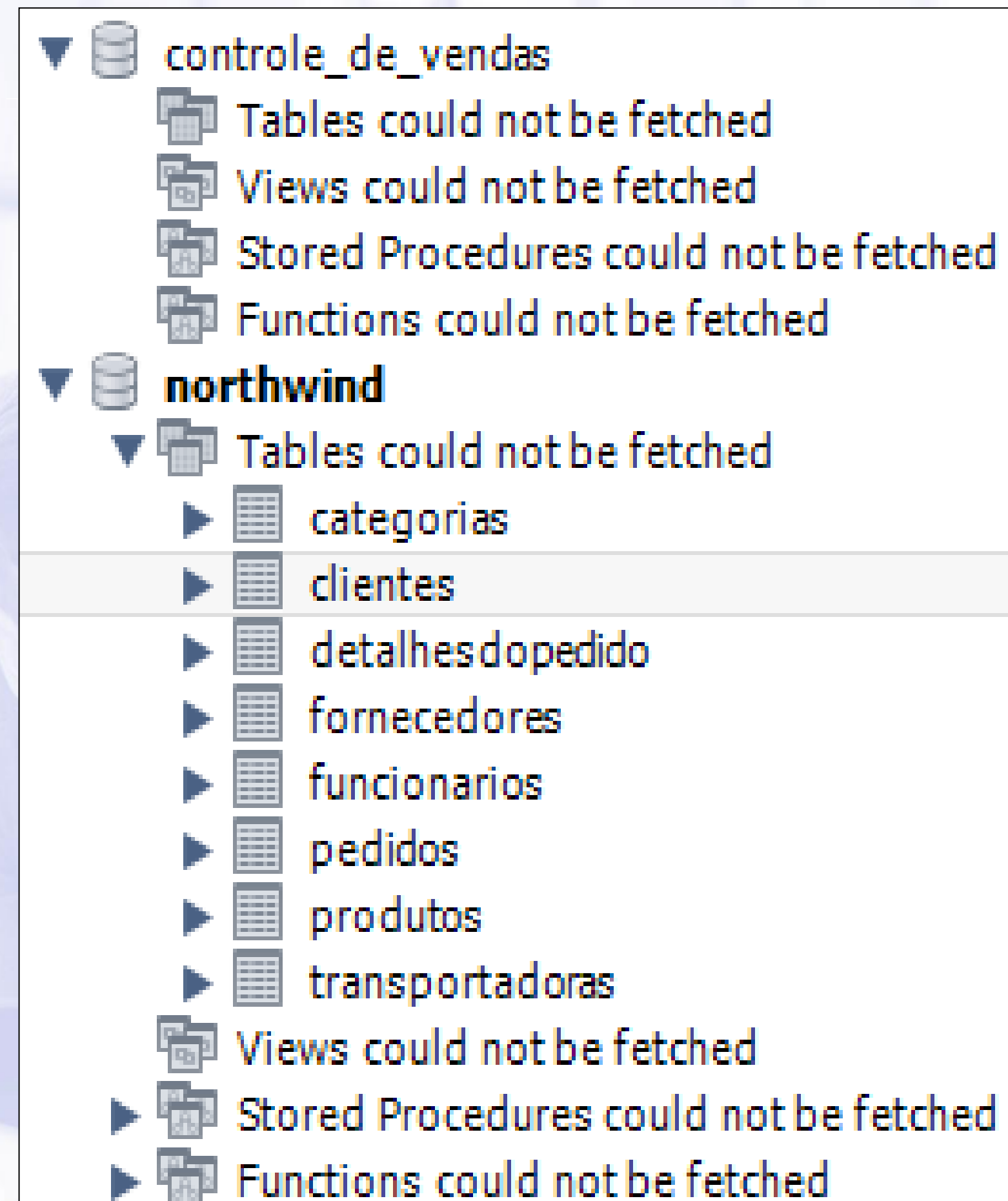


# Comandos SQL

## Comando USE

**USE** controle\_de\_vendas;

- Você perceberá que o prompt de comando do MySQL trará sempre o nome da base de dados em uso. Para operar outras bases de dados, basta apenas dar um comando USE e o nome da outra base.



# Comando Select



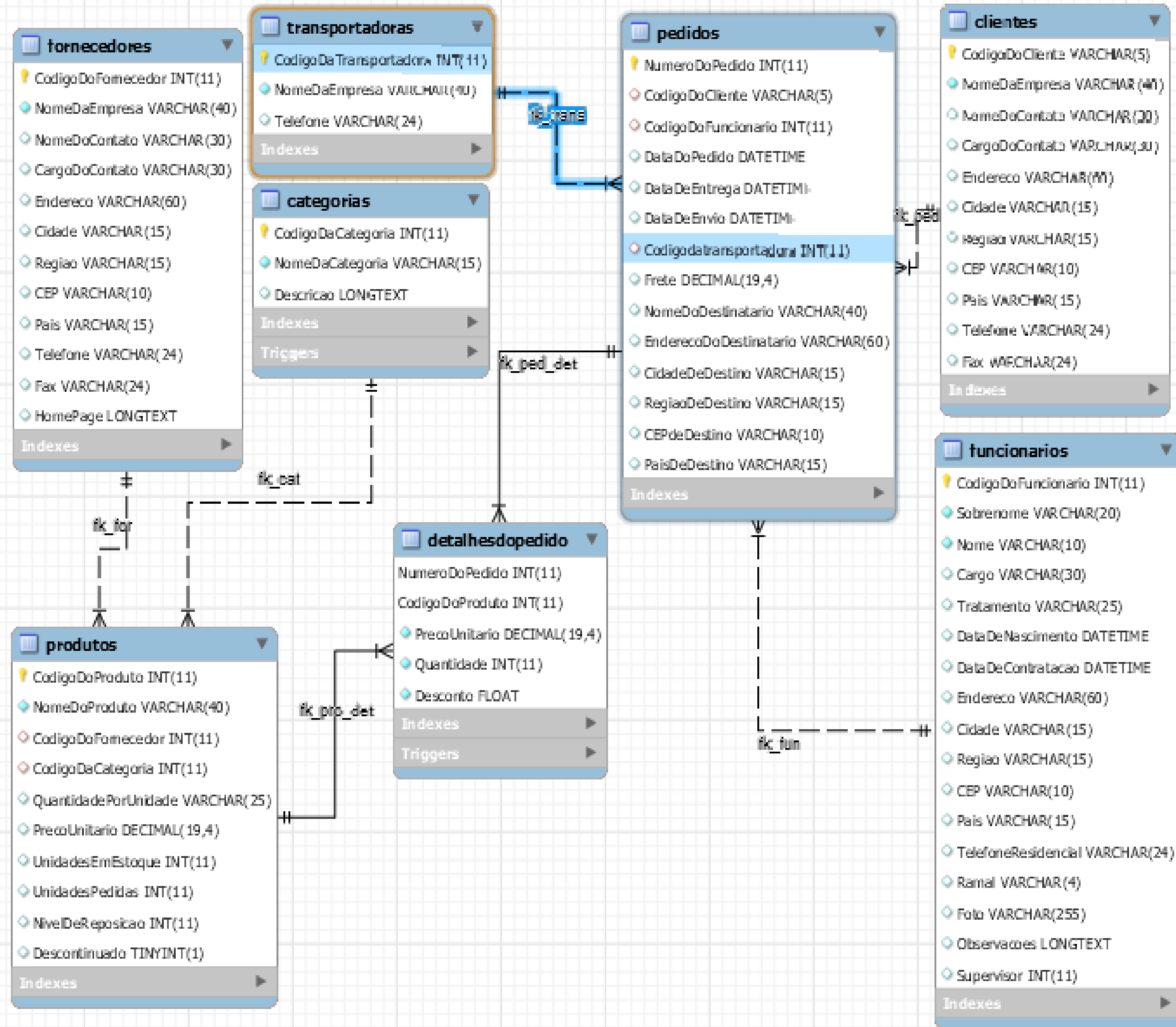
**ser**  
educacional

gente criando o futuro



# Comando Select

- No link a seguir você encontra um vídeo de como instalar o MySql e como preencher o banco de dados usando o arquivo que você encontrará na página da disciplina em logo após essa WEB (Banco.SQL).
- Link do vídeo
- <https://www.youtube.com/watch?v=drM4MKT8zBk>



# DQL – Comando Select

---



Comando `SELECT` permite recuperar informações existentes nas tabelas.

**`SELECT [DISTINCT] Colunas, Expressões, [AS nome-atributo]`** {O que se quer exibir}  
**`[FROM Tabelas]`** { Onde estão as informações a serem exibidas}  
**`[WHERE condição]`** {Filtros e Relacionamentos}  
**`[GROUP BY Colunas]`** { Colunas que queremos agrupar}  
**`[ HAVING BY Condição para as colunas agrupadas]`**  
**`[ORDER BY attr_name1 [ASC / DESC ]`** { Ordem de exibição}





# Comando SQL

- Realizando Operações:
  - SELECT COLUNAS (**Operações**)
  - FROM TABELA
  - WHERE **CONDIÇÃO**
  - Operadores Aritméticos:

+	<b>Soma</b>
-	<b>Subtração</b>
*	<b>Multiplicação</b>
/	<b>Divisão</b>

# Alterando estrutura da tabela



- Informando filtros para as seleções:
  - Operadores Relacionais:

=	Igual a
<>	Não igual a
<b>Between...and</b>	Entre
<b>In</b>	Na lista
>	Maior que
<	Menor que
<b>Is null</b>	É nulo
>=	Maior ou igual a
<=	Menor ou igual a
<b>Like</b>	Parecido

# Comando SQL - DATA

---

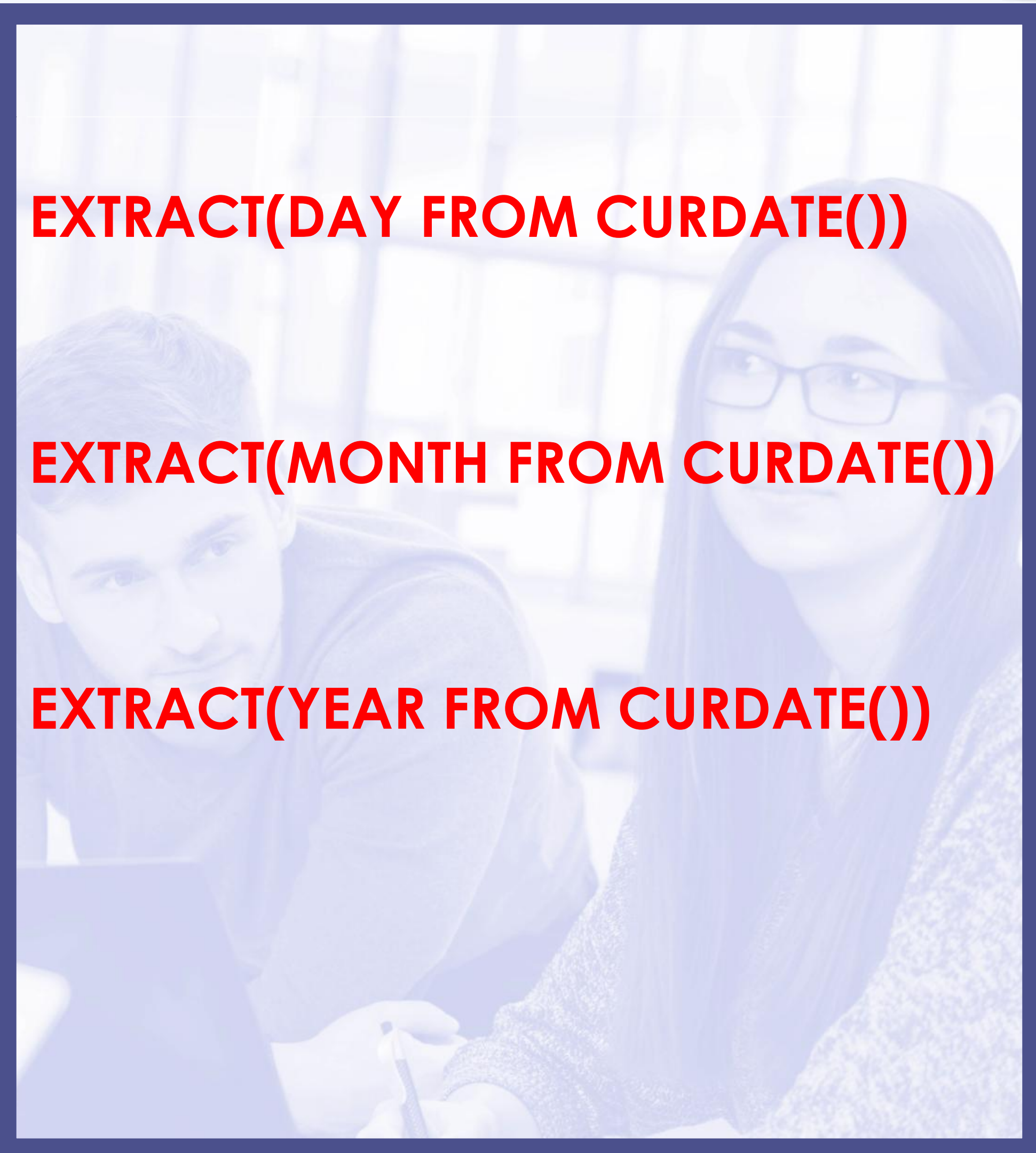


- Dia : DAY(Data);
- Mês: MONTH(Data);
- Ano: YEAR(Data);
- Data Corrente : CURDATE ou CURRENT\_DATE()
- Diferença entre duas datas é calculada em dias;
- Para o MySQL existem funções específicas como: DATEDIFF, PERIOD\_DIFF etc.

**EXTRACT(DAY FROM CURDATE())**

**EXTRACT(MONTH FROM CURDATE())**

**EXTRACT(YEAR FROM CURDATE())**





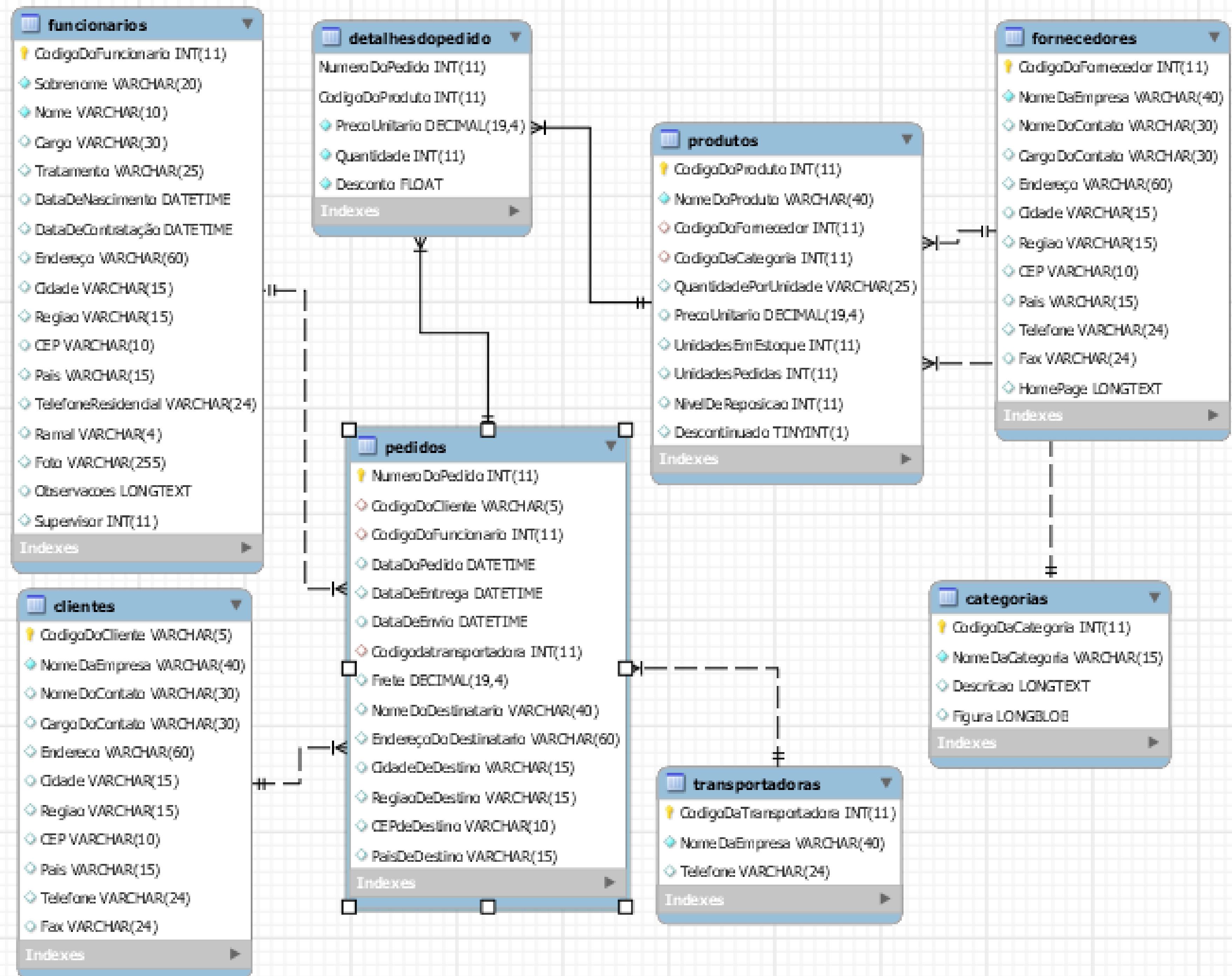
# Comando SQL - DATA



- **Funções para formatação de datas no MySQL**
  - Podemos aproveitar esse retorno “2012-08-2012” para demonstrar outra função muito interessante. A função “DATE\_FORMAT()” que recebe dois parâmetros(<data\_para\_formatar>, <formato\_desejado>) e tem a finalidade de alterar o formato da data, geralmente trabalhamos com datas nesse formato “MM/DD/AAAA”. Podemos alterar esse retorno da seguinte maneira.

Especificação	Descrição
%d	Dia do mês numérico(00..31)
%D	Dia do mês com sufixo (em Inglês)
%m	Mês, numérico(00..12)
%M	Nome do Mês(em Inglês)
%y	Ano, numérico (dois dígitos)
%Y	Ano, quatro dígitos numéricos

# Modelo usado nos comandos Select





# Comando Select

- Liste NOME (Nome e Sobrenome) e a Data de Nascimento de todos os funcionários;

**Select Concat (nome, ' ', sobrenome),  
datadenascimento  
From funcionarios;**

- Liste o Nome da Empresa e o telefone de todas as transportadoras;

**Select nomedaempresa, telefone  
from transportadoras;**







# Comando Select

- Liste da tabela de produtos Nome do produto e o valor armazenado em estoque (Qtd em Estoque \* Preço);

**Select nomedoproduto,  
unidadesemestoque \* precounitario  
From produtos**

- Liste da tabela produtos contendo código, nome, preço unitário e preço unitário reajustado em 30%;

**Select codigoproduto, nomedoproduto,  
precounitario, precounitario \* 1.30  
from produtos;**





# Comando Select

- Liste o nome e o endereço de todos os funcionários que moram nos EUA;

**Select nome, endereco**

**from funcionarios**

**Where pais = 'EUA';**

- Liste o código, nome e preço unitário de todos os produtos cujo preço unitário seja superior a 50.00;

**Select codigodoproduto, nomedoproduto,  
precounitario**

**From produtos**

**Where precounitario > 50**

**Order by precounitarios dec;**





# Comando Select

- Liste todos os Pedidos que tenham sido feitos no mês de janeiro de 1997;

**Select \* from pedidos**

**Where month(datadopedido) = 01**

**and year(datadopedido) = 1997;**

**Select \* from pedidos here**

**EXTRACT(MONTH FROM datadopedido) = 01**

**and**

**EXTRACT(YEAR FROM datadopedido) = 1996;**







# Comando Select

- Liste o nome e o endereço dos clientes que estejam localizados em madrid;

**Select nomedaempresa, endereco**

**From clientes**

**Where cidade = 'Madrid'**

- Liste o nome e o endereço dos clientes que estejam localizados no Brasil, Itália e Argentina;

**Select nomedaempresa, endereco**

**From clientes**

**Where pais in ('Brasil', 'Itália', 'Argentina')**





# Comando Select

- Liste os produtos que tenham preço unitário entre 50,00 e 70,00

**Select \* from produtos**

**Where precounitario between 50 and 70;**

- Liste o nome e o endereço das empresas que tenham seu nome iniciado pela letra 'B;

**Select nomedaempresa , endereco from clientes**

**Where nomedaempresa like 'B%'**





# Comando Select

- Até agora vimos como realizar pesquisas em uma única tabela. Contudo, na montagem do nosso modelo de dados sempre temos diversas tabelas. Logo é necessário sabermos como vincular a informação dessas tabelas de forma a mostrar a informação de maneira correta. A isto é dado o nome de união de tabelas (**join**).







# Comando Select

- Como visto anteriormente, a união entre as entidades do nosso modelo lógico se dá por meio de chaves primárias e estrangeiras. Essas chaves são, na representação física do modelo, as colunas que as tabelas têm em comum. No decorrer apresentarei as diversas formas de unir colunas e como implementá-las em SQL.



# Comando Select - Equijoin



- Para realizar a união de tabelas, basta acrescentarmos após a cláusula FROM do comando SELECT as tabelas que queremos unir. Devemos colocar na cláusula WHERE a condição de união das tabelas, ou seja, as respectivas chaves primária e estrangeira. Sintaxe:

## SELECT

```
Tabela1.coluna1, Tabela1.coluna2,  
Tabela2.coluna1, Tabela2.coluna2  
FROM Tabela1, Tabela2  
WHERE Tabela1.chave_primaria =  
Tabela2.chave_estrangeira
```

Obs.: Sempre que utilizamos mais de uma tabela em um select, é realizado um **produto cartesiano** entre as tabelas, isso é, uma combinação das linha de uma tabela com todas as linhas da outra tabela.

# Comando Select - Equijoin



- Pelo que vimos em aulas passadas, o nome das chaves primárias e o nome das chaves estrangeiras que são correspondentes são iguais, por esse motivo, a comparação entre as colunas, que garante a vinculação correta entre as linhas, deve ser possível diferença as tabelas as quais as colunas pertencem. Veja como ficaria o select apresentado anteriormente

Só é necessário se houver colunas com o mesmo nome em tabelas diferentes.

```
SELECT  
Tabela1.coluna1, Tabela1.coluna2,  
Tabela2.coluna1, Tabela2.coluna2  
FROM Tabela1 T1, Tabela2 T2  
WHERE T1.chave_primaria =  
T2.chave_estrangeira
```

```
SELECT T1.coluna1, T1.coluna2,  
T2.coluna1, T2.coluna2  
FROM Tabela1 T1, Tabela2 T2  
WHERE T1.chave_primaria =  
T2.chave_estrangeira
```



# Comando Select



- Liste todos pedidos realizados com o funcionário de nome “Nancy”;

**Select \***

**From pedidos p, funcionarios f**

**Where p.codigodofuncionario =  
f.codigodofuncionario**

**And nomedofuncionario = ‘Nancy’;**

- Liste o número do pedido e o nome da transportadora que fez a entrega;

**select numerodopedido, nomedaempresa**

**From pedidos p, transportadoras t**

**Where p.codigodatransportadora =  
t.codigodatransportadora**



# Join - Outer



- O Outer join possui o funcionamento um pouco diferente do anterior. Ao usar o Outer join, além de podermos retornar os registros das duas tabelas seguindo alguma relação, ainda podemos retornar registros que não entram nesta relação.

Existem dois tipos de Outer Join  
Left Outer Join  
Right Outer Join



# Join – Left

- Left Outer Join ou Left Join

Aplica o conceito de Outer Join na tabela que se encontrar à esquerda da relação (ou seja, o resultado vem da tabela à esquerda). No nosso exemplo, é o JOIN que resolve a situação 1, a consulta retorna apenas os clientes que ainda não efetuaram nenhum pedido.

**SELECT \***

**FROM Clientes c LEFT OUTER JOIN Pedidos p  
ON c.codigodocliente = p.codigodocliente  
WHERE p.codigodocliente IS NULL**





# Join - Right



- Right Outer Join ou Right Join

Aplica o conceito de Outer Join na tabela que se encontrar à esquerda da relação (ou seja, o resultado vem da tabela à esquerda). No nosso exemplo, é o JOIN que resolve a situação 2, a consulta retorna apenas os pedidos que foram realizados por clientes não cadastrados.

**SELECT \***

**FROM Clientes c RIGHT OUTER JOIN Pedidos p**  
**ON c.codigodocliente = p.codigodocliente**  
**WHERE c.codigodocliente is null;**





# Join - Full

•Todas as linhas de dados da tabela à esquerda de **JOIN** e da tabela à direita serão retornadas pela cláusula **FULL JOIN** ou **FULL OUTER JOIN**. Caso uma linha de dados não esteja associada a qualquer linha da outra tabela, os valores das colunas a lista de seleção serão nulos. Caso contrário, os valores obtidos serão baseados nas tabelas usadas como referência.

```
SELECT * FROM  
PEDIDOS P FULL JOIN CLIENTES C  
ON P.CODIGODOCLIENTE  
= C.CODIGODOCLIENTE
```





# Join - Cross

Cross join mostra combinação de todas as linhas das tabelas unidas. Nenhuma coluna comum será necessária para fazer um cross join. Quando você usa cross joins, o banco produz um produto cartesiano no qual o número de linhas no resultado seja igual ao número de linhas da primeira tabela multiplicado pelo número de linhas da segunda tabela. Por exemplo, se são 8 linhas na primeira tabela e 9 linhas na outra tabela SQL Server retorna um total de 72 linhas.







# Join - Self

•É um relacionamento igual ao Equijoin, o que lhe faz ser Self-Join, é que a segunda tabela é a mesma que a primeira. Muito conhecido como auto-relacionamento. Em um Self-Join, não é possível utilizar a cláusula using. Por exemplo, quero saber qual o cônjuge de uma pessoa cadastrada. O cônjuge também é uma pessoa cadastrada, e o relacionamento entre as informações ocorre pela coluna cpf\_conjuge.

```
select p.cpf_pessoa,p.nome, c.cpf_pessoa as  
"CPF Conjuge", c.nome as "Nome Conjuge"  
from tb_pessoa p join tb_pessoa c  
on (p.cpf_pessoa = c.cpf_conjuge);
```





# Comando Select

- Criar uma consulta que mostre o nome dos produtos, o nome da categoria a que pertence.

```
Select nomedoproduto, nomedaategoria  
From produtos p inner join categorias c  
On p.codigodacategoria =  
c.codigodacategoria
```

- Criar uma consulta que mostre o nome dos produtos, o nome da categoria a que pertence.

```
Select nomedoproduto, nomedaategoria  
From produtos p inner join categorias c  
On p.codigodacategoria =  
c.codigodacategoria
```





# Comando Select

- Criar uma consulta que mostre o nome dos produtos, o nome da categoria e o nome do fornecedor de cada produto a que pertence.

```
Select nomedoproduto, nomedacategoria,  
nomedaempresa  
From produtos p inner join categorias c  
On p.codigodacategoria =  
c.codigodacategoria  
Inner join fornecedores f  
on p.codigodoornecedor =  
f.codigodoornecedor
```





# Selects com funções Agregadas



**ser**  
educacional

gente criando o futuro



# Comando Select

- Funções Agregadas

<b>Avg(argumento)</b>	<b>Retorna a média dos valores do argumento</b>
<b>Max(argumento)</b>	<b>Retorna o maior valor do argumento</b>
<b>Min(argumento)</b>	<b>Retorna o menor valor do argumento</b>
<b>Sum(argumento)</b>	<b>Retorna o somatório dos valores do argumento</b>
<b>Count(argumento)</b>	<b>Retorna a número de linhas do argumento</b>

- Liste o maior preço, o menor preço e a média dos preços dos produtos;

```
Select max(precounitario),  
min(precounitario),  
avg(precounitario)  
From produtos
```



# Comando Select

- Funções Agregadas

<b>Avg(argumento)</b>	<b>Retorna a média dos valores do argumento</b>
<b>Max(argumento)</b>	<b>Retorna o maior valor do argumento</b>
<b>Min(argumento)</b>	<b>Retorna o menor valor do argumento</b>
<b>Sum(argumento)</b>	<b>Retorna o somatório dos valores do argumento</b>
<b>Count(argumento)</b>	<b>Retorna a número de linhas do argumento</b>

- Liste a quantidade de pedidos dos mês de janeiro de 1998;

```
Select count(*)  
from pedidos  
Where year(datadopedido) = 1998  
and month(datadopedido) = 01
```



# Select – Criar agrupamentos



- Cláusula GROUP BY:

Grupamento de colunas – Sumários – Não

```
SELECT COLUNAS  
FROM TABELA  
WHERE CONDIÇÃO  
GROUP BY CAMPOS
```

- Media de fretes por país;

```
Select avg(frete), pais  
From clientes  
Group by pais;
```

- Liste a quantidade de Clientes por país;

```
Select pais, count(*)  
From clientes  
Group by pais
```

- Liste a quantidade de pedidos no mês de Janeiro de 2007;

```
Select count(*),  
to_char(datadopedido, 'mm/yyyy')  
From clientes  
Group by to_char(datadopedido,  
'mm/yyyy')
```

# Select – Funções Agregadas

---



- Exercícios:
  - Liste a quantidade de pedidos dos mês de janeiro de 1998;

**Select count(\*) from pedidos**

**Where year(datadopedido) = 1998**

**and month(datadopedido) = 01**

# Select – Funções Agregadas

---



- Exercícios:
  - Liste o maior preço, o menor preço e a média dos preços dos produtos;

```
Select max(precounitario),  
min(precounitario),  
avg(precounitario)  
From produtos
```



# Select – Funções Agregadas

---



Cláusula GROUP BY:

Grupamento de colunas – Sumários – Não

**SELECT COLUNAS**

**FROM TABELA**

**WHERE CONDIÇÃO**

**GROUP BY CAMPOS**

# Select – Funções Agregadas

---



- Liste a quantidade de Clientes por país;

**Select pais, count(\*)**

**From clientes**

**Group by pais**

- Liste a quantidade de pedidos no mês de Janeiro de 2007;

**Select count(\*), to\_char(datadopedido, 'mm/yyyy')**

**From clientes**

**Group by to\_char(datadopedido, 'mm/yyyy')**

# Select – Funções Agregadas

---



- Media de fretes por país;

**Select avg(frete), pais**  
**From clientes**  
**Group by pais;**



# Selects com Subqueries



**ser**  
educacional  
gente criando o futuro

# Subqueries – Consultas Aninhadas

---



É possível integrar uma instrução SQL noutra. Quando tal é efetuado nas instruções **WHERE** ou **HAVING**, possuem uma construção de consulta secundária.

A sintaxe será a seguinte:

```
SELECT "nome_coluna1"  
FROM "nome_tabela1"  
WHERE "nome_coluna2" [Comparison Operator]  
      (SELECT "nome_coluna3"  
       FROM "nome_tabela2"  
       WHERE [condição])
```

# Predicados utilizados em Subqueries

---



- As subqueries são usadas em:
  - Predicados de comparação
  - Predicado IN
  - Predicados ALL ou ANY
  - Predicado EXISTS





# Subqueries

---

- Qual o código e nome dos Clientes que moram nas mesmas cidades que os fornecedores?

```
SELECT    codigodocliente, nomedaempresa  
FROM      clientes;
```

```
SELECT    cidade  
FROM      fornecedores;
```

# Subqueries

---



- Integração das duas consultas

```
SELECT      codigodocliente, nomedaempresa  
FROM        clientes  
WHERE       cidade in ( SELECT cidade FROM fornecedores);
```



# Subqueries

---

- Qual o número do pedido e o código do cliente que tem valor de frete maior que o valor médio do frete de 2008, organizado por código do cliente;

```
SELECT numerodopedido, codigodocliente  
FROM pedidos  
WHERE frete > (select avg(frete)  
                FROM pedidos  
                WHERE year(datadopedido) = 2008)  
  
Order by 2;
```





# Operador EXISTS

- Qual o nome dos clientes que tem pedidos cadastrados;

```
SELECT nomedaempresa  
FROM clientes  
WHERE EXISTS  
  (SELECT * FROM pedidos  
   WHERE clientes.codigodopedido = pedidos.codigodocliente)
```

**A condição é VERDADEIRA se o resultado da subquery não for vazio**



# Operador NOT EXISTS

---

- Qual o nome dos clientes que não tem pedidos cadastrados;

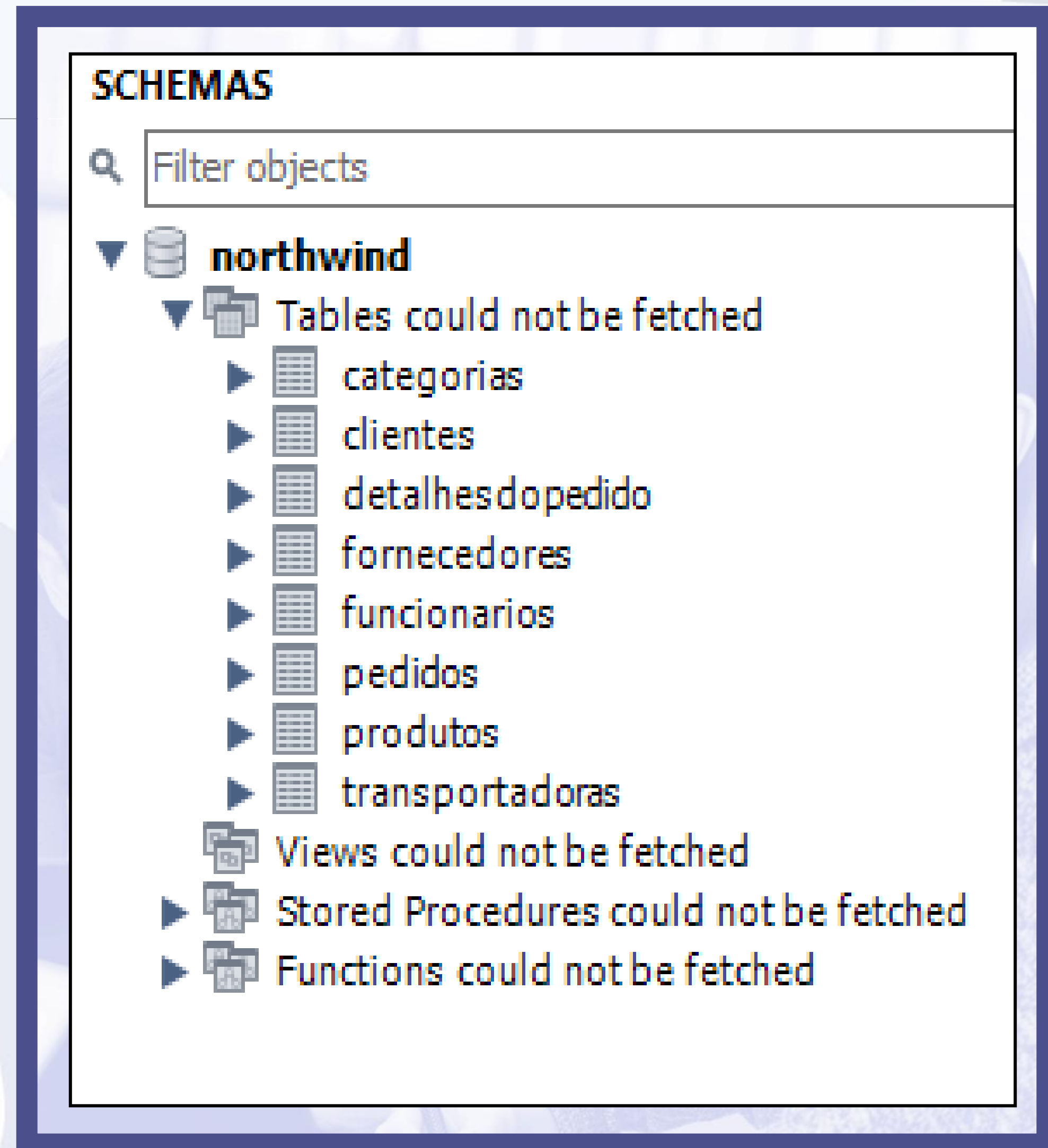
```
SELECT nomedaempresa  
FROM clientes  
WHERE NOT EXISTS  
  (SELECT * FROM pedidos  
   WHERE clentes.codigodopedido = pedidos.codigodocliente)
```

**A condição é VERDADEIRA se o resultado da subquery não for vazio**



# Comandos SQL;

- **SHOW DATABASES;**
- O SGBD organiza o banco de dados em diversas bases de dados e, em cada uma, podemos criar várias tabelas com seus atributos. Logo, ao nos referenciarmos ao nosso banco de dados, usaremos o termo “base de dados”, que são sinônimos, mas apenas para não haver confusão sobre qual base está sendo referida.
- Cada banco de dados organizado pelo SGBD é, portanto, composto por sua base de dados, com suas tabelas e atributos totalmente independentes e acessados separadamente de outras bases de dados no mesmo SGBD.



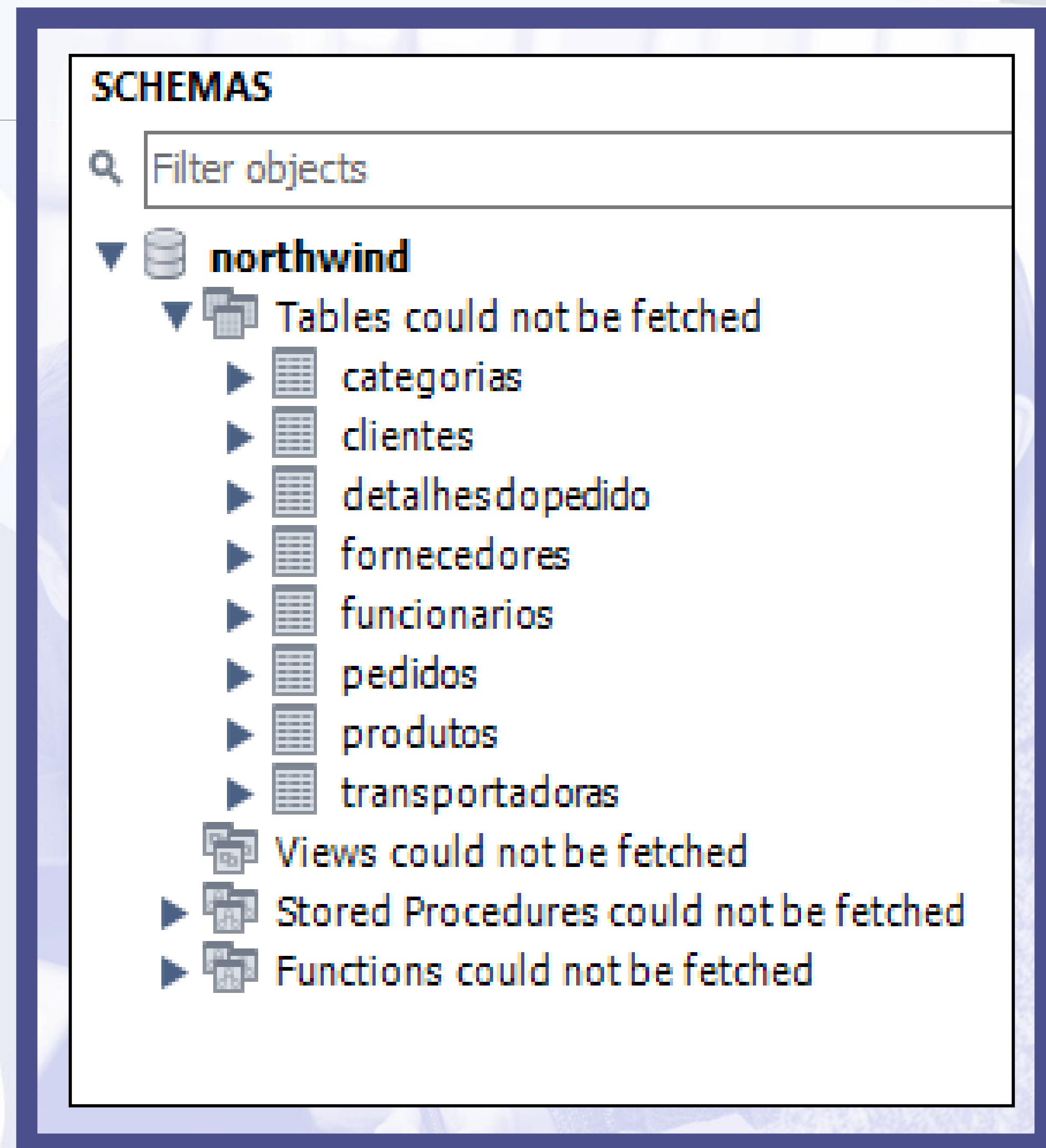




# Comandos SQL;

## CREATE DATABASES;

- Para criar uma nova base de dados chamada Controle\_de\_Vendas, mas, como o MySQL normalmente converterá o nome das bases de dados para minúsculo, vamos então referenciá-la sempre em minúsculo: **controle\_de\_vendas**.
- O comando para criação de bases de dados é **CREATE DATABASE**, seguido do nome da base de dados. Logo, o comando deverá ser digitado da seguinte forma  
**CREATE DATABASE** controle\_de\_vendas;





# Comandos SQL;

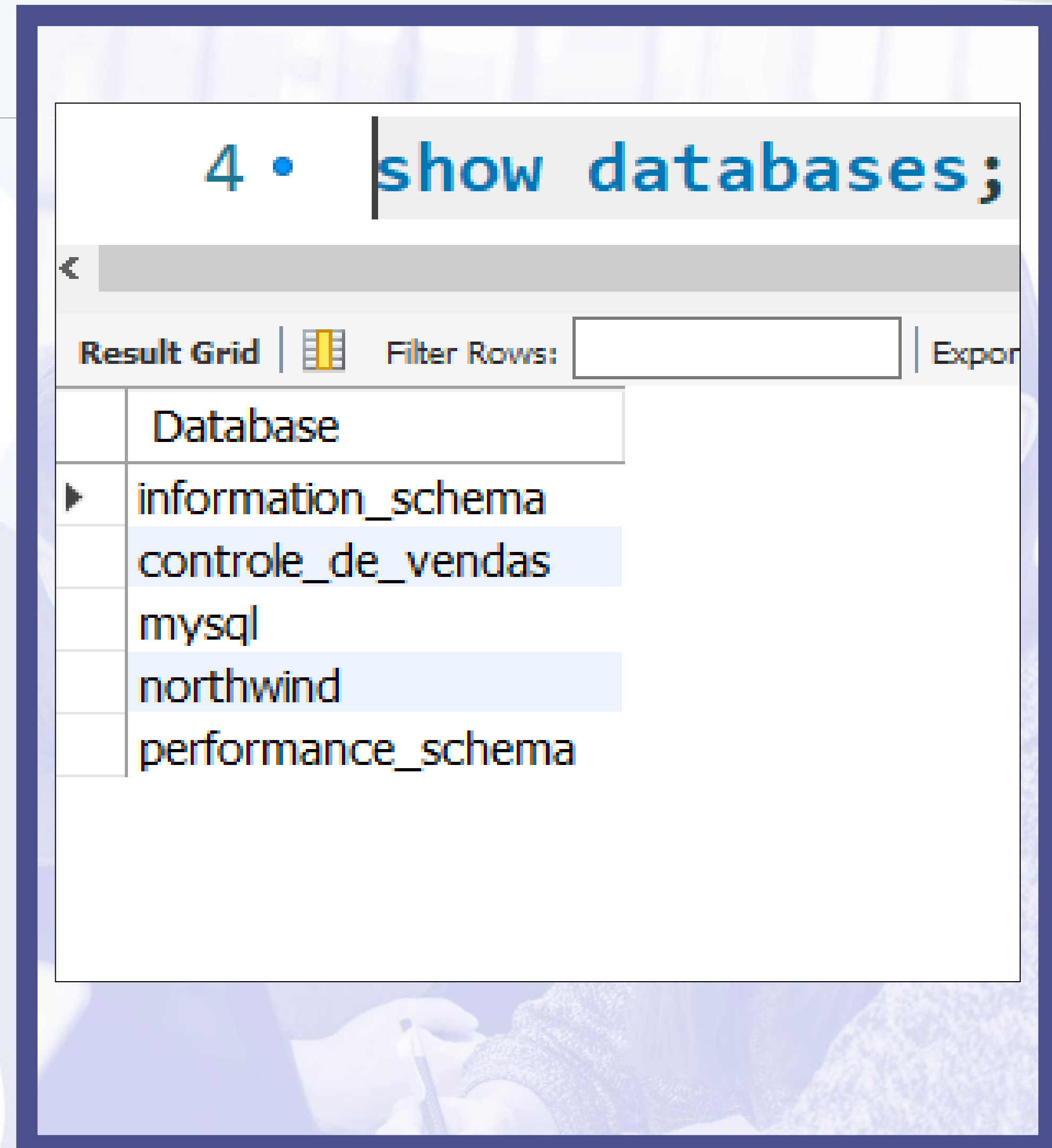
## Comando SHOW

**SHOW DATABASES;** – listará as bases criadas.

**SHOW TABLES;** – listará as tabelas criadas.

**SHOW CREATE TABLE** nome da tabela; – listará a estrutura da tabela indicada.

**SHOW CREATE DATABASE** nome da base de dados; – listará dados sobre a base de dados indicada.



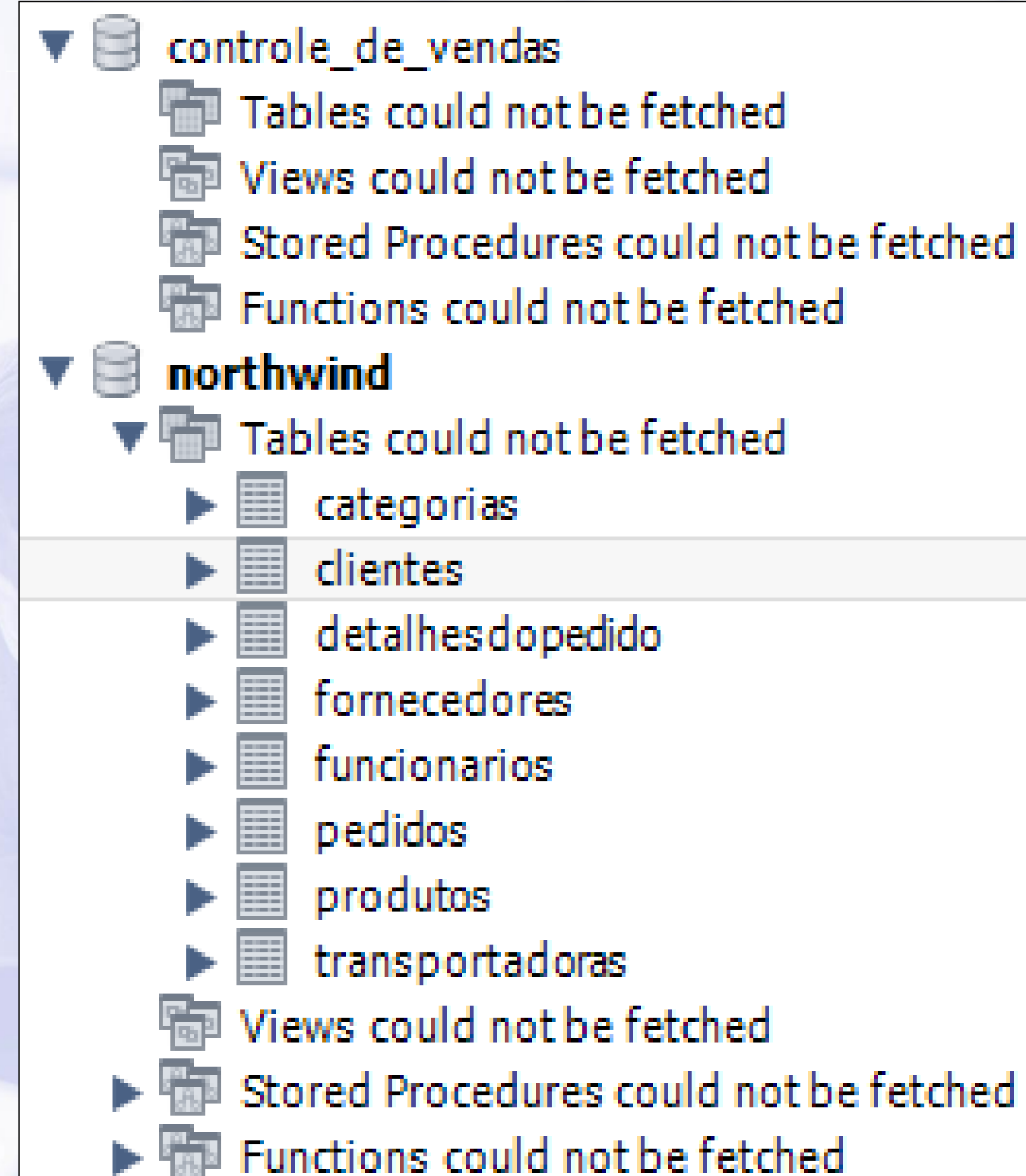


# Comandos SQL;

## Comando USE

**USE** controle\_de\_vendas;

Você perceberá que o prompt de comando do MySQL trará sempre o nome da base de dados em uso. Para operar outras bases de dados, basta apenas dar um comando USE e o nome da outra base.





# Comandos DDL



**ser**  
educacional  
gente criando o futuro



# Comandos SQL;

## Comando Create Table

```
CREATE TABLE NOME_TABELA  
(COLUNA1 TIPO(TAM,DEC) Nulidade,  
COLUNA2 TIPO(TAM,DEC) Nulidade,  
COLUNA3 TIPO(TAM,DEC) Nulidade,  
.  
.  
.  
COLUNA3 TIPO(TAM,DEC) Nulidade);
```

## Exemplo Create Table

```
CREATE TABLE Cliente (  
    Cod_cli INTEGER ,  
    Nome VARCHAR (50),  
    Endereco VARCHAR (50),  
    Telefone VARCHAR (20)  
);
```



# Comandos SQL;

## Comando Create Table

```
CREATE TABLE NOME_TABELA  
(COLUNA1 TIPO(TAM,DEC) Nulidade,  
COLUNA2 TIPO(TAM,DEC) Nulidade,  
COLUNA3 TIPO(TAM,DEC) Nulidade,  
.  
.  
.  
COLUNA3 TIPO(TAM,DEC) Nulidade);
```

### Exemplo Create Table com chave primária

```
CREATE TABLE Cliente  
(Cod_cli INTEGER primary key,  
Nome VARCHAR (50),  
Endereco VARCHAR (50),  
Telefone VARCHAR (20)  
);
```





# Comandos SQL;

## Comando Create Table

```
CREATE TABLE NOME_TABELA  
(COLUNA1 TIPO(TAM,DEC) Nulidade,  
COLUNA2 TIPO(TAM,DEC) Nulidade,  
COLUNA3 TIPO(TAM,DEC) Nulidade,  
.  
.  
.  
COLUNA3 TIPO(TAM,DEC) Nulidade);
```

**Exemplo Create Table com  
chave primária e auto  
incremento**

```
CREATE TABLE Cliente  
(cod_cli int primary key  
auto_increment, nome  
varchar(50),  
endereco varchar(50),  
telefone varchar(20));
```



# Comandos SQL;

- Exemplo **Create Table** com chave primária – modo padrão

**Create table** cliente

(cod\_cli integer not null,

nome varchar(50),

endereco varchar(50),

telefone varchar(50),

**constraint pk\_cliente primary key(cod\_cli));**

- Comandos para mostrar a tabela e estrutura das tabelas

**SHOW TABLES;** /\* Apenas mostra as tabelas \*/

**DESCRIBE** cliente; /\* Descreve a tabela \*/

**SHOW CREATE TABLE** cliente; /\* Mostra a estrutura da tabela \*/

3 • describe clientes;

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	Field	Type	Null	Key	Default	Extra
▶	CodigoDoCliente	varchar(5)	NO	PRI	NULL	
	NomeDaEmpresa	varchar(40)	NO	MUL	NULL	
	NomeDoContato	varchar(30)	YES		NULL	
	CargoDoContato	varchar(30)	YES		NULL	
	Endereco	varchar(60)	YES		NULL	
	Cidade	varchar(15)	YES	MUL	NULL	
	Regiao	varchar(15)	YES	MUL	NULL	
	CEP	varchar(10)	YES	MUL	NULL	
	Pais	varchar(15)	YES		NULL	
	Telefone	varchar(24)	YES		NULL	
	Fax	varchar(24)	YES		NULL	



# Comandos SQL;

- Criando restrições de integridade;

- Criando chave primaria:

`constraint pk_nome primary key(atributo)`

**Quando a chave é composta os atributos são apresentados separados por uma virgula**

**Nome da Chave Primária**

**Atributo que é a chave primária**

- Criando chave estrangeira:

`constraint fk_nome foreign key (atb_tabela)  
references Tbl_diferente (atb_tbl_dif)`

**Tabela de onde a chave estrangeira veio**

**Atributo da tabela de referência**



# Alterando estrutura da tabela



- **Alter Table**

<b>ADD</b> (incluir)	uma nova coluna
<b>MODIFY</b> (modificar)	o tamanho de uma coluna ou se a coluna deverá aceitar valores nulos
<b>DROP</b> (eliminar)	uma coluna existente
<b>RENAME</b> (trocar o nome de)	uma coluna existente
<b>RENAME TABLE</b> (trocar nome tabela)	
<b>CHANGE</b> (trocar os dados de uma coluna)	

# Alterando estrutura da tabela

---



- Eliminando uma coluna:

```
ALTER TABLE LISTA_DE_HOSPEDES  
DROP DESCONTO;
```

- Adicionando uma Coluna:

```
ALTER TABLE LISTA_DE_HOSPEDES  
ADD DESCONTO DECIMAL(2,2);
```

- Alterando uma coluna:

```
ALTER TABLE clienteb MODIFY nome  
VARCHAR(50) NOT NULL;
```

- Alterando uma coluna (MySQL):

```
ALTER TABLE clienteb CHANGE nome  
VARCHAR(50) NOT NULL;
```

- Trocando o nome de uma coluna:

```
ALTER TABLE LISTA_DE_HOSPEDES  
RENAME GARCOM SERVENTE;
```

- Alterando o nome da tabela:

```
RENAME TABLE CLIENTE1 TO CLINETEB;
```

# Alterando estrutura da tabela



- Criando restrições de integridade;
- Criando chave estrangeira:

```
constraint fk_nome foreign key (atb_tabela)  
references Tbl_diferente (atb_tbl_dif)
```

Tabela de onde a chave estrangeira veio

Atributo da tabela de referência

Tabela de referência

**ALTER TABLE** Produto

**ADD CONSTRAINT** fk\_Prod\_Forn **FOREIGN Key** (cod\_for) **references** Fornecedor  
(Cod\_for);



# Alterando estrutura da tabela

---



```
ALTER TABLE Produto  
ADD CONSTRAINT fk_Pro_For FOREIGN KEY (cod_fornecedor)  
REFERENCES Fornecedor (Cod_for);
```

Ou

```
ALTER TABLE Produto  
ADD CONSTRAINT fk_Pro_For FOREIGN KEY (cod_fornecedor)  
REFERENCES Fornecedor (Cod_for) ON DELETE NO ACTION ON UPDATE NO  
ACTION;
```

# Comandos DML



**ser**  
educacional  
gente criando o futuro



# Inserindo Dados

```
INSERT INTO Nome-Tabela [(col1, col2,...,  
    coln)] []-> opcional  
VALUES ( conteúdo1, conteúdo2,...,  
    conteúdoN)
```

- O tipo e o tamanho dos dados selecionados têm que ser compatíveis com as especificações das colunas na tabela de destino. Por exemplo, números podem ser incluídos em uma coluna de tipo caractere, mas o contrário não é aceito.

Exemplo:

```
INSERT INTO CATEGORIAS (  
    CODIGODACATEGORIA,  
    NOMEDACATEGORIA,  
    DESCRICAO, FIGURA)  
VALUES ( 99,  
    'CATEGORIA 99' ,  
    'DESCRIÇÃO 99',  
    '' );
```





# Inserindo Dados

```
INSERT INTO Nome-Tabela [(col1, col2,...,  
    coln)] []-> opcional  
VALUES ( conteúdo1, conteúdo2,...,  
    conteúdoN)
```

- O tipo e o tamanho dos dados selecionados têm que ser compatíveis com as especificações das colunas na tabela de destino. Por exemplo, números podem ser incluídos em uma coluna de tipo caractere, mas o contrário não é aceito.

Exemplo:

```
INSERT INTO EMPLOYEE  
VALUES
```

```
('Richard', 'K', 'Marini',  
    '653258653', '1962-12-30',  
    '98 Oak Forest, Katy, TX',  
    37000,  
    '987654321', 4);
```



# Inserindo Dados

- Alterando conteúdo de Linhas

```
UPDATE Nome-da-Tabela  
SET coluna = (valor, resultado de  
operacao)  
[Where condição];
```

Exemplo:

```
UPDATE CATEGORIAS  
SET descricao_categoria =  
'TESTE DE ALTERACAO'  
WHERE codigodacategoria  
= 99;
```

# Excluindo Dados

---

- Eliminando Linhas e uma tabela

**DELETE FROM Nome-da-Tabela**  
[Where condição];

Exemplo:

```
DELETE FROM CATEGORIAS  
WHERE codigodacategoria = 99;
```

```
DELETE FROM PEDIDOS  
WHERE  
    numerodopedido = 10409
```





# Alterando estrutura da tabela

---



O exemplo a seguir, com parâmetro **CASCADE**, também impede que um produto seja cadastrado sem um fornecedor válido, porém tem um comportamento diferente no caso de alterações ou exclusões do fornecedor. Se o código do fornecedor na tabela Fornecedor for alterado, automaticamente todos os seus produtos serão atualizados com o novo código. Se o fornecedor for excluído da tabela, todos os produtos com esse fornecedor serão também automaticamente excluídos. Isso é uma restrição parcial, já que permite alguns ajustes. Vejamos:

```
ALTER TABLE Produto ADD CONSTRAINT fk_Pro_For FOREIGN KEY  
(Cod_Fornecedor) REFERENCES Fornecedor (Cod_for) ON DELETE CASCADE ON  
UPDATE CASCADE;
```

# Criando Views



**ser**  
educacional  
gente criando o futuro



# Criação de View

- As View podem ser consideradas como tabelas virtuais. Regra geral, uma tabela tem um conjunto de definições e armazena fisicamente os dados. Uma vista também tem um conjunto de definições, que são criadas sobre tabela(s) ou outra(s) vista(s), e não armazena fisicamente os dados.
- A sintaxe para criar uma visão é a seguinte:
- **CREATE VIEW "nome\_vista" AS "Instrução SQL"**

## SCHEMAS



controle\_de\_vendas



Tables could not be fetched



Views could not be fetched



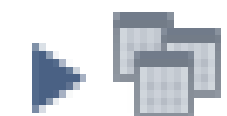
Stored Procedures could not be fetched



Functions could not be fetched



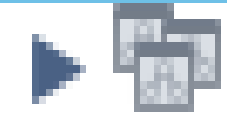
northwind



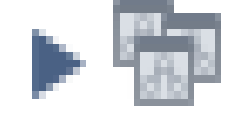
Tables could not be fetched



Views could not be fetched



Stored Procedures could not be fetched



Functions could not be fetched





# Criação de View

- Você pode colocar dentro de uma view o comando select feito com os seguintes critérios:
  - Consultas com união (Join)
  - Consultas com união (Union)
  - Consultas com Group By com função de totalização
  - Consultas com subconsultas (Subquery)
  - Consultas com Order By apenas com Top.

## SCHEMAS



controle\_de\_vendas



Tables could not be fetched



Views could not be fetched



Stored Procedures could not be fetched



Functions could not be fetched



northwind



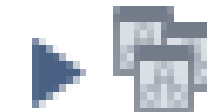
Tables could not be fetched



Views could not be fetched



Stored Procedures could not be fetched



Functions could not be fetched



# Criação de View

- Para criar uma view, utilizamos o comando CREATE VIEW.  
CREATE VIEW <view\_name>  
AS  
<instrução\_SELECT>

**CREATE view v\_pedido as**  
**Select numerodopedido,**  
**nomedaempresa, datadopedido**  
**From clientes a, pedidos b**  
**Where a.codigodocliente =**  
**b.codigodocliente;**

## SCHEMAS



controle\_de\_vendas



Tables could not be fetched



Views could not be fetched



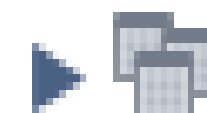
Stored Procedures could not be fetched



Functions could not be fetched



northwind



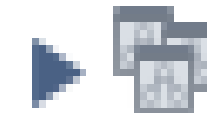
Tables could not be fetched



Views could not be fetched



Stored Procedures could not be fetched



Functions could not be fetched













# Criação de View

- Como fica o comando select após a criação da View.

**Select \* from v\_pedido;**

## SCHEMAS

Filter objects

- ▼  controle\_de\_vendas
  -  Tables could not be fetched
  -  Views could not be fetched
  -  Stored Procedures could not be fetched
  -  Functions could not be fetched
- ▼  northwind
  - ▶  Tables could not be fetched
  - ▶  Views could not be fetched
  - ▶  Stored Procedures could not be fetched
  - ▶  Functions could not be fetched



# Criando Índices



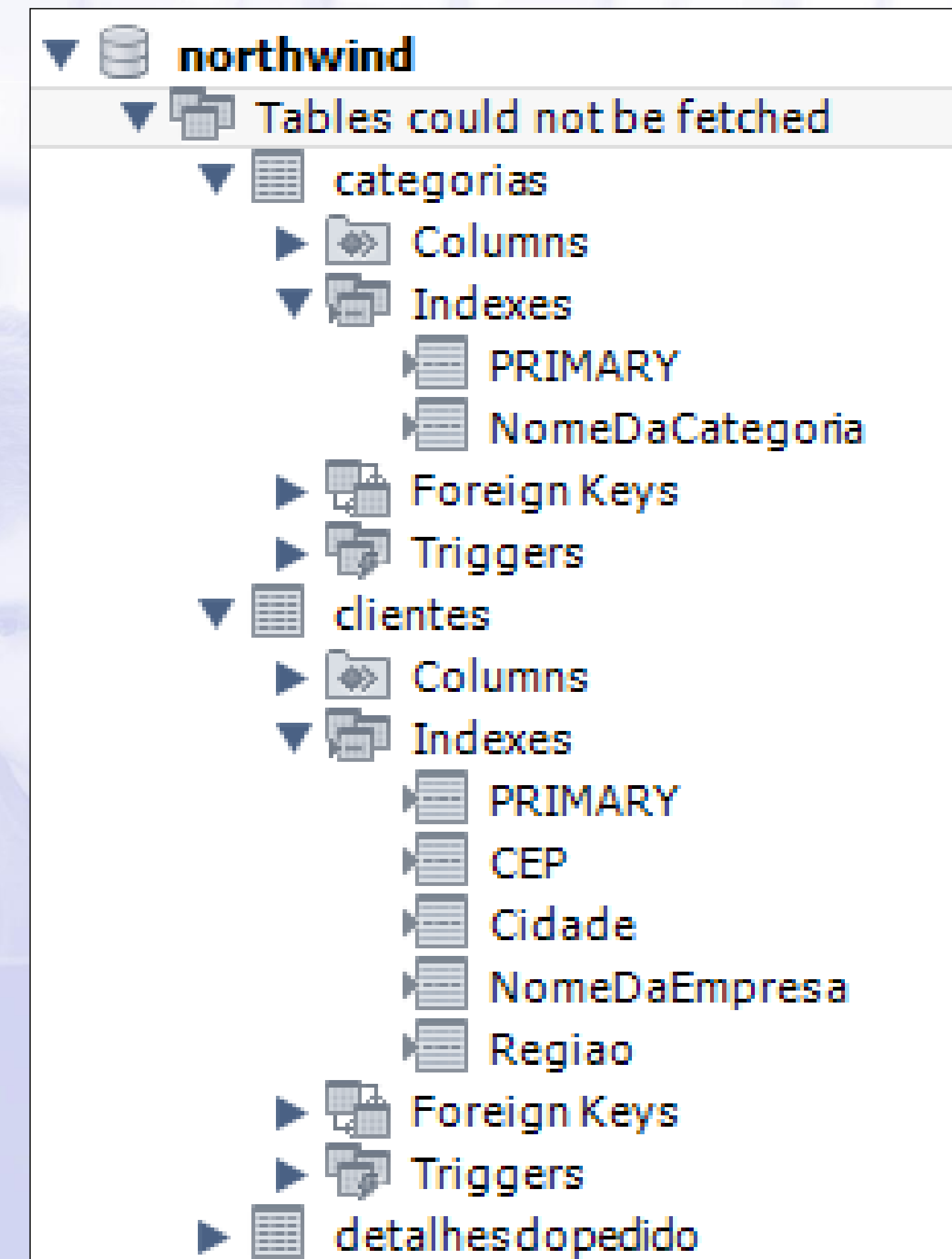
**ser**  
educacional  
gente criando o futuro

# Índices



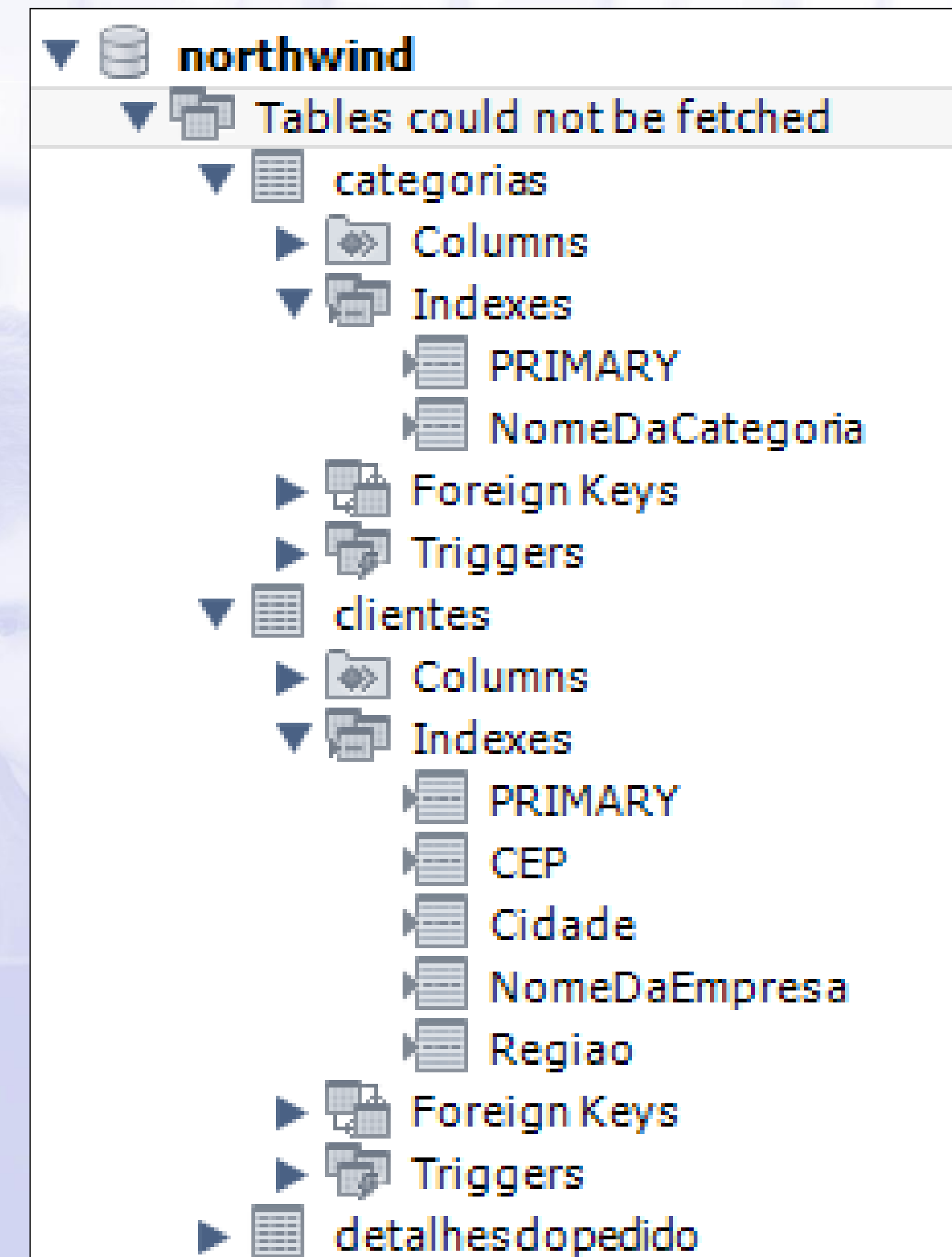
## Algumas Dicas quanto a índices

- Colunas para se indexar
  - PK's , FK's, colunas que se acessam por ranges (BETWEEN, > < )
  - Colunas que se usa para sort order
  - Colunas que se usa para grouping ou agregações
- Colunas para não se indexar
  - Coluna que você raramente referencia numa query
  - Colunas com alta cardinalidade como por exemplo Masculino e Feminino
  - Colunas com Ntex , Image, Text



# Índices

- “Um índice é um objeto de um esquema que pode acelerar a recuperação de linhas usando um ponteiro.”
- Existe para melhorar o processo de recuperação dos dados;
- A quantidade de índices afeta na performance de atualização das tabelas;
- **Sintaxe ANSI:**
  - Criar CREATE INDEX
  - Excluir DROP INDEX.
  - **CREATE [UNIQUE] INDEX nome\_índice ON nome\_tabela(coluna1, coluna2,.....);**





# Índices

- **Comando de Criação**

**CREATE**

**INDEX** nome\_do\_índice **ON** nome\_da\_tab  
ela(nome\_do\_campo);

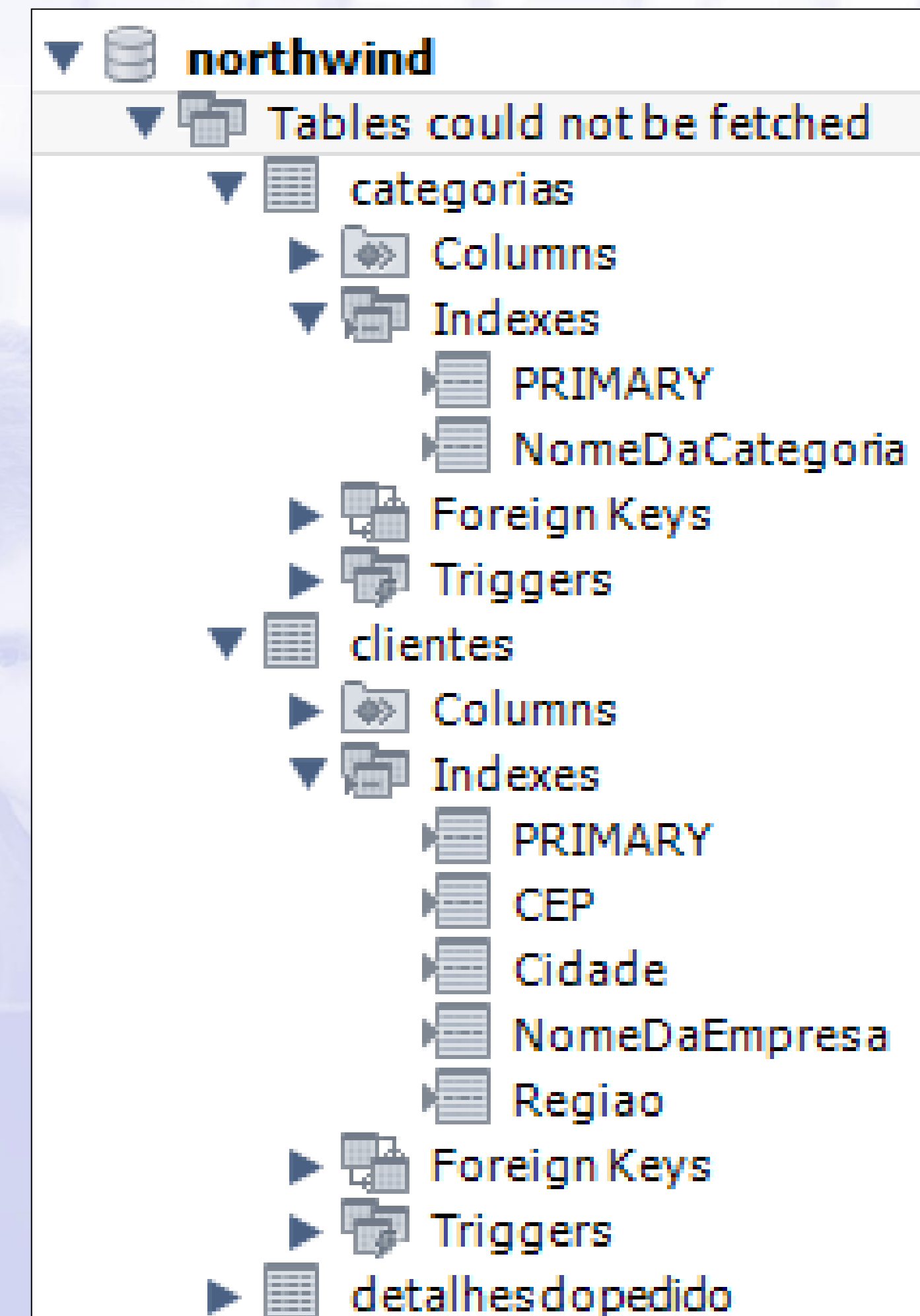
- **Exemplo:**

**CREATE**

**INDEX** indexnome **ON** Cliente(Nome)

- **Eliminar um índice**

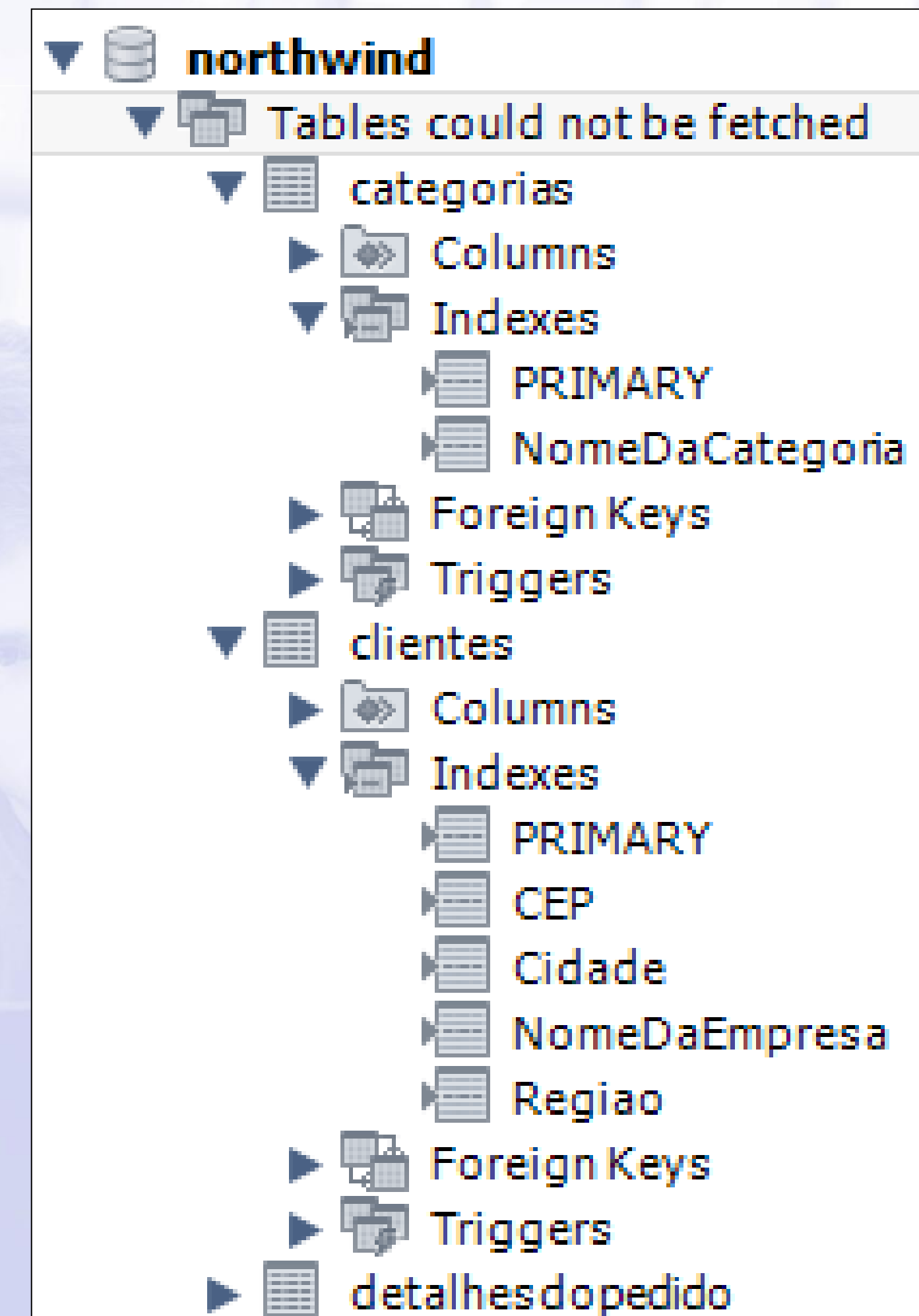
**DROP INDEX** nomecli **ON** Cliente;



# Índices

- É possível criar índice na criação da tabela

```
CREATE TABLE Livro(  
ID INT NOT NULL PRIMARY KEY,  
Codigo INT,  
Titulo VARCHAR(50),  
Autor VARCHAR(50),  
INDEX (Codigo),  
INDEX (Autor)  
);
```



# Criando Sequences



**ser**  
educacional

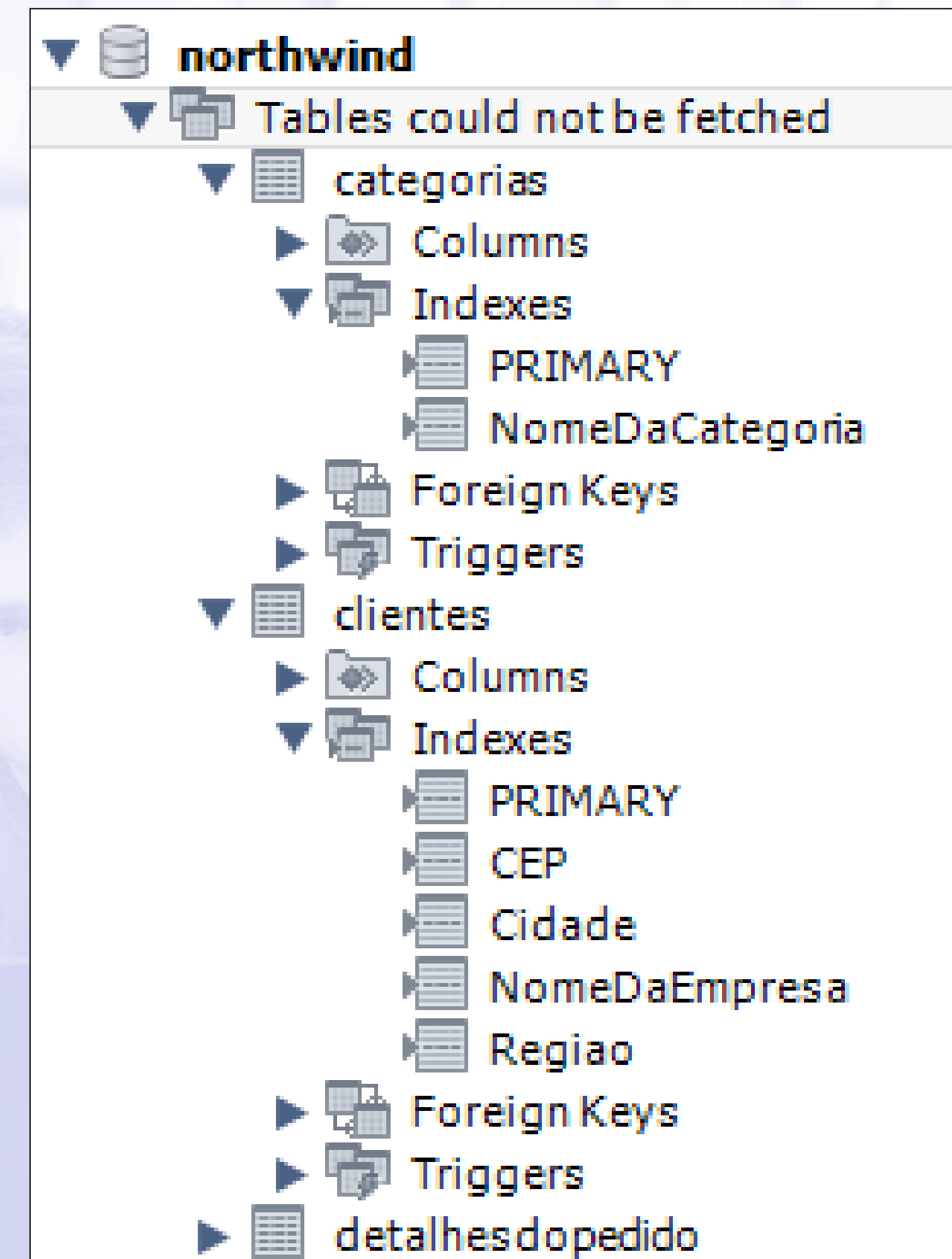
gente criando o futuro





# Criação de Sequencias

- O SQL possui um recurso para criar um objeto na base de dados que gera uma sequência automática de números. O objeto é um elemento externo às tabelas, mas seu valor, sempre incrementado, pode ser usado para alimentar atributos que necessitem de um campo com essa característica.





# Criação de Sequencias

- Uma sequência numérica pode ser útil, inclusive, por aplicações externas em que, ao acessarem o banco de dados, um novo número e exclusivo pode ser gerado.
- Contudo, dependendo do SGBD adotado, pode haver muitas variações deste recurso.
- O formato genérico do comando é:  
**CREATE SEQUENCE** objeto\_sequencial **START WITH** valor inicial **INCREMENT BY** incremento;

Para gerar um número de 1 em 1, iniciando em 1:

```
CREATE SEQUENCE Num_sequencial  
START WITH 1 INCREMENT BY 1;
```

Para gerar um número iniciando em 1000 e variando de 10 em 10.

```
CREATE SEQUENCE Num_seq START WITH 1000 INCREMENT BY 10;
```



# Constraint Check

- Observe que a cláusula CHECK e a condição da restrição podem ser usadas também para especificar restrições nos atributos e nos domínios e nas tuplas.
- O projetista do esquema poderia usar CHECK em atributos, nos domínios e nas tuplas apenas quando estiver certo de que a restrição só poderá ser violada pela inserção ou atualização das tuplas.
- O projetista deveria usar CREATE ASSERTION somente nos casos em que não for possível usar CHECK nos atributos, nos domínios ou nas tuplas, assim a verificação será implementada com maior eficiência pelo SGBD.

```
ALTER TABLE pedidos  
ADD CONSTRAINT chkstatus  
CHECK (status in (1,2,3,4));
```

```
ALTER TABLE funcionarios  
CONSTRAINT VERIF_SAL CHECK  
(SALARIO >= 10000),
```



# Tecnologias Emergentes



**ser**  
educacional

gente criando o futuro



# Tecnologia Emergentes

- Tecnologias emergentes tratam de tendências do emprego de tecnologias inovadoras, adaptadas, modernas ou não, mas que podem se tornar solução para as questões e necessidades crescentes no universo tecnológico, entre eles o dos bancos de dados. Muitas tendências podem se tornar realidade rapidamente e se confirmar como inovações aplicáveis, enquanto outras talvez sejam implantadas em futuros mais longínquos ou caírem no ostracismo, podendo voltar em algum momento.





# Tecnologia Emergentes

- Usufruir dos benefícios de **Big Datas**, **computação na nuvem**, **soluções via internet**, por exemplo, já é realidade em vários segmentos, com crescentes necessidades de serviços escaláveis, capazes de se expandir. Para algumas áreas mais específicas, com grande variedade de tipos de dados e informações desestruturadas, em várias áreas do conhecimento, sempre apareceram barreiras nos modelos engessados dos bancos de dados relacionais, mas agora podem se beneficiar da potencialidade dos não relacionais.





# Tecnologia Emergentes

- No segmento dos bancos de dados, vimos que ao longo da história, vários tipos de bancos de dados surgiram, no entanto, ainda hoje o banco de dados relacional é o tipo mais implementado no mundo, mas devido sua limitação em lidar com dados mais específicos e críticos para algumas áreas, os bancos de dados orientados a objetos e os não relacionais, por alguns chamados de NoSQL (não usam SQL), começam a pegar cada vez mais força e passam a ser considerados como possíveis soluções para as lacunas e demandas não atendidas nesta área. Abordaremos então algumas características desses tipos de bancos de dados.



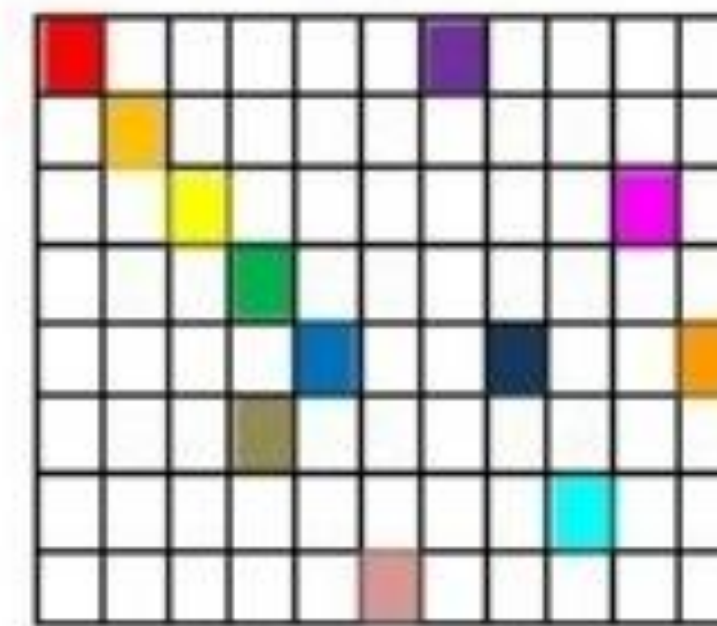


# Banco de dados não relacionais e suas aplicações

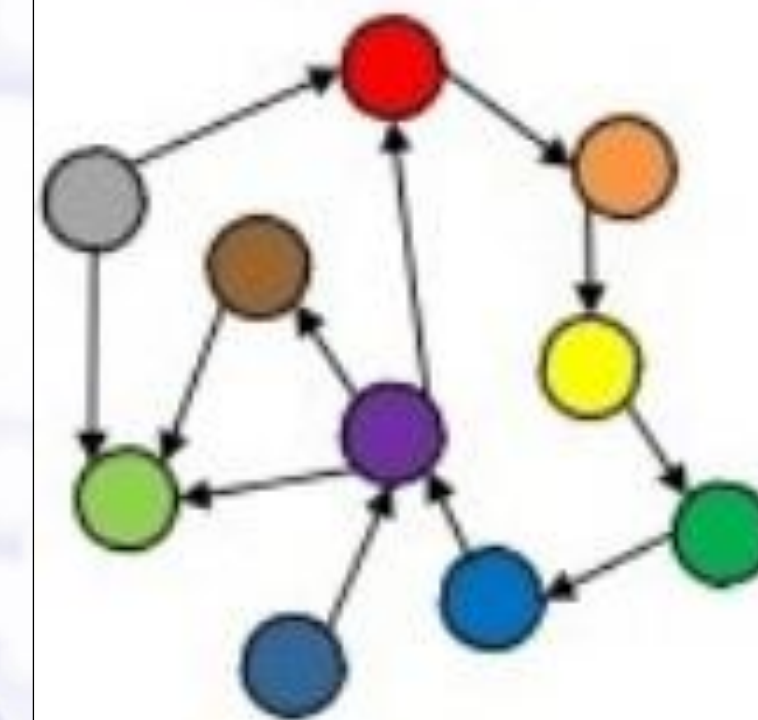


- Bancos de dados não relacionais possuem em sua essência a não estruturação dos dados disponibilizados, como fazem os modelos relacionais, com tabelas organizadas e bem definidas, seus atributos com tipos invioláveis em sua natureza e limitação. São, portanto, tratados como bancos de dados não estruturados ou semiestruturados, chamados até de **NoSQL**.

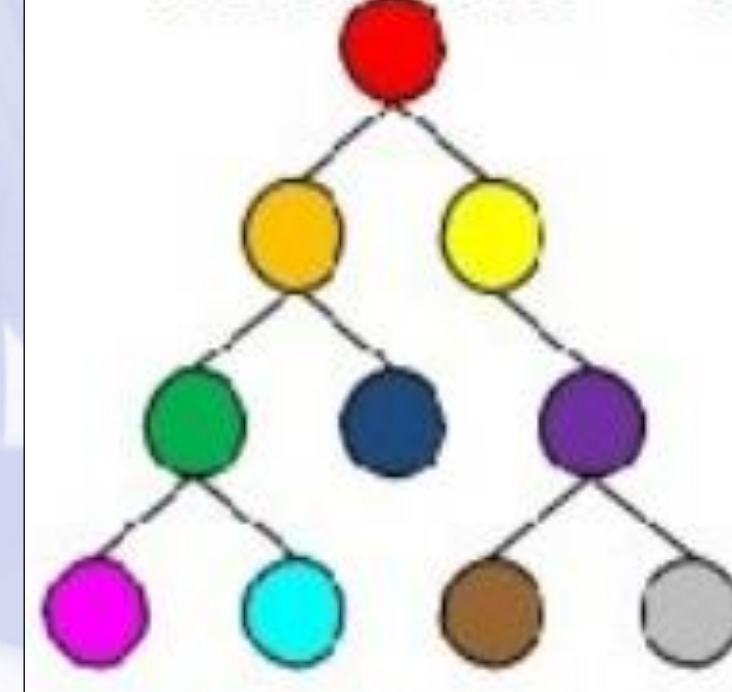
Colunar



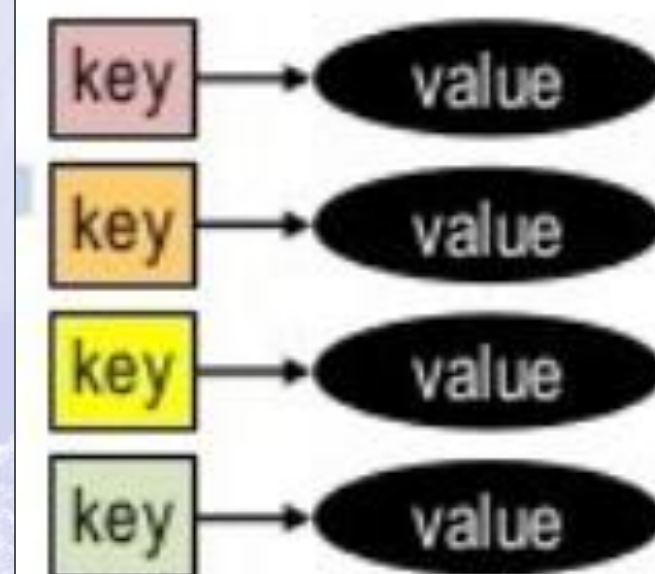
Grafos



Documento x



Chave-Valor

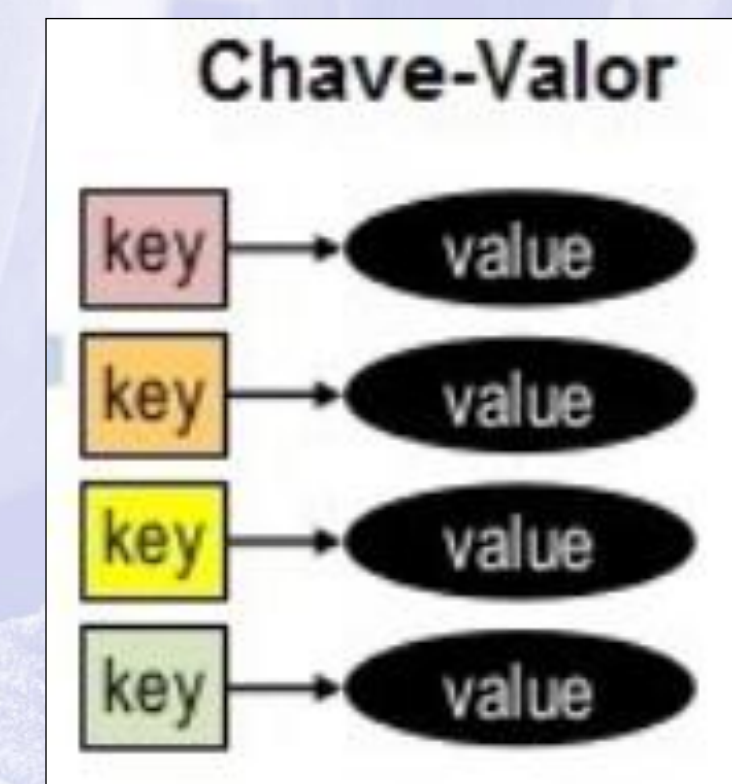
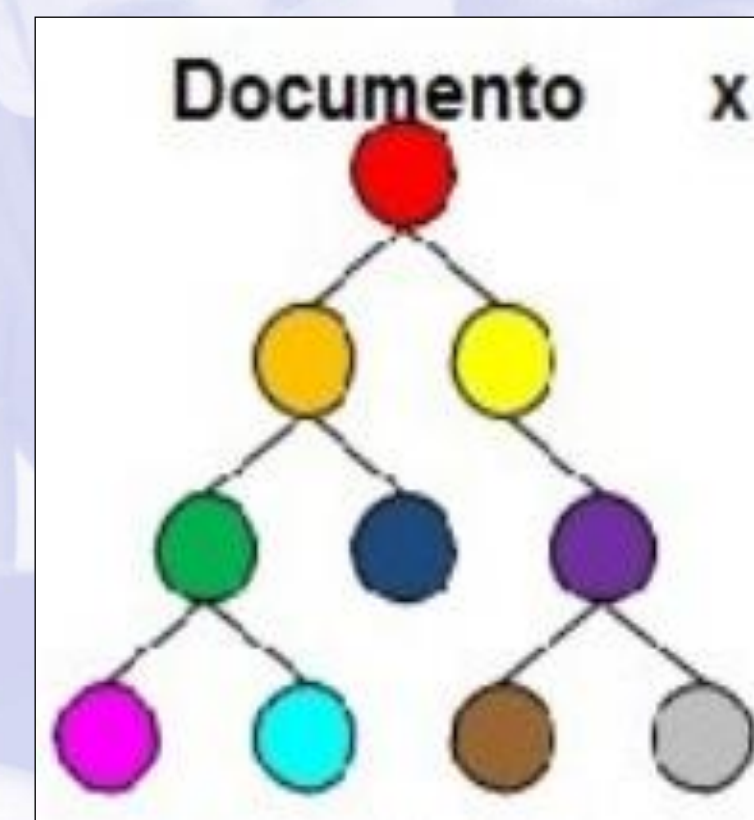
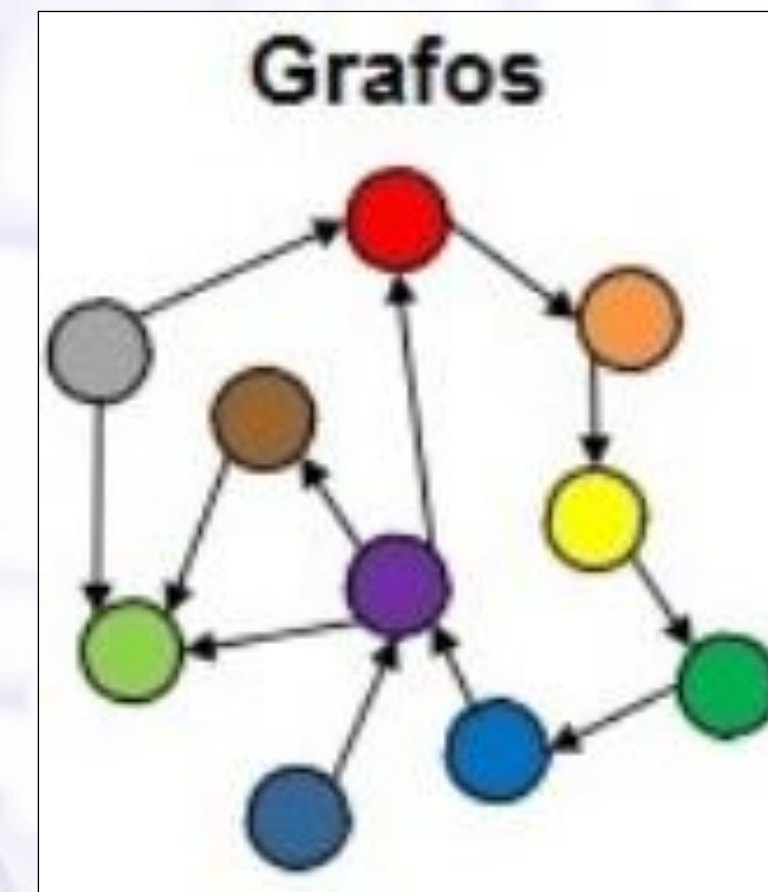
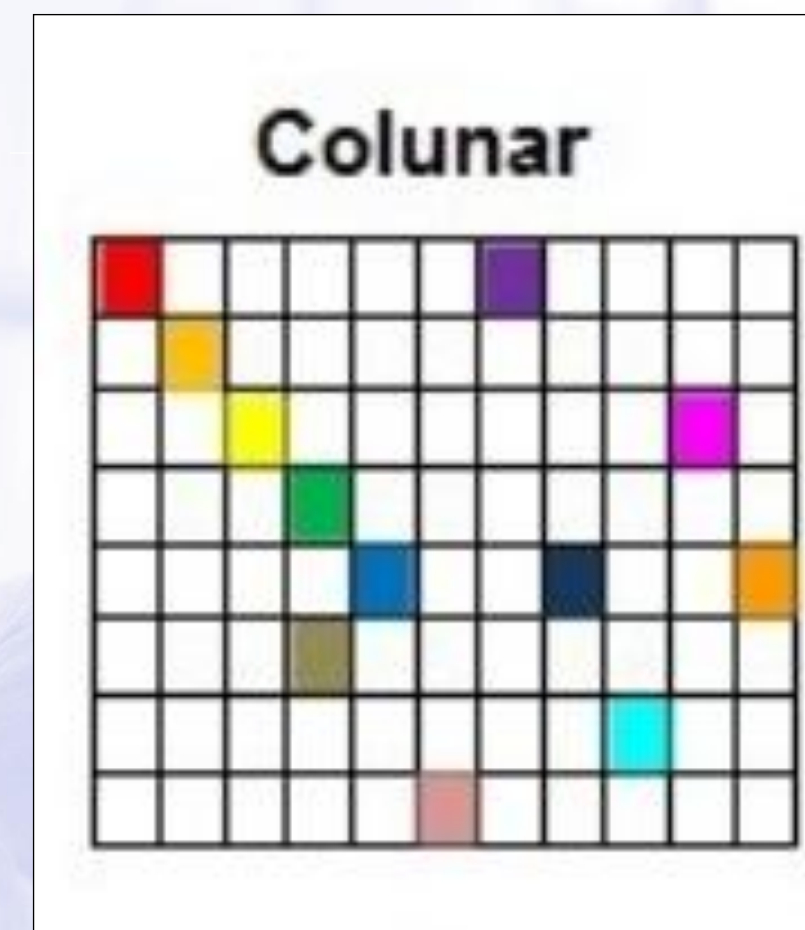




# Banco de dados não relacionais e suas aplicações



- Alguns autores não identificam bancos não relacionais como aqueles que simplesmente não usam SQL, mas há uma corrente tratando do tema de bancos de dados **não relacionais**, como NoSQL, mas isso não é um consenso, até porque existem soluções SQL, mesmo que parciais e limitadas, para manipulação de dados não relacionais. Há anos, autores já anunciavam a capacidade de o SQL Server permitir transações com fontes de não relacionais.





# Banco de dados não relacionais e suas aplicações



- A popularização e os benefícios do paradigma da programação orientada a objetos e suas diversas linguagens, como Java, C++, C#, entre outras, e as limitações impostas pelo modelo relacional, principalmente dentro do universo da web, levaram ao surgimento do modelo chamado “banco de dados orientado a objetos”. Esse tipo pode ser visto como uma extensão do modelo entidade-relacionamento (ELMASRI; NAVATHE, 2011).



# Banco de dados não relacionais e suas aplicações



- Para Silberschatz, Korth e Sudarshan (1999), os principais fornecedores de banco de dados suportam o modelo de dados objeto-relacional, que combina as características do modelo de dados orientado a objetos e o modelo de dados relacional. Bancos de dados orientados a objetos podem armazenar objetos e compartilhá-los para aplicações diferentes

## Opções de Bancos de Dados

Relacional

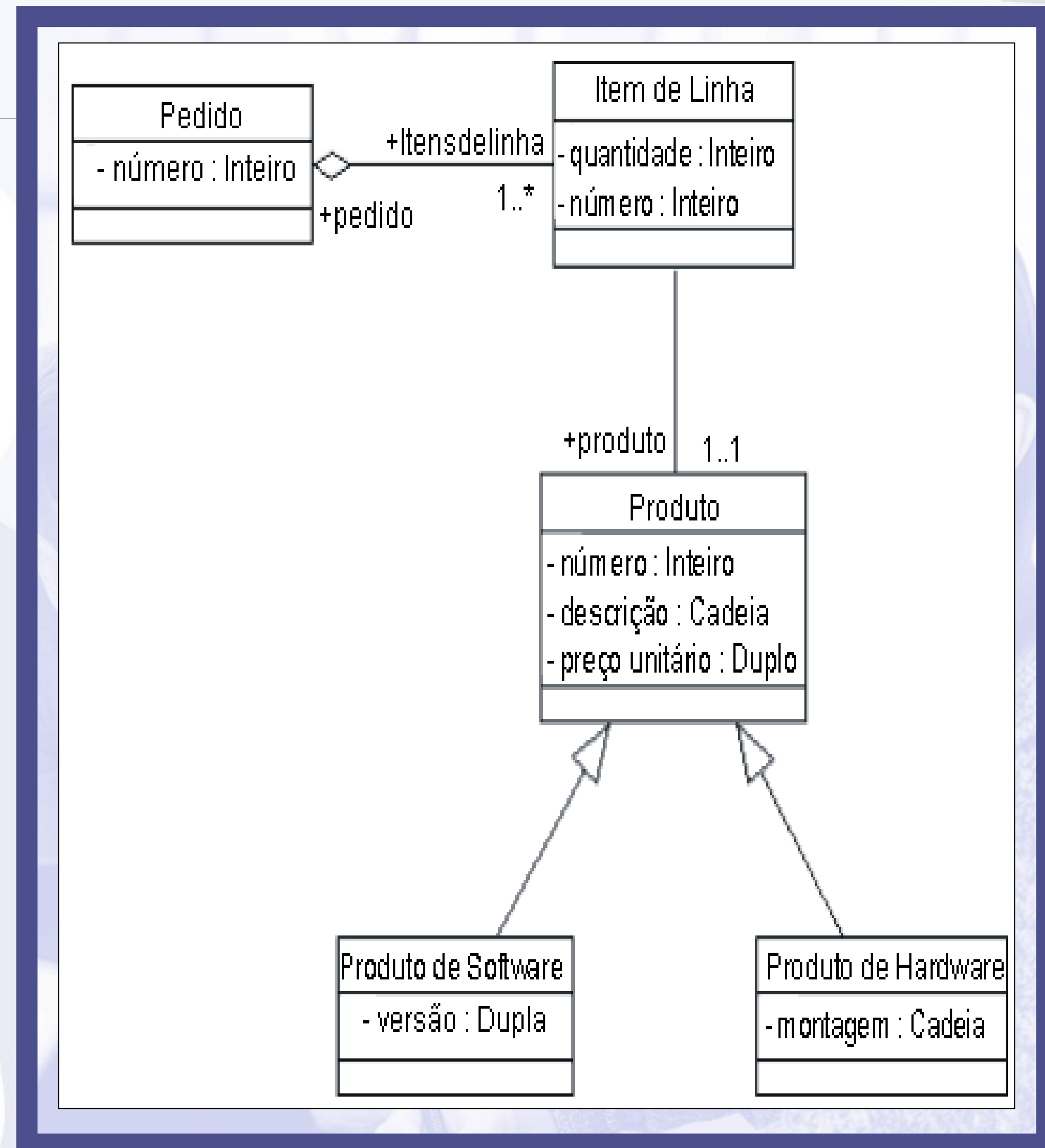
NoSQL



# Banco de dados não relacionais e suas aplicações



- Os principais conceitos da orientação a objetos que compõem a estrutura desse banco de dados são:
  - persistência de objetos,
  - objetos complexos,
  - presença de identificadores de objetos,
  - aplicação de herança,
  - métodos,
  - dados estruturados,
  - coleção,
  - encapsulamento e
  - polimorfismo.





# Segurança de banco de dados



**ser**  
educacional  
gente criando o futuro



# Segurança

- Definição:
  - Segurança em Banco de dados diz respeito à proteção do banco de dados contra ataques intencionais ou não intencionais utilizando-se ou não de meios computacionais
- O subsistema de segurança é responsável por proteger o BD contra o acesso não autorizado.





# Segurança

- Formas de acesso não autorizado:
  - leitura não autorizada
  - modificação não autorizada
  - destruição não autorizada
- O DBA tem plenos poderes para dar e revogar privilégios a usuários.
  - Criação de contas; Concessão de privilégios; Revogação de privilégios; Atribuição do nível de segurança







# Segurança

- O usuário tem um `auth_ID` que o identifica
  - Existe `PUBLIC` que representa todos usuários
- Privilégios são atribuídos/revogados:
  - Usuários
  - Papéis (Roles)
- O criador de um objeto é o dono do objeto e assim tem todos os privilégios sobre o objeto, podendo autorizar a outros usuários alguns(ou todos) destes privilégios.
- A opção **with grant option**, permite ao usuário que recebeu um privilégio repassar para quem quiser.





# Usuários

- Para criar usuários, usamos o comando **CREATE USER** e depois concedemos os “privilégios” de acesso com o comando **GRANT**. Caso o usuário não exista, é possível criá-lo também com esse comando.
- **CREATE USER** nome\_do\_usuario@host\_do\_usuario **IDENTIFIED BY** senha\_do\_usuario;

Exemplo:

```
CREATE USER renatoaf IDENTIFIED BY 'sen0701';
```

Ou

```
CREATE USER renatoaf@localhost IDENTIFIED BY 'sen0701';
```





# Usuários e Papéis

## ▪ Papéis (Roles)

- É um identificador ao qual pode-se atribuir privilégios que não existem a princípio. Então pode-se atribuir a um usuário este papel (conjunto de privilégios) com um único comando GRANT.
- Pode-se inclusive ao criar um papel usar outros papéis já cadastrados.
- Ex. PapelVendedor, PapelVendedorSapatos, PapelVendedorFrutas.

Exemplo:

```
CREATE ROLE nome-papel  
[WITH ADMIN  
{CURRENT_USER |  
CURRENT_ROLE}]
```

Para remover um papel:

```
DROP ROLE nome-papel;
```





# Usuários e Papéis

- **Existem papéis padrões na maioria dos SGBD:**
  - DBA: permite desempenhar o papel de administrados do banco de dados
  - Resource: permite criar seus próprios objetos
  - Connect: permite apenas se conectar ao banco de dados, mas deve receber os privilégios de alguém para acessar objetos.

Exemplo:

```
CREATE ROLE nome-papel  
[WITH ADMIN  
{CURRENT_USER |  
CURRENT_ROLE}]
```

Para remover um papel:

```
DROP ROLE nome-papel;
```

# Permissões de acesso em SQL



- O Comando **grant** é usado para conferir autorização. A forma básica deste comando é:

**grant** <lista de privilégios>  
**on** <nome da relação ou visão>  
**to** <lista de usuários>

Exemplos:

**grant select on agencia to U1, U2, U3**

**grant update on deposito to U1**

**grant references (nome-agencia)  
on agencia to U1**

Os privilégios a serem autorizados, em SQL, com o comando **grant** são:

**Select** - Autorização leitura

**Insert** - Autorização inserção

**Update** - Autorização  
atualização

**Delete** - Autorização  
eliminação

**Index** - Autorização índice

**References** - Autorização  
recursos

**Alter** - Autorização  
alteração

**Drop** - Autorização remoção

# Revogação de acesso em SQL



- Para revogar a autorização, o comando **revoke** é usado. Ele toma a forma quase idêntica àquela do comando grant:

**revoke** <lista de privilégios>  
**on** <nome da relação ou visão>  
**from** <lista de usuários>

```
revoke select on agencia from U1,  
U2, U3
```

```
revoke update on deposito from U1
```

```
revoke references (nome-agencia)  
on agencia from U1
```



# Ataques Típicos em Banco de Dados



## 1. Privilégios excessivos

- Quando usuários (ou apps) recebem privilégios de banco de dados que excedem os requisitos de sua função, esses privilégios podem ser usados para ganhar acesso a informações confidenciais.

## 2. Injeções SQL

- Um método popular que hackers usam para tomar controle de bancos de dados são as injeções SQL, mencionadas no tópico anterior. Aplicações são atingidas por injeções e o administrador de banco de dados precisa limpar a bagunça causada pelo código mal-intencionado inserido em *strings*.





# Ataques Típicos em Banco de Dados



## 3. Vulnerabilidades do sistema operacional

- As vulnerabilidades em sistemas operacionais podem levar a acessos não autorizados. Imagine que você deixou de instalar uma atualização e um hacker lançou um ataque baseado nessa nova brecha contra os seus sistemas.

## 4. Autenticação fraca

- Modelos de autenticação com baixa segurança permitem que hackers se valham de estratégias como ataques por força bruta e engenharia social para infiltrar bancos de dados.





# Ataques Típicos em Banco de Dados



## 5. Malware

- Uma ameaça perene como o malware é usada para roubar dados sensíveis por meio da infecção dos dispositivos de usuários legítimos e pode chegar ao seu banco de dados caso sua equipe não esteja atenta. Certos websites e programas devem ser evitados para prevenir a infecção.
- Dada a sua prevalência em publicidade na web e, como cavalo de tróia, em alguns programas gratuitos, será preciso instalar bloqueadores de anúncios e evitar o download de aplicativos não autorizados.





# Ataques Típicos em Banco de Dados



1. Implantar políticas de segurança em TI;
2. Evitar concessão de privilégios de acesso excessivos para posteriormente ficarem desatualizados e cair no esquecimento dos administradores;
3. Prevenir abusos de privilégios e uso inconsequente por maus profissionais;
4. Realizar testes manuais, por intermédio de ferramentas específicas diretamente nas aplicações do usuário para atestar a vulnerabilidade, como SQLMap, jSQL Injection, entre outras;
5. Manter o banco de dados em um servidor exclusivo, com acesso físico e lógico protegido de intrusão;



# Ataques Típicos em Banco de Dados



- 6.Prevenir o ambiente contra *malwares*;
- 7.Realizar auditorias com frequência;
- 8.Implantar política de backups eficientes;
- 9.Evitar exposição desnecessária de mídias de *storage* de backups.

Essas e outras atitudes combinadas poderão elevar o nível de segurança e nenhuma delas deverá ser subestimada nem valorizada a ponto de se classificar como absoluta e definitiva. Quando o assunto é segurança de dados, todo cuidado é pouco.





# PRÓXIMOS PASSOS



Revisão



# OBRIGADO



**Adilson da Silva**

**Mestre**

**[adilson.silva@sereducacional.com](mailto:adilson.silva@sereducacional.com)**