

# Materiais Complementares do Professor

## Orientação

### Materiais Complementares do Professor



Importante

Este é um espaço que utilizarei para disponibilizar materiais ou conteúdos complementares. Fique Atento!

# **APOSTILA BANCO DE DADOS**

---

**MINISTÉRIO DA EDUCAÇÃO  
UNIVERSIDADE ABERTA DO BRASIL**



GRADUAÇÃO EM  
**TECNOLOGIA EDUCACIONAL**  
LICENCIATURA

# **Introdução a Banco de Dados**

**Profa. Dra. Joyce Aline de Oliveira Marins**

**Profa. Dra. Gracyeli Santos Souza Guarienti**

**2019**

**Secretaria de Tecnologia Educacional  
Universidade Federal de Mato Grosso**

**MINISTÉRIO DA EDUCAÇÃO  
UNIVERSIDADE ABERTA DO BRASIL**



GRADUAÇÃO EM  
**TECNOLOGIA EDUCACIONAL**  
LICENCIATURA

# **Introdução a Banco de Dados**

**Profa. Dra. Joyce Aline de Oliveira Marins**

**Profa. Dra. Gracyeli Santos Souza Guarienti**

**2019**

**Secretaria de Tecnologia Educacional  
Universidade Federal de Mato Grosso**



MINISTÉRIO DA EDUCAÇÃO  
UNIVERSIDADE ABERTA DO BRASIL

# Introdução a Banco de Dados

Profa. Dra. Joyce Aline de Oliveira Marins

Profa. Dra. Gracyeli Santos Souza Guarienti

**2019**

Secretaria de Tecnologia Educacional  
Universidade Federal de Mato Grosso

**Ministro da Educação**

Abraham Weintraub

**Presidente da CAPES**

Anderson Ribeiro Correia

**Diretor Nacional da UAB**

Carlos Cezar Mordenel Lenuzza

**Reitora UFMT**

Myrian Thereza de Moura Serra

**Vice-Reitor**

Evandro Aparecido Soares da Silva

**Pró-reitor Administrativo**

Bruno Cesar Souza Moraes

**Pró-reitora de Planejamento**

Tereza Mertens Aguiar Veloso

**Pró-reitor de Cultura, Extensão e Vivência**

Fernando Tadeu de Miranda Borges

**Pró-reitora de Ensino e Graduação**

Lisiane Pereira de Jesus

**Pró-reitora de Pós-Graduação**

Ozerina Victor de Oliveira

**Pró-reitora de Pesquisa**

Patrícia Silva Osório

**Secretário de Tecnologia Educacional**

Alexandre Martins dos Anjos

**Coordenador da UAB/UFMT**

Alexandre Martins dos Anjos

**Coord. do Curso de Licenciatura em Tecnologia Educacional**

Silas Borges Monteiro

# SUMÁRIO

<b>UNIDADE 1 - Conceitos em Sistemas de Bancos de Dados..</b>	<b>9</b>
Referências.....	19
<b>UNIDADE 2 - Modelagem de Banco de Dados.....</b>	<b>23</b>
Referências.....	39
<b>UNIDADE 3 - Linguagem SQL - DDL (DATA DEFINITION LANGUAGE) .....</b>	<b>43</b>
Referências.....	56
<b>UNIDADE 4 - Linguagem SQL - DML (DATA MANIPULATION LANGUAGE) .....</b>	<b>59</b>
Referências.....	68



# UNIDADE 1

## BIBLIOTECA DE ÍCONES



**Reflexão** – Sinaliza que uma atividade reflexiva será desenvolvida. Para isso, sugerimos que leia a questão feita e anote o que você pensa a respeito da abordagem, antes de qualquer assimilação de novos conhecimentos. Você pode convidar seus colegas para debates, questionar a equipe de tutoria e docentes (usando a ferramenta *mensagem* ou *fórum*). No final do processo, faça uma síntese das ideias resultantes das novas abordagens que você assimilou e/ou construiu, de forma a se preparar para responder perguntas ou questionamentos sobre o assunto refletido.



**Pesquisa e Exercícios** – Indica uma atividade de pesquisa ou exercício propriamente dito, elaborada com a finalidade de conferir a sua compreensão sobre um determinado contexto informativo.



**Saiba mais** – Sugere o desenvolvimento de estudo complementar. No ambiente virtual do curso, na área de “Saiba Mais”, é possível localizar materiais auxiliares, como textos e vídeos, que têm por premissa apoiar o seu processo de compreensão dos conteúdos estudados, auxiliando-o na construção da aprendizagem.



**Atividades** – Aponta que provavelmente você terá uma chamada no seu Ambiente Virtual de Aprendizagem para desenvolver e postar resultados de seu processo de estudo, utilizando recursos do ambiente virtual.

**Vamos aos estudos?**

# UNIDADE 1

## Conceitos em Sistemas de Banco de Dados

Após a leitura deste capítulo, você será capaz de:

- Entender o que é um Banco de Dados e quais são suas características.
- Compreender das fases que compõem o projeto de Banco de Dados.
- Entender o conceito de um Sistema de Gerenciamento de Banco de Dados (SGBD);
- Conhecer as funções de um Administrador de Banco de Dados;

### 1 Introdução

Neste capítulo iremos entender os conceitos fundamentais de um Banco de Dados, quais são suas principais características e utilização. Mas antes disso é preciso entender a diferença entre dado, informação e conhecimento.



Segundo TOREY et al (2007):

- Dado: é o componente básico de um arquivo, é um elemento com um significado no mundo real, que compõe um sistema de arquivos. Como exemplo, podemos citar nome, sobrenome, cidade, bairro e outros.
- Informação: após a interpretação dos dados, é possível associar um significado aos dados ou processá-los. Normalmente a informação vem de convenções utilizadas por pessoas por meio de associações aos dados.
- Conhecimento: todo discernimento, obtido por meio de critérios, e apreciação aos dados e informações.

Como vimos, o dado é um componente básico para compor arquivo. Já registro nos dará uma informação completa, pois ele é formado por uma sequencia de dados juntos. Um exemplo de registro é a ficha de cliente de uma loja que deve conter seus dados pessoais tais como: nome, sobrenome, RG, CPF, endereço (MACHADO, 2008).

Quando temos muitos registros de várias pessoas que são pacientes de uma clínica criamos um arquivo chamado arquivo pessoa. Para ficar clara a diferença entre arquivo, registro e

dado, vamos ao exemplo de um cliente da loja de roupas, representando um de seus registros (RAMAKRISHNAN e GEHRKE, 2002).

O registro é composto por seis itens de dados (campos): código, CPF, nome, rua, bairro, cidade. Dentro do Banco de Dados, as fichas de todos os clientes que estão inseridas formarão o arquivo cliente.

#### FICHA DO CLIENTE

Código: ]

CPF:

Nome:

Rua:

Bairro:

Cidade:

Conforme as definições anteriores, temos que entender agora a definição de Banco de Dados que é ampla:

“Um Banco de Dados é um conjunto de arquivos relacionados entre si” (Chu, 1983)

“Um Banco de Dados é uma coleção de dados operacionais armazenados, sendo usados pelos sistemas de aplicação de uma determinada organização” (C. J. Date, 1985)

“Um Banco de Dados é uma coleção de dados relacionais” (Elmasri e Navathe, 2005).

“Um Banco de Dados é um objeto mais complexo, é uma coleção de dados armazenados e inter-relacionados, que atende às necessidades de vários usuários dentro de uma ou mais organizações, ou seja, coleções inter-relacionadas de muitos tipos diferentes de tabelas.” (TODREY et al,p.2,2007)”

Com base nas definições da literatura, podemos dizer que Banco de Dados é:

- Coleção de dados relacionados que tem informação sobre algo do mundo real, por exemplo, lojas, escritórios, bibliotecas ou banco;
- Possui coerência lógica entre os dados e significado;
- Um Banco de Dados sempre estará associado a aplicações onde existem usuários com interesse aos dados relacionados.

É importante entender que você pode ter um Banco de Dados automatizado (informatizado) utilizando as tecnologias computacionais ou não. Veja bem, uma lista telefônica é bom exemplo de um Banco de Dados que não utiliza informatização, porém ela possui todos os

dados de pessoas, empresas com números de telefone onde podemos consultar por meio de sobrenomes ou índices (ELMASRI e NAVATHE, 2011).



Pesquise outras formas de bancos de dados que não utilizam métodos computacionais. Quais são as maiores dificuldades encontradas para encontrar os dados armazenados em tais bancos?

## 1.2 Sistema De Gerenciamento De Banco de Dados

Podemos armazenar dados de duas maneiras: utilizando bancos de dados, ou então, arquivos de dados permanentes (SILBERSCHATZ, KORF e SUDARSHAN e 2012). Com a utilização de um Banco de Dados temos seguintes vantagens:

- Controle centralizado de dados: os dados ficam localizados em um único local e isso facilita o controle e acesso.
- Controle da redundância: como os dados estão centralizados existe otimização e redução do espaço de armazenamento e controle de redundância
- Compartilhamento de dados: com a utilização de um Banco de Dados sem redundância, o espaço de armazenamento é otimizado e facilita o compartilhamento de dados.
- Facilidade de acesso aos dados: estabelecimento de padrões devido a centralização dos dados e inter-relação de todos os registros;
- Independência de dados: os dados são independentes para cada registro dentro do Banco de Dados.

Segundo DATE (2004), um Sistema Gerenciador de Banco de Dados (SGBD) é um software genérico para manipular Banco de Dados. Ele permite a definição, construção e manejo de um Banco de Dados para diversas aplicações. Um SGBD possui uma visão lógica (projeto do BD), uma linguagem de definição de dados, linguagem de manipulação de dados e utilitários importantes.



Mas o que é linguagem de definição de dados e manipulação de dados?

Basicamente uma linguagem de definição de dados (DDL) é usada para definir o esquema conceitual do Banco de Dados. Na maioria dos SGBDs, a DDL também define as visões dos usuários, e algumas vezes as estruturas de armazenamento já uma linguagem de manipulação de dados (DML) é empregada para especificar as recuperações e atualizações do Banco de Dados (GARCIA-MOLINA, ULLMAN e WIDOM, 2008).

Os SGBDs também facilitam a conversão e a reorganização do Banco de Dados. Dessa forma podemos dizer que os SGBDs são programas de computador, que ajudam na:

- Definição e construção do Banco de Dados: que é o processo de criar uma estrutura inicial com tabelas e preenche-las com dados;
- Manipulação dos dados do Banco de Dados: são as operações de consultas alteração, exclusão realizadas nos dados.



Pesquise sobre os tipos de SGBDs disponíveis no mercado atualmente.

Com a utilização dos SGBDs, para manejo do Banco de Dados, foi possível otimizar e facilitar a manipulação dos dados, dessa forma seu uso cresceu durante o século XX (SILBERSCHATZ, KORF e SUDARSHAN, S, 2012). O Quadro 1 traz algumas aplicações de bancos de dados mais utilizadas pela sociedade.

#### Aplicações de Banco de Dados

Banco: para informações de cliente, contas, empréstimos e transações bancárias;

Universidades: informações de alunos, cursos e notas;

Linhas Aéreas: reservas e informações de horários;

Indústria: controlar a produção de itens da fábrica, estoques e pedidos.

Vendas: informações de cliente, produto e compras;

Biblioteca: informações de livros, nome dos alunos e solicitações.

Fonte: (SILBERSCHATZ, KORF e SUDARSHAN, S, 2012)

A Figura abaixo ilustra um Sistema de Banco de Dados, que é composto por dados, hardware, software e usuários. Observe que o Banco de Dados está dentro do SGBD (DATE, 2004).

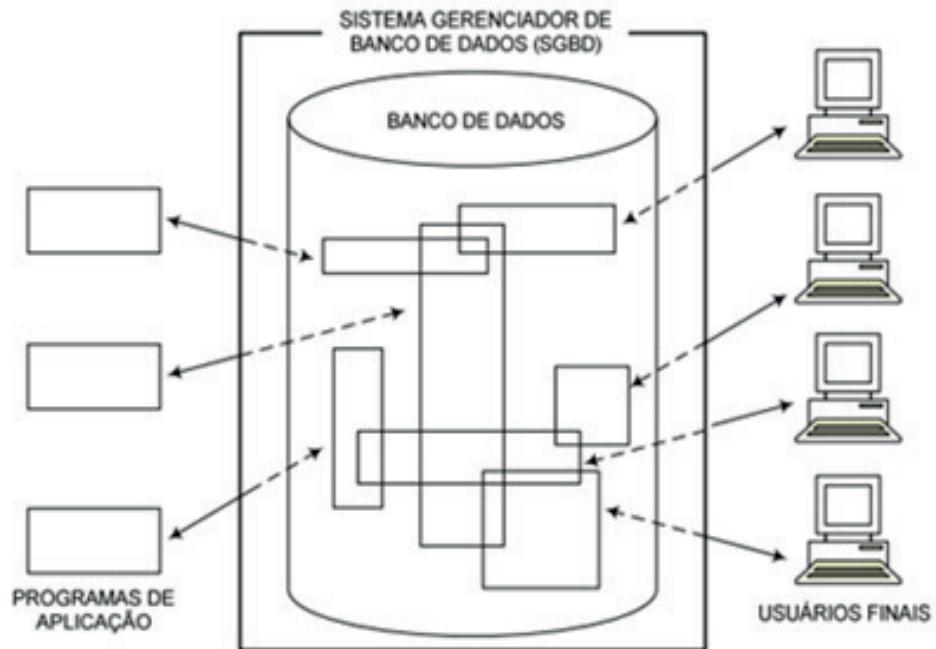


Figura 1: Representação de um sistema de Banco de Dados

Até agora vimos conceitos fundamentais para o entendimento da disciplina pois o objetivo deste capítulo é possibilitar aos cursistas uma introdução básica aos conceitos relacionados a sistemas de Bancos de Dados. Ressalvamos, no entanto, que as noções aqui apresentadas constituem apenas de noções básicas sobre Banco de Dados. Convidamos todos a verificar o material disponibilizado no SAIBA MAIS do Guia de Estudos e sugerimos a leitura dos livros indicados na seção Referências para aprofundamento e detalhes não abordados.

## 2 Visões Do Banco de Dados

Aplicações de Banco de Dados para se tornarem seguras e eficientes tornam-se complexas. Dessa forma os Administradores de Banco de Dados tem funções que devem facilitar a vida do usuário. É função do Administrador de Banco de Dados - *DataBase Administrator* (DBA) garantir que os dados estejam seguros e com desempenho satisfatório. Esse profissional é responsável por garantir (HEUSER, 2004):

- **Segurança do Banco de Dados:** garantir segurança aos dados, permitindo que apenas usuários com acesso adequado possam utilizar o banco.
- **Recuperação** deve sempre realizar procedimentos de backup para evitar que falhas façam que ocorra perda dos dados.

- **Disponibilidade:** ele tem responsabilidade de manter o Banco de Dados sempre disponível.
- **Suporte a equipe de desenvolvimento:** Bom relacionamento entre a equipe e o DBA para desenvolvimento e manutenção.
- **Implementação de Bancos de Dados:** Deve sempre realizar de forma adequada a implementação do banco.



Uma das principais razões para utilizar um SGBD é ter um controle central dos dados e dos programas de acesso a eles. Um DBA é responsável pelo controle sobre o sistema. Pesquise quais são outras responsabilidades de um DBA e veja sua importância.

Uma das funções essenciais ao DBA é utilizar abstração de dados. Com a utilização de abstração de dados, é possível esconder certos “detalhes” sobre como os dados estão armazenados e como é realizada a manutenção, para facilitar o entendimento do usuário (JUKIC, VRBSKY S e NESTOROV, 2013).

Segundo SILBERSCHATZ, KORF e SUDARSHAN (2012) existem três níveis básicos de abstração:

#### Níveis de Abstração

- **Nível físico:** Descreve como os dados estão armazenados. Este é o nível mais baixo de abstração.
- **Nível lógico:** Esse nível de abstração está acima do físico e descreve quais dados estão armazenados no BD e quais são suas relações. Descreve o Banco de Dados inteiro em termos de um pequeno número de estruturas relativamente simples.
- **Nível visões (View)** – Esse nível pode ser visto pelo usuário de diversas formas pois quem opera são os sistemas aplicativos. Esse nível existe para facilitar sua interação com o sistema, ou seja, o sistema pode fornecer muitas visões para o mesmo Banco de Dados.

A Figura 2 ilustra a relação entre os três níveis e a localização de cada uma das abstrações:

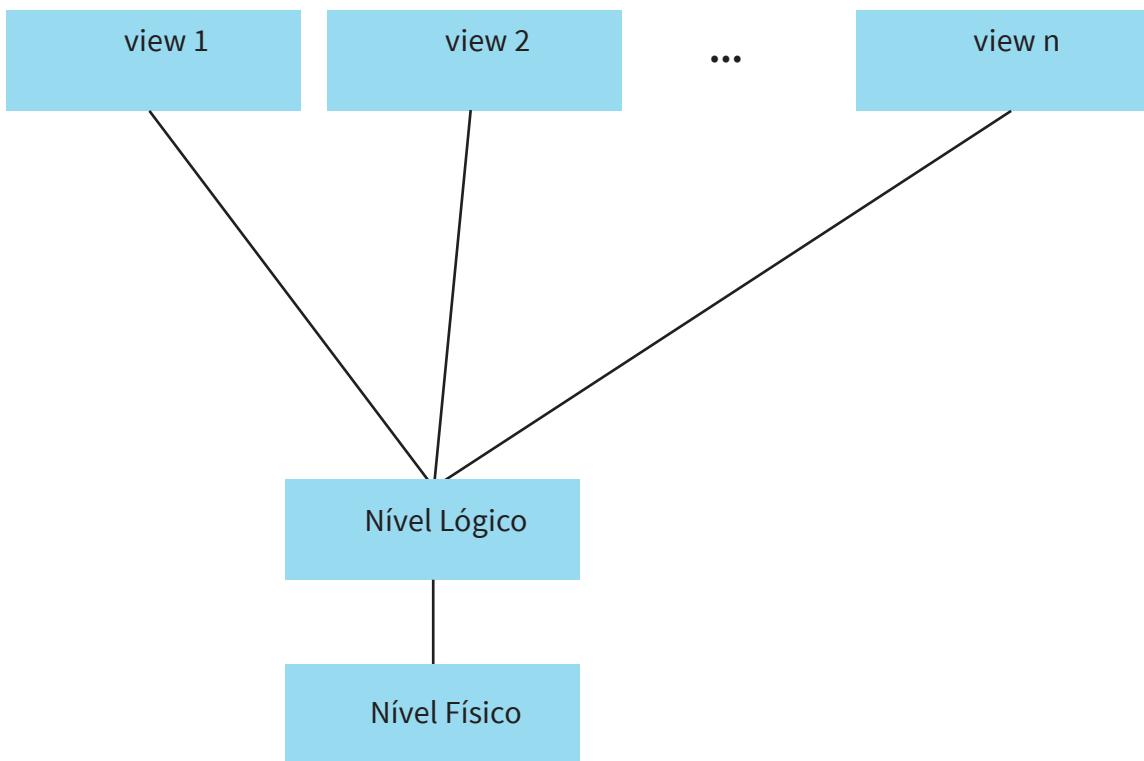


Figura 2: Os níveis de abstração de dados. (FONTE: SILBERSCHATZ, KORF e SUDARSHAN(2012))



Para ampliar o entendimento sobre os níveis de abstração de dados sugerimos que acesse o “Saiba Mais”. Leia o material em anexo.

## 2.1 Projeto de Banco de Dados

Os sistemas de Banco de Dados tem um ciclo de vida para sua execução. ELMASRI e NAVATHE (2011) elencam as etapas em oito fases:

- |   |
|---|
| <b>1. Definição do sistema:</b> nesta etapa é determinado o escopo do sistema, o deverá ser armazenado, e quais serão as operações realização assim como seus usuários. |
| <b>2. Projeto do Banco de Dados:</b> esta etapa é a criação do projeto conceitual, lógico e físico.   |
| <b>3. Implementação do Banco de Dados:</b> cria-se realmente o Banco de Dados, conforme esquemas definidos na etapa anterior.   |

<b>4.Carga ou conversão de dados:</b> nesta etapa, o Banco de Dados é preenchido com dados já existentes ou é preenchido manualmente.
<b>5.Conversão de aplicação:</b> nesta etapa os programas que antes acessavam o banco, são informados sobre as modificações do novo banco (etapa auxiliar).
<b>6.Teste e validação:</b> verifica-se tudo está funcionando de acordo com o planejamento.
<b>7.Operação:</b> é relativo à disponibilização do sistema para uso;
<b>8.Monitoramento e manutenção:</b> nesta etapa observa-se o funcionamento do sistema para possíveis ajustes.

(FONTE: ELMASRI e NAVATHE (2011))

Como vimos uma das etapas do ciclo de vida para execução de Sistemas de Banco de Dados, é Projeto do Banco de Dados que corresponde a 2<sup>a</sup> etapa, onde cria-se o modelo conceitual, lógico e físico.

## 2.2 Modelo Conceitual

Em um projeto de Banco de Dados normalmente utiliza-se o Modelo Entidade Relacionamento (Figura 3) para descrever quais são os requisitos que o usuário deseja no Banco de Dados (GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, 2008). No projeto conceitual é desenvolvido um modelo de alto nível para atender os requisitos elencados pelo usuário (solicitante do banco) durante a definição do sistema (CORONEL, MORRIS, e ROB, 2012).

Dessa forma o modelo conceitual irá descrever a realidade do ambiente real e o problema, sendo uma visão geral dos principais dados e suas relações, independente das restrições de implementação (JUKIC, N.; VRBSKY S.; NESTOROV, S., 2013).

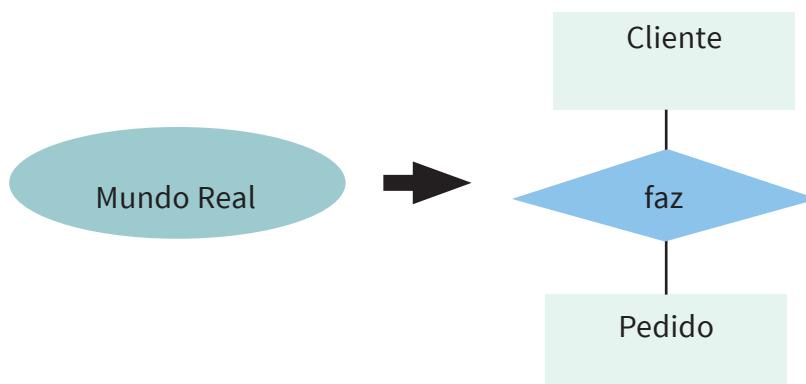


Figura 3: Modelo Entidade Relacionamento.

É possível observar que na Figura 3 onde é ilustrado o modelo Entidade Relacionamento de uma empresa onde o Cliente faz um pedido, é realizada uma abordagem conceitual de como de quais serão as entidades retratadas no Banco de Dados e quais serão suas relações.

Dessa forma podemos concluir que o Modelo Conceitual não irá dizer como será implementado o Banco de Dados por um SGBD tanto em forma física quanto em forma lógica (ELMASRI e NAVATHE, 2011).



Na fase de modelagem conceitual que iremos executar os trabalhos de construção de um modelo de dados. Acesse o saiba mais e veja quais são as ferramentas disponíveis para elaborar diagramas Entidade Relacionamento.

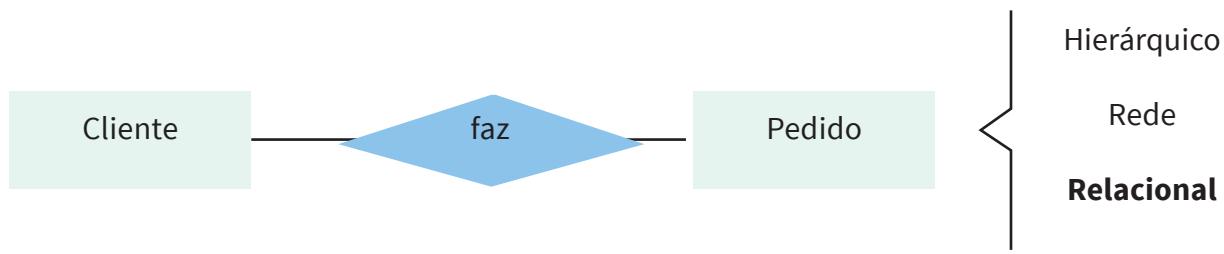
## 2.3 Modelo Lógico

A partir do modelo conceitual, ou seja, após a elaboração do modelo Entidade Relacionamento é possível ter uma visão geral de como será o sistema de Banco de Dados. O Modelo Lógico tem três abordagens atualmente possíveis: Relacional, hierárquica e rede.

- Modelo relacional: classifica os dados em tabelas que possui colunas e linhas, que possuem relacionamentos.(Veremos na Unidade 2)
- Modelo hierárquico: a organização é feita como uma árvore, onde cada registro tem um único “pai” e registros “irmãos” são colocados em uma ordem específica(CORONEL, MORRIS, e ROB, 2012).
- Modelo de rede: Cada conjunto consiste em um registro proprietário, e um ou mais registros de membro. Um registro pode ser um membro, em vários conjuntos, permitindo que esse modelo transmita relações complexas (HEUSER, 2004).

O modelo lógico irá descrever quais serão as estruturas que devem conter no Banco de Dados, sem considerar características específicas do SGBD, ou seja, não há preocupação em qual software será utilizado até este momento.

Entre os três modelos apresentados, o mais utilizado é o modelo Relacional. Trataremos sobre esse assunto com mais detalhes no próximo capítulo.



Pesquise outros tipos de Banco de Dados além do Relacional, Hierárquico e Rede. Pense em como utiliza-los em aplicações reais.

## 2.4 Modelo Físico

O Modelo Físico irá descrever as estruturas físicas de armazenamento de dados, como (HEUSER, 2004):

- Tamanho de campos.
- Tipos dos campos.
- Terminologia dos campos, que serão projetadas de acordo com os requisitos de processamento.
- Eficiência dos recursos computacionais.

Com o modelo físico é feita a implementação do Banco de Dados que envolve aspectos de software e de hardware que serão utilizados. O projeto físico, dependente de qual SGBD foi escolhido e então é especificado as estruturas de armazenamento, os índices e caminhos de acesso ao Banco de Dados (DATE, 2004).

Vimos nesta unidade aprendemos diversos conceitos como o que é dado, informação e relacionamento, também entendemos as definições de um Banco de Dados e um Sistema Gerenciador de Banco de Dados, e quais são os primeiros passos para a modelagem de um banco. Como vimos, o para realização da modelagem de um Banco de Dados é necessário realizar uma sequência de etapas, nos iremos abordar os mecanismos para elaboração da modelagem no próximo capítulo. Alguns exercícios são indicados no GUIA DE ESTUDOS e sua resolução é fortemente indicada.

## REFERÊNCIAS

- ELMASRI, RAMEZ; NAVATHE, SHAMKANT B. (2005) **Sistemas de Bancos de Dados.** Addison-Wesley, 4a edição em português.
- CHU, SHAO YONG. **Banco de Dados: organização, sistemas e administração.** São Paulo, editora Atlas, 1983.
- CORONEL, C.; MORRIS, S.; ROB, P. **Database Systems: Design, Implementation, and Management.** Cengage Learning, 2012.
- DATE, C. J. **Introdução a Sistemas de Bancos de Dados,** Editora Campus, 2004
- DATE, C. J., **Bancos de dados : fundamentos / C. J. Date ;** tradução Helio Auro Gouveia. Local: Rio de Janeiro, Editor: Ed. Campus, Ano: 1985.
- ELMASRI, R.; NAVATHE, S. B. **Sistemas de Bancos de Dados.** 6a. Ed. Addison-Wesley, 2011.
- GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. **Database Systems: The Complete Book,** 2. ed. Prentice Hall, 2008.
- HEUSER, C. A. **Projeto de Banco de Dados,** 5a. ed. Sagra Luzzato, 2004.
- JUKIC, N.; VRBSKY S.; NESTOROV, S. **Database Systems: Introduction to Databases and Data Warehouses.** Prentice Hall, 2013.
- MACHADO, F.N.R. **Projeto de Banco de Dados: uma visão prática.** 15a Ed. São Paulo: Érica, 2008.
- RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems,** 3a. Ed. McGraw-Hill, 2002.
- SILBERSCHATZ, A.; KORF, H. F.; SUDARSHAN, S. **Sistemas de Banco de Dados,** 6a. Ed. Elsevier, 2012.



# UNIDADE 2



# UNIDADE 2

## Modelagem de Banco de Dados

Após a leitura deste capítulo, você será capaz de:

- Entender os principais conceitos de cada fase de projeto de banco de dados.
- Conhecer e implementar os modelos de representação de dados para cada fase de projeto de banco de dados e seus elementos.
- Compreender os conceitos do modelo relacional
- Realizar a conversão do modelo Entidade Relacionamento para o modelo Relacional.

### 1 Introdução

Neste curso, estamos trabalhando os conceitos fundamentais a respeito de bancos de dados e sua utilização. Vimos na unidade anterior o conceito de Banco de Dados e Sistema Gerenciador de Banco de Dados (SGBD), porém, antes de utilizar a ferramenta propriamente dita, é necessário realizar a modelagem dos dados, para que seja definida uma estrutura adequada e que o Banco de Dados seja eficiente (Elmasri e Navathe, 2005). Dessa forma nosso próximo passo é entender como podemos realizar a modelagem de dados.



Afinal, o que é um modelo de dados?

O modelo de dados é uma junção de ferramentas possibilidades para descrição dos dados, sua semântica, relações e restrições de consistência. Ele é fundamental para criação da estrutura de um Banco de dados, pois dará sua definição. Os modelos de dados possibilidades uma forma de apresentação do projeto de banco de dados nos níveis Físico, Lógico e de Visão (view) (SILBERSCHATZ, KORF e SUDARSHAN e 2012).

Sendo assim, os modelos de dados podem ser classificados em quatro categorias (HEUSER, 2004):

- **Modelo relacional:** Utiliza a estrutura de tabela para representação dos dados e suas relações.
- **Modelo de entidade/relacionamento:** descreve entidades e relacionamentos entre elas. Este modelo será o foco deste capítulo.
- **Modelo de dados baseado em objeto:** este modelo descreve os dados com extensões do modelo Entidade-Relacionamento, porém tem características de orientação a objeto como encapsulamento, métodos (funções) e identidade de objeto.
- **Modelo de dados semi-estruturado:** totalmente antagônico ao modelo de dados, permitindo a especificação dos dados que itens individuais do mesmo tipo q que possam ter diferentes conjuntos de atributos para representar dados estruturados utiliza a XML (Extensible Markup Language).

Como vimos na Unidade 1, o modelo de rede e o modelo de dados hierárquico vieram antes do modelo de dados relacional, esses modelos estavam intimamente ligados a implementação da aplicação e complicava a tarefa de modelagem. Isso significa que, se houvesse algum problema com a aplicação todo o banco precisaria ser modelado novamente (CORONEL, MORRIS, e ROB, 2012).

Como o processo de abstração somente os elementos essenciais da realidade observada são enfatizados este processo chama-se modelo Conceitual. Com este modelo é possível pensar e expressar as propriedades estáticas e dinâmicas das aplicações auxiliando assim aplicações (DATE, 2004).

No nível conceitual (projeto conceitual) utilizam-se modelos semânticos, voltados para o entendimento dos usuários e projetistas envolvidos no projeto. No nível lógico temos o modelo relacional, objeto-relacional, que são implementados no SGBD, incluindo suas linguagens para definição e manipulação de dados (GARCIA-MOLINA, ULLMAN e WIDOM, 2008).

O modelo que iremos estudar é o modelo conceitual, chamado Modelo Entidade Relacionamento (E-R). Apresentaremos também a notação diagramática associada ao modelo ER, conhecida por diagramas ER. Esse modelo e suas variações são utilizados projetos a nível conceitual de aplicações de um banco de dados, e muitas ferramentas de projeto de um banco de

dados aplicam seus conceitos (SILBERSCHATZ, KORF e SUDARSHAN, S, 2012). Descreveremos os conceitos da estruturação de dados básica, e discutiremos seu uso no projeto de esquemas conceituais para aplicações de bancos de dados.



Pesquise sobre programas para construção de Diagramas Entidade Relacionamento.

## 2 Modelo Entidade - Relacionamento (MER)

Em 1976 Peter Chen publicou um trabalho intitulado “The Entity-Relationship Model: Toward the unified view of data”, no qual definia o processo de modelagem de dados. Este trabalho foi publicado e amplamente aceito, após divulgação passou a ser considerado o referencial definitivo para o processo de modelagem de dados (COUGO, 1997),

O modelo Entidade Relacionamento é composto por uma técnica de diagramação e de um conjunto de conceitos simples e serve como meio de representação dos próprios conceitos por ela manipulados (RAMAKRISHNAN e GEHRKE, 2002)

A proposta original de Peter Chen se estabeleceu e continua atualizada atualmente, e é baseada na formalização do óbvio, ou seja, a Lei do mundo:

“O mundo está cheio de coisas que possuem características próprias e que se relacionam entre si.”

Assim Chen visualizou um universo composto por um grupo básico de objetos chamados de entidades, com suas características próprias (os atributos) e relacionamentos entre esses objetos(entidades).

### 2.1 Representação Gráfica

Podemos representar de forma gráfica o mundo real, por meio do diagrama Entidade Relacionamento para então seguir com a implementação no Banco de Dados. Para realizar a representação de cada objeto utiliza-se retângulo para representar entidades, um losango para representar os relacionamentos e elipses para indicar os atributos, como mostra a Figura 1.

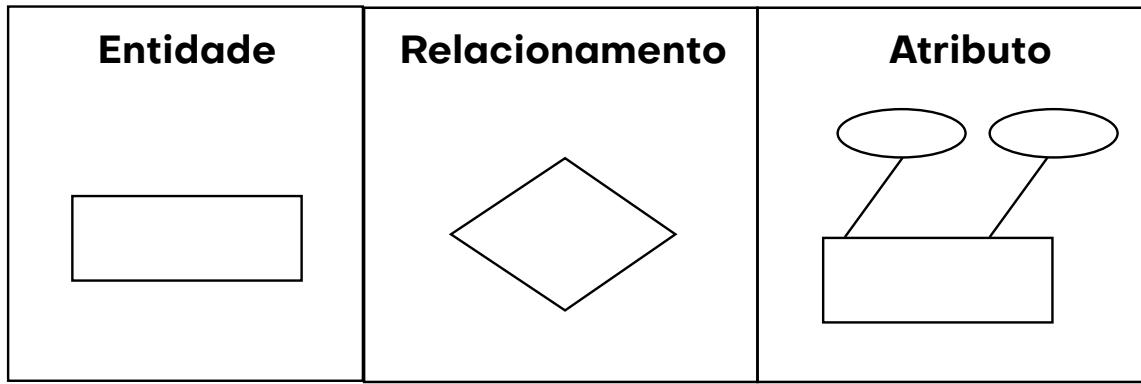


Figura 1: Representação Diagrama Entidade Relacionamento.

**Entidade** - É a representação abstrata de um objeto do mundo real sobre o qual iremos guardar informações. Exemplo: Funcionários, Departamentos, Empresa, Clientes, Fornecedores, Alunos, etc. Nenhuma entidade é igual a outra pois possui o conjunto de atributos diferentes.

**Instância de Entidade** - São os dados da entidade em determinado momento. Por exemplo: Em uma tabela pessoa temos os campos CPF = 035.745.888-55 e nome do cliente = ‘Maria’.

**Atributo** – São as características da entidade. Exemplo: CPF, RG, Nome, Endereço do fornecedor, Estado Civil do funcionário, Nome do aluno, Número da turma, etc.

**Relacionamento** - Representa a associação (relação) entre as entidades. Por exemplo: Um aluno está matriculado em uma disciplina. Se pensarmos na instância de aluno e de disciplina, pode-se citar a situação: O João matriculado na disciplina de ginástica onde:

- João - Elemento do atributo Nome do aluno da entidade Aluno;
- Ginástica - valor do atributo Nome da disciplina da entidade Disciplina;
- Matriculado - Relação existente entre um aluno e uma disciplina.





Acesse o ambiente virtual do curso, na área de “Saiba Mais” e veja os diferentes tipos de diagramas Entidade Relacionamento. Agora tente fazer um com um problema simples

## 2.2 Cardinalidade de Relacionamentos

Durante o processo de modelagem deve ser realizado o entendimento do relacionamento para poder estabelecer as regras de associações entre os elementos (entidades) (ELMASRI e NAVATHE, 2011).

Existem casos, em que uma regra pode ser verdadeira para uma entidade X e poderá ser observado como falsa para a outra entidade Y (JUKIC, N.; VRBSKY S.; NESTOROV, S., 2013). Quando observamos relações entre entidades, devemos fazer as seguintes perguntas para estabelecer o grau de associação (MACHADO, 2008):

- Com quantos elementos do tipo B se relaciona cada um dos elementos do tipo A?
- Dado um elemento do tipo B, com quantos elementos do tipo A ele se relaciona?

Tais perguntas representam a frequência com que existe o relacionamento, ou seja, o número de vezes que uma entidade se relaciona com a outra, denominada cardinalidade de relacionamentos.

A cardinalidade por meio de uma notação, que diz qual o grau de associação entre as entidades e esta notação fica entre o losango e o retângulo. Vejamos os exemplos a seguir:

### PRIMEIRO EXEMPLO: RELACIONAMENTO 1:1 (UM-PARA-UM)

Exemplo: **pessoa dirige um veiculo**

Se pensarmos na instância de PESSOA como João e no VEICULO caminhão temos:

- João - Elemento do atributo Nome da entidade PESSOA.
- Caminhão - Elemento do atributo Nome da entidade VEICULO.
- Dirige - Ligação entre pessoa e veiculo, sendo que:

Uma pessoa pode dirigir apenas um veiculo e um veiculo pode ter apenas um motorista.



## SEGUNDO EXEMPLO: RELACIONAMENTO 1:N OU N:1 (UM-PARA-MUITOS) E (MUITOS -PA-RA-UM)

Exemplo: **empregado trabalha em departamento**

Este é um exemplo clássico citado por (Elmasri e Navathe, 2005) :

- Pedro - Elemento do atributo Nome da entidade Empregado.
- Departamento de Contabilidade - Elemento do atributo Nome do Departamento da entidade Departamento.
- Trabalha - Relação entre um Funcionário e um Departamento, onde:

Um funcionário pode trabalhar em somente um departamento e

um departamento pode ter vários funcionários.



O que acontece caso inverta a cardinalidade dos relacionamentos?

Caso ocorra a inversão de cardinalidade, todo o modelo conceitual será modificado, e como mostrado na Figura abaixo em que houve a inversão, a implementação no banco seria: um empregado está lotado em VÁRIOS departamentos, mas UM departamento só pode ter um único empregado trabalhando nele.



Nestes casos, deve sempre fazer a verificação do modelo, a fim de evitar possíveis falhas.

### **TERCEIRO EXEMPLO: RELACIONAMENTO N : M (MUITOS-PARA-MUITOS)**

Exemplo: **alunos matriculados em disciplinas**

Exemplo de instância:

- Matheus está matriculado na disciplina Banco de Dados.
- Matheus - Elemento do atributo Nome da entidade Aluno.
- Banco de Dados - Elemento do atributo Nome da Disciplina da entidade Disciplina.
- Matriculado - Relação existente entre um aluno e uma disciplina, onde:

Um aluno pode estar matriculado em várias disciplinas e

cada disciplina pode ter vários alunos matriculados.



A leitura deste diagrama Entidade Relacionamento deve ser feita assim: Um aluno está matriculado em uma ou muitas disciplinas e uma disciplina tem vários alunos matriculados nela.

Devemos observar que no caso da cardinalidade (M:N e N;M) utilizam-se letras diferentes para que fique denotado que as duas entidades são grandezas diferentes, ou seja, pode ser atribuído valores diferentes.

## 2.3 Número De Elementos Que Participam Do Relacionamento

Até agora, vimos apenas relacionamento entre duas entidades para fins de facilitação de entendimento. Mas existem maneiras de dois ou mais elementos se relacionarem, na verdade, um relacionamento pode se estabelecer entre vários elementos e não somente entre dois (HEUSER, 2004).

Exemplos:

1. Relacionamentos Binários:

AVIÃO transporta CARGA

HOTEL localiza-se em CIDADE

REVISTA contém ANUNCIO

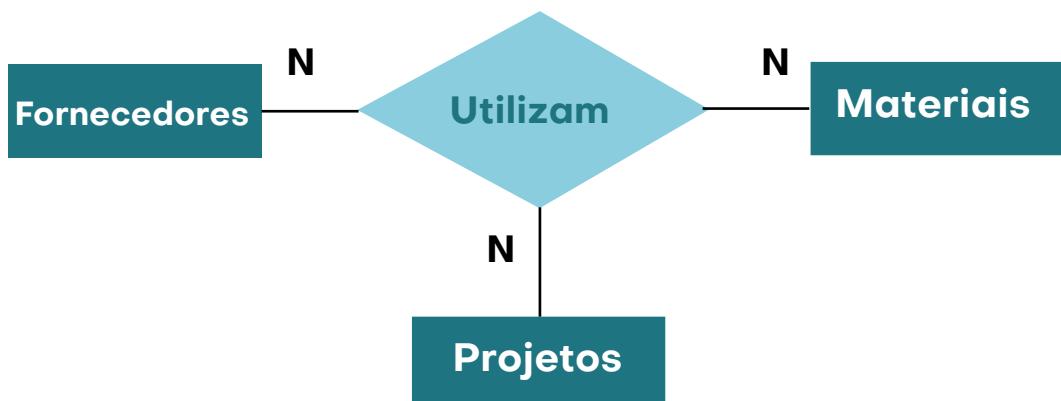
2. Relacionamentos Ternários:

FUNCIONÁRIO trabalha em PROJETO executando FUNÇÃO

PEÇA é fornecida por FORNECEDOR para um PROJETO

Em relacionamentos ternários, as associações de dois elementos não conseguem mapear a informação desejada. Dessa forma sempre que se queira a informação completa deve-se agregar os três elementos (entidades) (JUKIC; VRBSKY S e NESTOROV, 2013).

Como exemplo, podemos ver ilustrado no diagrama Entidade Relacionamento abaixo:



Geralmente os modelos que possuem mais de quatro entidades relacionadas são difíceis de serem implementados. Uma boa prática de modelagem é modelo e tentar modificá-lo.

## 2.4 Chaves

É importante aprender o conceito de chave em modelagem de Banco de Dados, pois ela implementa restrições que garantem a integridade referencial dos dados no banco de dados, ou seja, é uma garantia essencial para a segurança, quando bem modelada (SILBERSCHATZ, KORF e SUDARSHAN, 2012). Existem vários tipos de chave, como: chave candidata, chave primária, chave estrangeira e superchave.

Vamos entender cada uma delas!

Basicamente todos os atributos podem ser candidatas a chave. Claro, se ele tiver dados repetidos com muita frequência, não é um bom candidato à chave. Existem alguns critérios para eleger dentre as chaves candidatas a, por exemplo, chave primária. Por exemplo, se conseguirmos encontrar um atributo que sozinho consiga distinguir uma entidade cliente de outra? Como o atributo Código\_Umico de uma empresa do conjunto de entidade Funcionário e consegue, porque é um número único para cada Funcionário.

Bom, se você encontrar esse atributo, então ele será sua chave primária. Podemos assim dizer que a Chave Primária é a melhor chave candidata escolhida entre todas as candidatas, porque possibilita a identificação exclusiva de um registro (entidade) dentro do conjunto de registros da tabela do banco de dados.

Chave primária é o nome dado a chave candidata, escolhida por um projetista de banco de dados para identificar de forma única a entidade.

A chave primária é representada com um traço abixo do atributo escolhido para executar essa função:



- Chave estrangeira é o nome dado à chave primária de uma tabela que vai para outra tabela no intuito de realizar os relacionamentos entre elas e fazer referência. O nome estrangeira é justamente porque aquele campo não tem origem nessa tabela, sendo uma conexão entre as entidades. O campo vem de outra tabela considerada estrangeira.

Superchave é um conjunto de um ou mais atributos que, tomados coletivamente, permite-nos identificar unicamente uma entidade dentro de um conjunto de entidades. Todos os atributos que podem ser interessantes para ser chave são considerados superchave.



O que é um atributo interessante?

São aqueles em que seus dados não se repetem com frequência. Quando a superchave é mínima, ou seja, não existe um subconjunto, que seja superchave também, ela é chamada de chave candidata (SILBERSCHATZ, KORF e SUDARSHAN, S, 2012; HEUSER, 2004). Por exemplo, em uma tabela cliente temos o CPF, nome e data de nascimento de um cliente. Verifica-se nesse exemplo que a junção de CPF, nome cliente e data nascimento são superchave. CPF é chave candidata, porque é atributo que identifica de forma única a entidade e não tem como dividir, já nome cliente e data cliente não podem se dividir, pois esses atributos sozinhos se repetem com muita frequência. Portanto, os dois juntos são superchave mínima. Quando não há divisão também faz com que dois atributos juntos sejam uma chave candidata (ELMASRI e NAVATHE, 2011).



Existem outros tipos de chaves. Pesquise e veja qual sua melhor utilização.

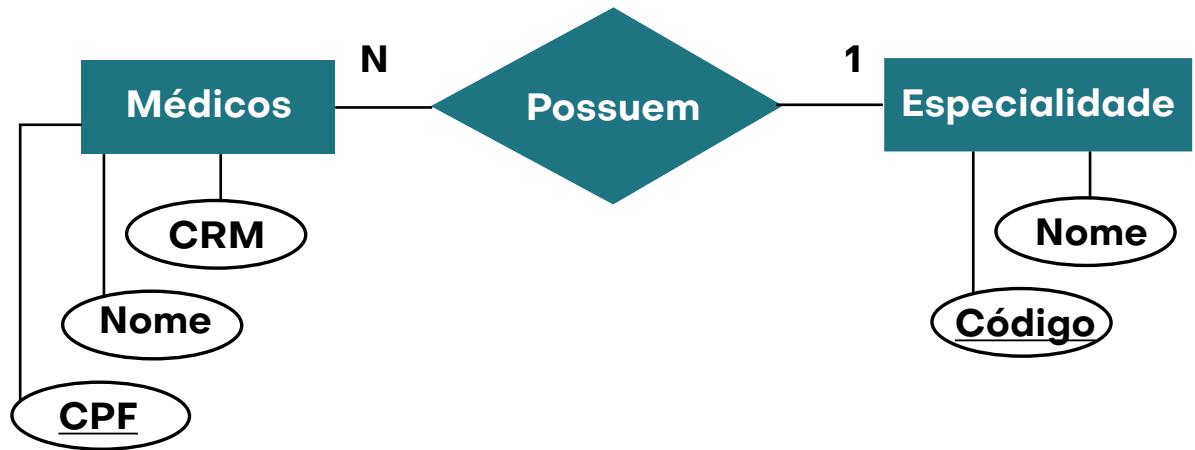
Agora que você sabe os conceitos principais do modelo Entidade Relacionamento, e como fazer um diagrama Entidade Relacionamento, vamos ver alguns exemplos completos que envolvem entidades, relacionamentos e suas cardinalidades:

- **Médicos possuem uma especialidade única**

Devemos sempre ler primeiro da esquerda para direita, ou seja, da entidade médicos para especialidade, depois voltar em sentido contrário.

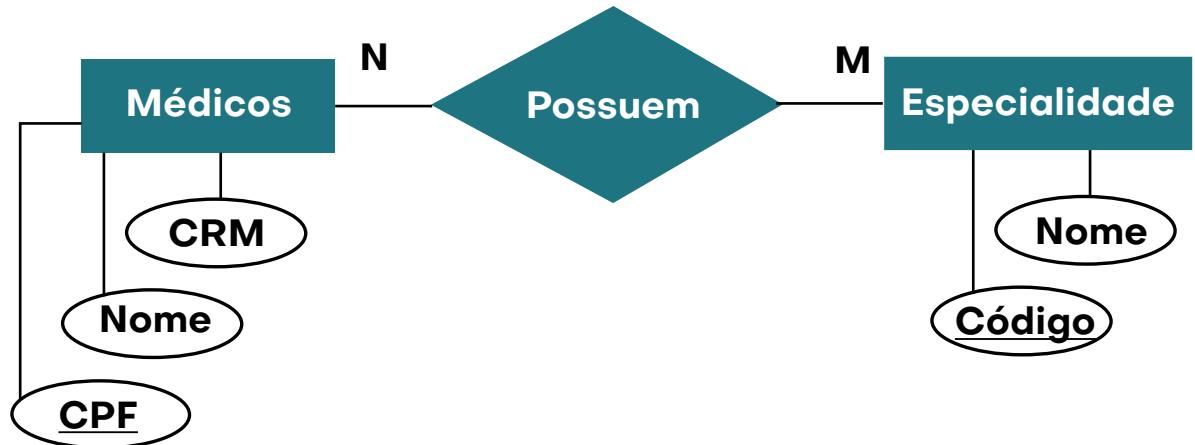
Exemplo: um médico possui uma e só uma especialidade, voltando, temos uma especialidade que pertence a um ou vários médicos (isso porque a cardinalidade em médicos diz que

pode ser n (várias).



- **Médicos podem ter mais de uma especialidade**

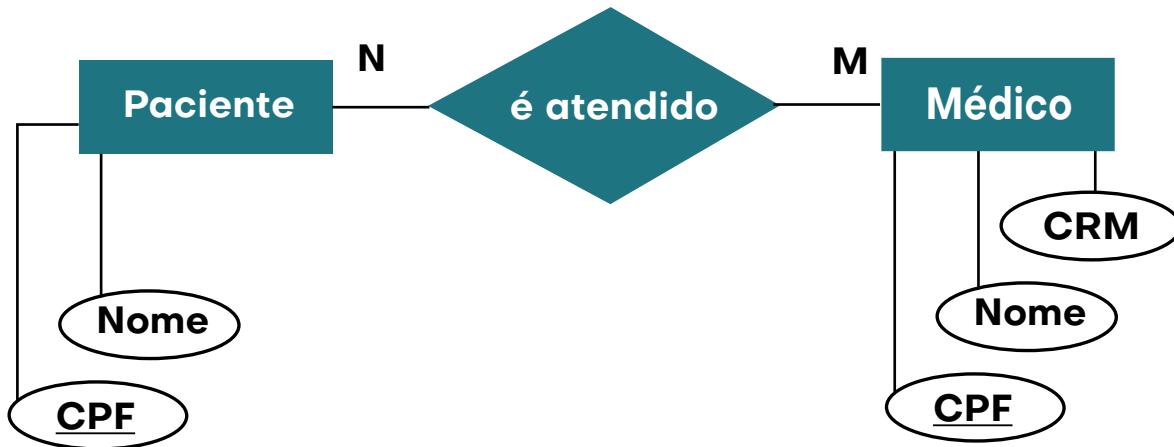
Neste caso um médico possui várias especialidades, e temos que uma especialidade pode pertencer a um ou vários médicos isso porque a cardinalidade em médicos diz que pode ser N (várias).



- **Um paciente é atendido por mais de um médico e um médico atende vários pacientes.**

Como vimos sempre devemos fazer a leitura da esquerda para a direita, e neste caso temos uma entidade diferente que é o paciente.

Neste caso um paciente é atendido por vários médicos e fazendo a leitura inversa temos um médico que atende a vários pacientes. Este é normalmente o cenário encontrado em aplicações de Banco de clínicas médicas.

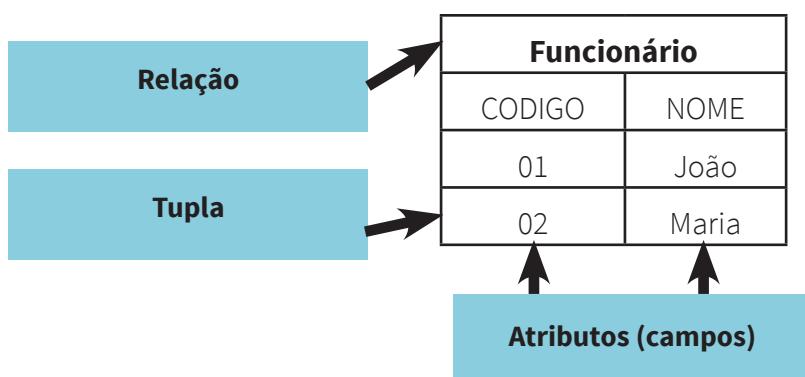


Pesquise as notações: Tipo de Entidade-Fraca, Tipo de Relacionamento Identificador, Atributo Multivalorado e Atributo Composto, para o diagrama Entidade Relacionamento. E tente utilizar cada uma dessas novas notações.

### 3 Teoria Relacional

A teoria relacional é usada para descrever um banco de dados por meio de conceitos simples, sendo uma forma diferente do diagrama Entidade Relacionamento (MACHADO, 2008). É usada para aperfeiçoar a visão dos dados para o projetista fazendo com que a visualização do banco de dados seja vista como um conjunto de tabelas compostas por linhas e colunas.

Outro conceito importante é a transparência que a teoria relacional, pois não é relevante ao usuário saber onde os dados estão nem como os dados estarão armazenados (ELMASRI e NAVATHE, 2005). O usuário manipula esses objetos disponibilizados em linhas e colunas das tabelas (Entidades) que possuem informações sobre o mesmo assunto, de forma estruturada e organizada.



A teoria Relacional possui premissas que definem uma tabela de dados (MACHADO, 2008):

1. Cada uma das tabelas é chamada de relação;
2. O conjunto de uma linha e suas colunas é chamado de tupla;
3. Cada coluna dessa tabela tem um nome e representa um domínio da tabela;
4. A ordem das linhas é irrelevante;
5. Não há duas linhas iguais (conceito de chave primária);
6. Usamos nomes para fazer referência às colunas;
7. A ordem das colunas também é irrelevante;
8. Cada tabela tem um nome próprio, distinto de qualquer outra tabela no banco de dados.

A representação por meio de um uma coleção de relações do modelo de dados relacional traz uma representação dos dados da base de dados como uma tabela propriamente dita. Informalmente, cada relação pode ser entendida como uma tabela ou um simples arquivo de registros.

Atenção! Ao tentar acessar as informações, é necessário busca-las nas tuplas de forma sequencial, as chaves são importantes para encontrar determinada tupla dentro do conjunto de tuplas da tabela de forma aleatória.

Conceitos Empregados No Modelo Relacional:

- Esquema do Banco de Dados: Descrição textual (ou gráfica) dos dados e suas estruturas. Consiste na descrição de esquemas de todas as tabelas do Banco de Dados.
- Instâncias do Banco de Dados: Conjunto de dados armazenados em um dado momento no Banco.
- Estado do Banco de Dados: Uma transação no Banco De Dados deve leva-lo de um estado consistente a outro estado consistente. Caso contrário essa transação deve ser desfeita.

Esse modelo utiliza linguagens para definição, manipulação e consulta do Banco de Dados. A linguagem utilizada no modelo relacional é a SQL (Structure Query Language). Nos próximos capítulos desta disciplina falaremos sobre a linguagem SQL (GARCIA-MOLINA, ULLMAN e WIDOM, 2008)..

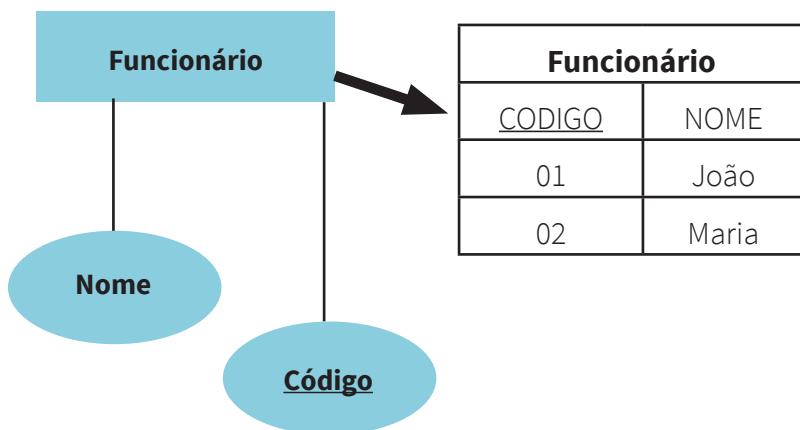


Mas e como é possível realizar a conversão do modelo Entidade Relacionamento para o modelo Relacional?

Ao desenvolver o diagrama Entidade Relacionamento, temos de forma clara quais são as entidades e seus atributos representadas em nível mais baixo.. Para realizar a conversão e não ter erros existe regras para sua realização veremos algumas aqui (MACHADO, 2008):

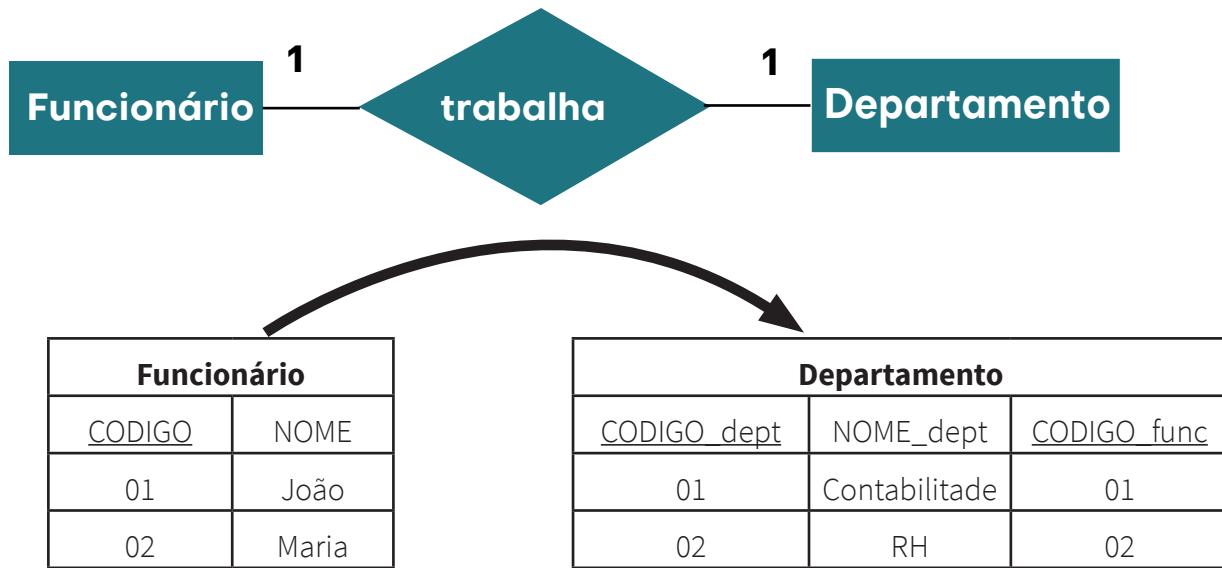
- **Mapeamento Das Entidades:**

Toda entidade deve ser uma tabela, e carregar todos os atributos (definidos para a entidade). Cada atributo vira um campo da tabela criada. Cada uma das chaves gera estruturas de acesso. A chave primária é projetada para não permitir ocorrências múltiplas e nem valores nulos.



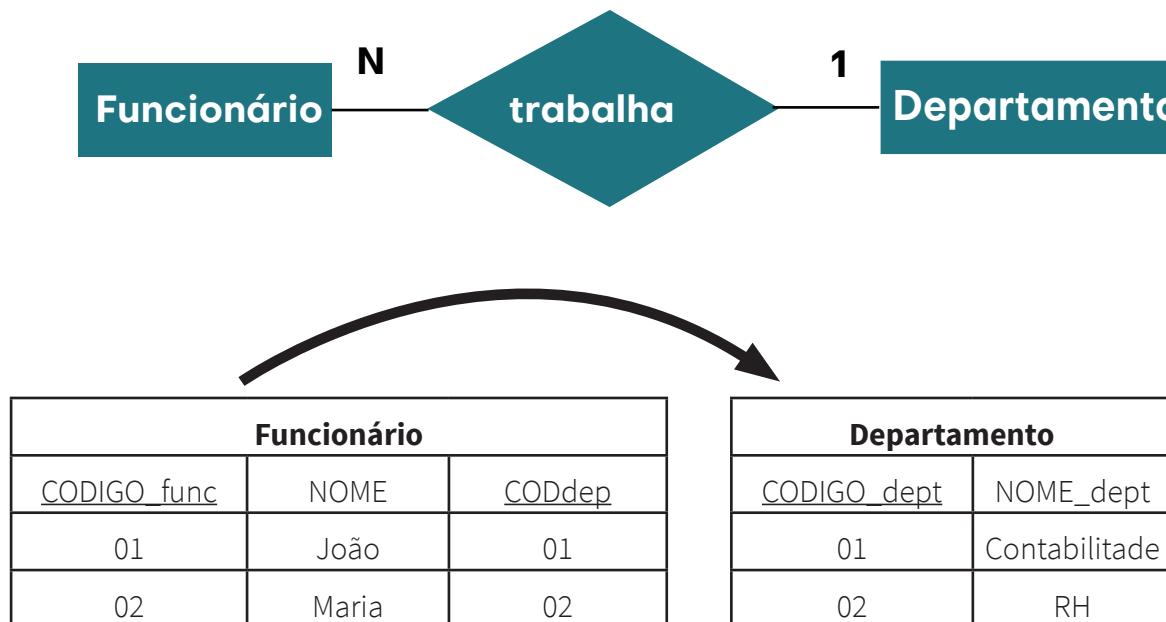
- **Mapeamento dos relacionamentos Relacionamento 1:1 (um-para-um):**

Continuaremos utilizando o exemplo ilustrado por Elmasri e Navathe (2005), que mostra a relação entre funcionário e departamento. Neste caso uma das entidades envolvidas deve carregar o identificador da outra para realizar a conexão (relacionamento) entre as tabelas, a escolha de qual tabela deverá carregar o atributo conexão deve acontecer conforme a conveniência do projeto. Uma dica é que se as operações que manipulam esse Banco realizam um grande numero de operações sobre o conjunto de dados do DEPARTAMENTO como no exemplo abaixo, então, a melhor solução neste caso, seria colocar como chave na tabela departamento.



- **Relacionamento 1:N (um-para-muitos)**

Neste caso a entidade, cuja relação é do tipo várias (N), é carregada o identificador da entidade (tabela), cuja conectividade é 1 (chave estrangeira), e os atributos do relacionamento.



- **Relacionamento N:M (muitos-para-muitos)**

Vejamos agora o relacionamento de muito para muitos, neste caso a relação irá tornar-se uma tabela que carrega os atributos (se houver) e os identificadores das entidades que ele relaciona. Esse é o único caso em que um relacionamento torna-se uma tabela.



Funcionário	
CODIGO_func	NOME_func
01	João
02	Maria

É ALOCADO	
CODIGO_FUNC	CODIGO_PROJ
01	100
02	200

Projeto	
CODIGO_proj	NOME_proj
100	VIVA
200	NATUREZA

Após realizar a conversão Do Modelo Entidade-Relacionamento para o Modelo Relacional é possível implementar em um SGBD o modelo desenvolvido. Veremos na próxima unidade como criar um banco de dados utilizando a linguagem de definição de dados (DDL) em Linguagem SQL(Structured Query Language).

Para uma leitura complementar, acesse o Saiba Mais e leia o material sobre modelagem de banco de dados em anexo. Trabalhamos até aqui uma visão geral de modelagem de Banco de Dados no módulo dois de nossa disciplina. Em caso de dúvidas sobre o conteúdo estudado, sugerimos que retome a leitura do material e acesse a área Saiba Mais no AVA para consultar os materiais complementares. Ou converse com seu tutor e apresente os seus questionamentos.

Na sequência, passaremos para o Módulo III... Seguimos adiante!

## REFERÊNCIAS

- COUGO, Paulo S. **Modelagem Conceitual e Projeto de Banco de Dados.** Rio de Janeiro: Campus, 1997.
- ELMASRI, RAMEZ; NAVATHE, SHAMKANT B. (2005) **Sistemas de Bancos de Dados.** Addison-Wesley, 4a edição em português.
- CORONEL, C.; MORRIS, S.; ROB, P. **Database Systems: Design, Implementation, and Management.** Cengage Learning, 2012.
- DATE, C. J. **Introdução a Sistemas de Bancos de Dados,** Editora Campus, 2004
- ELMASRI, R.; NAVATHE, S. B. **Sistemas de Bancos de Dados.** 6a. Ed. Addison-Wesley, 2011.
- GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. **Database Systems: The Complete Book,** 2. ed. Prentice Hall, 2008.
- HEUSER, C. A. **Projeto de Banco de Dados,** 5a. ed. Sagra Luzzato, 2004.
- JUKIC, N.; VRBSKY S.; NESTOROV, S. **Database Systems: Introduction to Databases and Data Warehouses.** Prentice Hall, 2013.
- MACHADO, F.N.R. **Projeto de Banco de Dados: uma visão prática.** 15a Ed. São Paulo: Érica, 2008.
- RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems,** 3a. Ed. McGraw-Hill, 2002.
- SILBERSCHATZ, A.; KORF, H. F.; SUDARSHAN, S. **Sistemas de Banco de Dados,** 6a. Ed. Elsevier, 2012.



# UNIDADE 3



## UNIDADE 3

# Linguagem SQL – DDL (DATA DEFINITION LANGUAGE)

Após a leitura deste capítulo, você será capaz de:

- Entender a Linguagem SQL
- Utilizar os comandos da linguagem DDL- Linguagem de Definição de Dados para criar a estrutura do banco de dados.

### 1 Introdução

Na Unidade II, vimos como projetar um modelo conceitual utilizando o modelo Entidade Relacionamento, em seguida vimos como transformar o modelo conceitual em um modelo com tabelas chamado modelo relacional que tem maior proximidade com a estrutura de um Banco de dados. Agora nesta unidade, iremos estudar uma linguagem padrão chamada SQL (Structured Query Language ou Linguagem de Consulta Estruturada), muito usada por todos os fabricantes de Sistema de Gerenciamento de Banco de Dados.



Vamos entender um pouco o que é a linguagem SQL?

A linguagem SQL é um padrão de linguagem de consulta de Banco de Dados que usa uma combinação de construtores em Álgebra e Cálculo Relacional e possui as seguintes características principais (ELMASRI e NAVATHE, 2005):

- Linguagem de definição de dados (DDL)
- Linguagem interativa de manipulação de dados (DML)
- Definição de Visões
- Autorização de usuários

- Integridade de dados
- Controle de Transações

Segundo Machado (2008), a primeira versão da linguagem SQL recebeu o nome de SEQUEL, entre 1976 e 1977, por razões jurídicas e depois de revisada, teve seu nome alterado para SQL. A IBM tinha um projeto grandioso chamado System R e foi logo colocado em execução e surgiram novas alterações na linguagem.

Houve um sucesso grande com a nova maneira para realização de consulta em banco, e a utilização da linguagem SQL aumentou. Dessa forma os SGBD's começaram a adquirir a linguagem como padrão DB2 da IBM, Oracle da Oracle Corporation, Sybase da Sybase Inc., Microsoft SQL Server da Microsoft, etc. Dessa forma a SQL tornou-se um padrão em bancos relacionais, faltava apenas tornar-se de direito. Em 1982, o American National Standard Institute (ANSI) tornou a SQL o padrão oficial de linguagem em ambiente relacional sendo hoje utilizada na grande maioria dos sistemas de Bancos de Dados relacionais, tais como MySQL, DB2, SQLServer.

Neste curso, estudaremos os comandos existentes na linguagem SQL:

- **Comandos DDL (Data Definition Language):** Conjunto de comandos responsáveis pela criação, alteração e deleção da estrutura das tabelas e índices de um sistema (DATE, 2004).
- **Comandos DML (Data Manipulation Language):** Conjunto de comandos responsáveis pela consulta e atualização dos dados armazenados em um banco de dados (HEUSER, 2004). Veremos esses comandos na Unidade 4.

## 2 Comando DDL

Vamos revisar o que vimos na Unidade 2 sobre o modelo relacional:

- Formado por tabelas
- Linhas representam os registros (tuplas)
- Colunas representam atributos (campos).

Vimos também que cada registro deve ser único e é diferenciado por uma chave primária que só permite registro de determinado valor nesse campo.

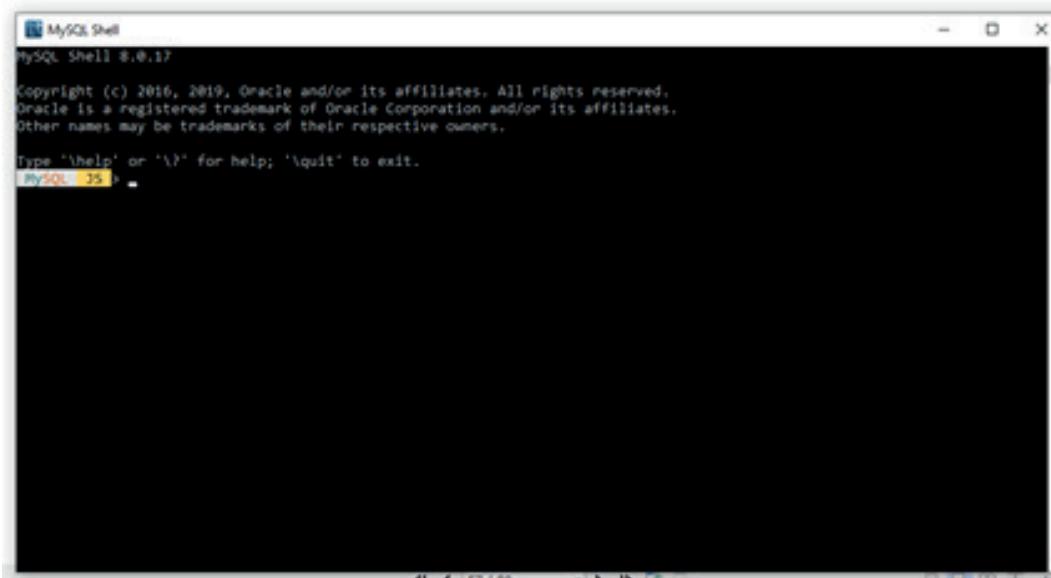
Por meio da DDL definimos estruturas do banco, isso significa que com essa linguagem realmente podemos criar um banco de dados em um SGBD, com: tabelas, visões, sequenciais e outras estruturas (HEUSER, 2004). O resultado no uso da DDL constitui em um arquivo chamado de dicionário ou diretório de dados, ele é um arquivo de metadados<sup>1</sup> (ELMASRI e NAVATHE, 2005).

Resumindo a DDL nos fornece o conjunto de comandos que nos possibilitam a criação de tabelas do modelo relacional, comandos que identificam chaves, comandos que apagam ou alteram estruturas das tabelas.

Para utilizar a linguagem SQL, com os comandos DDL, utilizaremos o SGBD MySQL<sup>2</sup>, que é livre, e possui grande aplicabilidade em diversos tipos de sistemas. Ele é uma plataforma modular que oferece uma para criação de estruturas de projetos de Bancos de Dados. Existem três versões diferentes do MySQL, utilizaremos a MySQL Standard Edition que se adapta tanto a pequenos projetos ou aplicativos web de menor escala (MYSQL, 2019).

Ao realizar o download do MySQL, ele irá requisitar a configuração de uma senha para o localhost, então devemos cadastrá-la e seguir cada etapa até a finalizar a instalação.

Após a instalação será iniciado um terminal com todas as ferramentas de Banco de Dados, e se você selecionou durante a instalação as ferramentas complementares como Workbenck, todas irão abrir após a instalação, conforme mostra Figura abaixo:



---

1 Metadados: dados a respeito de dados (informações adicionais). Em um sistema de Banco de Dados, esse arquivo ou diretório é consultado antes que o dado real seja manipulado.

2 Você pode baixar o SGBD e ter mais informações no site: <https://dev.mysql.com/downloads/installer/>

Observe que o cursor mostra a seguinte sintaxe:



Digite \sql para ativar o modo da linguagem SQL. No cursor do terminal deve aparecer da seguinte forma:

A screenshot of a terminal window titled 'MySQL Shell' with version '8.0.17'. The window shows the following text:

```
Copyright (c) 2016, 2019, Oracle and/or its affiliates. All rights reserved.  
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.  
Other names may be trademarks of their respective owners.  
  
Type '\help' or '\?' for help; '\quit' to exit.  
MySQL JS > \sql  
Switching to SQL mode... Commands end with ;  
MySQL SQL >
```

The terminal window has a dark background and light-colored text. The cursor is visible at the end of the command 'MySQL SQL >'.

Dessa forma o terminal estará habilitado a trabalhar com a linguagem SQL, porém é necessária a realização do login como usuário root, pois em um banco de dados, o principal critério é a segurança. Sendo assim para criar uma conexão com nosso usuário devemos digitar:

```
\connect root@localhost
```

E logo em seguida digitar a senha cadastrada quando foi realizado o download.

No terminal é realizada a seguinte pergunta: “Save password for ‘root@localhost’? [Y]es/[N] o/Ne[v]er (default No):” Você pode digitar Y, para deixar salva sua senha, N, para não salvar e sempre realizar este processo toda vez que abrir o terminal e trabalhar com o Banco de Dados.

**AVISO! Não é uma prática interessante armazenar senhas de Banco de Dados, pois pode causar falhas de segurança!**

Para criar as tabelas no Banco de dados, primeiro devemos saber quais são os tipos de dados mais comuns:

Para dados do tipo	Tipo definido	Tamanho
Caracteres	Char(n), varchar (n)	Armazena até n bytes
Numérico exato	Decimal (p,e) ou numeric (p, e)	Depende
Número aproximado	Float, real	8 bytes e 4 bytes
Número inteiro	Int, smallint, tinyint	4 bytes, 2 bytes e 1 byte
Data e hora	Date, Time, smalldatetime	8 bytes, 4 bytes
Texto e imagens	Text, image, ntext	Variável
Monetário	Money, smallmoney	8 bytes, 4 bytes

### **Numéricos (inteiros e exatos):**

Smallint – Guarde os valores numéricos, em 2(dois) bytes, compreendidos entre o intervalo de -32768 a +32767.

Int- (Integer) - Armazena valores numéricos, em quatro bytes binários, compreendidos entre o intervalo -2.147.483.648 a +2.147.483.647.

Tinyint- usa 8 bits (1 byte) números não negativos de 0 a 255.

### **Alfanuméricicos ou caracteres:**

Os tipos Varchar (n) e Char (n) permitem o armazenamento em um campo alfanuméricico de n caracteres, onde n deve ser menor ou igual a 254 caracteres. Por exemplo: o tipo char (30) reserva na memória 30 espaços fixo e o tipo varchar (30) permite uma variação no tamanho, até o limite de 30 caracteres.

### **Campo data**

O tipo Date define um campo que irá guardar datas.

### **Campo Hora**

Para guardar a hora o use o tipo Time.

## Números aproximados

Para valores decimais sem exatidão é possível utilizar o tipo float, mas ele não tem precisão suficiente para vários dígitos.

## Texto e imagem

Os tipos text, guarda texto. É possível armazenar imagens com o tipo image.

Vamos utilizar agora alguns comandos para **criar, alterar e apagar tabelas** com os tipos de itens anteriores para elucidar os conceitos aprendidos com a linguagem SQL utilizando o SGBD MySQL.



Então vamos conhecer os comandos DDL?

Os comandos da DDL SÃO:

- CREATE (criação de estrutura),
- ALTER (alterar estrutura)
- DROP (permite remover ou excluir uma estrutura).

O Primeiro passo, para criar seu Banco de Dados é cria-lo! Vejamos o exemplo de como criar um banco de dados:



```
CREATE DATABASE banco_empresa;
```

Ao digitar o código acima no MySQL, ele irá criar uma base de dados chamada **banco\_empresa**.

Uma observação importante é que um código só será executado e finalizado quando encontrar o ponto-e-vírgula (;).

Antes de criar as tabelas propriamente ditas devemos dar o comando:



```
USE banco_empresa;
```

Com o comando USE, estamos selecionando o Banco de Dados que iremos criar nossas tabelas. Após criar o banco de dados, ele não terá nenhuma tabela com atributos, portanto, o próximo passo é criar as tabelas e suas relações. Veremos agora a sintaxe do comando **Create Table**:



```
CREATE TABLE <nome-tabela>(<nome-coluna> <tipo-dado> [NOT  
NULL],  
  
PRIMARY KEY (<nome-coluna-chave>),  
  
FOREIGN KEY (<nome-coluna-chave-estrangeira>) REFERENCES  
<nome-tabela-origem> ON DELETE [RESTRICT]  
[CASCADE]  
[SET NULL]);
```

Onde (ELMASRI e NAVATHE, 2005).:

- **< nome-tabela >** - Representa o nome da tabela que será criada.
- **< nome-coluna >** - será representado o nome dos campos que serão criados e deve ser colocado um campo após o outro, separado por vírgula.
- **< tipo-do-dado >** - Define o tipo e tamanho dos campos definidos para a tabela.
- **NOT NULL** – para chave primária, essa clausula é obrigatória, pois indica que esse atributo deve ter um conteúdo.

- **NOT NULL WITH DEFAULT** - Preenche o campo com valores pré-definidos, de acordo com o tipo do campo, caso não seja especificado o seu conteúdo no momento da inclusão do registro (ELMASRI e NAVATHE, 2011). Os valores pré-definidos são:
  - Campos numéricos - Valor zero.
  - Campos alfanuméricos - Caractere branco.
  - Campo formato Date - Data corrente.
  - Campo formato Time - Horário no momento da operação.
- **PRIMARY KEY** (nome-coluna-chave-primária): Define qual atributo será a chave primária da tabela. Caso ela tenha mais de uma coluna como chave, elas deverão ser relacionadas entre os parênteses.
- **FOREIGN KEY** (nome-coluna-chave-estrangeira) **REFERENCES** (nome-tabela origem): serve para definição de chaves estrangeiras, ou seja, os campos que são chaves primárias de outras tabelas. Na opção REFERENCES devemos escrever o nome da tabela de onde veio a chave estrangeira, ou seja, a tabela de origem onde o campo era a chave primária (HEUSER,2004).
- **ON DELETE** – caso haja exclusão de um registro na tabela de origem e existe um registro correspondente nas tabelas filhas essa opção diz os procedimentos que devem ser feitos pelo SGBD (SILBERSCHATZ e SUDARSHAN, 2012). As opções disponíveis são:
  - RESTRICT - Opção default (padrão). Não permite a exclusão na tabela de pai (tabela de origem) de um registro, cuja chave primária exista em alguma tabela filha.
  - CASCADE – Realiza a exclusão em todas as tabelas filhas que possuam o valor da chave que será excluída na tabela pai.
  - SET NULL - Atribui o valor NULO nas colunas das tabelas filha que contenham o valor da chave que será excluída na tabela pai.

Para facilitar a visualização de cada uma das nomenclaturas vejamos o exemplo para criar uma tabela de um funcionário:



```

CREATE TABLE funcionario (
    Cod_func INT [NOT NULL],
    nome VARCHAR (20)
    PRIMARY KEY (Cod_func)
    FOREIGN KEY (cod_dept) REFERENCES departamento
    ON DELETE [RESTRICT]);

```

Para o modelo relacional de uma empresa em que temos um ou vários funcionários, que podem ser alocados em vários projetos (Relação (N:M)) temos o seguinte diagrama:

Funcionário	
CODIGO_func	NOME_func
01	João
02	Maria

É ALOCADO	
CODIGO_FUNC	CODIGO_PROJ
01	100
02	200

Projeto	
CODIGO_proj	NOME_proj
100	VIVA
200	NATUREZA

Para criar a tabela Funcionário usamos o seguinte código:



```
CREATE TABLE Funcionario (
    CODIGO_func INTEGER NOT NULL AUTO_INCREMENT,
    NOME_FUNC VARCHAR(20) , PRIMARY KEY(CODIGO_func));
```

Para criar a tabela Alocação usamos o seguinte código:



```
CREATE TABLE Alocacao(
    CODIGO_func INTEGER NOT NULL,
    CODIGO_proj INTEGER NOT NULL,
    FOREIGN KEY(CODIGO_func) REFERENCES
    Funcionario(CODIGO_func),
    FOREIGN KEY(CODIGO_proj) REFERENCES Projeto(CODIGO_
proj));
```

Para criar a tabela Projeto usamos o seguinte código:



```
CREATE TABLE Projeto (
    CODIGO_proj INTEGER NOT NULL AUTO_INCREMENT,
    NOME_proj VARCHAR(30) , PRIMARY KEY(CODIGO_proj));
```

Para o modelo relacional de uma banda e músico (cuja cardinalidade é 1:N) conforme mostra figura abaixo, temos as tabelas:

Banda
Codigo_banda
nome_banda
02

Musico
Codigo_musico
Codigo_banda
Nome_musico
tempo_experiencia



```
CREATE TABLE Banda (
    Código_banda INTEGER NOT NULL AUTO_INCREMENT,
    Nome_banda VARCHAR(20) NULL,
    PRIMARY KEY(Código_banda));
```



```
CREATE TABLE Musico (
    código_musico INTEGER NOT NULL AUTO_INCREMENT,
    Código_bandaFK INTEGER NOT NULL,
    nome_musico VARCHAR(20),
    tempo_experiencia VARCHAR(20),
    PRIMARY KEY(código_musico),
    FOREIGN KEY (Código_bandaFK)
    REFERENCES Banda (Código_banda));
```

Outro comando da linguagem DDL é o comando **ALTER**. Com esse comando podemos alterar a estrutura de uma tabela, por exemplo, adicionar mais atributos a uma tabela (Entidade). Essa cláusula permite acrescentar modificar e alterar nomes e formatos de colunas de tabelas.

Sintaxe:



```
ALTER TABLE <nome-tabela>
ADD <nome-coluna> <tipo-do-dado>
ALTER TABLE <nome-tabela>
MODIFY <nome-coluna> <tipo-do-dado> [NULL]
```

Onde cada campo representa:

- nome-tabela - nome da tabela que será atualizada.
- nome-coluna - nome da coluna que será criada.
- tipo-do-dado - tipo e tamanho dos campos definidos para a tabela.
- ADD <nome-coluna> <tipo-do-dado> - inclusão de uma nova coluna na estrutura da tabela. Na coluna correspondente a esse campo já existente será preenchido o valor NULL (Nulo).
- MODIFY <nome-coluna> <tipo-do-dado> - Permite a alteração na característica da coluna especificada.

#### **Exemplos do comando Alter Table:**

Vamos supor que a empresa precisa do endereço de todos os funcionários. Na tabela Funcionário trabalhada logo acima temos os atributos (CODIGO\_func, NOME\_func) precisamos adicionar o atributo endereço na tabela, digitando o Código abaixo:



```
ALTER TABLE Funcionario
ADD endereço char(20);
```

Caso necessite alterar o tipo de algum atributo de sua tabela, por exemplo, alterar o endereço para varchar(30), então o código seria esse:



```
ALTER TABLE Funcionario MODIFY endereço VARCHAR(30) NOT NULL;
```

E para alterar o nome do atributo, por exemplo, trocar endereço por endereço\_completo, utilizariammos o seguinte comando:



```
ALTER TABLE Funcionario CHANGE endereço endereço_completo  
varchar(30) NOT NULL;
```

Finalizando, para deletar o endereço, ou seja, a empresa não necessita guardar os dados de endereço em seu banco de dados, utilizariammos o seguinte código:



```
ALTER TABLE Funcionario DROP endereço;
```

E a ultima cláusula da linguagem DDL que iremos abordar aqui é o comando que permite deletar a estrutura da tabela do Banco de Dados, ou seja, com esse comando a tabela inteira será excluída e eliminada! Esse é o comando **DROP TABLE <nome-tabela>**.

- Onde: nome-tabela - Representa o nome da tabela que será deletada.

Exemplos do comando **DROP TABLE**, utilizando as tabelas desta Unidade.

**Drop Table** alocacao;

**Drop Table** banda;

**Drop Table** funcionario;

**Drop Table** musico;

**Drop Table** projeto;



Quando utilizar o comando DROP?

Utilize o comando DROP, apenas quando revisar sua tabela e tiver certeza sobre a exclusão dos dados!

Finalizamos essa Unidade do nosso curso que trata sobre os comandos DDL, na próxima Unidade veremos os comandos utilizados para realizar manipulações em um Banco de Dados.

## REFERÊNCIAS

ELMASRI, RAMEZ; NAVATHE, SHAMKANT B. (2005) **Sistemas de Bancos de Dados**. Addison-Wesley, 4a edição em português.

DATE, C. J. **Introdução a Sistemas de Bancos de Dados**, Editora Campus, 2004

ELMASRI, R.; NAVATHE, S. B. **Sistemas de Bancos de Dados**. 6a. Ed. Addison-Wesley, 2011.

HEUSER, C. A. **Projeto de Banco de Dados**, 5a. ed. Sagra Luzzato, 2004.

MACHADO, F.N.R. **Projeto de Banco de Dados: uma visão prática**. 15a Ed. São Paulo: Érica, 2008.

MYSQL, 2019 <<https://www.mysql.com/why-mysql/>>

SILBERSCHATZ, A.; KORF, H. F.; SUDARSHAN, S. **Sistemas de Banco de Dados**, 6a. Ed. Elsevier, 2012.

# UNIDADE 4



# UNIDADE 4

## Linguagem SQL – DML (DATA MANIPULATION LANGUAGE)

Após a leitura deste capítulo, você será capaz de:

- Compreender a linguagem de manipulação de dados SQL
- Aplicar as consultas e manipulações dos dados armazenados com os comandos DML.

### 1 Introdução

Nesta unidade, estudaremos como manipular dados em um Banco de Dados, ou seja, como realizar buscas, inserir dados, atualizar dados, apagar dados.

A instrução utilizada para consultas no banco de dados é a **SELECT**, para realização de inserções de novos dados utilizamos o **INSERT**, para remoção de linhas contendo dados utilizamos o comando **DELETE**, e o **UPDATE**, para atualizações nos dados.

- **INSERT** - INSERÇÃO
- **UPDATE** - ATUALIZAÇÃO
- **DELETE** - APAGAR
- **SELECT** - SELECIONAR

### 2.1 INSERT

Vamos começar com o comando **INSERT**, usado para adicionar dados nas tabelas do Banco de Dados. Com este comando vamos inserir dados em nossas tabelas.

Função:

- Incluir um novo registro em uma tabela do Banco de Dados.

Sintaxe:



```
INSERT INTO <nome-tabela> (<nome-coluna>, <nome-coluna>)  
VALUES (<relação dos valores a serem incluídos>);
```

Onde cada campo representa:

**<nome-tabela>**: o nome da tabela que será incluída como um novo registro.

**<nome-coluna>**: o nome de uma ou mais colunas que terão o conteúdo no momento da operação de inclusão.

Podemos omitir os nomes das colunas, se a ordem dos valores que iremos inserir corresponder a mesma ordem em que foram criados. Caso contrário, é necessário informar todos os campos da tabela.

Para facilitar o entendimento da linguagem vejamos como exemplo adicionar os dados de um Funcionário:

Funcionário
Código: 01
Nome: João
Endereço: Rua Dom Bosco

Para inserir os dados deste novo funcionário no sistema de banco de dados da empresa utilizaremos o seguinte comando:



```
INSERT INTO FUNCIONARIO  
    (CODIGO_func, NOME_func, ENDEREÇO)  
VALUES (1, "João", "Rua Dom Bosco");
```

Seguindo o exemplo do modelo relacional criado na Unidade 2 podemos inserir os dados de uma banda e logo em seguida de um músico:



```
INSERT INTO BANDA (codigo_banda, nome_banda)
```

Código da banda: 1

Nome da Banda: Forever

```
VALUES (1,"FOREVER");
```



```
insert into MUSICO (codigo_musico, Código_bandaFK,
```

Código Musico: 1

Código Banda: 1

Nome Musico: Ananias

Tempo Experiência: 3 anos

```
values (1,1,"Ananias", "3 anos");
```



Acesse a plataforma, e pesquise como inserir um grande volume de dados, por exemplo, grandes empresas armazenam muitos dados e necessitam migrar tais dados de um banco para outro, como isso pode ser feito? Pesquise e encontre a melhor solução para inserção de dados em Banco de Dados .

## 2.2 UPDATE

Após inserir os dados no banco, pode ser necessário realizar atualizações, por exemplo: um funcionário trocou de endereço, ou está trabalhando em outro projeto. Para estes casos, utilizamos o comando de atualização de Banco UPDATE.

Sintaxe:



```
UPDATE <nome-tabela>
SET <nome-coluna> = <novo conteúdo para o campo>
WHERE <condição>
```

Cada campo representa (ELMASRI e NAVATHE, 2005):

**<nome-tabela>** - o nome da tabela que terá seus dados alterados.

**<nome-coluna>** - o nome de uma ou mais colunas que terão o conteúdo alterado.

**<condição>** - a condição para a seleção dos registros que serão atualizados. Podemos indicar uma condição para que um ou vários registros possam ser alterados, quando não se especifica, todos os dados são alterados.

Vejamos um exemplo:

- Atualize o endereço do funcionário João:



```
UPDATE Funcionario
SET endereço = "Rua Pedro de Araújo"
WHERE nome_func = "João";
```

Caso a empresa queira atualizar o salário de todos os funcionários do projeto GEOCIÊNCIAS em 10%. Utilizaremos o seguinte código:



```
UPDATE Funcionario
SET SALARIO = SALARIO * 1.1
WHERE nome_projeto = "GEOCIENCIAS";
```

- Atualize o tempo de experiência do músico Ananias para 4 anos:



```
UPDATE musico  
SET tempo_experiencia = "4 anos"  
WHERE nome_musico = "Ananias";
```



Pesquise e tente implementar os exemplos acima no SGBD MySQL.

## 2.3 DELETE

Quando utilizamos a requisição para remover um dado do banco utilizamos a instrução **DELETE**. Essa requisição é descrita da mesma forma de uma consulta vejamos a sintaxe (SILBERSCHATZ e SUDARSHAN, 2012):

Sintaxe:



```
DELETE FROM <nome-tabela>  
WHERE <condição>
```

Cada campo representa:

**<nome-tabela>** - o nome da tabela que terá seus dados deletados.

**< condição>** - a condição para apagar os dados da tabela, que pode ser um ou vários registros.

Como exemplo, imagine que a mesma empresa que contratou o João para execução de um projeto, ao finalizar este, não necessitou mais dos serviços de João. Dessa forma os dados de João devem ser excluídos do Banco de Dados.



**DELETE FROM** funcionario

**WHERE** nome\_func = "João" **AND** codigo\_func=1;

Outro exemplo seria o músico Ananias decidiu sair da Banda FOREVER, pois irá seguir carreira solo. Os dados dele devem ser retirados do banco de dados da seguinte forma:



**DELETE FROM** musico

**WHERE** nome\_musico = "Ananias" **AND** codigo\_bandaFK=1;



Perceberam algo diferente nos dois últimos códigos?

Utilizamos o operador “AND” para selecionar o funcionário João, pois é necessário informar também qual é o código único deste funcionário, já que no banco de dados podemos ter vários funcionários com o mesmo nome, porém somente um único funcionário terá o código 1 (chave primária); o mesmo caso se aplica para o músico, pois podemos ter vários músicos com o nome Ananias, porém queremos retirar apenas aquele que está na Banda Forever que possui código 1.

A linguagem SQL oferece os seguintes operadores lógicos:

CONECTORES LÓGICOS	
Utilizados para enumerar duas ou mais condições na condição	
OPERADOR	SIGNIFICADO
AND	“E” LÓGICO
OR	“OU” LÓGICO
NOT	NEGAÇÃO



Pesquise e execute os comandos acima utilizando o SGBD MySQL utilizando os conectores lógicos.

## 2.4 SELECT

Vimos como inserir, como atualizar e apagar dados do banco de dados. Agora veremos a cláusula mais utilizada em um sistema de banco de dados: a consulta! Sistemas de empresas, de banco e sites de modo geral necessitam realizar buscas no banco de dados de forma periódica, portanto é muito importante entender como realizar buscas em um banco de dados.

A estrutura básica do comando de busca de banco de dados **SELECT** consiste em três cláusulas (ELMASRI e NAVATHE, 2005):



```
SELECT *  
FROM <nome-tabela>  
WHERE <condição>
```

Em que cada um dos campos corresponde:

**SELECT**: seleciona os atributos desejados na consulta;

<\*> : significa que todos os atributos da tabela selecionada serão retornados; Caso seja especificado aqui os nomes dos atributos ou expressões ou constantes ou funções deve ser conectadas por operadores aritméticos e parênteses.

**<nome-tabela>** - o nome da(s) tabela(s) que possui as colunas que serão selecionadas na consulta.

**<condição>** - condição para selecionar os dados, podendo retornar um ou vários dados.

**< nome-coluna >** - caso não seja utilizado o <\*> devemos especificar os nomes das colunas que queremos retornar os dados.

Quando trabalhamos com consultas, é normal que seja utilizado expressões para retornar os dados solicitados pelo usuário. Temos os seguintes operadores para relacionar duas expressões em SQL:

Operador	Significado
=	Igual
!=	Diferente
<>	Diferente
>	Maior
!>	Não maior (menor ou igual)
<	Menor
!<	Não menor (maior ou igual)
>=	Maior ou igual
<=	Menor ou igual

Veja a seguir como realizar uma consulta para obter os nomes dos funcionários utilizando para isso a tabela FUNCIONARIO:



```
SELECT nome_func  
FROM FUNCIONARIO;
```

Quando precisamos recuperar os dados de mais de uma coluna, devemos simplesmente declarar os nomes das colunas separados por vírgula. Vejamos a seguir, por exemplo, como obter os nomes e endereço de todos os funcionários:



```
SELECT nome_func, endereço  
FROM FUNCIONARIO;
```

Caso, queiramos recuperar apenas os funcionários cujo salario é igual a R\$1000,00, utilizáramos o seguinte comando:



```
SELECT *
FROM FUNCIONARIO
WHERE SALARIO = 1000;
```

Observe que neste caso, apenas faríamos uma consulta e verificaríamos quais são os funcionários com o salario igual a R\$1000,00.

Podemos utilizar a cláusula **WHERE** de diferentes formas, e colocar vários tipos de sentenças que irão depender dos critérios utilizados para busca e podem ser utilizados quaisquer operadores lógicos ou aritméticos. Para a **WHERE**, têm-se outras cláusulas específicas que podem ser utilizadas para aprimorar as buscas. São elas:

- **DISTINCT** <nome-coluna>
- **GROUP BY** <nome-coluna>
- **HAVING** <condição>
- **ORDER BY** <nome-campo> ASC/DESC

Cada uma dessas cláusulas pode realizar as seguintes operações:

**DISTINCT:** retira dados repetidos de uma busca, quando utilizado.

**GROUP BY:** esta cláusula só pode ser usada em conjunto com o HAVING, pois ela agrupa os campos especificados pela coluna indicada.

**HAVING:** diz qual será a condição para agrupamento dos dados da cláusula do GROUP BY.

**ORDER BY:** ordena os dados de uma busca de forma crescente (ASC) ou decrescente (DESC).

Veremos agora exemplos de cada uma dessas cláusulas, utilizando a tabela Funcionário. Vamos buscar em nosso banco de dados todos os nomes de projetos sem repetições. Para isso utilizaremos o seguinte comando:



```
SELECT DISTINCT nome_projeto
FROM FUNCIONARIO;
```

Agora imagina que no sistema da Empresa seja necessária a realização de uma agenda de funcionários em ordem alfabética, dessa forma, utilizariamos o seguinte comando:



```
SELECT nome_func  
FROM FUNCIONARIO  
ORDER BY nome_func ASC ;
```

Da mesma forma em que é possível ordenar os nomes dos funcionários em ordem alfabética também é possível ordenar os nomes dos projetos em ordem decrescente:



```
SELECT nome_proj  
FROM FUNCIONARIO  
ORDER BY nome_proj DESC ;
```

Outra clausula comumente usada em consultas de banco de dados é o agrupamento de dados, onde podemos verificar e agrupar, por exemplo, os funcionários que estão trabalhando com uma carga horária superior a 30 horas semanais:



```
SELECT nome_func  
FROM FUNCIONARIO  
GROUP BY carga_horaria  
HAVING carga_horaria>'30 horas' ;
```

Nesta unidade, vimos o conteúdo básico sobre manipulação de banco de dados, agora, acesse o Saiba Mais. veja o material adicional e faça os exercícios para melhor aprendizado.

## Referências

SILBERSCHATZ, A.; KORF, H. F.; SUDARSHAN, S. **Sistemas de Banco de Dados**, 6a. Ed. Elsevier, 2012.

ELMASRI, RAMEZ; NAVATHE, SHAMKANT B. (2005) **Sistemas de Bancos de Dados**. Addison-Wesley, 4a edição em português.



UNIVERSIDADE FEDERAL  
DE MATO GROSSO



**SETEC**  
SECRETARIA DE  
TECNOLOGIA EDUCACIONAL



# **Livro - Banco de Dados**

---

<https://biblioteca-a.read.garden/viewer/9786556900186/capa>

# Vídeo 01 - Introdução a Modelagem de Dados

---

## Vídeo 01

Introdução a Modelagem de Dados



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para accesar.

# Vídeo 02 - Modelo entidade-relacionamento e Diagrama ER

---

## Vídeo 02

Modelo entidade-relacionamento e diagrama ER



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para acessar.

# Vídeo 03 - Normalização de Dados - 1FN

## **Vídeo 03**

Normalização de Dados - Primeira Forma Normal.



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

# Vídeo 04 - 2<sup>a</sup> Forma Normal

## **Vídeo 04**

Normalização de Dados - Segunda Forma Normal



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

# Vídeo 05 - 3<sup>a</sup> Forma Normal

## **Vídeo 05**

Normalização de Dados - Terceira Forma Normal



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

# Vídeo 06 - 4<sup>a</sup> Forma Normal

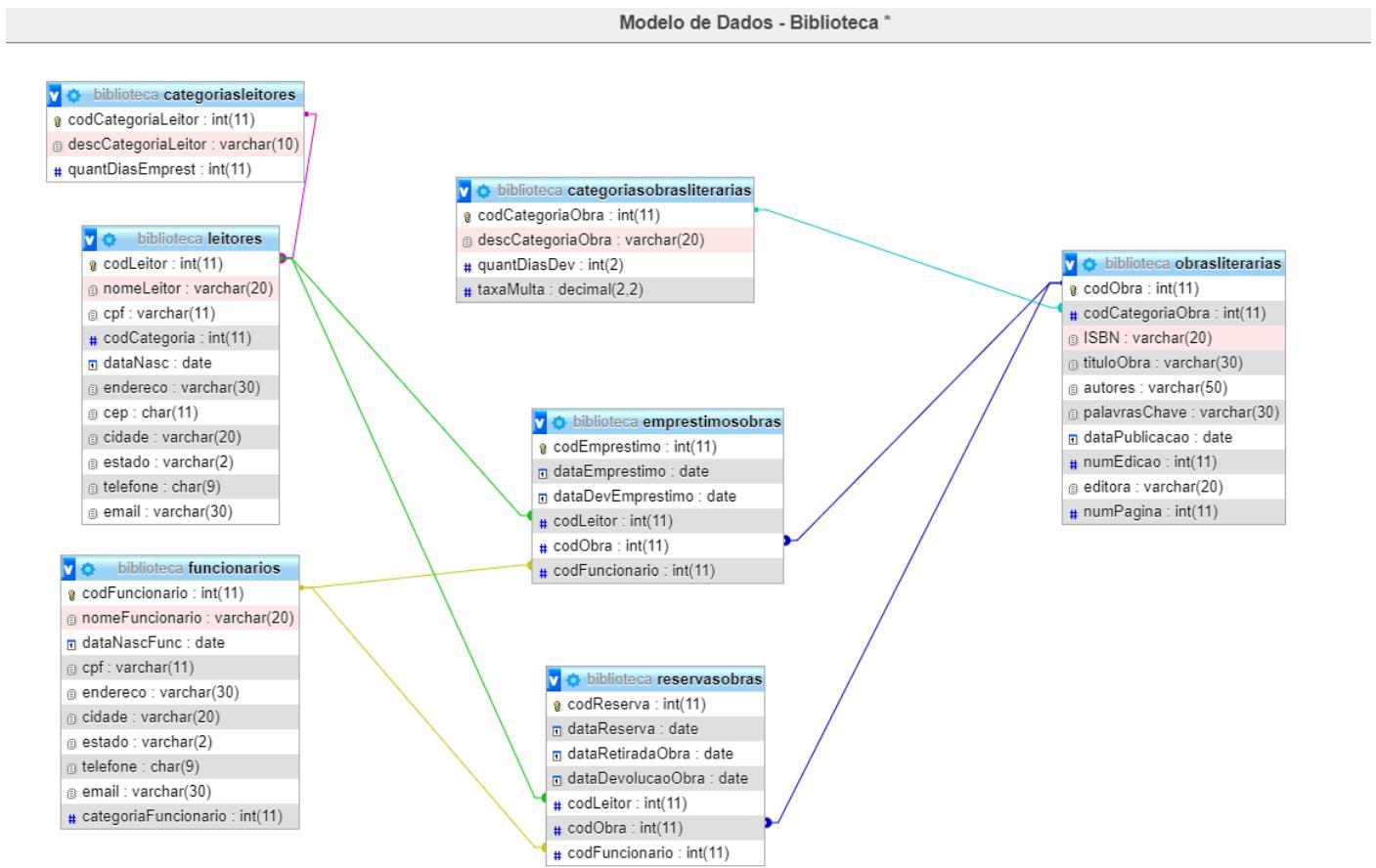
## **Vídeo 06**

4<sup>a</sup> Forma Normal



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.

# Modelo de dados Biblioteca



# **Base de dados Biblioteca**

---

Base de Dados - Biblioteca

# Vídeo 07 - Importando a base de dados

## Biblioteca

---

### Vídeo 07

Importando a base de dados.



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para acessar.

# Download WamppServer

---

Download do servidor WamppServer - sugestão para criar o banco de dados SQL.

<https://www.wampserver.com/en/download-wampserver-64bits/>

Como instalar: <https://www.youtube.com/watch?v=1cnXP8uw5gI>

# Vídeo 08 - Criando uma base de dados SQL

---

## Criando uma base de dados SQL



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para accesar.

# Vídeo 09 - Inserindo dados no Banco de Dados

---

## Inserção de dados no banco de dados



Aponte a câmera para o código e accese o link do conteúdo ou clique no código para accesar.

# **JOGO - BANCO DE DADOS**

---

Teste os seus conhecimentos:

Jogo: <https://wordwall.net/pt/resource/59572584/jogo-associa%C3%A7%C3%A3o-banco-de-dados>

# APOSTILA SQL

# A04 SQL Básica

Prof. Dr. George H. G. Fonseca

CDD003 Fundamentos de Banco de Dados  
Pós Graduação em Ciência dos Dados  
Universidade Federal de Ouro Preto

Março de 2020



UFOP  
Universidade Federal  
de Ouro Preto



UFOP  
Universidade Federal  
de Ouro Preto

# Sumário

- 1 Introdução
- 2 Definição e Tipos de Dados SQL
- 3 Consultas SQL
  - Nomes de Atributos Ambíguos, Pseudônimo e Variáveis de Tupla
  - Nomes de Atributos Ambíguos, Pseudônimo e Variáveis de Tupla
  - Cláusula WHERE Não Especificada e Asterisco
  - Tabelas como Conjuntos em SQL
  - Correspondência de Padrão de Substring
  - Operações Aritméticas
  - Ordenando os Resultados de uma Consulta
  - Funções de Agravação
  - Agrupamento: Cláusulas GROUP BY e HAVING
  - Estrutura Geral de uma Consulta SQL
- 4 Declarações INSERT, DELETE e UPDATE
- 5 Referências

# Introdução

- *Structured Query Language (SQL)* é a linguagem padrão para manipular bancos de dados relacionais.
- A SQL foi desenvolvida nos anos 70 pela IBM e foi adotada por várias outras companhias que desenvolvem SGBDs.

# Definição e Tipos de Dados SQL

- SQL usa os termos **tabela**, **linha** e **coluna** para os termos relacionais relação, tupla e atributo, respectivamente.
- O principal comando SQL para definição de dados é o CREATE, que pode ser usado para criar esquemas, tabelas, tipos dentre outros.

# Definição e Tipos de Dados SQL

```
CREATE SCHEMA UNIVERSIDADE;
```

```
CREATE TABLE UNIVERSIDADE.ALUNO
(
    Matricula          INT          NOT NULL,
    Nome               VARCHAR(45)  NOT NULL,
    DataNascimento    DATE        NOT NULL,
    Endereco           VARCHAR(45)  NOT NULL,
    CCodigo            INT          NOT NULL,
    PRIMARY KEY (Matricula),
    FOREIGN KEY (CCodigo) REFERENCES CURSO(Codigo)
);
```

# Definição e Tipos de Dados SQL

- Os comandos DROP e ALTER são responsáveis por excluir e alterar uma tabela respectivamente.
- Esses comandos não serão abordados em detalhes pois ferramentas CASE apresentam a funcionalidade de gerar e executar esses comandos de forma intuitiva via interfaces gráficas.

```
DROP TABLE ALUNO;
```

```
ALTER TABLE PROFESSOR  
ADD COLUMN 'Endereco' VARCHAR(45) NOT NULL AFTER 'Salario';
```

# Consultas SQL

- Consultas mais simples serão apresentadas e gradualmente consultas mais complexas serão introduzidas.
- A forma básica da declaração SELECT é formada pelas três cláusulas SELECT, FROM, WHERE e tem a seguinte forma:

```
SELECT <lista de atributos>
FROM   <lista de tabelas>
WHERE  <condicoes>;
```

- Em SQL, os operadores lógicos básicos de comparação são  $=$ ,  $<$ ,  $\leq$ ,  $>$ ,  $\geq$  e  $\neq$ . Eles correspondem respectivamente aos operadores de álgebra relacional  $=$ ,  $<$ ,  $\leq$ ,  $>$ ,  $\geq$  e  $\neq$ .

# Consultas SQL

**Consulta 1.** Recuperar a data de nascimento e salário dos professores cujo nome é 'George Fonseca'.

```
SELECT DataNascimento , Salario  
FROM PROFESSOR  
WHERE Nome='George Fonseca ';
```

## Consultas SQL

**Consulta 2.** Recuperar o CPF e o salário de todos os professores que trabalham nos departamento cujo campus é 'Ipatinga'.

```
SELECT CPF, Salario  
FROM PROFESSOR, DEPARTAMENTO  
WHERE Campus='Ipatinga' ANDCodigo=DCodigo;
```

- Na cláusula WHERE da Consulta 2 a condição Campus='Ipatinga' é uma **condição de seleção** que seleciona a tupla particular de interesse da tabela DEPARTAMENTO, pois Campus é um atributo de DEPARTAMENTO.
- A condição Codigo=DCodigo é chamada **condição de junção**, pois combina duas tuplas: uma de DEPARTAMENTO e outra de PROFESSOR sempre que o valor de Codigo em DEPARTAMENTO é igual ao valor de DCodigo em PROFESSOR.

# Nomes de Atributos Ambíguos, Pseudônimo e Variáveis de Tupla

- Na SQL o mesmo nome pode ser usado para mais de um atributo em diferentes tabelas.
- Se esse for o caso e uma consulta multi-tabela se refere a dois ou mais atributos com o mesmo nome, devemos qualificar o nome do atributo de acordo com o nome da tabela para prevenir ambiguidade.
- Isso é feito prefixando o nome da tabela ao nome do atributo, separando-os por ponto.

```
SELECT PROFESSOR.CPF, PROFESSOR.Salario  
FROM PROFESSOR, DEPARTAMENTO  
WHERE DEPARTAMENTO.Campus='Ipatinga' AND  
DEPARTAMENTO.Codigo=PROFESSOR.DCodigo ;
```

# Nomes de Atributos Ambíguos, Pseudônimo e Variáveis de Tupla

- É possível ainda renomear nomes de tabelas para nomes mais curtos criando um **pseudônimo**, como no exemplo da Consulta 3.
- De fato essa prática é recomendável para facilitar a **legibilidade** das consultas SQL.

**Consulta 3.** Recupere o código e o nome das disciplinas que pertencem ao curso de 'Medicina'.

```
SELECT D.Codigo , D.Nome  
FROM DISCIPLINA AS D, CONTEM AS C, CURSO AS CS  
WHERE D.Codigo=C.DCodigo AND CS.Codigo=C.CCodigo  
      AND C.Nome='Medicina';
```

## Cláusura WHERE Não Especificada e Asterisco

- Uma cláusula WHERE faltando significa que não há condição na seleção das tuplas, assim, todas as tuplas especificadas na cláusula FROM serão selecionadas.

**Consulta 4.** Selecione os CPFs de todos os professores.

```
SELECT    CPF
FROM      PROFESSOR;
```

**Consulta 5.** Selecione os CPFs de todos os empregados e todas as combinações de CPFs de professores e nomes de departamentos.

```
SELECT    E.CPF, D.Nome
FROM      EMPREGADO AS E, DEPARTAMENTO AS D;
```

## Cláusura WHERE Não Especificada e Asterisco

- Para recuperar os valores de todos os atributos das tuplas selecionadas não é necessário listar todos os atributos na SQL, é possível especificar apenas um asterisco (\*).

**Consulta 6.** Recuperar todos os atributos dos professores que trabalham no DEPARTAMENTO cujo código é 'DECOM'.

```
SELECT *
FROM   PROFESSOR
WHERE  DCodigo='DECOM'
```

## Cláusura WHERE Não Especificada e Asterisco

**Consulta 7.** Recuperar todos os atributos dos professores e os atributos do DEPARTAMENTO em que eles trabalham para cada empregado do departamento 'Computacao'.

```
SELECT P.*  
FROM PROFESSOR AS P, DEPARTAMENTO AS D  
WHERE P.DCodigo=D.Codigo AND D.Nome='Computacao';
```

# Tabelas como Conjuntos em SQL

- SQL não trata uma tabela como um conjunto, e sim como um multi-conjunto, assim tuplas duplicadas podem aparecer como resultado de uma consulta.
- Se deseja-se eliminar duplicatas dos resultados de uma consulta SQL a palavra-chave DISTINCT deve ser usada na cláusula SELECT, especificando que apenas tuplas distintas devem permanecer no resultado.

**Consulta 8.** Recuperar todos os salários distintos de todos os professores.

```
SELECT DISTINCT Salario  
FROM PROFESSOR;
```

# Tabelas como Conjuntos em SQL

- A SQL incorporou diretamente algumas das operações da teoria dos conjuntos, como operações de união (UNION), diferença (DIFFERENCE) e interseção (INTERSECT).
- Os resultados dessas operações são conjuntos de tuplas, assim, duplicatas são eliminadas do resultado.
- As operações sobre conjuntos se aplicam apenas a tabelas com tipos compatíveis, ou seja, as tabelas para as quais a operação está sendo processada têm que ter os mesmos atributos e na mesma ordem.

# Tabelas como Conjuntos em SQL

**Consulta 9.** Listar todos os nomes de todas as pessoas (professores e alunos) que participaram da oferta de disciplinas no ano de 2020.

```
(SELECT P.Nome
  FROM PROFESSOR AS P, OFERTA AS O
 WHERE P.CPF=O.PCPF AND O.Ano=2020)
UNION
(SELECT A.Nome
  FROM ALUNO AS A, OFERTA AS O
 WHERE A.Matricula=O.AMatricula AND O.Ano=2020);
```

# Correspondência de Padrão de Substring

- Uma funcionalidade da SQL permite a comparação de apenas partes de caracteres de uma string, usando o operador de comparação LIKE.
- Isso pode ser usado para **correspondência de padrões de strings**.
- Strings parciais são especificadas usando dois caracteres especiais: % substitui um número arbitrário de zero ou mais caracteres e sublinhado (\_) substitui um único caractere.

## Correspondência de Padrão de Substring

**Consulta 10.** Recuperar todos alunos cujo endereço contém 'Ipatinga'.

```
SELECT    Nome  
FROM      ALUNO  
WHERE    Endereco LIKE '%Ipatinga%';
```

**Consulta 11.** Recuperar todos os professores nascidos nos anos 80.<sup>1</sup>

```
SELECT    Nome  
FROM      PROFESSOR  
WHERE    DataNascimento LIKE '_ _8 _ _ _ _ _';
```

---

<sup>1</sup>O formato de data empregado na SQL é aaaa-mm-dd

# Operações Aritméticas

- Outra característica da SQL permite o uso de operações aritméticas nas consultas.
- Os operadores aritméticos padrão para adição (+), subtração (-), multiplicação (\*) e divisão (/) podem ser aplicados a valores numéricos ou atributos com domínios numéricos.

**Consulta 12.** Mostrar os salários resultantes se cada professor que do deptamento 'Computacao' recebesse um aumento de 10%.

```
SELECT P.Nome, P.Salario * 1.1 AS SalAumentado  
FROM PROFESSOR AS P, DEPARTAMENTO AS D  
WHERE D.Codigo=P.DCodigo AND D.Nome='Computacao';
```

# Ordenando os Resultados de uma Consulta

- A SQL permite ao usuário ordenar as tuplas resultantes de uma consulta pelos valores de um ou mais dos atributos que aparecem no resultado da consulta usando a cláusula ORDER BY.

**Consulta 13.** Recuperar a lista dos professores e dos departamentos nos quais eles estão trabalham ordenados pelo nome do departamento e, dentro de cada departamento, ordenar alfabeticamente pelo nome.

```
SELECT      D.Nome, P.Nome  
FROM        DEPARTAMENTO AS D, PROFESSOR AS P  
WHERE       D.Codigo=P.DCodigo  
ORDER BY    D.Nome, P.Nome;
```

# Ordenando os Resultados de uma Consulta

- A ordem padrão é a ordem ascendente dos valores.
- Pode-se especificar a palavra-chave DESC se deseja-se ver os resultados na ordem decrescente de valores.
- É possível ainda limitar o número de tuplas retornadas com a cláusula LIMIT para qualquer tipo de consulta.
- Por exemplo, caso se desejasse os nomes dos departamentos em ordem decrescente a cláusula ORDER BY ficaria:

```
ORDER BY D.Nome DESC, P.Nome  
LIMIT      5;
```

# Funções de Agregação

- **Funções de agregação** são usadas para summarizar informações de múltiplas tuplas em uma única-tupla sumário.
- Várias funções de agregação existem como a contagem de elementos (COUNT), o mínimo (MIN) e máximo (MAX) de uma série de valores, a soma (SUM) e a média (AVG) de um conjunto de dados.
- Há ainda funções de agregação para cálculos estatísticos mais avançados, porém estas não serão abordadas aqui.
- Essas funções podem ser usadas em uma cláusula SELECT.

## Funções de Agregação

**Consulta 14.** Recuperar a soma dos salários de todos os professores, o salário máximo, o salário mínimo e o salário médio.

```
SELECT    SUM (Salario), MIN (Salario),
          MAX (Salario), AVG (Salario)
FROM      PROFESSOR;
```

- Pode-se usar a palavra-chave AS para criar renomear os nomes das colunas resultantes da consulta:

```
SELECT    SUM (Salario) AS TotalSal ,
          MIN (Salario) AS MinSal ,
          MAX (Salario) AS MaxSal ,
          AVG (Salario) AS MedSal
FROM      PROFESSOR;
```

## Funções de Agregação

**Consulta 15.** Recuperar o número total de professores que trabalham no departamento 'Computacao'.

```
SELECT COUNT (*)
FROM PROFESSOR AS P, DEPARTAMENTO AS D
WHERE D.Codigo=P.DCodigo AND D.Nome='Computacao';
```

# Agrupamento: Cláusulas GROUP BY e HAVING

- Muitas vezes deseja-se aplicar funções de agregação a subgrupos de tuplas em uma relação, onde esses subgrupos são baseados em alguns valores de atributos.
- Por exemplo, se desejarmos encontrar o salário médio dos empregados em cada departamento.
- Nesses casos precisa-se particionar as tabelas em conjuntos sem sobreposição (ou grupos) de tuplas.
- Cada grupo consistirá das tuplas que têm o mesmo valor para um determinado **atributo de agrupamento**.
- A cláusula GROUP BY tem essa finalidade. A cláusula GROUP BY especifica os atributos de agrupamento, que devem também aparecer na cláusula SELECT.

## Agrupamento: Cláusulas GROUP BY e HAVING

**Consulta 16.** Para cada departamento, recuperar o código do departamento, o número de professores no departamento e seu salário médio.

```
SELECT      DNumero , COUNT (*) , AVG ( Salario )
FROM        PROFESSOR
GROUP BY    DNo;
```

# Agrupamento: Cláusulas GROUP BY e HAVING

- Algumas vezes deseja-se recuperar valores para funções de agregação apenas para grupos que satisfazem determinada condição. Isso pode ser feito pela cláusula HAVING.

**Consulta 17.** Para cada departamento, no qual mais de dois professores trabalham, recuperar o código do departamento, o nome do departamento e o número de professores que nele trabalham.

```
SELECT      D.Codigo , D.Nome, COUNT (*)  
FROM        PROFESSOR, DEPRTAMENTO  
WHERE       D.Codigo=P.DCodigo  
GROUP BY    D.Codigo , D.Nome  
HAVING     COUNT (*) > 2;
```

# Estrutura Geral de uma Consulta SQL

- Em qualquer consulta SQL as cláusulas **SELECT** e **FROM** são **obrigatórias**. As cláusulas **WHERE**, **GROUP BY**, **ORDER BY**, **HAVING** e **LIMIT** são **opcionais**, porém, caso presentes, devem ocorrer nessa ordem.

```
SELECT      <lista de atributos>
FROM        <lista de tabelas>
| WHERE     <condições>
| GROUP BY <atributos>
| ORDER BY <atributos>
| HAVING    <condições>
| LIMIT     <número de linhas>;
```

# Declarações INSERT, DELETE e UPDATE

- O comando INSERT adiciona uma tupla a uma tabela.
- Deve-se especificar o nome da relação e um lista de valores para a tupla.
- Esses valores devem estar na mesma ordem em que os atributos foram especificados no comando CREATE TABLE.

```
INSERT INTO ALUNO VALUES
(
    '2018005',
    'Mariana Gomes',
    '2001-02-28',
    'Rua Pedro A. Cabral, Belo Horizonte - MG',
    '1'
);
```

# Declarações INSERT, DELETE e UPDATE

- O comando DELETE remove uma tupla de uma tabela.
- Ele inclui uma cláusula WHERE, de forma similar às consultas, para selecionar as tuplas a serem deletadas.

```
DELETE FROM ALUNO  
WHERE Nome='Caio Nunes';
```

# Declarações INSERT, DELETE e UPDATE

- O comando UPDATE é usado para modificar os dados de uma ou mais tuplas de uma tabela.
- Assim como no comando DELETE, uma cláusula WHERE deve ser especificada para definir quais tuplas sofrerão atualização.

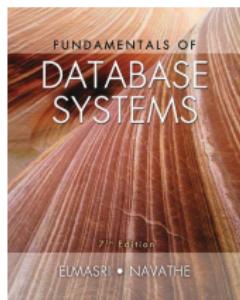
```
UPDATE      ALUNO  
SET          Endereco='Av. Ipanema , Itabira – MG'  
WHERE       Nome='Ana Pedrosa ';
```

# Declarações INSERT, DELETE e UPDATE

- Múltiplas tuplas podem ser atualizadas ao mesmo tempo em um único comando UPDATE, como no exemplo abaixo, que aumenta o salário dos professores do departamento de código '3' em 5%.

```
UPDATE PROFESSOR  
SET Salario=Salario *1.05  
WHERE DCodigo='3';
```

# Referências



# **CARDINALIDADE EM UM SGBD**

---

## CARDINALIDADE EM UM SGBD

A **cardinalidade**, em um Sistema de Gerenciamento de Banco de Dados (SGBD), refere-se à relação entre as entidades em um modelo de dados. Ela indica quantas ocorrências de uma entidade podem estar associadas a um certo número de ocorrências de outra entidade por meio de relacionamentos. A cardinalidade é uma parte essencial do projeto de banco de dados e ajuda a definir como as tabelas se relacionam entre si.

Existem dois tipos principais de cardinalidade: **cardinalidade mínima** e **cardinalidade máxima**.

### **Cardinalidade Mínima:**

A cardinalidade mínima indica o número mínimo de ocorrências que uma entidade pode ter em relação a outra entidade em um relacionamento. Ela é representada por símbolos ou palavras-chave, geralmente:

0: Indica que uma entidade pode não estar associada a nenhuma ocorrência da outra entidade.

1: Indica que uma entidade deve estar associada a pelo menos uma ocorrência da outra entidade.

### **Cardinalidade Máxima:**

A cardinalidade máxima indica o número máximo de ocorrências que uma entidade pode ter em relação a outra entidade em um relacionamento. Novamente, é representada por símbolos ou palavras-chave:

0 ou 1: Indica que uma entidade pode estar associada a no máximo uma ocorrência da outra entidade.

1: Indica que uma entidade está associada exatamente a uma ocorrência da outra entidade.

0..n: Indica que uma entidade pode estar associada a várias ocorrências da outra entidade.

1..n: Indica que uma entidade está associada a pelo menos uma ocorrência da outra entidade, mas pode estar associada a várias.

Um exemplo de como isso pode ser aplicado é considerando uma relação entre duas entidades: "Cliente" e "Pedido". A cardinalidade pode ser definida da seguinte forma:

Um cliente pode fazer zero ou mais pedidos: Cliente (0..n) - Pedido (1..n).

Um pedido deve estar associado a exatamente um cliente: Pedido (1) - Cliente (1).

A definição correta da cardinalidade ajuda a garantir a integridade dos dados e a modelar as relações entre as entidades de forma precisa dentro de um banco de dados.

No PostgreSQL, a cardinalidade é geralmente definida ao criar ou modificar tabelas por meio da linguagem SQL. Aqui está como você pode definir a cardinalidade em um relacionamento usando o PostgreSQL:

**Suponha que você tenha duas tabelas: "Cliente" e "Pedido", e deseja definir a cardinalidade entre elas.**

-- Criando a tabela Cliente

```
CREATE TABLE Cliente (
    id_cliente SERIAL PRIMARY KEY,
    nome VARCHAR(100)
);
```

-- Criando a tabela Pedido

```
CREATE TABLE Pedido (
    id_pedido SERIAL PRIMARY KEY,
    id_cliente INT REFERENCES Cliente(id_cliente),
    descricao TEXT
);
```

Neste exemplo, o relacionamento entre "Cliente" e "Pedido" é definido através da coluna "id\_cliente" na tabela "Pedido" que faz referência à coluna "id\_cliente" na tabela "Cliente". Isso estabelece um relacionamento entre as tabelas.

Agora, para definir a cardinalidade mínima e máxima, você precisa entender o contexto das suas necessidades específicas. No exemplo acima, a cardinalidade é a seguinte:

**Um cliente pode fazer zero ou mais pedidos: Cliente (0..n) - Pedido (1..n).**

**Um pedido deve estar associado a exatamente um cliente: Pedido (1) - Cliente (1).**

Observe que a cardinalidade mínima é expressa pela coluna que é uma chave estrangeira (**REFERENCES Cliente(id\_cliente)**), que define a relação mínima de um pedido com um cliente. A cardinalidade máxima é determinada pelo uso da chave estrangeira e a maneira como as associações são estabelecidas.

Lembre-se de que a definição da cardinalidade pode variar com base nas regras de negócios específicas do seu aplicativo e nos requisitos do banco de dados.

A **cardinalidade máxima na chave estrangeira** é determinada pela forma como você configura as restrições de integridade referencial na tabela. No PostgreSQL, você pode usar as **palavras-chave ON DELETE e ON UPDATE** junto com a definição da chave estrangeira para controlar a cardinalidade máxima e o que acontece quando registros relacionados são excluídos ou atualizados.

Vou apresentar alguns exemplos para esclarecer como definir a cardinalidade máxima na chave estrangeira:

Um cliente pode fazer zero ou mais pedidos (Cliente 0..n - Pedido 1..n):

```
CREATE TABLE Cliente (
    id_cliente SERIAL PRIMARY KEY,
    nome VARCHAR(100)
);
```

```
CREATE TABLE Pedido (
    id_pedido SERIAL PRIMARY KEY,
    id_cliente INT REFERENCES Cliente(id_cliente) ON DELETE CASCADE, -- DELETE CASCADE significa que se um cliente for excluído, seus pedidos também serão excluídos.
    descricao TEXT
);
```

Nesse caso, estamos **usando ON DELETE CASCADE** para indicar que, se um cliente for excluído, todos os seus pedidos associados também serão excluídos. Isso está alinhado com a cardinalidade máxima de 1..n entre Cliente e Pedido.

Um pedido deve estar associado a exatamente um cliente (Pedido 1 - Cliente 1):

```
CREATE TABLE Cliente (
    id_cliente SERIAL PRIMARY KEY,
    nome VARCHAR(100)
);
```

```
CREATE TABLE Pedido (
    id_pedido SERIAL PRIMARY KEY,
    id_cliente INT REFERENCES Cliente(id_cliente) ON DELETE RESTRICT, -- RESTRICT significa que a exclusão do cliente é impedida se houver pedidos associados.
    descricao TEXT
);
```

Nesse exemplo, usamos **ON DELETE RESTRICT** para indicar que a exclusão de um cliente é restrita se houver pedidos associados a ele. Isso está alinhado com a cardinalidade máxima de 1 entre Pedido e Cliente.

Lembre-se de que as opções **ON DELETE** e **ON UPDATE** permitem controlar o que acontece quando há modificações nas chaves estrangeiras (quando um registro referenciado é excluído ou atualizado). As opções disponíveis incluem **CASCADE**, **RESTRICT**, **SET NULL** e **SET DEFAULT**, dependendo do comportamento desejado.

# SQL - JOIN

## SQL - JOIN



Aponte a câmera para o código e acesse o link do conteúdo ou clique no código para acessar.