

# AULA CINCO

## STRUCTURED QUERY LANGUAGE (SQL)



# CONTEÚDO



**03** INTRODUÇÃO

---

**09** COMANDO SELECT

---

**15** CLÁUSULA WHERE

---

**17** OPERADORES RELACIONAIS, LÓGICOS E “CURINGAS”

---

**25** COMANDO ORDER BY

---

**27** COMANDO INNER JOIN

---

**32** CRIANDO CONSULTAS COMPLEXAS

---

# MODELO ENTIDADE RELACIONAMENTO (MER)

O Modelo Entidade Relacionamento (também chamado MER) é um modelo conceitual utilizado na Engenharia de Software para descrever os objetos (**entidades**) envolvidos em um domínio de negócios, com suas características (**atributos**) e como elas se relacionam entre si (**relacionamentos**).

Em geral, este modelo representa de forma abstrata a estrutura que possuirá o banco de dados da aplicação. O banco de dados poderá conter várias outras entidades, tais como chaves e tabelas intermediárias, que podem só fazer sentido no contexto de bases de dados relacionais.



# ENTIDADES

Os objetos ou partes envolvidas um domínio, também chamados de entidades, podem ser classificados como físicos ou lógicos, de acordo sua existência no mundo real.

**Entidades físicas:** são aquelas realmente tangíveis, existentes e visíveis no mundo real, como um cliente (uma pessoa, uma empresa) ou um produto (um carro, um computador, uma roupa).

**Entidades lógicas:** são aquelas que existem geralmente em decorrência da interação entre ou com entidades físicas, que fazem sentido dentro de um certo domínio de negócios, mas que no mundo real não são objetos físicos (que ocupam lugar no espaço). São exemplos disso uma venda ou uma classificação de um objeto (modelo, espécie, função de um usuário do sistema).

# ENTIDADES

Podemos classificar as entidades segundo o motivo de sua existência:

**Entidades fortes:** são aquelas cuja existência **independe de outras entidades**, ou seja, por si só elas já possuem total sentido de existir. Em um sistema de vendas, a entidade **produto**, por exemplo, independe de quaisquer outras para existir.

**Entidades fracas:** ao contrário das entidades fortes, as fracas são aquelas que **dependem de outras entidades** para existirem, pois individualmente elas não fazem sentido. Mantendo o mesmo exemplo, a entidade **venda depende da entidade produto**, pois uma venda sem itens não tem sentido.

**Entidades associativas:** entidades associativas são um conceito MER e da modelagem de banco de dados, que representam relacionamentos complexos entre entidades, geralmente conceitos de relacionamentos N:N. São usadas para armazenar informações adicionais sobre a relação entre duas ou mais entidades. São implementadas usando tabelas associativas, que mapeiam duas ou mais tabelas referenciando as **chaves primárias** de cada tabela. Exemplo: relacionamento entre uma tabela **alunos** e uma tabela **professores**, na prática, como se trata de um relacionamento N:N, resolve-se utilizando uma tabela intermediária chamada **disciplina**.

# RELACIONAMENTOS

De acordo com a quantidade de objetos envolvidos em cada lado do relacionamento, podemos classifica-los de três formas:

**Relacionamento 1:1 (um parra um):** cada uma das duas entidades envolvidas referenciam obrigatoriamente apenas uma unidade da outra. Por exemplo, em um banco de dados de currículos, cada usuário cadastrado pode possuir apenas um currículo na base, ao mesmo tempo em que cada currículo só pertence a um único usuário cadastrado.

**Relacionamento 1:N (um para muitos):** uma das entidades envolvidas pode referenciar várias unidades da outra, porém, do outro lado cada uma das várias unidades referenciadas só pode estar ligada uma unidade da outra entidade. Por exemplo, em um sistema de plano de saúde, um usuário pode ter vários dependentes, mas cada dependente só pode estar ligado a um usuário principal.

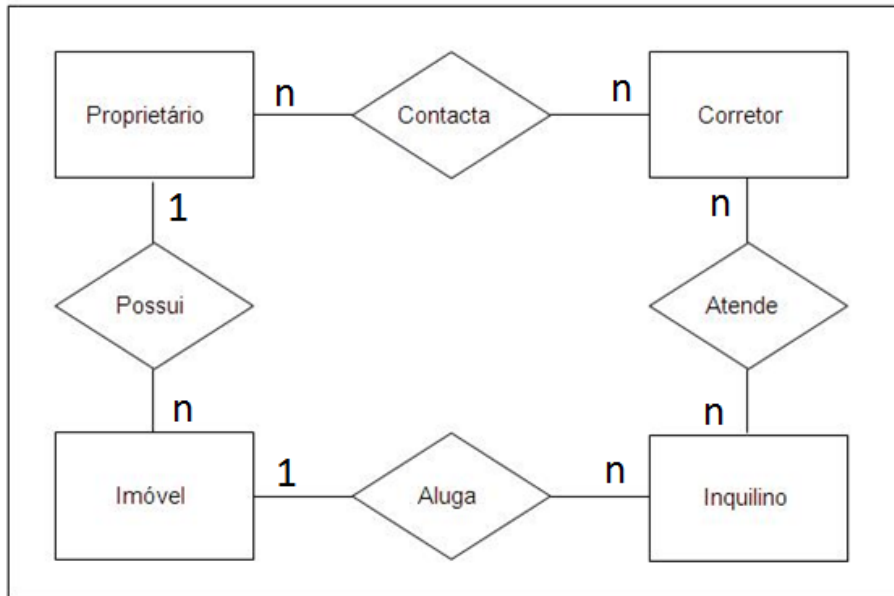
**Relacionamento N:N (muitos para muitos):** neste tipo de relacionamento cada entidade, de ambos os lados, podem referenciar múltiplas unidades da outra. Por exemplo, em um sistema de biblioteca, um título pode ser escrito por vários autores, ao mesmo tempo em que um autor pode escrever vários títulos.

# ATRIBUTOS

Atributos são as características que descrevem cada entidade. Por exemplo, um **cliente** possui **nome**, **endereço** e **telefone**. Durante a análise de requisitos, são identificados os atributos relevantes de cada entidade naquele contexto, de forma a manter o modelo o mais simples possível e consequentemente armazenar apenas as informações que serão úteis futuramente.

Uma pessoa possui atributos pessoais como cor dos olhos, altura e peso, mas para um sistema que funcionará em um supermercado, por exemplo, estas informações dificilmente serão relevantes.

# DIAGRAMA ENTIDADE RELACIONAMENTO (DER)



O Diagrama Entidade Relacionamento (DER) é a representação gráfica do MER.

No exemplo representado pelo diagrama ao lado temos as seguintes entidades e relacionamentos:

**Proprietário contata Corretor** (um proprietário pode contatar vários corretores e um corretor pode ser contatado por vários proprietários);


**Corretor atende Inquilino** (um corretor pode atender vários inquilinos e um inquilino pode ser atendido por vários corretores);

**Inquilino aluga Imóvel** (um inquilino aluga um imóvel e um imóvel pode ser alugado por vários inquilinos);


**Proprietário possui Imóvel** (um proprietário possui vários imóveis e um imóvel pertence a apenas um proprietário).



# DIAGRAMA ENTIDADE RELACIONAMENTO (DER)

 **Lucidchart** Produto ▾ Casos de uso ▾ Recursos ▾ Preços Falar com Vendas Cadastre-se gratuitamente

Novidades na Lucid: Conheça nossos mais recentes recursos, projetados para ajudar equipes ágeis a colaborar melhor, alinhar objetivos e esclarecer dúvidas. [Ler mais →](#)



### Programa para fazer diagrama entidade relacionamento online

O Lucidchart é uma solução de diagramação inteligente que tem as ferramentas para ajudar você a criar um diagrama ER online em poucos minutos. Importe seus próprios dados ou comece do zero. Veja e construa o futuro em qualquer lugar com o Lucidchart.

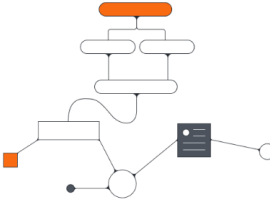
Faça um diagrama ER gratuito

ou continuar com


Fazer login

Fazer login

Fazer login

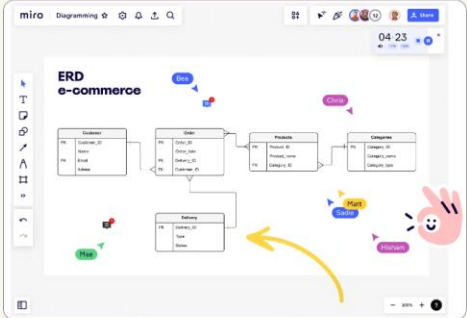


[www.lucidchart.com](http://www.lucidchart.com)

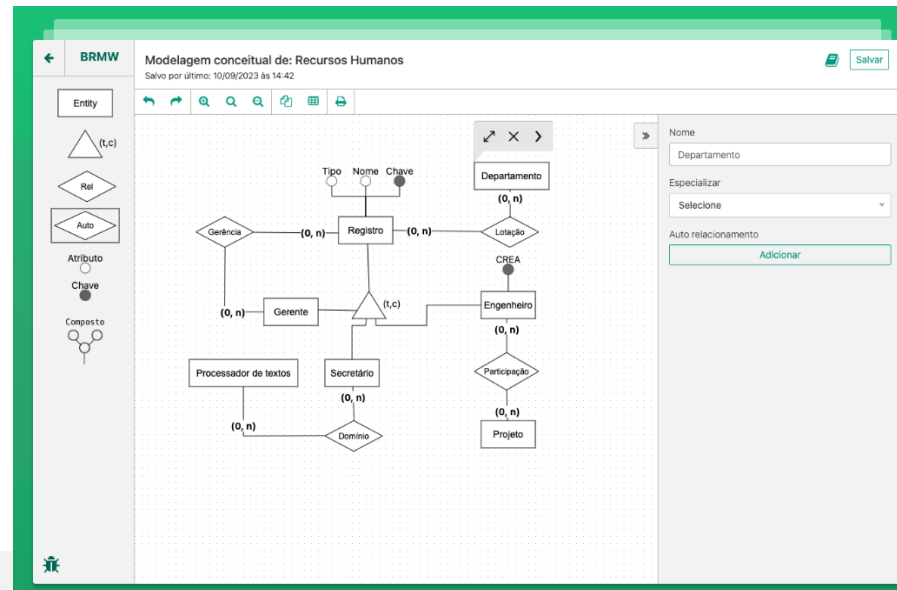
 **miro** O que é a Miro? ▾ Soluções ▾ Recursos ▾ Enterprise ▾ Clientes ▾ Preços Fale com Vendas Login

## Mapeie sistemas com diagramas de entidade e relacionamento


Vincule e compreenda os diferentes elementos de seu banco de dados e mostre como eles interagem com uma ferramenta para fazer diagramas de entidade e relacionamento (ER) online. Reduza a complexidade e visualize como as entidades do sistema se conectam e se sobrepõem, trazendo agilidade às equipes.



[miro.com](https://miro.com)



[www.brmodeloweb.com](http://www.brmodeloweb.com)

 **CYNGULA**

Copyright © 2015. Cyngula Treinamentos MEI. Todos os direitos reservados. | [www.cyngula.com.br](http://www.cyngula.com.br) | [prof.jobel@gmail.com](mailto:prof.jobel@gmail.com)

9

# SQL – STRUCTURED QUERY LANGUAGE

## Structured Query Language

*Linguagem Estruturada de Consultas*

SQL é uma **linguagem padrão** para **acesso e manipulação** de **bancos de dados**.

# CONSULTAS



**CONSULTAS TRANSFORMAM DADOS EM INFORMAÇÕES!**

# O QUE O SQL PODE FAZER?

1. Pode **executar consultas** contra os bancos de dados;
2. Pode **obter dados** do banco de dados;
3. Pode **inserir registros** em um banco de dados;
4. Pode **atualizar registros** em um banco de dados;
5. Pode **excluir registros** em um banco de dados;
6. Pode **criar novos bancos de dados**;
7. Pode **criar novas tabelas** em um banco de dados;
8. Pode **criar procedimentos automáticos** em um banco de dados;
9. Pode **criar visualizações** em um banco de dados;
10. Pode **definir permissões em tabelas, procedimentos e visualizações**.

# SINTAXE

O SQL é composto por comandos e funções:

## COMANDOS:

Realizam operações nos bancos de dados, por exemplo, filtrar, classificar, excluir, etc.

## FUNÇÕES:

Modificam e retornam dados ao usuário, por exemplo, converter em maiúsculas, separar *strings* de texto, etc.

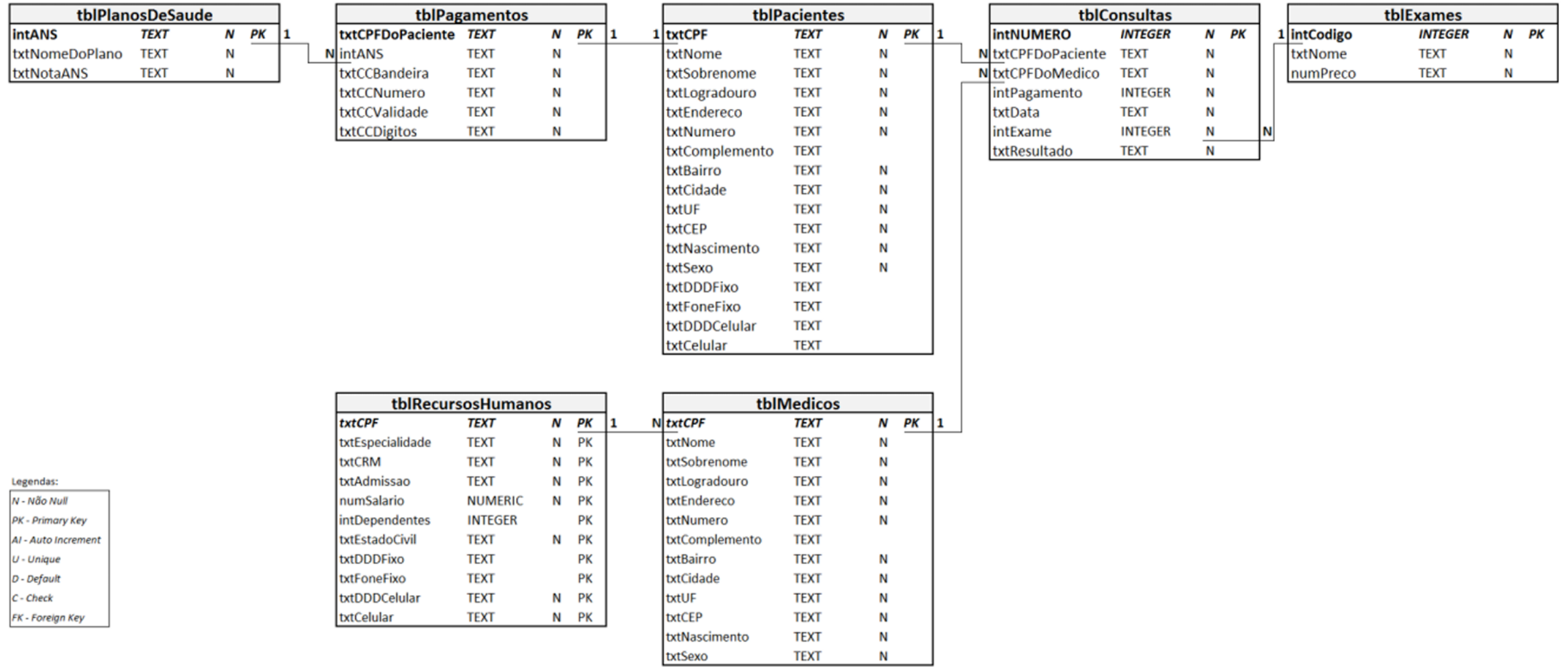




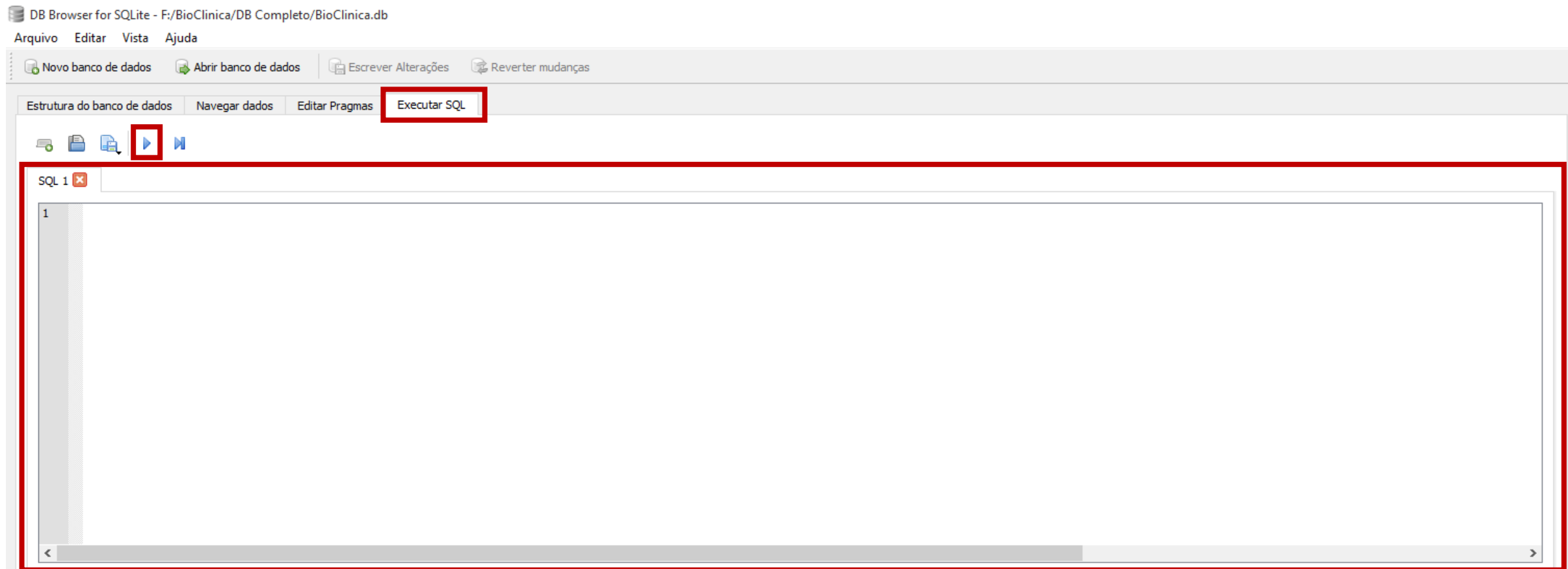
# PARTE I

# CONSULTAS

# BIOCLINICA.db



# BIOCLINICA.db



# COMANDO SELECT

O comando **SELECT** é usado para **selecionar dados** em um banco de dados. O resultado do comando **SELECT** é armazenado em uma tabela resultado chamada de **RESULT-SET**.

**SELECT**  $\left[ \begin{array}{c} * \\ \text{ou} \\ \text{campo 1, campo 2, ..., campo n} \end{array} \right] \text{ **FROM** tabela}$

# COMANDO SELECT

**SELECT \* FROM** tblPacientes

**SELECT \* FROM** tblMedicos

**SELECT \* FROM** tblRecursosHumanos



# COMANDO SELECT

```
SELECT txtCPF, txtNome FROM tblPacientes
```

```
SELECT txtCPF, txtNome, txtSobrenome FROM tblPacientes
```

```
SELECT txtCPF, txtNome, txtSobrenome FROM tblMedicos
```

# COMANDO SELECT

```
1  SELECT  
2  txtNome, txtSobrenome, txtCidade, txtUF  
3  FROM  
4  tblPacientes
```

# COMANDO SELECT DISTINCT

Em uma tabela, um campo pode conter diversos valores duplicados, o comando **SELECT DISTINCT** é utilizado para retornar apenas valores distintos (**únicos**) do campo.

**SELECT DISTINCT**

*campo*

**FROM** *tabela*

# COMANDO SELECT DISTINCT

**SELECT DISTINCT** txtNome **FROM** tblPacientes

**SELECT DISTINCT** txtCidade **FROM** tblPacientes

**SELECT DISTINCT**  
txtCPFDDoMedico  
**FROM**  
tblConsultas

# CLÁUSULA WHERE

**CLÁUSULA** é um sinônimo de **CONDIÇÃO** ou **CRITÉRIO**. A cláusula **WHERE** é usada para **filtrar registros**. A cláusula **WHERE** permite extrair somente os registros que **satisfaçam** um **critério** especificado.

**SELECT** campo 1, campo 2, ... , campo n

**FROM** tabela

**WHERE** <condição>



# CLÁUSULA WHERE

```
SELECT txtNome, txtDDDCelular, txtCelular, txtCidade  
FROM tblPacientes  
WHERE txtCidade = 'MANAUS'
```

```
SELECT txtCPF, intDependentes  
FROM tblMedicos  
WHERE intDependentes = 0
```

**OBSERVAÇÃO:** Quando o(s) campo(s) na condição forem do tipo **TEXT** a condição deve vir entre aspas simples (').

# OPERADORES RELACIONAIS DA CLÁUSULA WHERE

Operador	Descrição
=	Igual à
<>	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual à
<=	Menor ou igual à
<b>BETWEEN</b>	Valores dentro de uma faixa de dados
<b>LIKE</b>	Parecido com
<b>IN</b>	Especifica múltiplos valores para um campo

# OPERADORES LÓGICOS DA CLÁUSULA WHERE

Os operadores lógicos **AND (E)**, **OR (OU)** e **NOT (NÃO)** são usados para filtrar registros com **base em mais de uma condição**.

Operador	Descrição
<b>AND</b>	O operador <b>AND</b> exibe um registro se tanto a primeira condição <b>E</b> a segunda condição forem <b>verdadeiras</b> .
<b>OR</b>	O operador <b>OR</b> exibe um registro se a primeira condição <b>OU</b> a segunda condição forem verdadeiras.
<b>NOT</b>	Os campos com valores listados após o operador <b>NOT</b> não será mostrados na consulta.

# CARACTERES CURINGA (WILDCARDS)

CARACTER	DESCRIÇÃO
%	Substitui muitos caracteres.
_	Substitui um único caractere.

# CLÁUSULA WHERE COM OPERADORES RELACIONAIS

```
SELECT txtNome, txtSobrenome, txtCidade  
FROM tblPacientes  
WHERE txtCidade <> 'SANTOS'
```

```
SELECT txtCPF, intDependentes  
FROM tblRecursosHumanos  
WHERE intDependentes > 0
```

```
SELECT txtCPF, numSalario  
FROM tblRecursosHumanos  
WHERE numSalario < 26000
```



# CLÁUSULA WHERE COM OPERADORES RELACIONAIS

```
SELECT txtCPF, numSalario  
FROM tblRecursosHumanos  
WHERE numSalario >= 25612
```

```
SELECT txtNomeDoPlano, intNotaANS  
FROM tblPlanosDeSaude  
WHERE intNotaANS <= 60
```

# CLÁUSULA WHERE COM OPERADORES RELACIONAIS E LÓGICOS

```
SELECT txtCPF, numSalario  
FROM tblRecursosHumanos  
WHERE numSalario BETWEEN 28000 AND 32000
```

```
SELECT txtNome, txtSobrenome  
FROM tblPacientes  
WHERE txtNome LIKE 'JO%'
```

```
SELECT txtNome, txtSobrenome, txtCidade  
FROM tblPacientes  
WHERE txtCidade IN ('CUIABA', 'CAMPINAS')
```

# CLÁUSULA WHERE COM OPERADORES RELACIONAIS E LÓGICOS

```
SELECT txtNome, txtSobrenome, txtCidade  
FROM tblPacientes  
WHERE txtCidade BETWEEN 'C' AND 'M'
```

```
SELECT txtNome, txtSobrenome, txtCidade  
FROM tblPacientes  
WHERE txtCidade NOT BETWEEN 'C' AND 'M'
```

```
SELECT txtNome, txtSobrenome, txtNascimento  
FROM tblPacientes  
WHERE txtNascimento NOT LIKE '%1980'
```

# CLÁUSULA WHERE COM CARACTERES CURINGA

```
SELECT txtNome, txtSobrenome  
FROM tblPacientes  
WHERE txtNome LIKE 'J%'
```

```
SELECT txtNome, txtNascimento  
FROM tblPacientes  
WHERE txtNascimento LIKE '%1980'
```

```
SELECT txtNome, txtDDDDCelular, txtCelular  
FROM tblPacientes  
WHERE txtDDDDCelular LIKE '2_'
```

# COMANDO ORDER BY

O comando **ORDER BY** é usado para classificar um ou mais campos em ordem crescente ou decrescente. A palavra-chave **ORDER BY** classifica os registros em ordem crescente por padrão. Para classificar os registros em ordem decrescente, você pode usar a palavra-chave **DESC**.

**SELECT** *campo 1, campo 2, ..., campo n*

**FROM** *tabela*

**ORDER BY** *campo 1 ASC | DESC, campo 2 ASC | DESC*

# COMANDO ORDER BY

```
SELECT txtNome, txtSobrenome  
FROM tblPacientes  
ORDER BY txtNome
```

```
SELECT txtNome, txtSobrenome  
FROM tblPacientes  
ORDER BY txtNome DESC
```

```
SELECT txtNome, txtDDDDCelular, txtCelular  
FROM tblPacientes  
ORDER BY txtDDDDCelular, txtCelular
```

# COMANDO INNER JOIN

Uma cláusula **SQL JOIN** é usado para combinar **campos** de **duas ou mais tabelas, com base em um campo comum entre eles**. O tipo mais comum de se juntar é: **INNER JOIN** (junção simples). Um **INNER JOIN** retorna todas as linhas de várias tabelas onde a condição de junção for atendida.

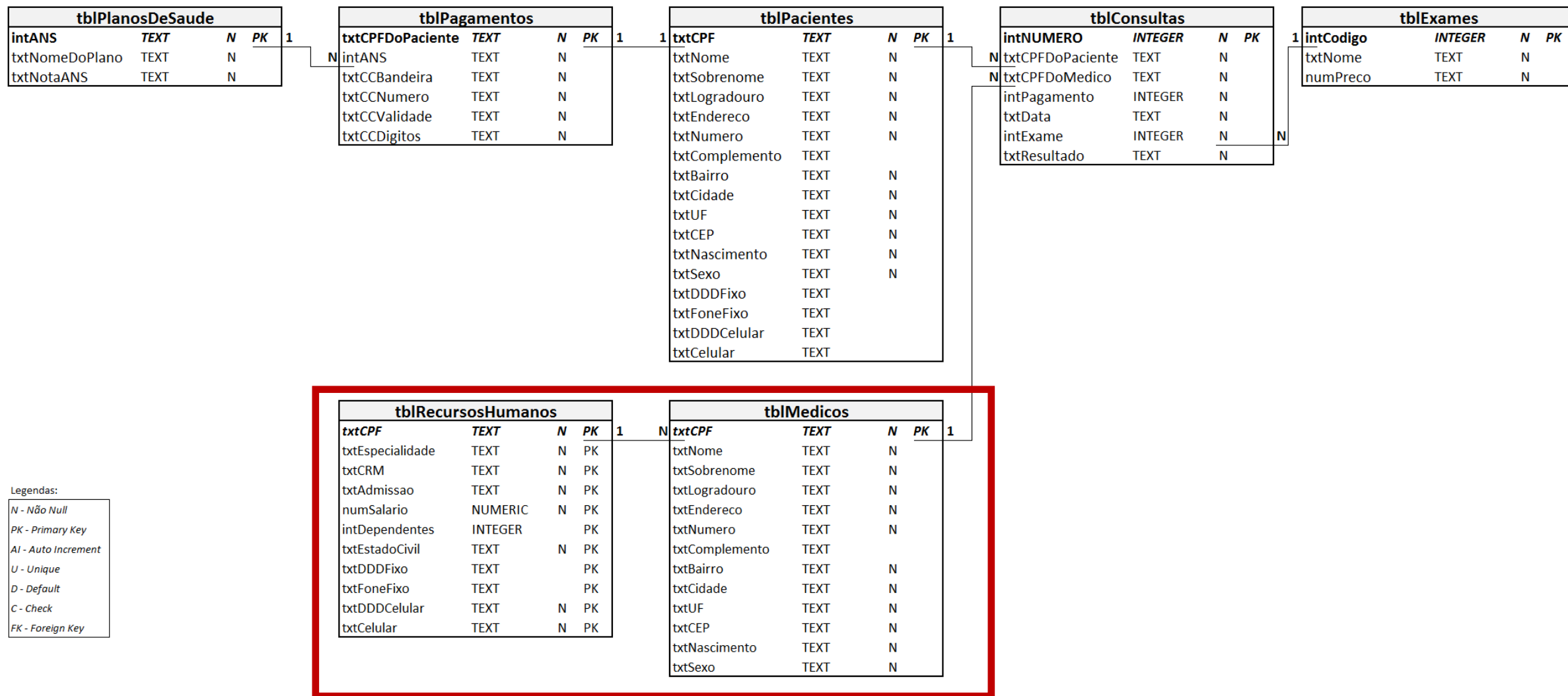
**SELECT** tabela.campo 1, tabela.campo 2, ..., tabela.campo n

**FROM** tabela 1

**INNER JOIN** tabela 2

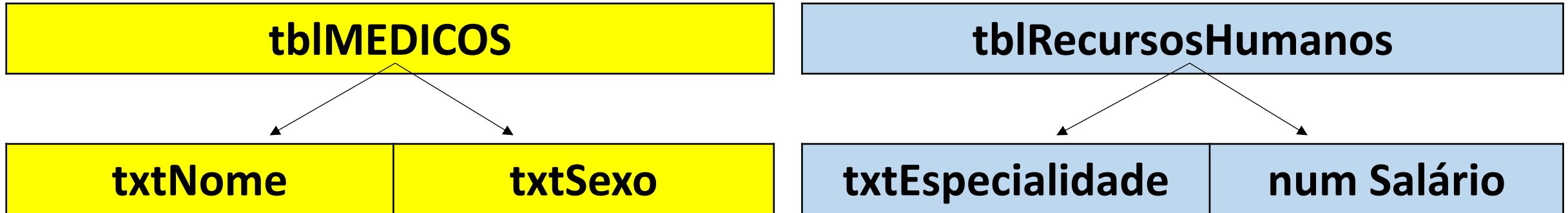
**ON** <condição>

# COMANDO INNER JOIN





# COMANDO INNER JOIN



**SELECT**

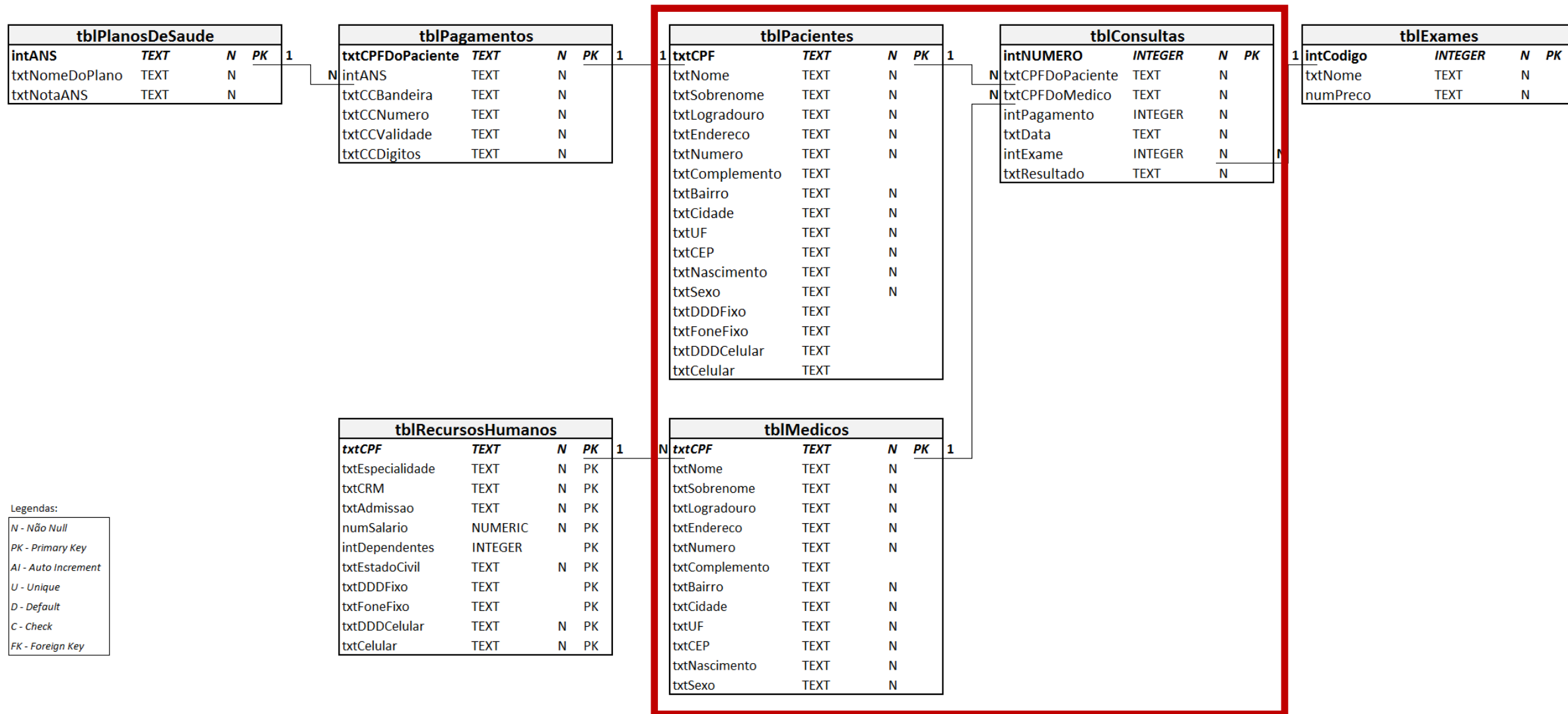
`tblMedicos.txtNome, tblMedicos.txtSexo,`  
`tblRecursosHumanos.txtEspecialidade, tblRecursosHumanos.numSalario`

**FROM** `tblMedicos`

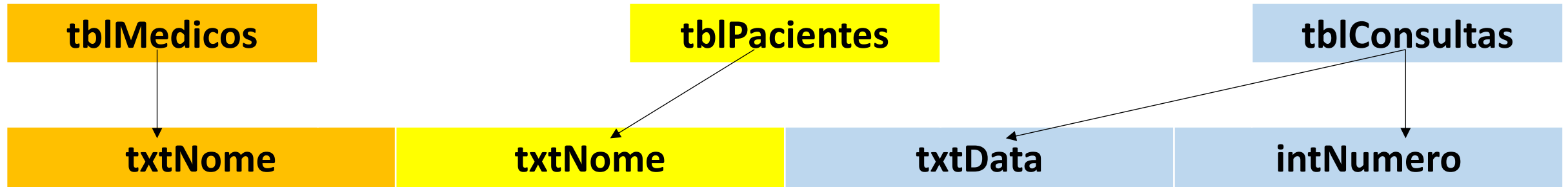
**INNER JOIN** `tblRecursosHumanos`

**ON** `tblMedicos.txtCPF = tblRecursosHumanos.txtCPF`

# COMANDO INNER JOIN



# COMANDO INNER JOIN



**SELECT**

**tblMedicos.txtNome, tblPacientes.txtNome, tblConsultas.txtData, tblConsultas.intNumero**

**FROM** **tblConsultas**

**INNER JOIN** **tblPacientes**

**ON** **tblPacientes.txtCPF = tblConsultas.txtCPFDoPaciente**

**INNER JOIN** **tblMedicos**

**ON** **tblMedicos.txtCPF = tblConsultas.txtCPFdoMedico**

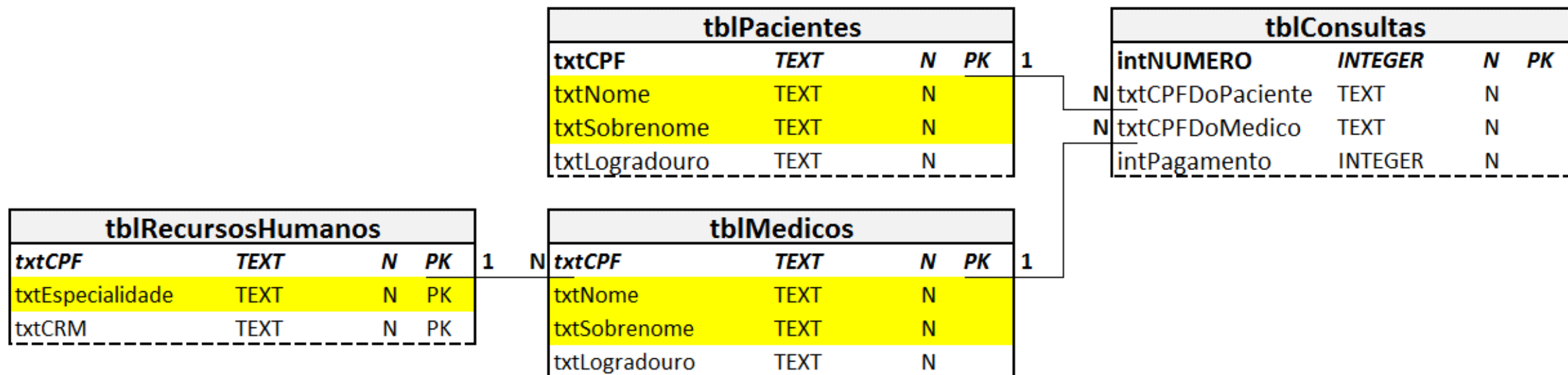
# CRIANDO CONSULTAS COMPLEXAS

Para criar consultas complexas, é **mais fácil dividir a execução da mesma em consultas mais simples**. Por exemplo, usando o banco de dados BioClinica.db, vamos efetuar a seguinte consulta:

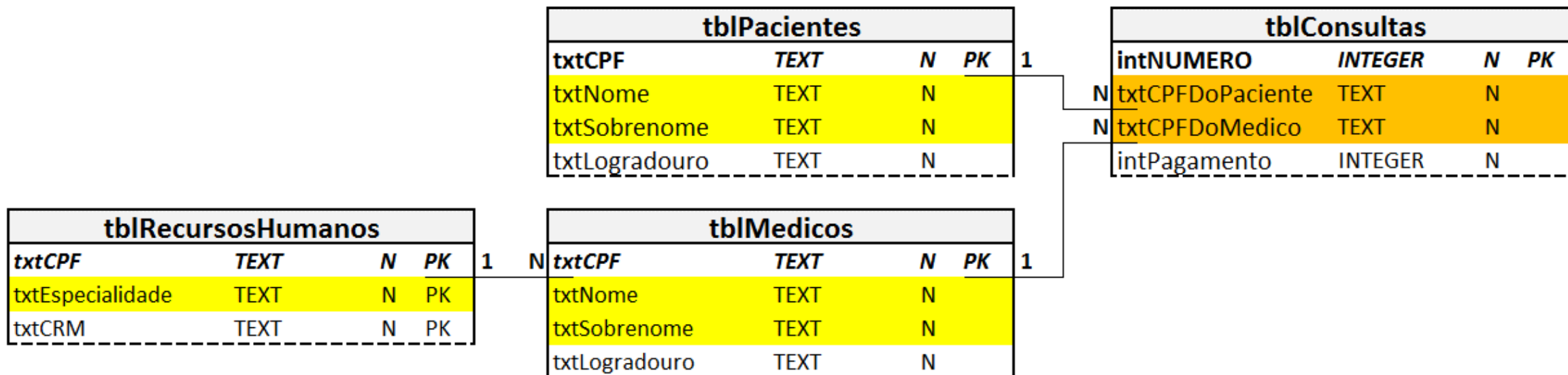
**Listar todos os pacientes que foram atendidos por médicos cardiologistas.**

Para isso teremos que usar quatro (4) tabelas.

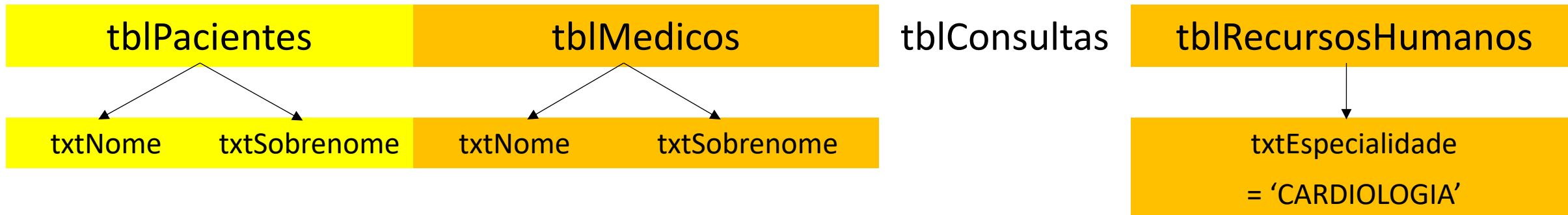
# CRIANDO CONSULTAS COMPLEXAS



# CRIANDO CONSULTAS COMPLEXAS



# CRIANDO CONSULTAS COMPLEXAS



# PRIMEIRA PARTE: SELECIONANDO PACIENTES E CONSULTAS

**SELECT**

**tblPacientes.txtNome, tblPacientes.txtSobrenome**

**FROM** **tblConsultas**

**INNER JOIN** **tblPacientes**

**ON** **tblConsultas.txtCPFDoPaciente = tblPacientes.txtCPF**



## SEGUNDA PARTE: ACRESCENTANDO MÉDICOS

```
SELECT  
tblPacientes.txtNome, tblPacientes.txtSobrenome,  
tblMedicos.txtNome, tblMedicos.txtSobrenome  
FROM tblConsultas  
INNER JOIN tblPacientes  
ON tblConsultas.txtCPFDoPaciente = tblPacientes.txtCPF  
INNER JOIN tblMedicos  
ON tblConsultas.txtCPFdoMedico = tblMedicos.txtCPF
```

# TERCEIRA PARTE: ACRESCENTANDO ESPECIALIDADE

```
SELECT
tblPacientes.txtNome, tblPacientes.txtSobrenome,
tblMedicos.txtNome, tblMedicos.txtSobrenome,
tblRecursosHumanos.txtEspecialidade
FROM tblConsultas
INNER JOIN tblPacientes
ON tblConsultas.txtCPFDoPaciente = tblPacientes.txtCPF
INNER JOIN tblMedicos
ON tblConsultas.txtCPFdoMedico = tblMedicos.txtCPF
INNER JOIN tblRecursosHumanos
ON tblRecursosHumanos.txtCPF = tblMedicos.txtCPF
WHERE
tblRecursosHumanos.txtEspecialidade = 'CARDIOLOGIA'
```

# TERCEIRA PARTE: ACRESCENTANDO ESPECIALIDADE

```
SELECT
tblPacientes.txtNome, tblPacientes.txtSobrenome, tblMedicos.txtNome,
tblMedicos.txtSobrenome, tblRecursosHumanos.txtEspecialidade
FROM tblConsultas
INNER JOIN tblPacientes
ON tblConsultas.txtCPFDOPaciente = tblPacientes.txtCPF
INNER JOIN tblMedicos
ON tblConsultas.txtCPFdoMedico = tblMedicos.txtCPF
INNER JOIN tblRecursosHumanos
ON tblRecursosHumanos.txtCPF = tblMedicos.txtCPF
WHERE
tblRecursosHumanos.txtEspecialidade = 'CARDIOLOGIA'
ORDER BY tblPacientes.txtNome
```